Modeling Real-time Rendering

Chee-Kien Gabriyel Wong and Jianliang Wang

Nanyang Technological University, Singapore

Abstract

The real-time rendering process is well known to be extremely dynamic and complex. This paper presents a novel approach to modeling this process via the system identification methodology. Given the process's dynamic nature arising from the possible myriad variations of render states, polygon streams and the non-linearities involved, we describe a modeling approach using neural networks with supervised training from application-generated data. By comparing the outputs of the neural network model's representation of the rendering process with actual empirical data, we discuss the accuracy of our approach in relation to the practical issues of integrating this study to real-world applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Interaction techniques

1. Introduction

The key challenge in real-time computer graphics is to provide the highest level of realism in the generated imagery at interactive frame rates given a fixed set of computing resources. Although research done in the past attempted to address the frame latency problem by providing mathematical models of the rendering process, these models were often primitive because they were derived from some approximate experimentation or they depended on specific application-level data structures (such as projecting render times based on small polygon count quantities [FS93] and using a particular scene description format [WW03]). Furthermore, with the advent of more powerful and programmable consumer graphics hardware in the coming years, the rendering pipeline is being used in an increasingly complex manner to achieve ultra-realistic visual effects. Consequently, it could be progressively challenging to adopt these models into the current applications as hardware and software technolgy continue to evolve.

While much of past research focused on finding an accurate formulation to describe real-time rendering, this paper discusses their inadequacies with reference to the real-world applications of today and proposes an approach by which an accurate model of the rendering process may be obtained through the system identification methodology [Lju87].

2. Previous Work

Over a decade ago, Funkhouser and Sequin [FS93] described a predictive approach to controlling frame rate through a rendering system that was defined by a benefit-cost model. The elements of this model consists of an *object tuple*, *(O,L,R)* where O is the instance of an object, L, being the level-of-detail of the object instance and R, the rendering algorithm associated to it. An algorithm based on constrained optimization was developed to select the best level-of-detail to match the user selected frame rate. The cost defined in terms of rendering time was:

$$Cost(O,L,R) = max \begin{Bmatrix} C_1 Poly(O,L) + C_2 Vert(O,L) \\ C_3 Pix(O) \end{Bmatrix} \qquad (1)$$

where Poly is the polygon count, Vert is the number of vertices, Pix is the time taken per pixel stage of rendering an object and C_1, C_2 and C_3 are constant coefficients specific to a rendering algorithm and machine.

Apart from the inaccuracy mentioned by Wimmer and Wonka [WW03] on this formulation in terms of accounting for just polygons and vertices of visible objects instead of the actually transformed ones, it is important to note that the polygon processing performance of graphics hardware is generally non-linear. Hook and Bigos [HB97] provided empirical evidence that the time required to process a single polygon or vertex varies at different overall quantities even with some fixed parameters such as the rendering state and display resolution. As a result, the calculation of polygon processing time based on the best fitting line across the stated four level-of-details (LODs) of the sample object may not be an accurate basis for rendering time estimation per polygon. At the same time, this non-linear polygon processing characteristic of graphics hardware would weaken Aliaga and Lastra's [AL99] proposed rendering time formulation based on:

$$RenderingTime = C_1 * NumberOfVisibleTriangles \qquad (2)$$

since C_1 is defined as a static triangle processing rate given by the hardware. Wimmer and Wonka [WW03] provided a thorough insight into the rendering process by listing several other underlying factors that contribute to the total rendering time. Their proposed rendering time formulation was:

$$RT = ET_{system} + max(ET^{CPU}, ET^{GPU}) \qquad (3)$$

with

$$ET^{CPU} = ET^{CPU}_{nr} + ET^{CPU}_{r} + ET^{CPU}_{mm} + ET^{CPU}_{idle} \qquad (4)$$

and

$$ET^{GPU} = ET^{GPU}_{fs} + ET^{GPU}_{r} + ET^{GPU}_{mm} + ET^{GPU}_{idle} \qquad (5)$$

where

- *RT* is the total render time
- *ET*, the estimated time
- *system*, all other system tasks
- *nr*, the non-rendering tasks
- *r*, the rendering tasks
- *mm*, the memory management tasks
- *idle*, the processor idle or waiting time
- *fs*, frame setup

Although this formulation attempts to be all-encompassing by considering the key and important processes associated to rendering, there are however too many subtasks subsumed under various elements of the equation. Hence, the practicality of such a formulation becomes questionable since it could be impossible or non-trivial to measure these subtasks unless the test scenes are sufficiently simple.

2.1. Frame time

While the term *rendering* technically refers to the generation of graphics or imagery, it is of common knowledge that it includes other related processes. The reason is twofold - before dedicated hardware for rendering was introduced the central processing unit and main memory was shared amongst graphical and non-graphical processes; all practical computer-generated graphics applications such as as virtual reality walkthroughs, games and simulation software does not consist of pure rendering functions alone. This is best illustrated in many well known real-time rendering toolkits such as Performer [RH94] and OpenSG [KWE03] in which processes are classified into the *APP*, *CULL* and the *DRAW* stages. Examples of such processes from the *APP* stage are database manipulation, networking, input-output control and logic computation. Hence, the formulation of a frame time estimation framework requires not only inclusion of relevant time quantities but also the possibility of tracking or measuring them as well. This is especially non-trivial in applications such as games where content and state changes are dynamic. For instance, it is possible to transfer a non-rendering process that used to run on the CPU to the GPU based on the programmable graphics hardware today. However, this may not be captured by Wimmer and Wonka's formulation in Equation 5.

3. System identification

System identification is the process of establishing mathematical models of dynamic systems based on observed data from these systems [Lju87]. Depending on the *a priori* information available, a system may be modeled out of empirical data (*black-box* modeling) or some existent guiding physical principles (*white-box* modeling). Since it is non-trivial to formulate a function that is comprehensive in terms of describing real-time rendering, this research is focused on the *black-box* modeling approach using measured data. Figure 1 illustrates the procedures in system identification. It basically describes an iterative process by which a system model is derived from measured data through selection of a near-fitting model structure and subsequently computing accurate parameters for this model such that it eventually exhibits identical behavior as the actual system. The derived model is validated using an alternative set of input data different from those used for identification. Due to its wide adoption in many modeling and control applications and its data-driven nature, the system identification process was used in our research as a means to establish a credible model for describing the rendering process.

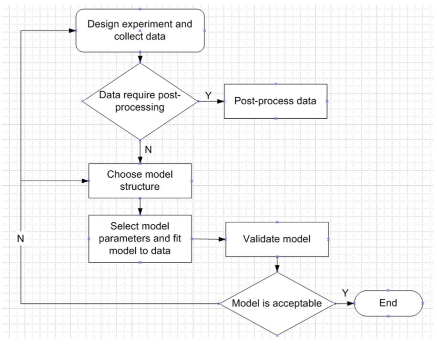
Design experiment and collect data
Data require post-processing
Y
Post-process data
N
Choose model structure
N
Select model parameters and fit model to data
Validate model
Model is acceptable
Y
End

Figure 1: *The system identification process.*

4. Modeling using neural networks

The first *artificial neuron* proposed by McCulloch and Pits [MP43] mimicks the function of the biological neuron through a multiple-input-single-output model. It is essentially a processing unit that sums up the weighted values of its inputs to produce an intermediate output. This output is then fed as the input to an activation function that produces the final output. With layers of interconnected neurons, an *artificial neural network* (ANN) is formed and is frequently used to simulate the function of many systems. Figure 2 illustrates the structure of the artificial neuron. ANNs need to be trained in order to capture the character-

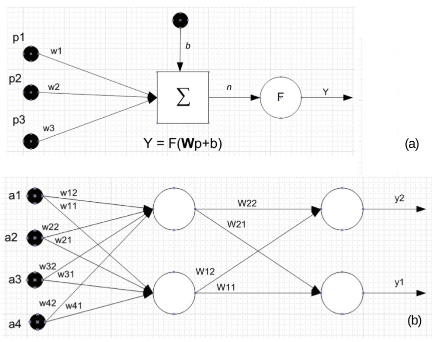

Figure 2: *(a): The perceptron neuron. (b): A multi-layer perceptron network (MLP).*

istics of the systems they model after. Since its inception, training algorithms for neural networks such as the back-propagation [RHW86] and Levenberg-Marquardt [Mar63] methods were developed to compute the weights and bias for the inputs.The neural network is adopted in our research for modeling the rendering process because of its ability to capture information from complex, non-linear, multi-variate systems without the need to assume any underlying data distribution or mathematical models. In recent years, there has also been increasing popularity in using multi-layer perception networks due to its impressive successes in real-world applications such as pattern recognition and control applications.

5. Implementation and Results

5.1. Test setup

Empirical data consisting of the triangle count per frame and frame rate were collected from two different applications running on a Pentium IV, 3.2Ghz processor with 1GB RAM and NVidia's GeForce 6800 graphics board. This was done via the user's free input of the camera's movement and orientation to simulate common navigation characteristics in virtual environments. The objective is to capture a wide range of polygon loads and a good combination of rendering features so that the rendered frames reflect properties which are representative of the application. Both applications rendered the animated frames in real-time. The first application was custom developed to encompass most common rendering parameters in applications such as textures, fog, lighting, animation, shaders, moderate depth complexity and varying polygon loads. It consists of a scene populated by hundreds of different instances of a bumpmapped ogre's head, an animated grassland using shaders and also moving clouds on a skydome. The second application is a popular game title, *Serious Sam 2*, published by 2KGames (`www.croteam.com`). Sample screenshots from the applications are provided in Figure 3. A randomly chosen game level was used and data were collected from trial runs of the game. Microsoft's DirectX tool, PIX (`http://www.microsoft.com/windows/directx/default.aspx`) together with NVidia's NVPerfKit (`http://developer.nvidia.com/object/nvperfkit_home.html`) and its instrumentation driver were used as the data collection tools due to its good support and ease-of-use for detecting low-level software and driver information. The next step

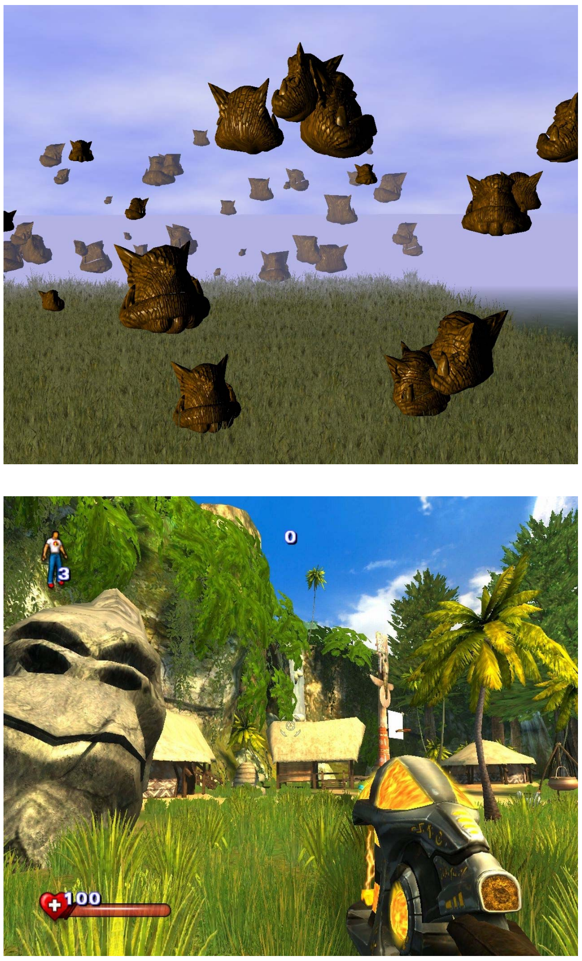

Figure 3: *Screenshots from (a) the first test application and (b) the second test application, Serious Sam 2, a game published by 2K Games.*

was to process the collected data by using the *Neural*

Network Plant Identification Tool [DB93] from Matlab. In accordance to the system identification methodology described in Section 3, a neural network was first selected as the model structure. The collected data were fed into the neural network to train it to generate an accurate mapping of the relationship between the input triangle count and the output frame rate. Different neural network structures and parameters were tried in order to obtain the best fitting model. This process went on in an iterative manner until the performance objective was met. This performance objective is simply a numeric quantity describing the difference between the predicted and actual frame rate. The same procedure was repeated for the second application.

5.2. Results

From the experiments conducted, the neural network used to model the first application consists of a MLP network with two layers, six units and three delay units in each of the input and output channels. The second neural network differs from the first with just four delay units in both the input and output channels. An example of the neural network used to model the first application is shown in Figure 4. The graphs

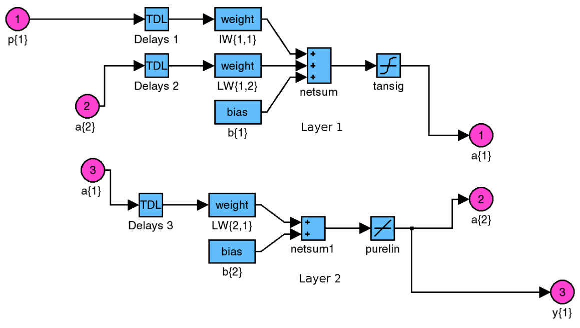

Figure 4: *The neural networks used for capturing the system properties of the first application.*

at the top-right and bottom-right in Figure 5 refer to the actual and predicted output frame rates of the application and neural network respectively. The difference between them is shown in the graph at the bottom-left in the same figure. The graph at the top-left corner shows the input (triangle count per frame) to the neural network model over the test period. It was observed that the mean difference between the frame rates generated by the neural network model and the actual application to be 0.00455. In the same order of arrangement, the graphs for the second experiment using the game are shown in Figure 6. The neural network was able to model closely the characteristics of the rendering process in the second application with a mean difference of 0.00896 in terms of frame rate. All networks were trained using the Levenberg-Marquardt algorithm over 200 epochs for over 5000 frame samples.

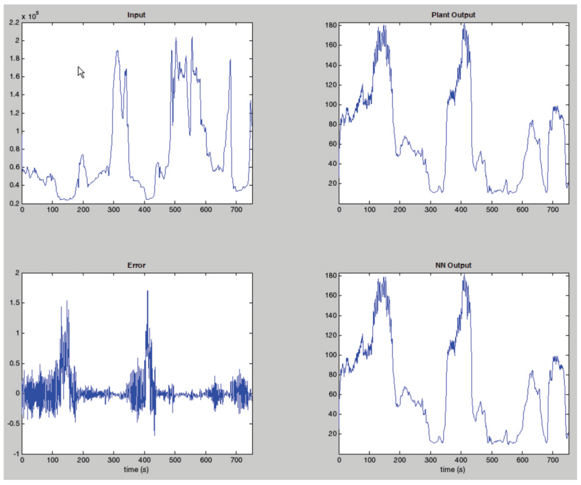

Figure 5: *The results generated by the neural network compared to the actual output in the first application.*

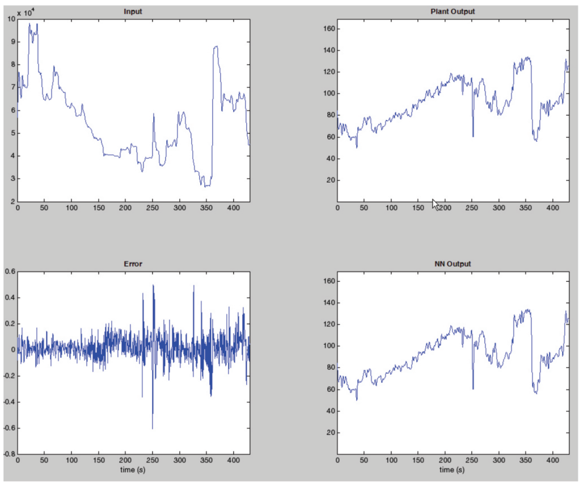

Figure 6: *The results generated by the neural network compared to the actual output in the second application.*

5.3. Discussion

The objective of our tests was to derive a model of the rendering process of an application. While it may seem ideal to have a single model for all applications however this is impractical because basically rendering software systems are dynamic and they vary in terms of the number of components contributing to the final render time. For example, it is common that applications differ in the type and number of processes in the *APP* stage (such as network communication, application logic and input-output processes). Trying to derive a model for this stage is tantamount to finding a general description of the non-drawing processes which is nontrivial. On the hand, over generalizing the model by grouping these processes together as a single consituent of the total rendering time limits the usefulness of the model since the breakdown is then not known and thereby cannot be examined in detail or controlled. Nevertheless, it is possible to derive a model at *application level* such that good results can be obtained when the model is tested with varying conditions within the scope of the application (such as different combat situations in a game level or across levels). This explains our approach of determining a single but useful model of a particular rendering process which is application-specific.

From Section 2, it is apparent that the rendering process consists of more than a single component or input. There is no contradiction to this fact even though we derived just the relationship between triangle count and frame rate using a neural network because the focus is to expose an input quantity (triangle count) that can be easily measured while keeping the other factors in the *black-box*. The purpose of doing so is related to future development of this concept such as applying suitable control strategies on the model by controlling the input. To illustrate, one powerful usage of this modeling approach is to develop a software component that is able to vary the polygon count per frame through usage of level-of-detail techniques or culling. By inserting this controlling element to a feedback loop, stable and interactive frame rates can be achieved during run-time.

As described in Section 5.2 the structural similarity between the two neural networks should not carry an implication that the two applications' features are similar because the weights of the two neural networks are different after training. Furthermore the game *Serious Sam 2* itself consists of more complex processes such as artificial intelligence and audio processing than the first application. However, this opens up additional scope for studies into finding possible relationship that may exist among various neural networks structures and their corresponding rendering processes.

On practical aspects and applicability of this research, data collection can be done easily via freely available tools and in a way transparent to the user. Once a satisfactory model is derived, there is no need to re-train the network unless the application changes. From the experiments, training of our neural network takes less than three minutes for a dataset of approximately 5000 data points on a typical Pentium IV system and this can be done off-line.

6. Conclusion

We described an approach to modeling real-time rendering using the system identification approach. The inadequacies in previous rendering time formulations due to incomplete or inaccurate descriptions were discussed. We then introduced neural networks as a means to model the rendering process and compared the outputs of the neural network models with those of the actual applications. Finally, a discussion on practical issues in implementing this approach was provided.

References

[AL99] Aliaga D. G., Lastra A.: Automatic Image Placement to Provide a Guaranteed Frame Rate. In *SIGGRAPH 1999, Computer Graphics Proceedings* (Los Angeles, 1999), Rockwood A., (Ed.), Addison Wesley Longman, pp. 307–316. 2

[DB93] Demuth H., Beale M.: *Neural Network Toolbox for use with MATLAB - User's Guide*. The Mathworks, Cochituate Place, 24 Prime Park Way, Natick, MA, USA, 1993. 4

[FS93] Funkhouser T. A., Séquin C. H.: Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. In *Proceedings of ACM SIGGRAPH Conference* (Aug 1993), pp. 247–254. 1

[HB97] Hook B., Bigos A.: 3D Acceleration Demystified, Part II: The Benchmarks. http://www.gamasutra.com/features/19970601/3d_acceleration_demystified.htm. 1

[KWE03] Klein T., Weiler M., Ertl T.: A Volume Rendering Extension for the OpenSG Scene Graph API. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 95. 2

[Lju87] Ljung L.: *System Identification, Theory for the User*. Prentice Hall, 1987. 1, 2

[Mar63] Marquardt D. W.: An Algorithm for Least-squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics 11*, 1 (1963), 431–444. 3

[MP43] McCulloch W., Pitts W.: A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics 5* (1943), 115–133. 2

[RH94] Rohlf J., Helman J.: IRIS Performer: A High Performance Multiprocessing Toolkit for Real-time 3D Graphics. In *SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1994), ACM Press, pp. 381–394. 2

[RHW86] Rumelhart D. E., Hinton G. E., Williams R. J.: *Learning Internal Representations by Error Propagation*, vol. 1. MIT Press, Cambridge, MA, 1986. 3

[WW03] Wimmer M., Wonka P.: Rendering Time Estimation for Real-Time Rendering. In *Rendering Techniques 2003 (Proceedings of the Eurographics Symposium on Rendering 2003)* (Jun 2003), Christensen P., Cohen-Or D., (Eds.), Eurographics, Eurographics Association, pp. 118–129. 1, 2