International Journal of Computer Architecture and Mobility (ISSN 2319-9229)
Volume 7 -Issue 9, September 2019

Predictive Distributed Rendering for Three Dimensional Games

Bhanu Dutta Parashar
Shri Vaishnav Institute of Information Technology,
Shri Vaishnav Vidyapeeth Vishwavidyalaya ,Indore

Abstract

Rendering three Dimensional Games is a very computational intensive task, and with the rise in complexity of meshes and trend of photorealism in games requires a very powerful Graphics Processing Unit along with a Multicore CPU, I hereby propose a proof of work document to validate an approach to utilize processing power of different computers connected via a network to be utilized to render these graphics for a single node, allowing usage of idle CPU cycles of computers available over a LAN or over the Internet. This process predicts the upcoming frames and renders them on different computers and then send the rendered frames to the computer needing them, this will work in a distributed environment similar to Torrent, and a caching mechanism can ensure to minimise the delay even in case of incorrect predictions. A mathematical approach is taken as a proof with some assumptions as per needed to prove the practical viability of this approach along with calculating the minimum and the recommended conditions which allows to render at a constant rate.

1. Introduction

Rendering is a task which require massive parallel computational power, this is nowadays achieved by special dedicated circuit called GPU , i.e. Graphics Processing Unit which is a setup of large number of parallel ALU cores, along with a dedicated memory, while the individual cores are pretty weak in terms of processing speed when compared to a CPU core, GPUs operate at comparatively higher clock speeds and the number of cores is significantly higher than that of CPUs. This allows GPU to quickly perform large number of operations in parallel.

Rendering include tasks calculating vertex coordinates, fragment and color data along etc. These tasks are independent for each part of frame and pixel and can be processed individually in parallel.

GPUs use this powerful parallel processing capability along with the usage of pipelining to heavily powerup the process of rendering, the GPU rendering pipeline is shown in figure 1.

“A computer graphics pipeline, rendering pipeline or simply graphics pipeline, is a conceptual model that describes what steps a graphics system needs to perform to render a 3D scene to a 2D screen.” [1]

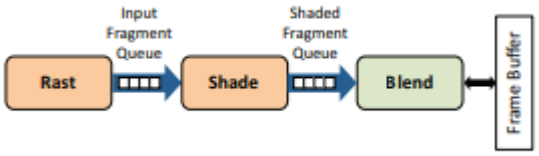

Figure 1: Graphics Pipeline [2]

But GPUs have drawbacks in economical and physical aspects , GPUs are costly for the average user and generate a lot of heat and draw a lot of power, hence it’s not usually viable for common users to have powerful GPUs in their computers , but this constraint restrains them to run high end games over weak computers, so an approach that allows to run such games in a sandboxed environment while rendering over a network is a highly plausible solution.
But processing over networks has 2 drawbacks[3]:

1. The speed of transfer of frames over network is comparatively slow for real time rendering.
2. The knowledge of which frame to render is known just before it is needed.

Due to these drawbacks if the frames are tried to be rendered as and when they are needed over the network, they won't be available on time to the needed computer giving rise to huge lag and a drop in frames per second count.
For this purpose, it is needed that we priorly get a set of frames which are most probable to be needed to be displayed, so as to display them as and when needed.
To get this prior knowledge, it is needed to predict which frame(s) is/are to be rendered, and then render them over different computers. To do so a prediction system is needed.

To select the frame out of the set of frames, it is also required to maintain a mapping of game parameters, user control parameters and frames.

2. Basic Architecture

In simple terms the basic architectures involves predicting the users next actions based on his/her previous history of actions and behavior , and use this to predict what game control action, is the user going to take next with highest probability.
Then calculate the next game state according to these user parameters, and using this information, render the next frame prior to its actual need.
This process can be run in probability gradient over user control parameters, i.e. take discrete changes from the highest probability user parameters to it's opposite parameters according to the set of user parameters and with each larger distance to the original user parameter, render the frames at different machine(s) with increasing routing distance between the main node and rendering nodes.

The rendering pipeline principle can still be applied over a set of machines divided equally among all.
Each frame might take time to render but since they are rendered prior to use, these frames can be sent and stacked in cache. Since Future frames are predicted, this can be done in a loop of continuously upgrading calculated game and predicted user control parameters, but each time an iteration is made exponential number of frames are needed to be rendered, hence a dynamic threshold based on number of nodes and network capacity is to be placed to limit the number of frames rendered.

i. Game Parameter Translator: Game parameters are defined as "set of properties that define a given state of game", these parameters include game dependent data such as - "player location", "player level", "enemy location" etc.

The number of such parameters and their domains vary too much hence a module that normalizes and provides dimension reduction functionality over this, and converting it into a format that can be used for training a neural network.

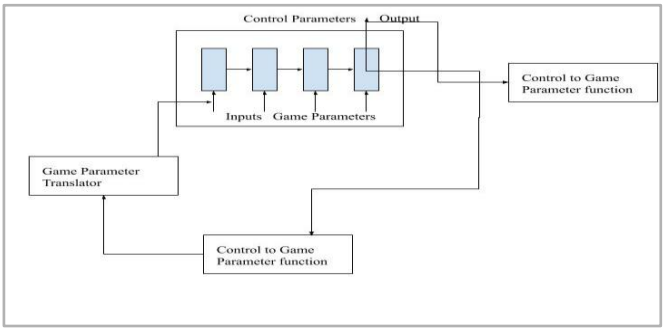

Figure 2: Architecture

ii. Control to Game Parameter Function: The approach need to implemented as a framework, just like how games need to be recompiled with different configuration for different platforms, in the same way, a Control

to Game Parameter function is intended to be designed as part of game development which will mathematically simulate the game and map user controls to next frame game parameters.

iii. Cache Map: Each frame when rendered on some other system is sent to the main system, the cache is saved in a table for
each frame number of the game, and each table contains a list of frames one out of which is the next probable frame.

For a frame 'k':

Let the set of Game Parameters be 'G^k'.
Let the set of User Controls be 'U^k'.

The Control to Game function 'f' is defined as:

$$f(G^k,U^k) = G^{k+1} \quad \text{-eq(1)}$$

And Control Function 'g' is defined as:

$$U^k = g(G^k) \quad ; if\ k=0$$
$$U^k = g(G^k,U^{k-1}) \quad ; if\ k>0 \quad \text{-eq(2)}$$

Let for a RNN based prediction system 'P', the number of RNN states be 'n'.

Let time required to calculate prediction be 'T_l'.

Also the number of nodes need to be the same as the number of RNN States, i.e 'n'.

Let the predicted User Controls by RNN be 'U_p^k' .

So let probability of correctly predicting User Control be 'p'.

i.e. $P(U_p^k = U^k) = p$ -eq(3)

Also Let,

iv. RNN based Prediction System: For predicting the next frame of the game, an RNN is used which is trained on previous game history and by developers and then further tweaked by user's personal play history.
This prediction system then generates the next user control parameters and these parameters are then used to calculate game parameters and this goes on in iterations of
RNN.

3. Proof Of Work

T_f = Frame Transfer Time over network
T_r= Frame Render Time
T_t = Parameters Transfer Time

Let $T_d = T_f + T_r + T_t + T_l$

So now we need to find out the relationship between the number of nodes ,the probability of correctly predicting a frame and time delays to find out the practicality of the approach.

Initialization:

1. Render G^0.
2. Give Input $G^0$to RNN.
3. Get U^0.
4. Calculate G^1

Now after this for further processing the calculations can be made as 2 cases.

1. **Case 1:p** = 1, i.e 100% accurate RNN.(Ideal Case).

In this case each time the rendered frame can be fetched from the cache on top of it.

So the frames rendered per second are:

$F = n/(T_d)$ fps -eq(4)

2. **Case 2:** Otherwise, i.e. practical case.

Let each time cache fails there is a penalty of 'T_p'.

Probability of Cache fail be '**q**'
$q = (1-p)$. -eq(5)

Now since each time cache fails , time required to fetch frame increases exponentially.

i.e. $T_p \propto 2^n$. -eq(6)
So,
$T_p = k*2^n$ -eq(7)

Let 'm' be the average number of directly connected nodes to each node.
So Time delay due to cache miss is:

$= (k*2^{n/m})*q$. -eq(8)

So Frames rendered per second are:

$$F = n/(T + (k*2^{n/m})*q)$$
$$= n/(T + (k*2^{n/m})*(1-p)) \quad \text{-eq(9)}$$

So the equation 8 defines a relation between number of nodes and probability of predicting correct output for a given Time delay and other constant parameters, for maximizing the frames per second.

So if we take a constant value for 'p' , we can plot a graph for number of nodes vs frames per second.

So Taking the following assumptions for the sake of plotting the graph and understanding the nature of plot.

Let

$p = 0.6$[4]
$K = 1$
$T = 0.5s$
$m = 50$

The plotted graph in figure 3.
It shows the nature of graph as that while initially frames per second increases with number of nodes but there comes a saturation after a point because the number of directly connected nodes is fixed.

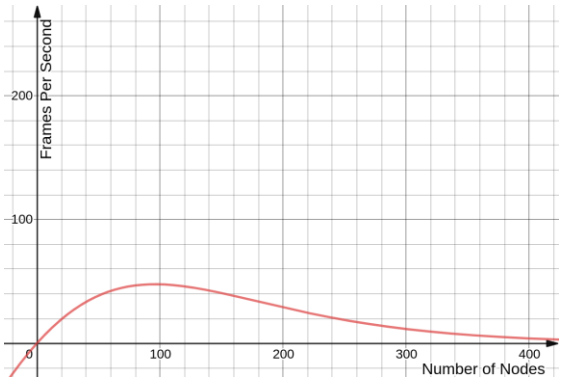

Figure 3: Number of nodes vs Frames per second plot

4. Conclusion

So it can be concluded that it is actually practical to implement a "Predictive Distributed Rendering for Three Dimensional Games" , but there is a need to have a large number of directly connected nodes.

Hence in a LAN network with mesh topology, very high frames per second can be achieved very easily. As the figure 3 suggests, with average configurations frames per second of around 40 and above is achievable, and with optical fibres and recent advancements this can be further increased.

The Best configuration is to have
$m = n/2$
i.e. half nodes should be directly connected. Hence a constant trade off between number of

nodes and number of directly connected nodes is to be maintained to generate best results.

Also it is concluded that due to caching effects of network speed over rendering speed can be minimized.

5. References

[1]https://en.wikipedia.org/wiki/Graphics_pipeline.
[2]GRAMPS: A Programming Model for Graphics Pipelines JEREMY SUGERMAN and KAYVON FATAHALIAN and SOLOMON BOULOS.
[3] Distributed Parallel Computing in Networks of Workstations — A Survey Study.
[4]Predicting Player Moves in an Educational Game: A Hybrid Approach.