Imprecise Computation and Load Sharing in Computer Generated Imaging Systems

Marc Berger
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716 USA

Wei Zhao
Department of Computer Science
University of Adelaide
Adelaide, SA 5001 Australia

Abstract

The fast rendering of complex scenes is a fundamental problem in imaging systems. Imprecise computation is a two level scheme for executing screen update tasks. The *full* level specifies the maximum computation requirement of a rendering task while the *reduced* level specifies the minimum computation requirement which will render a region imprecisely but acceptably. Load sharing is a task allocation scheme which shares the working load of the system among all processors, better utilizing each processor and thereby reducing overall response time. This paper describes an integration of imprecise computation and load sharing techniques which reduce image generation time while maintaining high image quality. No one has formally studied this integrated system and we do this with an analytical model which evaluates and compares CGI systems using only one of the techniques with systems which use both techniques. Our study demonstrated that the coupled system features faster task response time and high image quality even when the system has been over-loaded.

Keywords: image synthesis, real-time image display, distributed computer system, imprecise computation, task allocation, load sharing/balancing.

Introduction

One of computer graphics most challenging tasks is to generate realistic images in real-time. While recent technological advances in hardware and software have brought about major strides towards this lofty goal, the major stumbling block is that the combination of rendering speed and high quality image performance requirements play off against each other. As the number of objects describing an image increases so does computation time, and this increase is often linear.

Computer Generated Imaging (CGI) system designers are often forced to choose between realism and speed, depending upon the requirements of the system or the inclination of the designer. CAD systems often draw wire frame images to achieve real-time response but are willing to tolerate acceptable delays of several seconds to produce shaded images. In [9] an adaptive refinement process, and spare CPU cycles, is used to quickly render an image at different levels of complexity. At the other extreme,"fast" ray tracing of complex scenes consisting of hundreds of thousands of objects consume several hours of processing time on single processor systems [11]; but this time decreases directly as the number of processors increases [1].

High performance visual training simulation systems, such as Redifussion/Evans & Sutherland CT6 image generator [19], sacrifice visual exactness for speed to achieve a high level of scene fidelity while maintaining a screen update rate of 20 - 30 hertz. These specially designed systems, costing in excess of 5 million dollars, attain this performance by using a high degree of parallelism in which work is distributed among several processor nodes. Crucial to their performance is a well designed data base which supports dynamically varying multiple levels of detail which is based upon the distance an object is viewed. Typically, as an object gets closer to the viewer unresolved objects are transformed to resolved objects by increasing detail. Objects that are located far from the viewer or at the edges of the field of view are still imme-

diately recognizable although displayed in reduced detail [22]. This technique takes advantage of the vast redundancy of visual cues in a typical scene and the ability of the brain to fill in detail based upon previous experiences [2].

These attempts to reduce image generation time can be formalized into two categories: imprecise computation and distributed computing. The technique of *imprecise computation,* recently studied by a number of researchers [10, 15, 16, 17], chooses to trade off computational accuracy for faster response time. That is, for some applications, a partial execution of an imaging task, which yields a faster response time and imprecise-but-acceptable results, may be better than a full execution of a task which yields the highest possible precision of computation with a longer response time. A critical issue in an imprecise system is that the partial execution of tasks should be properly controlled in order to keep the computation quality at an "acceptable" level.

On the other hand, computer networks impact task response time by *sharing the working load* of the individual nodes (i.e., transferring tasks) among all nodes in the system [20]. In this way, the computation resources on each node can be better utilized and the task response time can be reduced. The computation quality of a load sharing system is never degraded. However, as we will see, its ability to reduce the response time is very limited compared with an imprecise system.

In this paper we introduce a new model that properly couples imprecise computation and load sharing techniques in local area networks to reduce processing time while maintaining a high visual quality. Clearly, any set of criteria used to measure the effectiveness of an image generation system is different for real-time and non-real-time systems [6]. For our analysis, we have chosen three important performance metrics: 1) the mean task waiting time, 2) the mean task served computation time, and 3) the fraction of the display screen which is precisely processed. The first metric assesses the image generation time while the latter two indicate the image quality.

Computation Model

We adopt a *two level imprecise computation model* for rendering tasks. Typically a task is a request to update a specified part of the scene. In our system, each task has two levels of computation time requirements: the *full level* requirement specifying the maximum computation time needed by the task, and the *reduced level* computation requirement defining the minimum computation time needed by the task. If a task is allocated this reduced amount of processing time, its corresponding screen area is rendered imprecisely but acceptably.

There are several features in scene generation that could be affected by a reduced level of computation. These include, but are not restricted to, geometric modelling, motion dynamics, lighting characteristics, texture detail and aliasing artifacts. In general, tasks that update areas in the center of the screen have a higher precise computational requirement than tasks updating areas near the edges of the screen.

We note that this simple two level task computation model covers a wide range of application tasks. For example, if a task's reduced level computation time is zero, then this task can be dropped in case the system is too busy. On the other hand, if the reduced level computation time is the same as the the full level one, then this task is so important that it should never be partially processed.

Each update request (task) has its unique processing requirement which depends upon the visual complexity of its designated screen area. To simplify the analysis, in this paper, we assume that the (full level) task computation time is exponentially distributed with a mean of $1/\mu$. The reduced level computation time of a task is a fraction R of the full level computation time of the same task, where R is a constant (an overall average) between 0 and 1. Consequently, the reduced level computation time is also exponentially distributed, with a mean of R/μ. In [3], Chong and Zhao showed that the derived analytical formulas, assuming a system with constant R, are good approximations for systems in which the R parameter is not constant but a random variable. A theoretical investigation of this issue is beyond the scope of this study, and is currently being undertaken in a separate paper [5].

System Model

The functional requirements of a CGI system can be decomposed into three processes:

1. Scene Manager - This process, residing in the host computer, selects the level of detail for those objects that are potentially visible within the current field of view. It

also sends out task requests for an area of the screen to be updated. The display screen is divided into equal size blocks of pixels to reduce communication overhead that would occur if just one pixel was transferred at a time. These requests are randomly routed to a rendering processor node.

2. Renderer - This process performs the viewing transformation, clipping and finally calculates the color of each pixel in the block it receives from the scene manager. Identical code consisting of different rendering algorithms is loaded onto each processor.

3. Displayer - This process collects the results from the renderer process and transmits them to the display device.

The scene manager's major functions are to decide what information to send and where to send it. In some systems, the scene manager (perhaps in conjuction with a controller process) will monitor the load of the rendering processor nodes [22]. This would allow the scene manager to dynamically change the level of detail and select a lightly loaded processor. We refrain from doing this in our system since this is a cause of a potential bottleneck. Instead we distribute the scheduling function to the processor nodes.

The set of identical nodes executing the renderer process form a local area network connected by an ethernet. We assume that screen update tasks arrive to the scene manager from the external world randomly as a Poisson process. These tasks can be generated either through interactive actions (eg. CAD or flight simulations) or through system requests (eg. animated sequences). Each task is then randomly passed on to a rendering processor node arriving as a Poisson process with arrival rate λ_e. In the case a task needs to be transferred to another node, we assume that an average communication overhead d, representing communication and transfer processing delays, is required to transfer a task.

We call $\rho_e = \lambda_e/\mu$ the *external load* of the system. ρ_e represents, in terms of the task full level computation, the amount of computation requirement proposed to a node from the scene manager. Note that ρ_e is larger than the real load a node experiences (i.e., the processor's ***utilization***) if any of the tasks are served at the reduced level.

Load Sharing Scheme

The *load sharing (task allocation) scheme* is invoked upon an external rendering task arrival at a node. It makes the decision *where* the task is to be executed. The objective of this load sharing scheme is to distribute (share) the working load among all the nodes such that the processors on each node can be better utilized and the over all response time of tasks can be reduced. When a task is allocated to a node by the load sharing scheme, the task is put in a task queue, waiting to be executed by the processor on the node.

The load sharing scheme we adopt is as follows:
Upon an external task arrival at a node ν, the total number of tasks on node ν is checked. This number includes the task executing and those on the node's queue.
If the total number of tasks on node ν is at least S, a system parameter denoting *send*, then an attempt is made to transfer the newly arrived external task to some other node. This is done as follows: node ν broadcasts over the local area network the information of the newly arrived task. Then each other node sends node ν its current queue length. Because nodes respond to the request at different times, depending upon their current status, we are not permitting node ν to collect information from *all* the nodes. Instead node ν collects this information from a total of L nodes, a system parameter denoting the response *limit*. If the number of tasks in any responding node, ν', is less than A, another system parameter denoting *accept*, the newly arrived external task is transferred to ν' and executed.
In the case a newly arrived external task is not transferred to other nodes, the task is queued in a *ready queue*, and will be executed locally on node ν.

We note that a node should not accept any task from another node if it is sending out its own external tasks. Hence, the upper bound of parameter A is S. This fact is used in the analysis and evaluation of the system performance in the rest of the paper.

Node Scheduling Scheme

When the processor is ready to execute a task, the *node scheduling scheme* is invoked to decide *how* the task is to be executed — at the full level or the reduced level. The objective of task scheduling is to improve the task response time while maintaining computation quality as high as possible.

The node scheduling scheme we adopt for this study is as follows: When a processor node is ready to execute a task, the total number of tasks in the node is checked. If the number of tasks in the node is less than P, a system parameter denoting *partial processing*, the task's computation is performed at the full level, otherwise the task is performed at the reduced level. While a task is in execution, the number of tasks in the system is not monitored. A task receives a full level computation even if the number of tasks in the system exceeds P (due to new arrivals during its execution), as long as at the beginning of its execution, the number of tasks is less than P.

Performance Metrics

For our imprecise CGI system with the load sharing and scheduling schemes proposed in the last section, the following performance metrics are important: The first performance metric is *the Normalized Mean Task Waiting Time*, denoted as W_n.

$$W_n = \frac{W_q + dP(\text{a task is transferred})}{\text{Mean Task Full Level Comp. Time}} \quad (1)$$

where W_q is the mean time spent waiting in the ready queue, the mean task full level computation time is $1/\mu$, and d is the average delay of transferring a task.

Using Little's result [13, 14], we can write

$$W_q = N_q/\lambda_e. \quad (2)$$

where N_q is the mean number of tasks waiting to be served. Let j be the number of tasks on a node upon an external task arrival. With our task allocation and scheduling schemes,

$$P(\text{task transfer}) = P(j \geq S)(1 - P^L(j \geq A)). \quad (3)$$

In the right side of the above formula, the first term is the probability that the node at which a task arrives has at least S tasks when it arrives. The second term means that among L probes regarding transferring this task, at least one of them is successful, and hence the new task is transferred. Hence

$$W_n = \frac{N_q/\lambda_e + dP(j \geq S)(1 - P^L(j \geq A))}{1/\mu} \quad (4)$$

$$= N_q/\rho_e + DP(j \geq S)(1 - P^L(j \geq A)). \quad (5)$$

where $D = d\mu$. We call D the *normalized communication overhead*. It represents the communication overhead measured in the unit of the mean task full level computation time.

W_n assesses the speed of generating an image. As W_n approaches 0, tasks spend less time waiting to begin their execution and hence complete their execution faster. If all tasks were fully processed, W_n would completely measure the performance of the imaging system. However, with the addition of imprecise computation, the quality of the image must also be measured.

The second performance metric we consider is *the Normalized Mean Task Served Computation Time*, denoted as Q_c. This metric is defined as

$$Q_c = \frac{\text{Mean Task Served Comp. Time}}{\text{Mean Task Full Level Comp. Time}} = \frac{\text{Mean Task Served Comp. Time}}{1/\mu} \quad (6)$$

where the Mean Task Served Computation Time is the average computation time a task receives. This metric indicates the average computation quality over all tasks executed. Because not all tasks receive the full level computation time, Q_c is a value between R and 1. The larger the value of this metric, the higher the average quality of computation resulting in a more detailed image.

Let U be the utilization factor of the processor. Q_c can be computed as follows:

$$Q_c = \frac{\text{Mean Task Served Comp. Time}}{1/\mu} = \frac{U/\lambda_e}{1/\mu} = U/\rho_e \quad (7)$$

The third performance metric is *the Fraction of the Screen that is Fully Processed, Q_t*. This is defined as (for a give time interval)

$$Q_t = \frac{\text{Mean Size of Screen Area Fully Proc.}}{\text{Mean Size of Screen Area Proc.}} = \frac{\text{Mean Number of Tasks Fully Proc.}}{\text{Mean Number of Tasks Executed}}. \quad (8)$$

where each task updates the same amount of screen area. Q_t specifies how many tasks receive the full level (precise) computation. With this definition of Q_t, we see that

$$\text{Mean Task Svd. Comp.Time} = Q_t\frac{1}{\mu} + (1 - Q_t)\frac{R}{\mu}. \quad (9)$$

That is, by (6),

$$Q_t + R(1 - Q_t) = Q_c. \quad (10)$$

Substituting (7) into (10) and solving for Q_t we have

$$Q_t = \frac{U - R\rho_e}{(1 - R)\rho_e}. \quad (11)$$

N_q, U, $P(j \geq S)$, and $P(j \geq A)$ are derived later in the paper.

Performance Analysis

To compute the performance metrics, we first construct an approximate analytical model for our system. Once the analytical model is solved, the performance metrics can be easily calculated.

To perform the analysis of the imprecise distributed system, a Markov model of the system can be constructed. However, the state space is very large and complex. In previous studies (e.g., [7]), certain approximation assumptions are made so that the system model can be decomposed and/or simplified. It turns out that this approximation analysis is necessary and successful in the sense that it is usually asymptotically exact and it simplifies the analysis which helps to reveal the important aspects of the system characteristics. Because of this, we also take an approximation approach in this paper.

We assume that the state of each node is stochastically independent of the state of any other node. With this assumption, each node can be analyzed separately. Further, because the network is homogeneous, the system performance measurements can be obtained by analyzing any one of the individual nodes.

We further assume that tasks arriving at a node from other nodes form a Possion process with rate of λ_i. In addition, when the total number of tasks on a node is S or more, the arrivals of the external tasks which cannot be transferred from this node[1] also form a Possion process with rate of $\lambda_{e'}$. With these latter two assumptions, the arrival rates to the queue on a node can be determined: Let j be the total number of tasks on a node. The arrival rate to the ready queue on a node is $\lambda_e + \lambda_i$ if $j < A$; λ_e if $A \leq j < S$, and $\lambda_{e'}$ if $j \geq S$.

In the rest of this paper, we denote $\lambda_e + \lambda_i$ by λ, $(\lambda_e + \lambda_i)/\mu$ by ρ, and $\lambda_{e'}/\mu$ by $\rho_{e'}$.

We examine the system behaviour only when it is in a stable state. It is easy to see that the condition for the system to be stable is

$$R\rho_e < 1, \tag{12}$$

that is,

$$\rho_e < 1/R, \tag{13}$$

where $\rho_e = \lambda_e/\mu$. We assume that the above condition is always satisfied throughout this paper.

With the assumptions stated above, the state of a node can be represented by a pair (i, j) where the integer $i \geq 0$ indicates the total number of tasks in the node, and j is either 0 or 1, where 1 corresponds to state when the system is performing reduced level computation and 0 corresponds to state when the node is performing full level computation or is idle. Denote the equilibrium probability that the node is in state (i, j) by $P_{i,j}$. $P_{i,j}$ is solved in [24]. The results are as follows for $0 < i < P, P \leq i < A, A \leq i < S$, and $i \geq S$, repectively:

$$P_{i,0} = \begin{cases} \rho^i P_{0,0} \\ \frac{\rho^i}{(1+\rho)^{i-P+1}} P_{0,0} \\ \frac{\rho^A \rho_e^{i-A}}{(1+\rho)^{A-P}(1+\rho_e)^{i-A+1}} P_{0,0} \\ \frac{\rho^A \rho_e^{S-A} \rho_{e'}^{i-S}}{(1+\rho)^{A-P}(1+\rho_e)^{S-A}(1+\rho_{e'})^{i-S+1}} P_{0,0} \end{cases} \tag{14}$$

and

$$P_{i,1} = \begin{cases} 0 \\ \frac{R\rho^{i+1}}{(1+\rho)^{i-P+1}} \frac{1-B^{i-P+1}}{1-B} P_{0,0} \\ \frac{R\rho^A \rho_e^{i-A}}{(1+\rho)^{A-P}} \left\{ \frac{\rho_e(1-B_e^{i-A+1})}{(1+\rho_e)^{i-A+1}(1-B_e)} + \frac{R^{i-A+1}\rho(1-B^{A-P})}{1-B} \right\} P_{0,0} \\ \frac{R\rho^A \rho_e^{S-A} \rho_{e'}^{i-S}}{(1+\rho)^{A-P}} \left\{ \frac{1}{(1+\rho_e)^{S-A}} \left[\frac{\rho_{e'}(1-B_{e'}^{i-S+1})}{(1+\rho_{e'})^{i-S+1}(1-B_{e'})} + \frac{R^{i-S+1}\rho_e(1-B_e^{S-A})}{1-B_e} \right] + \frac{R^{i-A+1}\rho(1-B^{A-P})}{1-B} \right\} P_{0,0}. \end{cases}$$

where B, B_e, and $B_{e'}$ are defined as

$$B = R(1+\rho), \tag{15}$$

$$B_e = R(1+\rho_e), \tag{16}$$

$$B_{e'} = R(1+\rho_{e'}). \tag{17}$$

For $0 < i \leq A, A < i < P, P \leq i < S, i \geq S, i < P$ and

[1] Due to all the L probes being unsuccessful.

$P \leq i < S$, respectively:

$$P_{i,0} = \begin{cases} \rho^i P_{0,0} \\ \rho^A \rho_e^{i-A} P_{0,0} \\ \frac{\rho^A \rho_e^{i-A}}{(1+\rho_e)^{i-P+1}} P_{0,0} \\ \frac{\rho^A \rho_e^{S-A} \rho_{e'}^{i-S}}{(1+\rho)^{S-P}(1+\rho_e)^{i-S+1}} P_{0,0} \end{cases} \quad (18)$$

$$P_{i,1} = \begin{cases} 0 \\ \frac{R\rho^A \rho_e^{i-A+1}}{(1+\rho_e)^{i-P+1}} \frac{1-B_e^{i-P+1}}{1-B_e} P_{0,0} \\ \frac{R\rho^A \rho_e^{S-A} \rho_{e'}^{i-S}}{(1+\rho_e)^{S-P}} \left\{ \frac{\rho_{e'}(1-B_{e'}^{i-S+1})}{(1+\rho_{e'})^{i-S+1}(1-B_{e'})} \right. \\ \qquad \left. + \frac{R^{i-S+1}\rho_e(1-B_e^{S-P})}{1-B_e} \right\} P_{0,0} \end{cases} \quad (19)$$

where B_e and $B_{e'}$ are defined in (16) and (17) respectively.

For $0 < i \leq A, A < i \leq S, S < i < P, i \geq P, i < P$, and $i \geq P$, respectively:

$$P_{i,0} = \begin{cases} \rho^i P_{0,0} \\ \rho^A \rho_e^{i-A} P_{0,0} \\ \rho^A \rho_e^{S-A} \rho_{e'}^{i-S} P_{0,0} \\ \frac{\rho^A \rho_e^{S-A} \rho_{e'}^{i-S}}{(1+\rho_{e'})^{i-P+1}} P_{0,0} \end{cases} \quad (20)$$

$$P_{i,1} = \begin{cases} 0 \\ \frac{R\rho^A \rho_e^{S-A} \rho_{e'}^{i-S+1}}{(1+\rho_{e'})^{i-P+1}} \frac{1-B_{e'}^{i-P+1}}{1-B_{e'}} P_{0,0} \end{cases} \quad (21)$$

where $B_{e'}$ is defined in (17).

Note that all the above expressions of the state probabilities are given in terms of $P_{0,0}$. With the normalization condition that

$$\sum_{i=0}^{\infty} (P_{i,0} + P_{i,1}) = 1, \quad (22)$$

we solve $P_{0,0}$ as follows:

$$P_{0,0} = \left\{ \frac{\sum_{i=0}^{\infty} (P_{i,0} + P_{i,1})}{P_{0,0}} \right\}^{-1}. \quad (23)$$

With the solution expressions of $P_{i,j}$ ((14), (15), (18), (19), (20), (21), and (23)), the parameters involved in computing the state probabilities are λ_e, λ_i, $\lambda_{e'}$, μ, P, A, S, and R. Among these parameters, for a given system, λ_i and $\lambda_{e'}$ are unknown. To solve the unknown parameters λ_i and $\lambda_{e'}$, we need to establish two new equations.

Let j be the total number of tasks on a node when an external task arrives at the node. According to our task allocation scheme, when $j \geq S$, if using up to L probes, a node cannot find another node to send a newly arrived external task, this task has to be processed locally. Hence, the arrival rate of tasks to the queue when $j \geq S$ is

$$\lambda_{e'} = \lambda_e P^L(j \geq A). \quad (24)$$

where the last term of the right side is the probability that none of the L probes succeeds.

In the homogeneous system that we consider in this paper, the arrival rate of transferred tasks must equal the rate of task transfers. Hence,

$$\lambda_i P(j \leq A) = (\lambda_e - \lambda_{e'}) P(j \geq S)(1 - P^L(j \geq A)) \quad (25)$$

With the expressions of $P_{i,j}$, the solution of equations (24) and (25) may be found numerically by any appropriate method.[2] All the values of parameters in $P_{i,j}$ are now known, and hence the values of $P_{i,j}$ are computed.

Once the state probabilities are found, the terms needed in determining the performance metrics W_n, Q_c, and Q_t can be computed easily. The probability that there are K or less tasks in a node is given by

$$P(i \leq K) = \sum_{i=0}^{k} (P_{i,0} + P_{i,1}) \quad (26)$$

For processor utilization, we have

$$U = 1 - P_{0,0} \quad (27)$$

The mean queue length in a node can be computed as

$$N_q = \sum_{i=1}^{\infty} (i-1)(P_{i,0} + P_{i,1}). \quad (28)$$

And the mean number of tasks in a node is

$$N = \sum_{i=1}^{\infty} i(P_{i,0} + P_{i,1}). \quad (29)$$

Substituting (26), (28) and (27) into (5), (7), and (11) respectively, the values of the performance metrics W_n, Q_c, and Q_t can be computed.

[2] We used fixed point iteration to obtain the numerical results.

Performance Sensitivity

The performance of the CGI system is directly related to the values of the parameters L, S, and A which are *directly used* in load sharing. We now examine this sensitivity.

From [24], we found that as L increases, the response time decreases while the image quality increases (ie. W_n decreases and Q_c and Q_t increase) — an improvement of overall system performance. However, after $L = 5$, the improvement of further increasing L is not significant. The detailed analysis can be found in [24]. Similar observations have been made in the system without using imprecise computation model [7].

Parameter S is the threshold for determining if an attempt should be made to transfer a newly arrived external tasks. From our analysis we have the following observations: The system performance metrics, W_n, Q_c, and Q_t, are all affected by selection of S. Too large or too small of an S value would result in a poorer system performance. There is no single value of S which can minimize W_n and maximize Q_c and Q_t at the same time. For example, in the case the external load is 1.1, $S = 7$ results in the minimum W_n, while $S = 5$ yields the maximum Q_c and Q_t. However, the difference between the S value which results in the minimum W_n and the one which yields the maximum Q_c and Q_t is usually small. Any value between these two S values (e.g., $S = 5, 6$, or 7 in the case external load is 1.1) should produce reasonably good performance.

A question of practical interest then is what value should S be? In [10], S has been determined as a function of N — the mean number of tasks on a node. We noticed a rule of thumb for determining the value of parameter S: if $S = N$, the system performance is at or very close to the optimum: W_n is minimized and Q_c and Q_t are maximized. For example, in the case the external load is 1.2, at $S = 7$, $N \approx S$. At that time, W_n is at its minimum while Q_c and Q_t are near their maximum.

Parameter A is used to determine when a node should accept tasks from other nodes. If the total number of tasks on a node is less than A, then this node can accept tasks from other nodes. A node should not accept any task if it has started to send out its own task. Hence, as we discussed at the beginning of this paper, A should not be larger than S — the threshold for sending tasks. So the upper limit of A is S. The lower bound of A is obviously 1.

Here, we are interested how a value of A between its upper bound and lower bound affects the system performance W_n, Q_c, and Q_t. Specially, we are concerned how a value of parameter A should be selected such that the system has the best or near best performance. From our analysis the following observations can be made. The selection of parameter A clearly affects the system performance. Too small or too large values of parameter A would result in a poorer system performance. Let A_{ow} be the value of A such that the system has the lowest mean task waiting time W_n. That is, A_{ow} is a non increasing function of parameter D. By the fact that A's selection is also based on its upper bound S, we may propose to calculate A_{ow} approximately as

$$A_{ow} = S - \alpha(D) \tag{30}$$

where $\alpha(D)$ is a non-decreasing function of D and $0 \leq \alpha(D) \leq S$. Our data suggest that $\alpha(1) = 1$, $\alpha(5) = 3$, etc. The other values of $\alpha(D)$ may be determined by an appropriate interpolation method. Also, we further note that when $A = A_{ow}$, the values of Q_c and Q_t is at or very close to their optimal (highest) values. Hence, (30) may be used to calculate the value of parameter A which can yield the optimal or near optimal performance.

Conclusion

This work represents a successful integration of the use of both load sharing and imprecise computation techniques in CGI systems to reduce image generation time while maintaining a high visual quality. To evaluate this new system model, we proposed three performance metrics — the normalized mean task waiting time (W_n), the normalized mean task served computation time (Q_c), and the fraction of the screen that is fully processed (Q_t). These performance metrics, together, assess the quality of the image and the speed at which it generated. An analytical model was developed to evaluate the system performance. Using the results from the analytical model, we compared the performance of CGI systems which use either load sharing or imprecise computation with systems which use both imprecise compuation and load sharing. It was found that using both imprecise computation and load sharing techniques are necessary. Comparatively, using only one of the two techniques alone, a system could

suffer from sluggish performance or low image quality, especially when the system load is high.

References

[1] P. Aktkin, S. Ghee and J. Packer, "Transputer architectures for ray tracing", *Proceedings: Computer Graphics 87, Computer Animation*, 1987.

[2] R. Clapp, "Comparisons of performance in various visual systems common to simulation", *Summer Computer Simulation Conference*, 1986.

[3] E. K. P. Chong and W. Zhao, "Performance Evaluation of Imprecise Computer Systems," submitted for journal publication, September 1988.

[4] E. K. P. Chong and W. Zhao, "Equilibrium Behaviour of Queues with Queue Length Dependent Random Service Rates," in preparation.

[5] R. Deyo, J. Briggs, P. Doenges, "Getting Graphics in Gear: Graphics and Dynamics in Driving Simulation", *Proceedings: Computer Graphics*, 1988.

[6] D. Eager, E. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Trans. Software Eng.*, vol. SE-12, No. 5, pp 662-675, 1986.

[7] D. Eager, E. Lazowska, and J. Zahorjan, "The Limited Performance Benifits of Migrating Active Processes for Load Sharing", *Perfromance Evaluation Review*, Vol. 16, No. 1, pp 63-72, May 1988.

[8] L. Bergman, H. Fuchs, E. Grant, and S. Spach, "Image Rendering by Adaptive Refinement", *Proceedings: Computer Graphics*, 1986.

[9] C. H. Hsu and J. W. Liu, "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems", in *Proc. IEEE 6th Inter. Conf. on Distributed Computing Systems*, pp 216-223, 1986.

[10] T. Kay and J. Kayiya, "Ray Tracing Complex Scenes" *Proceedings: Computer Graphics*, 1986.

[11] B. Kim and D. Towsley, "Dynamic Flow Control Protocols for Packet-Switching Multiplexers Serving Real-Time Multipacket Messages," *IEEE Transactions on Communications*, Vol. COM-34, No. 4, April 1986.

[12] L. Kleinrock, *Queueing Systems — Volume 1: Theory & Volume 2: Computer Applications*, John Wiley and Sons, 1975 (Vol. 1) & 1976 (Vol. 2).

[13] J. D. C. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research*, vol. 9, pp 383–387, 1961.

[14] K. Lin, S. Natarajan, and J. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems", in *Proc. of IEEE Real-Time Systems Symposium*, 1987.

[15] J. Liu, K. Lin, and C. Liu, "Concord Prototype System and Real-Time Scheduling," *Proc. of IEEE 4th Workshop on Real-Time Operating Systems*, July 1987.

[16] J. Liu, K. Lin, and S. Natarajan, "Scheduling Real-Time, Periodic Jobs Using Imprecise Results," *Proc. of IEEE Real-Time Systems Symposium*, 1987.

[17] D. Shorrock, "Visual systems - The state of the art", *Summer Computer Simulation Conference*, 1986.

[18] J. Stankovic, "A Perspective on Distributed Computer Systems," *IEEE Trans. Comp.*, Vol. C-33, No. 12, 1984.

[19] J. Yan, "Advances in Computer-Generated Imagery for Flight Simulation", *IEEE Computer Graphics and Applications*, August 1985.

[20] Y.Yemini, "A Bang-Bang Principle for Real-Time Transport Protocols," *Proc. SIGCOMM '83 Symp. Commun. Architect. Protocols*, 1983.

[21] W. Zhao and M. Berger, "An Analytic Model for Imprecise Distributed Computer Systems", Australian Computer Science Communications, Jan. 1990.