Published in final edited form as:

Expert Syst Appl. 2015 July 15; 42(12): 5245–5255. doi:10.1016/j.eswa.2015.02.026.

Optimization Model for Web Based Multimodal Interactive Simulations

Tansel Halic^{*}, Woojin Ahn⁺, and Suvranu De⁺

Tansel Halic: tanselh@uca.edu; Woojin Ahn: ahnw@rpi.edu; Suvranu De: des@rpi.edu

^{*}Computer Science Department, University of Central Arkansas, 201 Donaghey Ave., Conway, Arkansas, 72035

⁺Department of Mechanical, Aerospace and Nuclear Engineering, Rensselaer Polytechnic Institute, Troy, New York, 12180

Abstract

This paper presents a technique for optimizing the performance of web based multimodal interactive simulations. For such applications where visual quality and the performance of simulations directly influence user experience, overloading of hardware resources may result in unsatisfactory reduction in the quality of the simulation and user satisfaction. However, optimization of simulation performance on individual hardware platforms is not practical. Hence, we present a mixed integer programming model to optimize the performance of graphical rendering and simulation performance while satisfying application specific constraints. Our approach includes three distinct phases: *identification*, *optimization* and *update*. In the *identification* phase, the computing and rendering capabilities of the client device are evaluated using an exploratory proxy code. This data is utilized in conjunction with user specified design requirements in the *optimization* phase to ensure best possible computational resource allocation. The optimum solution is used for rendering (e.g. texture size, canvas resolution) and simulation parameters (e.g. simulation domain) in the *update* phase. Test results are presented on multiple hardware platforms with diverse computing and graphics capabilities to demonstrate the effectiveness of our approach.

Keywords

Interactive Web environments for Medicine; Optimization; Simulation; Virtual reality; Web-based interaction

Introduction

Multimodal interactive simulations (MIS) are highly demanding in terms of computational resources. Such systems have wide applications in medical training, flight simulation, video

© 2015 Published by Elsevier Ltd.

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

gaming, movies and other industries. MIS systems especially virtual reality applications require multiple components to work synergistically to achieve a high degree of immersion. These may include realistic visual rendering engine to display photo-realistic images and physics engine to simulate real world phenomena including rigid body motion, deformation of solids, fluid flow, and multi-physics phenomenon. Multimodal interactions require interfaces such as mouse, trackballs, haptic and tracking devices. Enabling interactions with multiple hardware devices while maintaining expected realism for both visualization and simulation require complex algorithms and considerable CPU time. However, faster execution rate is necessary in applications where real-time interactivity is mandatory. This is especially challenging when multiple hardware platforms must be supported with diverse hardware specifications.

Unfortunately, conventional MIS systems are designed to work with specific software and hardware settings. Enabling cross-platform compatibility (Halic et al., 2011; Maciel et al., 2010) may not be the right approach as it requires *a priori* knowledge of the platform, intricate compilation and installation procedures (Engelke et al., 2013). This greatly limits accessibility, usage and portability. Besides, such an approach may not be feasible on proprietary target platforms. On the other hand, rapid growth of web and ubiquitous computing platforms such as tablets and smart phones have facilitated the accessibility, mobility and usage of applications regardless of the underlying platform. To access such advantages, applications should be platform independent and be accessible over the World Wide Web. The web introduces greater flexibility as the applications do not need to be installed as in the case of traditional native applications. MIS systems on the web also allow for low development and maintenance costs. This requires the use of open standards of the web (e.g. HTML5). Device anonymity in terms of hardware and the software needs to be maintained. Hence, migrating MIS to the web offers true portability and platform accessibility along with widespread usage.

The Software Framework for Multimodal Interactive Simulations (SoFMIS) is a platform-independent framework designed to facilitate development of 3D interactive applications over the web using WebGL (Halic et al., 2012). WebGL is a plug-in free JavaScript based visualization technology, released in 2011 (now part of HTML5 standard) allows for the development of realistic and interactive 3D applications on the web browsers (“WebGL Specification”, 2012). Applications based on SoFMIS can therefore operate on multiple web browsers running on a range of hardware platforms including desktops and mobile computing devices. The performance of visualizations and simulations depends heavily on the hardware platform, which significantly affects user experience. With hardware upgrade cycles becoming shorter, designing an application that can capitalize on the prospective upgrades becomes even more challenging.

Platform independent applications not only need to detect target hardware but also adjust their performance to future changes of hardware. This process is often more challenging than the development of the application itself. Hence, it is essential to develop a strategy that allows automatic detection of target hardware resources and optimize resources to run most efficiently on the device. In this paper, we address this issue by developing a strategy that is

capable of computing visualization and simulation parameters based on the client hardware while respecting the constraints set by the application.

One common use of MIS systems is Virtual Reality (VR) based web applications (Arnab et al., 2011). In VR applications, the realism of the application is highly affected by the execution constraints. For instance, VR based simulation with haptic interface devices; force feedback computation needs to be performed at 1Khz execution rate for smooth touch sensation. Visual rendering rates entails execution rate at least 30Hz for real-time interactivity. Any failure to meet these execution constraints due to lack of optimization of the device computing resources drastically reduces the user experience, which causes disruption in the sense of immersion.

Our performance optimization framework models the performance of visualization and simulation using non-linear mixed integer programming (Floudas, 1995). This model satisfies the visualization and simulation performance constraints while maximizing the visual quality for a specific device. The optimization does not make any assumption on hardware capabilities. The device capabilities are parameterized and extracted in the identification phase, which respects device anonymity. The subsequent pre-optimization phase of the simulation is performed on the server side resulting in elimination of overhead on the client device. This makes the approach more suitable for low profile devices such as smart phones.

Our optimization model also takes into consideration the constraints set on the quality of visual rendering and performance of physical simulations at the application level. Factors such as load due to visualization and the programmability of GPU are also considered. Texture sizes, display resolution, canvas up-scaling ratio and geometry discretization level are determined as outputs. Computing the visualization and simulation parameters before the start of simulation prevents any unwanted effects during the application execution such as visual artifacts existing in previously proposed real-time approaches. Unlike existing approaches our model also accounts for run-time performance of simulation to avoid slowdowns, lag or freeze due to scarce CPU resources. Our optimization approach has been implemented and tested in the context of SoFMIS. However, the methods used here may also be applicable to other software environments.

1. Literature Review

Real time measurement and control of applications involving visualization and simulation is an on-going area of research. Tack et. al. (N. Tack, 2004) proposed real-time control of applications involving 3D graphics content targeted for mobile terminals such as PDAs. Their aim was to decode different 3D content while considering the dynamic load on the device. They derived a heuristic algorithm to approximate the content to be rendered for a 3D scene. Their control algorithm estimates the rendering time by analytically approximating the rendering pipeline ("Mesa Home Page," 2012). Continuous processing power on the client and server sides is thus necessary. This type of analytical modeling assumes that the entire rendering pipeline is carried out on the CPU with a single thread, and is also not suitable for the estimation of execution time for current modern GPU-based

systems which include multiple cores (hence multiple threads) and system on chip systems (SoC) (Wolf et al, 2008) type architectures. Finally, in the current platforms separate acquisition of each parameter corresponding to rendering stages such as vertex and triangle clipping, rasterization of line, triangle and pixels is not practical due to lack of driver support at the application level.

In further work from the same group (K. Tack, Lafruit, Catthoor, & Lauwereins, 2006; Nicolaas Tack, Lafruit, Catthoor, & Lauwereins, 2005) they defined a cost function and used an unconstrained optimization model to increase visual quality. They performed computation of view dependent Pareto graphs offline and associated the graph for each object in the scene to change in the level of detail (LOD) during run-time. In their approach, optimization of the textures and artifacts due to level-of-detail changes were not considered. Wimmer and Wonka (Wimmer & Wonka, 2003) tried to estimate the rendering time with analytical approximation based on heuristics. The heuristics are based on triangle count, transformed vertex count, cost derived in (Funkhouser & Séquin, 1993) and estimation based on sum of the cost of total transformed vertex and total generated pixels. Their work focused on estimation of rendering time rather than a framework for optimization of performance on multiple hardware devices.

A signature-based profiling approach was presented in (Mochocki et al., 2006) for estimation of rendering performance. In this approach, “signature” refers to information generated from modifying the rendering routine. Their idea was to collect and update information (signature) during rendering and estimate the performance of the next frame from previously collected signatures using a distance metric. The extensive data collection process requires modifications to the original application and native support from the hardware platform (e.g. hardware performance counters) to record accurate performance metrics, which may not be feasible for many applications. Their validation results were based on software simulation of ARM chips rather than actual physical devices. Their major goal was to predict workload to save energy by adjusting voltage/frequency scaling of the SoC chips instead of changing quality offline/online or performance of an application.

Wong and Wang (Gabriyel Wong & Wang, 2008; G. Wong & Wang, 2010) tried to estimate 3D rendering performance using a black box-type linear parametric model and neural networks. In their study, input parameters to the model such as parameters affecting the adaptive controller, rendering process, quality of service assurance are not explicitly conveyed. Their validation results were limited to desktop machines with no guarantee on the scalability to other systems such as mobile or tablet devices. The work did not describe the training process and calibration of the model and more importantly computation and memory overhead for devices that have minimal resources.

Ngoc et. al. (Ngoc et al., 2003) proposed a middleware that decides allocation of the hardware and software resources for embedded systems. Their system targeted the field-programmable gate arrays (FPGA) architecture. The model proposed is for discrete set of computational tasks designated for batch processing. They did not consider hardware constraints such as memory. The set of applications that they aimed for are not clear from the work. Moreover, no validation studies were presented.

Optimization of power consumption by generating power-performance tradeoffs have been explored in the literature. Thompson et al. (Thompson et al., 2009) proposed a model driven approach to understand the application power consumption for mobile and ubiquitous computing systems. Their approach aims at providing a modeling environment for application developers to artificially create an application that has similar features as the final design. Their modeling tools then generate a synthetic code to emulate the computational requirements of the final application that could be used by developers and designers to estimate the power consumption and performance of a mobile device during application execution. The objective, therefore, was to provide guidance to application developers instead of performance optimization. In (Jejurikar & Gupta, 2002) a heuristic algorithm was introduced to minimize due performance-energy tradeoff for a set of pre-defined tasks for embedded systems. This is a scheduler-level algorithm (Ben-Itzhak et al. 2010), where thread allocations on multicore devices is based on maximizing the average performance over consumed energy.

A scenario based performance allocation algorithm was proposed in (N. Miniskar, Munaga, Wuyts, & Catthoor, 2009; N. R. Miniskar et al., 2009) for multiple processor system on chip systems (MPSoC) (Wolf et al., 2008). They created scenarios including Pareto optimal pre-scheduling information for each application in the design phase. These Pareto graphs were only for energy consumption versus performance balance trade-offs. During the application, the scenario monitoring component detects the pre-computed scenario for the application that aims to minimize energy consumption while meeting application performance requirements. They did not describe the situation where the pre-computed optimal graphs cannot be detected. Their methods require additional changes and also support from the device platform. In addition, the method introduced additional memory and computational loads.

Adaptive rendering for cloud-based mobile gaming was attempted in (Wang & Dey, 2010). In their architecture, the game content was streamed to the client. They characterized rendering parameters such as communication and computational costs that affect the server side output image. This output stream was adaptively changed based on loads at the client device. A major drawback of this technique is that the rendering settings were determined based on a predefined set of discrete adaptive levels. The adaptive algorithm on the client side creates a lag during the response to the increasing load. Preda et al. (Preda et al. 2008) adopted a similar approach for online gaming. They addressed the discrete set of LOD by subdividing only significant regions of the geometry to prevent immediate and unpleasant changes in the rendering. This method, like other similar approaches, introduces additional computational overhead at the client side.

A quality of service manager service was proposed in (Van et al., 2002) on top of the operating system. They also proposed a quality controller module for each domain specific application and a general quality manager to oversee the overall resources of the computing platform. Although the details of the algorithm are unclear, it appears to distribute the resources based on application priority during run time execution. This ad-hoc method tries to increase the resources of the application as long as the global resource manager can

support the load. The overhead of their run-time scheduling of system resources and response time were not reported.

Rosenbaum et. al. (Rosenbaum et al. 2011) introduced a device adaptive approach. Their approach was very specific to data visualization, where data can be progressively altered on the server side for a particular device. They assumed that they knew all device hardware features *a priori* and used thresholds based on the device metrics during run-time. A similar approach was proposed by Kim et. al. ((Kim et al., 2004), where they employed a MPEG-2 framework to stream 3D data for facial animation applications. They used an animation framework extension of MPEG-4 (Bourges-Sévenier et al., 2004) to adaptively stream to the client device considering device and bandwidth limitation of the client terminal. In their method, they rated the devices according to rating factors such as device computing capability and network ratio. However, critical parameters including screen resolution, memory and non-fixed graphics rendering capability were not included. Their approach also requires additional memory and computational resources on the client side. Another approach by introduced web based stereoscopic visualization application named CoWebViz (Kaspar et al. 2012). They allowed interactive volume rendering around 10fps on client-side browsers with using MJPEG technology. As the proposed approach is based on stream processing (e.g. sending JPEG images with various resolutions) over the network, the performance of their application has been modeled with basic algebraic equations that correlate the frame rate and latency to the network bandwidth. The disadvantage of their model is applicable for visualization applications as opposed to interactive simulation.

Unlike previous approaches, our approach does not assume that the simulation and visualization scenes run flawlessly on the client device. Nor do we make any device specific assumptions (Ngoc et al., 2003; Rosenbaum et al., 2011). The visual scene data can be beyond the physical capabilities of the device in terms of rendering and storage, in which case unpleasant rendering artifacts may appear (see Results section). This can be addressed by considering the physical limitations of the device.

Previous works also assume that the scene shading is based on fixed shading (Kim et al., 2006). This does not include complex rendering stages. Therefore, the aim of majority of the previous works (Preda et al., 2008; Rosenbaum et al., 2011; K. Tack et al., 2006; Nicolaas Tack et al., 2005) has been to only increase geometry tessellation, which may result in artifacts such as visual distortions caused by continuous changes from high resolution mesh to low resolution mesh, missing frames due to overhead of subdivision or LOD selection during the execution. The assumption is that the computational resources, especially on the CPU side, are available for online processing. In addition, meshes are static and do not have interactivity (Gobbetti & Bouvier, 2000; K. Tack et al., 2006).

The web based VR applications, with recently introduced native web browser support (e.g. MozVR of Mozilla Firefox and WebVR for Chrome) for VR kits (e.g. Head Mounted Display Oculus Rift and hand motion tracking device Leap Motion), will become more pervasive. The simulation and visualization of VR performance optimization for heterogeneous devices will be necessary. Performance optimization with the previous methods that impose additional computation load on the client device is challenging

especially for low-end devices. Moreover, a method that assumes a priori device information is not feasible for such applications.

Our approach, on the other hand, takes into account the simulation and visualization constraints with no any assumption regarding the device capabilities. We also incorporate multiple objectives such as increasing texture quality and discretization, allocating resources for interactivity and other application specific constraints imposed by the application developer. Our approach can be applied or adopted to any specific MIS applications such as VR based visualization and simulation.

2. Material and Methods

2.1. Optimization model

As discussed above, our overall approach has three phases as shown in Figure 1: identification, optimization and update, which are explained in detail in the sub-sections below:

2.1.1. Identification—The web browsers sends user-agent string along with the HTTP Request (a message sent over to the server at each request), defined in HTTP protocol (HTTP/1.1). This user-agent string is the only data field that contains information about client platform. The information, however, is very limited and only depicts the web-browser type, version, operating system information or web browser layout engine (e.g. Software component of web browser that generates and displays GUI components in a web page at the client side) of the client. While some device type can be determined for some devices such as tablets and smart phones (with using a user-agent string in the HTTP header), it is not possible and also not feasible to identify any hardware specification of the client device. Hence, when a device performs a HTTP request to run a multimodal interactive application, a benchmark identification test is performed to estimate the device performance. This is achieved using a small proxy JavaScript code (Listing-1) which is automatically sent to the client device.

For a visualization scenario, e.g., the proxy code initializes a dummy visualization scene and incrementally loads the GPU of the client. This benchmark code estimates the hardware related runtime parameters that are required by the optimization model. During the rendering of this dummy scene, all frame rates are recorded locally and then uploaded to the server side which is saved in a database for later use for this device. The unique number generated specific to the client device is saved both on the client side and on the server side to prevent any redundant benchmarks. This unique ID is sent to the device as a HTTP cookie with expiration date which is set as 90 days. This enables us to perform a benchmark only once for any device within the 90 days expiration period. When the client requests the simulation for the second time during this period, we only perform a simple check from the database to retrieve the client side device information. When the client connects after the expiration date, the client will be notified with a benchmark renewal request. In the case of any hardware changes before the expiration date, the user needs to manually request a new benchmark using GUI options available on the web page of our framework.

In the identification phase, In addition to dynamic performance data, the proxy code also queries the client GPU to identify static device specific constraints.

2.1.2. Optimization—In this phase, the accumulated data from the first phase is used to compute the optimal visualization and simulation quality within user constraints. We discuss the optimization model for visualization below, followed by how to modify the model to account for simulation performance.

Optimization for visualization quality: Let O represent a set of ‘ n ’ objects in the scene; i.e., $card(O) = n$. Our objective (J) is to maximize the total number of nodes ($N_j, j \in O$), the texture parameters for each object to be rendered (d_j, b_j, a_j, s_j see below for explanation) and minimize the deterioration effect (L) by maximizing the canvas size (c) of the frame buffer. We also aim to minimize the variation of the mesh resolution (μ) between the objects.

In our optimization model, we categorize the constraints into hard and soft constraints. Hard constraints are the limitations arising from the physical client device capability. Number of supported textures, number of textures accessible from shaders, depth buffer bits etc. are examples of hard constraints. The hard constraints determined from the identification phase can be directly substituted into the optimization model. They may be dynamic or *static*. For example the maximum color bit support of a frame buffer is a static hard constraint which cannot vary with time; while the total number of nodes that can be rendered can easily vary based on current GPU/CPU load and bus traffic and hence is a dynamic hard constraint. It might often become challenging to accurately identify some hard constraints as the constraints themselves change with changing device states. On the other hand, the soft constraints are restrictions set by the designer or the user of the simulation. Examples of such constraints include minimum rendering frame rates, the total number of the mesh size or deviation of mesh sizes.

We thus define an optimization problem in Table-1, that defines an objective function J , defined in Eq. (A.1), to be maximized with respect to several constraints specified in Eq. (A. 2) through Eq. (A.18). We describe the various constraints employed in further details below.

Texturing Constraints: Texturing constraints place limitations on the size and number of textures employed for the rendering. In our formulation, for any object ‘ j ’, d_j, b_j, a_j, s_j represent diffuse for decaling, bump for creating fine surface details, ambient occlusion for attenuation of light and occlusion over the surfaces, and specular mapping for reflectivity and shininess of the surface fragments respectively in the rendering scene. These texture mapping types cover most applications, while others such as environment maps or alpha maps may also be possible.

Additional maps apart from visualization including perlin noise and displacement maps can be similarly incorporated in the objective function and as well as in the constraint equations defined with Eq. (A.2), (A.12) and (A.16). In the model, the texture weights ($w_{d_j}, w_{b_j}, w_{a_j}, w_{s_j}$) are based on their visual importance. The higher the weight for a given texture the more important the texture becomes for the final rendering. The importance of the texture can

vary based on the application requirement. However, in general, bump maps are given greater importance than the other textures (Mali™ GPU Application Optimization Guide, 2011). When detailed decal information is needed, e.g. in high resolution environment maps, diffuse texture is given precedence. Similar to the texture weights described above, w_j , refers to weights of objects that are proportional to its visual importance in the overall scene.

In our formulation, T_{ji} in Eq. (A.2) represents the binary decision variable of the optimum selected texture size. However, in order to ensure the selection of only one texture size for an object, constraint in Eq. (A.3) is added. Eq. (A.2) constrains a texture size (K_{ij}) to be power of two to ensure higher quality texture during the reduction and magnification process in a GPU (Fernando, 2004). In order to achieve that we define α and p as;

$$\alpha_i = 2^i \text{ and } p \leq \log_2(T_{\max}) \quad (1)$$

For special texture sizes, e.g., rectangular texture α_i can be chosen as a linear multiple of a texture element size.

Any texture size (K_{ij}) is constrained by the device capabilities. Therefore, we assume that the texture size of a particular map is bounded by a maximum ($T_{\max}$) and a minimum ($T_{\min}$) value as in Eq. (A.16). $T_{\min}$ is generally set by the user. Otherwise, the minimum texture size is chosen as 2×2 by default. Here, $T_{\max}$ is obtained by the proxy code during the identification process and indicates the maximum texture resolution that the GPU supports with the fragment and vertex shader. In the objective function, weight (w_T) for textures indicates the quality measure of texture with respect to the nodes. If the maximum texture size ($T_{\max}$) is far less than the GPU memory of the device ($T_{\max} \ll GMM$), then this value can be assumed to be some small value as the model will maximize the texture sizes without affecting node size. In this paper, we set $w_T = 1$.

In addition to individual constraints on each texture type, the total size of each texture cannot exceed the GPU texture space for an application (G) which is defined in Eq. (A.12). Our proxy code can determine the maximum number of textures that can be fetched from the vertex or fragment shader. In Eq. (A.12), GPU texture memory requirement (G) is bounded by $t * T_{\max}$, where 't' is the total number of textures supported by the device. In addition, texture memory requirement should be less than GPU memory (GMM) which is expressed in Eq. (A.15).

Node Constraints: These constraints set limitations on minimum and maximum number of nodes in the rendering of an object. Eq. (A.5) bounds the minimum mesh size by $N_{\min}$, which is application specific and ensures that the mesh resolution is sufficient to preserve shape details. The maximum mesh size for an object is limited to $N_{\max}$ in Eq. (A.5). This number may be provided by the user, otherwise we extract this value from maximum

achieved number of nodes divided by the total number of objects $\frac{M_{\max}}{n}$ from the identification phase. In Eq. (A.4), the total number of mesh sizes are constrained to $M_{\max}$ which is also obtained from the identification phase.

Another constraint in our model is the absolute value of the deviation D_j of N_j from the average discretization ($\frac{\sum_{i=1}^n N_j}{n}$). It is bounded from above by v - a number determined by the application designer based on the requirement. If v is not set by the user, the maximization problem will favor the object that has more weight compared to the others in the objective function (J). In a rendering scene it is more desirable to have approximately close mesh sizes. Objects that have trivial representations in the scene such as planar surfaces, or objects with less curvature or large surfaces can be removed from the constraint equations in Eq. (A.7) to Eq. (A.9) to reduce visualization loads. In order to realize this constraint as an absolute value, we split it into two constraints: Eq. (A.8) and Eq. (A.9) (Williams, 1999). Therefore, the sum (μ) of absolute values indicated as abs_D_j in Eq. (A.10) for each object needs to be less than v as indicated

$$\sum_{j=1}^n abs_D_j - v < 0 \quad (2)$$

Canvas Constraints: These constraints limit the canvas size when the rendering resolution is set. Our benchmarks test the limits of the client device by applying different canvas sizes. Canvas is the final rendering layer on a web page. After objects are rasterized in the frame buffer, final colors are computed and the final image is composited on the canvas. This implies that the canvas size can be different than the frame buffer size. The benefit of having different sizes is favorable especially for low end devices. With the smaller frame buffer size and larger canvas size we can render higher resolution meshes by decreasing the size of the frame buffer but increasing the canvas size.

However, this can decrease the quality of the final image if the gap is large (such as half the ratio) due to up-scaling of the final image. The penalty of the effect is reflected in the objective function with term $w_L L$, where w_L is the weight for the effect and L is the total number of the nodes that are required to be rendered when the canvas size is reduced. The absolute value of this parameter increases as the canvas size reduces. Reduction of canvas size therefore requires rendering higher resolution meshes to compensate for the quality difference. As an example, if the canvas size reduction is one third of the original size but the achievable number of nodes for that canvas size with regards to the benchmark results are not greater than one third, our model will rule out the canvas reduction in the optimum solution set. Therefore, canvas size reduction can be useful only if the ratio of increase in the total number of nodes is greater than the canvas size reduction ratio.

We incorporate the canvas size in the formulation by including the maximum rendered mesh capacity. For example, a reduced canvas will provide greater capacity to render higher resolution meshes but it will be penalized by the objective function weight. The constraint arising from the canvas size variable (c_j) is added in the model in Eq. (A.13). In this equation, r is the total canvas resolution, γ represents the maximum number of nodes rendered for the targeted frame rates obtained directly from the proxy code and the C places a constraint on the total number of nodes. In our model, we made canvas size decision

variable (c_i) binary. Therefore, c_i can be either one or zero and the sum of the c_i (such as; $\sum_{i=1}^r c_i = 1$) is equal to one as only one canvas size can be selected at a time.

Memory Constraints: GPU memory varies with different client devices. Therefore hard constraints should be placed on the memory utilized for the visualization and simulation in order to work within the memory bounds. Assuming the application will run in the full screen mode, the proxy code tests the client GPU with varying number of nodes and triangles in that mode. Therefore the GPU memory is computed by reserving the necessary memory allocation needed for full screen display. This includes sizes of frame buffer, front and back buffers, stencil buffer and depth buffer. The computation of the size of these buffers is carried out based on the proxy code information. For instance, a 1280×800 resolution supporting all color channels including alpha channels needs 6.0 Mb with dual buffer. This memory requirement is included as a multiplier with the coefficient ρ in Eq. (A. 15). The memory space requirement is limited by the value of GMM in the model that includes frame buffer sizes, depth buffer, stencil buffer, mesh sizes (β refers to memory allocation for a mesh) and total texture footprint in the memory.

Optimization for simulation performance: So far, we have discussed considerations that pertain only to visualization. Multimodal interactive simulations allow physical interactions with objects in the scene. Thus, our environment also supports such simulations. At present we support position based dynamics (PBD) for deformable objects (Müller et al., 2007) and heat transfer simulation based on the finite difference method (Cozzi & Riccio, 2012).

The simulation model is the extension of our visualization model. In addition to the visualization model, the simulation model introduces the performance constraint for each simulation. The performance constraint is a soft one of the form

$$f_{\min} < f_i < f_{\max}, \forall i \in S \quad (3)$$

where f_i indicates the performance of simulation of object i that is a member of the set of objects being simulated in the scene (S). f_i is also a function of number of nodes (N). We assume here that ' m ' out of ' n ' objects in the scene are being simulated and $S \subset O$. Based on the performance data in the identification phase, we may linearly interpolate the benchmark data to estimate the number of nodes for $f_{\max}$ and $f_{\min}$. $f_{\max}$ and $f_{\min}$ are lower and upper limits on the number of nodes being simulated. Solution to the actual problem may not be achievable unless computing resources are balanced for all simulations in the scene. Therefore, pre-allocation of resources should be performed within the device computation limits. In order to ensure that all allocations are within the computational limit, the following constraint is included in the model

$$\sum_{i=1}^m \frac{f_i}{F_i} \leq 1 - r, i \in S \quad (4)$$

where F_i represents the maximum number of nodes that the simulation can execute. This may be computed with the benchmark data and may be taken as the number of nodes when

the simulation runs at approximately one frame per second. Therefore, $\frac{f_i}{F_i}$ is the ratio indicating device usage, which cannot exceed unity. In our formulation, we limit it to $1 - r$, where r represents the normalized reserved percentage for a device that is to prevent saturation or guaranteeing allocation enough space for spikes during run-time execution.

Recall that in our optimization formulation for visualization, the objective function in Eq. (A.1) tries to maximize the visual quality. With the given simulation constraints, our aim is to maximize both visual quality and simulation performance. In order to incorporate simulation performance maximization into the formulation, we need to modify the objective function in Eq. (A.1) as the follows

$$J = \frac{W_T \sum_{j=1}^n (w_{dj}d_j + w_{bj}b_j + w_{aj}a_j + w_{sj}s_j) + \sum_{j=1}^{n+m} w_j N_j - w_\mu \mu - w_L L}{\sum_{i=1}^m f_i} \quad (5)$$

The visualization and simulation formulation presented above is based on non-linear mixed-integer programming (Diwekar, 2008). We can linearize the model by using fractional programming (Schaible & Shi, 2004) based on the Charnes-Cooper transformation process (Schaible, 1974);

$$y_0 = \frac{1}{\sum_{i=1}^m f_i} \quad (6)$$

Now the objective function becomes;

$$J = y_0 w_T \sum_{j=1}^n (w_{dj}d_j + w_{bj}b_j + w_{aj}a_j + w_{sj}s_j) + \sum_{j=1}^{n+m} w_j N_j y_0 - w_\mu \mu y_0 - w_L L y_0 \quad (7)$$

With variable transformations of the form $y_j = N_j y_0$, this objective function can be recast into the form shown in Table 2, with α_j corresponding to the weight for each decision variable in Eq. (1). Similar transformations need to be carried out to convert all original decision variables to new ones. The new problem is exactly the same as the original one except the following additional constraint for the y_0

$$\sum_{i=1}^m Y_i - 1 = 0 \quad (8)$$

where $Y_i = f_i y_0$ is a new decision variable. This transformation allows us to eliminate the original decision variables. The linearized system can be solved using a simplex method. The IBM CPLEX linear programming solver ("IBM - Mathematical Programming," 2012) is used. After the solution process, we computed each original decision values in the post-processing in CPLEX solver by applying appropriate inverse transformations.

The solution of the above optimization problem may not be always feasible. The feasibility of the solution depends on the constraints set for the simulation and device capabilities. For example, if the simulation performance constrain f_{max} amounts to a value that is less than the visualization node constraint N_{min} , then the problem automatically becomes unfeasible for the device. Another problem would be in cases where the total computing capability of the

device may not be sufficient for the all the simulations. In this case, the sum $\sum_{i=1}^m \frac{f_i}{F_i}$ will exceed unity. This means the device cannot run both simulations with the requested constraint at the same time. Since the constraint is violated, the model cannot produce a feasible solution.

This new problem can be expressed as in Table-2. In Eq. (A.20), c_i are simulation performance constraint coefficients (such as $y_0 f_i$) introduced due to the linear transformation. y_0 from the linear transformation is required to be a positive number as $N_j > 0$, Eq. (A.18). The A_{ij} are the coefficients for the original model constraints.

2.1.3. Update—In this last step, we generate the device data for the specific client based on the optimization phase. If the mesh size resulting from the optimization phase is larger than the original mesh, we subdivide the mesh. Otherwise we decimate the mesh to decrease its size. Similarly, for textures, the proper sizes from the application repository are sent to the client device. If the total number of supported textures of the device is less than the total number of textures in the application, the optimum solution automatically fragments the textures and more than one texture is used. A texture atlas is generated for the scene and texture coordinates are modified accordingly. The WebGL canvas and simulation mesh sizes are similarly set based on the optimal solutions.

3. Results

We performed a series of visualization and simulation tests to verify that our approach successfully generates the optimum visual quality and performance for each of three devices being tested: a MacBook Pro, a Samsung Chrome Book Series 5 with Chrome OS, and an Apple iPad 2. The Macbook Pro is equipped with 4-i7 cores (2.0GHz for each core), 4GB physical memory, ATI Radeon™ 6490M 256 GB GPU card with a maximum 1440×900 display resolution and Windows 7 64bit. The Chrome book has 1.66-GHz Intel Atom N570 with 2GB physical memory and NM10 graphics chipset integrated video card. The iPad 2 is based on Apple A5 SoC 1GHZ chip with 1024×768 display resolution. We used a Acer Aspire AS7736Z Laptop on the server side whose specification is 2.20 GHz CPU with two cores and 3GB physical memory and Windows 7 64 bit. The clients and server are connected over a 54 Mbs local Wi-Fi network.

We choose a virtual surgical simulation scene for our tests related to laparoscopic adjustable gastric banding (LAGB) (Sankaranarayanan et al., 2010) which has four tissue structures; liver, stomach, lesser omentum (connective tissue between stomach and liver), and gastrohepatic ligament (surrounds top portion of stomach). Our models have the default base meshes for liver, stomach, lesser omentum and gastrohepatic ligaments as 2526, 1411, 867, and 1250 respectively.

Our optimization framework requires information about the device capabilities and specifications which are accumulated with the benchmark tests. In each of the benchmark tests, the devices are first connected to the server through a browser URL. On the server side, the PHP script handles the client requests. During the initial connection, the server generates a unique ID for each client device and then automatically sends the proxy code to the device. The proxy code identifies the device capabilities and performs two benchmark tests. The first is the **visualization benchmark** where performance is measured with increasing load. The load is varied by changing the frame buffers and canvas sizes. The second benchmark test is the **simulation benchmark** where performance is measured after increasing the load for simulation. To prevent any performance deterioration during the data collection process, the results are written into a local HTML5 web storage and then asynchronously streamed to the server. As mentioned previously, these two benchmark tests are carried out in the first phase and are required for computing the optimum solution in the optimization phase.

In the update phase, visualization and simulation meshes are prepared for each object based on the optimization solution for the device. If the optimum number of nodes for a mesh is greater than the default number of nodes, then the mesh is subdivided and more nodes are added. Conversely, if the optimum number of nodes is greater than the default mesh, the original mesh is decimated. During the subdivision and decimation processes, the texture coordinates and shape boundaries of each object are preserved. As a result, each device has the optimum visualization mesh sizes for each object since the solution is subject to the limits of visualization capability. For instance, the optimum solution for the visualization mesh for the liver model on the MacBook Pro has 7007 nodes, while the default mesh consists of 2526 nodes. Therefore, in the update phase, the mesh is tessellated until a mesh with a number of nodes close to the optimum is reached.

Visualization test results

In the visualization test, we limit the frames per second (FPS) to be 15 for the Chrome Book, 100 for the MacBook Pro, and 25 for the iPad 2. In the optimization model, weights (w_j) are assumed as 0.4, 0.3, 0.2, and 0.1 for the liver, stomach, lesser omentum and gastrohepatic ligament models, respectively. Based on the visual importance, the texture weights are chosen as 0.4, 0.3, 0.2 for w_{dj} , w_{bj} and w_{sj} , respectively. No ambient maps are used in the scene. The texture weight (w_T), and deterioration weight (w_L) are assumed to sum up to one. Weight for deviation (w_μ) is chosen as 0.1. The term v is chosen as half of the total number of nodes in the default scene (6054 nodes). We used two different error norms for evaluation of visualization performance. The raw error (RE) is defined as

$$RE = \frac{|FPS_{obtained} - FPS_{target}|}{FPS_{target}} \quad (9)$$

The normalized error (NE) considers the initially given FPS range $[FPS_{target} - \delta, FPS_{target} + \delta]$ and is defined as

$$NE = \begin{cases} \frac{|FPS_{obtained} - (FPS_{target} - \delta)|}{FPS_{target} - \delta}, & \text{if } FPS_{obtained} < FPS_{target} \\ \frac{|FPS_{obtained} - (FPS_{target} + \delta)|}{FPS_{target} + \delta}, & \text{if } FPS_{obtained} > FPS_{target} \end{cases} \quad (10)$$

where δ is set as 3 FPS and 5 FPS for the visualization and simulation tests, respectively. This range affects the search algorithm, which allows more flexibility in the selection of the benchmark data. The purpose of utilizing a NE formulation is also to account for undershoot and overshoot of the obtained FPS with regards to the target FPS with upper/lower range indicated by δ . Intuitively, if the obtained FPS range is lower than the target FPS, the RE can be clearly understood to be a result of this shortfall. However, any obtained FPS that exceed the target FPS can also be seen as error when utilizing a RE formulation. This is precisely why we use a normalized error formulation that accounts for conditions when the obtained FPS is within the target FPS range. Both RE and NE data are shown for the default and optimized visualizations in Figure 2.

Results indicate that 12, 94.3 and 21 FPS were achieved in the optimized case for the Chrome Book, MacBook Pro and iPad 2 respectively. In the non-optimized case, an average of 6.7, 171 and 35 FPS was achieved for the three devices. As seen in Figure 2a, the default visualization has a higher error percentage for all devices compared to the optimized results. For the Chrome Book, a high error in default visualization cases is due to highly tessellated meshes that are beyond the device capabilities to render at the target rate. On the other hand, in the MacBook Pro and iPad 2, the error is because of underutilization of the device GPU. This means that some devices can easily handle visualization of more complex meshes than the default meshes for a specified rendering rate. The high error in the optimized case for the Chrome Book exists due to the range (δ) during use of benchmark data obtained from the first phase. In Figure 2b, the error is computed with *NE* that considers this range and it is clearly seen that the performance is within the performance FPS range in the optimization case.

In the visualization test, the optimum number of total object nodes are computed to be 20716, 4291, 40128 for the MacBook Pro, Chrome Book, and iPad 2, respectively. Apart from dramatic error reductions and changes in optimized object nodes, overall visual quality was also improved for each of the objects. These visualization test results are shown in Figure 3. In the Chrome Book, the default condition creates noticeable visual artifacts (Figure 3(a)) due to insufficient texture memory of the GPU. This results in the abnormal black surface surrounding the properly rendered textures. The optimization model resolves the abnormality in Figure 3(b). For the MacBook Pro and iPad 2, the resultant scenes with a default scene have less detail on the surface due to flaws of specular lights and bump mappings as shown in Figure 3 (c) and (e). At these regions the minute surface details are indistinguishable. However in the optimized model, these surface details become more noticeable. These result from both high mesh resolution and texture sizes.

It is known that high-resolution meshes produce better visual quality than low-resolution meshes. This is because quality features are diminished when the mesh is made coarser. For instance, surface detail features are lost due to interpolation of surface normals and tangents. In both devices, the artifacts are noticeably observed on the surface of the objects in the

default rendering. Moreover, in the optimized rendering, the meshes are tessellated based on device capability. In the MacBook Pro and iPad 2, the default meshes are subdivided and more nodes are generated whereas in the Chrome Book the default meshes are decimated.

In order to show the difference in visual quality, we increased the luminescence and zoomed onto the surface of the liver for the MacBook Pro case. Images for both the optimized and non-optimized cases were taken. Next, we performed a grayscale conversion from the RGB images and applied a canny edge detection to expose and highlight the surface details on the liver. These results are shown in Figure 4. Figure 4a represents the optimized and non-optimized rendering of the liver surface. Figure 4b shows the canny edge detection results for the same optimized and non-optimized rendering images. The image on the right top in Figure 4b clearly shows that the non-optimized version of the visualization output cannot show minute details on the surface. The edge detection reveals that there is not much variation in the intensity of the image. However, the bottom Figure 4b shows that even tiny details in the optimized case can be spotted on the surface. The density of details is higher than the non-optimized case. Ultimately, this indicates that our optimization model successfully enhances visual quality by increasing the density of object surface details.

Simulation Test Results

In the simulation benchmark test, we incorporated the following two physics simulations in the scene:

Simulation-1

A thin tissue structure is simulated using the position based dynamics method (Müller et al., 2007). The model consists of 625 nodes.

Simulation-2

A 2D heat transfer simulation is performed on the image space domain using the finite difference method (Cozzi & Riccio, 2012) with 81,081 nodes.

We limited the FPS range of Simulation-1 to be within 30 and 40 and Simulation-2 to be within 40 and 50, respectively. The optimum solutions for Simulations 1 and 2 are computed to be 2025 and 136,800 nodes for the MacBook Pro, 225 and 39852 for the Chrome Book, and 64 and 1568 nodes for the iPad 2, respectively. The performance error for each simulation is computed based on the normalized error (NE) metric and plotted in Figure 5. In the case of the MacBook Pro shown in Figure 5(a), the test with Simulation-1 reported a FPS of 37.6 and is within the defined performance range. The error of Simulation-2 is only 4.4% in the optimized case. In the Chrome Book, the error is 1.6% and 24.4% in the optimized cases for Simulations 1 and 2, respectively. The large optimized errors of Simulation-2 stems from the larger weight ($w_1 > w_2$) used in Simulation-1 compared to Simulation-2. The JavaScript engine in the Chrome OS is fairly slow compared to regular laptop browsers. This is partially why the optimized error for Simulation-2 in the Chrome Book is higher than other devices.

An infeasible solution of the optimization model indicates that the given device cannot handle the defined simulation constraints. In these cases, we simply remove the lower bound performance constraint (f_{min}) of the simulation to find an optimum solution. However, as expected this increases the error. In the iPad 2 case, our model indicates infeasibility in the solution. Therefore, we removed the lower bound constraints for both simulations to find an optimum feasible point. This relaxation in the constraint increased the optimization error as seen in Figure 5c. As a result, the iPad 2 performed inferiorly compared to other devices. Once again, this is due to the low performance of the JavaScript execution rather than the device capability itself, especially since the rendering performance of the iPad 2 is superior to the Chrome Book.

4. Discussion

In this paper, we presented a performance optimization model for interactive visualization and simulation for ubiquitous environments such as the web. Our proposed approach provides a generic solution for multiple devices regardless of their computing capabilities. Our model successfully utilizes the client hardware considering the image quality and performance. We validated our model by carrying out visualization and simulation tests. Since our method is based on offline computations, it does not impose additional burdens on the client side. Moreover, it is based on the actual benchmark data for each device, which makes the method dynamically adaptable to a range of hardware devices. In addition, our method is robust enough to cope with hardware changes (such as a hardware upgrade) of the client device. Client hardware information is queried and benchmarked only once and the accumulated benchmark data is stored in database servers. This removes any redundant benchmark tests of identifying client device capabilities.

The optimization model for visualization incorporates various parameters including shader codes, total supported textures and total textures by the fragment, vertex shader, or display size. Although the model can easily accommodate a range of shading and rendering techniques, we plan to incorporate more complex rendering techniques (such as multi-pass rendering, deferred shading, etc) in the visualization model as part of future work.

The simulation model optimizes client performance depending on the device capability. Currently the model takes frame rate data as a criterion. Future work may include expansion of the optimization model by incorporating simulation accuracy and convergence metrics as additional criteria. The issue of optimum utilization for upcoming computing platforms such as hardware-based parallel processing on the client device ("WebCL - Heterogeneous parallel computing in HTML5 web browsers," 2011.) could also serve to extend the current work.

Acknowledgments

This project was supported by National Institutes of Health (NIH) Grant NIH/NHLBI 1R01HL119248-01A1, NIH/NIBIB 2R01EB005807, 5R01EB010037, 1R01EB009362 and 1R01EB014305.

Appendix A

Table 1
Visualization optimization model

<p>The visualization optimization problem: maximize: subject to:</p> <p>maximize:</p> $J = w_T \sum_{j=1}^n (w_{dj}d_j + w_{bj}b_j + w_{aj}a_j + w_{sj}s_j) + \sum_{j=1}^n w_j N_j - w_\mu \mu - w_L L$ <p>subject to:</p>	(A.1)
$\sum_{i=1}^p \alpha_i T_{ji} - K_{lj} = 0, \forall K_{lj} \in K_l, l \in 0,$	(A.2)
$\sum_{i=1}^p T_{ji} \leq 1 \text{ and } \sum_{i=1}^p T_{ji} > 0$	(A.3)
$\sum_{j=1}^n N_j \leq M_{\max}, N_j \in N,$	(A.4)
$N_j \geq N_{\min} \text{ and } N_j \leq N_{\max}, \forall N_j \in N,$	(A.5)
$\sum_{i=1}^n N_j - nS = 0$	(A.6)
$N_j - S - D_j = 0, \forall j \in 0$	(A.7)
$-D_j \leq \text{abs}_- D_j, \forall j \in 0$	(A.8)
$D_j \leq \text{abs}_- D_j, \forall j \in 0$	(A.9)
$\sum_{j=1}^n \text{abs}_- D_j = \mu$	(A.10)
$\mu < v$	(A.11)
$\sum_{j=1}^u \sum_{l=1}^n K_{lj} - G = 0, n = \text{card}(\theta), u = \text{card}(K_1)$	(A.12)
$\sum_{i=1}^r \gamma_i c_i - C = 0,$	(A.13)

$nS < C$	(A.14)
$G + \beta nS + \sum_{i=1}^r \rho c_i \leq GMM$	(A.15)
$K_{li} \leq T_{\max}, K_{li} \geq T_{\min} \forall K_{li} \in K_l, i \in 0$	(A.16)
$G < t * T_{\max}$	(A.17)
$N_j > 0, C > 0$	(A.18)

Table 2
Linearized optimization model

The simulation optimization problem: Maximize: $J = \sum_{j=1}^{m+n} \alpha_j y_j, j \in S \cup \theta$ Subject to:	(A.19)
$\sum_i c_i y_i - 1 = 0$	(A.20)
$\sum_j A_{ij} y_j - b_i y_0 \leq 0$	(A.21)
$y_0 > 0$	(A.22)

References

- Arnab, S.; Petridis, P.; Dunwell, I.; de Freitas, S. Tactile interaction in an ancient world on a web browser. *International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)*. 2011. Retrieved from <https://curve.coventry.ac.uk/open/file/4a458109-7562-e2c3-66fb-017f90a4c169/1/Tactile%20interaction.pdf>
- Ben-Itzhak Y, Cidon I, Kolodny A. Performance and power aware cmp thread allocation modeling. *High Performance Embedded Architectures and Compilers*. 2010:232–246.
- Bourges-Sévenier M, Jang ES. An introduction to the MPEG-4 animation framework extension. *Circuits and Systems for Video Technology*, IEEE Transactions on. 2004; 14(7):928–936.
- Cozzi, P.; Riccio, C., editors. *OpenGL Insights*. 1st. A K Peters/CRC Press; 2012.
- Diwekar, UM. *Introduction to applied optimization*. Vol. 22. Springer Verlag; 2008.
- Engelke T, Becker M, Wuest H, Keil J, Kuijper A. MobileAR Browser – A generic architecture for rapid AR-multi-level development. *Expert Systems with Applications*. 2013; 40(7):2704–2714.10.1016/j.eswa.2012.11.003

- Fernando, R. GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Addison-Wesley Professional; 2004.
- Floudas, CA. Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press; USA: 1995.
- Funkhouser, TA.; Séquin, CH. Proceedings of the 20th annual conference on Computer graphics and interactive techniques. New York, NY, USA: ACM; 1993. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments; p. 247-254.
- Gobbetti E, Bouvier E. Time-critical multiresolution rendering of large complex models. Computer-Aided Design. 2000; 32(13):785–803.10.1016/S0010-4485(00)00068-3
- Halic T, Ahn W, De S. A framework for 3D interactive applications on the web. SIGGRAPP Asia 2011 Posters. 2011:58.
- Halic T, Ahn W, De S. A Framework for Web Browser-Based Medical Simulation Using WebGL. Accepted in 19th Medicine Meets Virtual Reality. 2012
- Halic T, Venkata SA, Sankaranarayanan G, Lu Z, Ahn W, De S. A software framework for multimodal interactive simulations (SoFMIS). Studies in Health Technology and Informatics. 2011; 163:213–217. [PubMed: 21335791]
- IBM - Mathematical Programming: Linear Programming, Mixed-Integer Programming and Quadratic Programming - IBM ILOG CPLEX Optimizer - Software. (n.d.). Retrieved May 21, 2012, from <http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/>
- Jejurikar R, Gupta R. Energy aware task scheduling with task synchronization for embedded real time systems. Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems. 2002:164–169.
- Kaspar M, Parsad NM, Silverstein JC. An optimized web-based approach for collaborative stereoscopic medical visualization. Journal of the American Medical Informatics Association. 2012 amiajnl-2012-001057. 10.1136/amiajnl-2012-001057
- Kim H, Joslin C, Di Giacomo T, Garchery S, Magnenat-Thalmann N. Adaptation mechanism for three dimensional content within the MPEG-21 framework. In. Computer Graphics International, 2004 Proceedings. 2004:462–469.10.1109/CGI.2004.1309248
- Kim HS, Joslin C, Di Giacomo T, Garchery S, Magnenat-Thalmann N. Device-based decision-making for adaptation of three dimensional content. The Visual Computer. 2006; 22(5):332–345.
- Maciel A, Sankaranarayanan G, Halic T, Arikatla VS, Lu Z, De S. Surgical model-view-controller simulation software framework for local and collaborative applications. International Journal of Computer Assisted Radiology and Surgery. 2010:1–15.
- Mali™ GPU Application Optimization Guide. ARM; 2011.
- Mesa Home Page. (n.d.). Retrieved April 4, 2012, from <http://www.mesa3d.org/>
- Miniskar N, Munaga S, Wuyts R, Catthoor F. Pareto based runtime manager for overlapped resource sharing. Proc ECES Workshop, Edegem, Belgium. 2009
- Miniskar, NR.; Hammari, E.; Munaga, S.; Mamagkakakis, S.; Kjeldsberg, PG.; Catthoor, F. Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation. Berlin, Heidelberg: Springer-Verlag; 2009. Scenario Based Mapping of Dynamic Applications on MPSoC: A 3D Graphics Case Study; p. 48-57.
- Mochocki BC, Lahiri K, Cadambi S, Hu XS. Signature-based workload estimation for mobile 3D graphics. Proceedings of the 43rd annual Design Automation Conference. 2006:592–597.
- Müller M, Heidelberger B, Hennix M, Ratcliff J. Position based dynamics. Journal of Visual Communication and Image Representation. 2007; 18(2):109–118.
- Ngoc NP, Lafruit G, Deconinck G, Lauwereins R. Terminal QOS management on run-time reconfigurable platforms. Third PA3CT-symposium. 2003:22–23.
- Preda M, Villegas P, Moran F, Lafruit G, Berretty RP. A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering. Vis Comput. 2008; 24(10):881–888.10.1007/s00371-008-0284-2
- Rosenbaum R, Gimenez A, Schumann H, Hamann B. A flexible low-complexity device adaptation approach for data presentation. Proceedings of Electronic Imaging-Visualization and Data Analysis 2011. 2011

- Sankaranarayanan G, Adair JD, Halic T, Gromski MA, Lu Z, Ahn W, et al. De S. Validation of a Novel Laparoscopic Adjustable Gastric Band Simulator. Will Appear in Surgical Endoscopy. 2010
- Schaible S. Parameter-free convex equivalent and dual programs of fractional programming problems. *Mathematical Methods of Operations Research*. 1974; 18(5):187–196.
- Schaible S, Shi J. Recent developments in fractional programming: single-ratio and max-min case. *Nonlinear Analysis and Convex Analysis*. :493–506. n.d.
- Tack K, Lafruit G, Catthoor F, Lauwereins R. Platform independent optimisation of multi-resolution 3D content to enable universal media access. *The Visual Computer*. 2006; 22(8):577–590.
- Tack, N.; Lafruit, G.; Catthoor, F.; Lauwereins, R. Proceedings of the tenth international conference on 3D Web technology. New York, NY, USA: ACM; 2005. Pareto based optimization of multi-resolution geometry for real time rendering; p. 19-27.
- Tack N, Morán F, Lafruit G, Lauwereins R. 3D graphics rendering time modeling and control for mobile terminals. Proceedings of the ninth international conference on 3D Web technology. 2004:109–117.
- Thompson C, White J, Dougherty B, Schmidt D. Optimizing mobile application performance with model-driven engineering. *Software Technologies for Embedded and Ubiquitous Systems*. 2009:36–46.
- Van Raemdonck W, Lafruit G, Steffens EFM, Otero Perez CM, Bril RJ. Scalable 3D graphics processing in consumer terminals. *Multimedia and Expo, 2002, ICME '02 Proceedings 2002 IEEE International Conference on*. 2002; 1:369–372. vol.1. 10.1109/ICME.2002.1035795
- Wang S, Dey S. Rendering adaptation to address communication and computation constraints in cloud mobile gaming. *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*. 2010:1–6.
- WebCL - Heterogeneous parallel computing in HTML5 web browsers. (n.d.). Retrieved August 30, 2011, from <http://www.khronos.org/webcl/>
- WebGL Specification. (n.d.). Retrieved from <http://www.khronos.org/registry/webgl/specs/latest/>
- Williams, HP. *Model Building in Mathematical Programming*, 4th Edition. 4th. Wiley; 1999.
- Wimmer, M.; Wonka, P. Proceedings of the 14th Eurographics workshop on Rendering. Aire-la-Ville, Switzerland: Switzerland: Eurographics Association; 2003. Rendering time estimation for real-time rendering; p. 118-129. Retrieved from <http://dl.acm.org/citation.cfm?id=882404.882422>
- Wolf W, Jerraya AA, Martin G. Multiprocessor system-on-chip (MPSoC) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2008; 27(10):1701–1713.
- Wong, G.; Wang, J. Proceedings of The 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry. New York, NY, USA: ACM; 2008. Control theory based real-time rendering; p. 40:1-40:3.
- Wong G, Wang J. Dynamics of 3D polygonal rendering. *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*. 2010:766–771.10.1109/SIBIRCON.2010.5555085

Highlights

We proposed a optimization model for web based interactive simulations

Proposed method includes identification, optimization and update phases

Nonlinear model can be converted to linear optimization model

Optimization model handles the infeasible solutions

Final model effectively modifies simulation based on device specification

Figure 1. Optimization framework phases

(a)

(b)

Figure 2. (a) Raw and (b) normalized error in FPS for Chrome OS, MacBook Pro and iPad 2

Figure 3. Rendered image quality for the liver model before and after optimization, respectively on the Chrome Book (a,b); MacBook Pro (c,d) and iPad 2 (e,f)

Figure 4. MacBook Pro optimized versus non-optimized: (top left) default rendering (bottom left) optimized rendering in grayscale image (right top) edge detection on non-optimized (right bottom) edge detection on optimized case

Figure 5. Normalized error for simulation performed on the MacBook Pro (a), Chrome Book (b) and iPad 2 (c)

Listing 1

Client side proxy code

```

initWebGL();

 $T_{max} \leftarrow \text{getTexMax}();$ 
 $t \leftarrow \text{getNbrTex}();$ 
 $GMM \leftarrow \text{getGPUMem}();$ 
 $t_{vert} \leftarrow \text{getVertexTexSize}();$ 
 $\rho_{dbbits} \leftarrow \text{getDepthBufferBits}();$ 
 $\rho_{fbbits} \leftarrow \text{getFBufferBits}();$ 
 $\rho_{sbbits} \leftarrow \text{getStencilBufferBits}();$ 
 $Fb_{max} \leftarrow \text{getMaxFBSize}();$ 
 $p \leftarrow \min(\text{parseShader}(); \log_2(T_{max}));$ 
initShader();
forall  $i \in O$  do
    forall  $Fb_i$  do
        forall  $c_i$  do
            generate Mesh( $N_i$ );
            set Display( $c_i$ ;  $Fb_i$ );
            clear Stream(FPSstream);
            while  $\text{time}_{passed} < \text{time}_{testTime}$  do
                renderScene();
                FPSstream  $\leftarrow \text{getFPS}();$ 
            end
            asyncUpload( $N_i$ ;  $Fb_i$ ;  $c_i$ ; FPSstream);
            increase( $N_i$ );
        end
    end
end
forall  $i \in S$  do
    while Avg (FPSstream)  $\geq 1$  do
        generateMesh( $N_i$ );
        clear Stream(FPSstream);
        while  $\text{time}_{passed} < \text{time}_{testTime}$  do
            runSimulation();
             $F_i \leftarrow \text{getFPS}();$ 
            FPSstream  $\leftarrow F_i$ ;
        end
        asyncUpload( $N_i$ ; FPSstream);
        increase( $N_i$ );
    end

```

end

asyncUpload(T_{max} , t ; GMM ; t_{vert} ; ρ_{dbbits} ; ρ_{fbbits} ; ρ_{sbbits} ; Fb_{max} ; p);

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript