GPU-based Motion Matching for Crowds in the Unreal Engine

NAGARAJ RAPARTHI, Department of Visualization, Texas A&M University
ERIC ACOSTA, Val G. Hemming Simulation Center, Uniformed Services University
ALAN LIU, Val G. Hemming Simulation Center, Uniformed Services University
TIM MCLAUGHLIN, Department of Visualization, Texas A&M University

Fig. 1. Crowd Simulation using Motion Matching

Motion Matching is a computationally expensive animation selection process where a motion capture database is regularly searched to identify the best frame of animation to be played. This study presents a process used to compute these calculations in parallel by using multiple GPU threads. The described work is shown to greatly reduce the computational time of CPU - based Motion Matching within the Unreal Engine.

CCS Concepts: • **Computation methodologies** → **Animation**; **Procedural Animation**.

Additional Key Words and Phrases: animation, motion matching, unreal engine, hardware acceleration, compute shaders

ACM Reference Format:
Nagaraj Raparthi, Eric Acosta, Alan Liu, and Tim McLaughlin. 2020. GPU-based Motion Matching for Crowds in the Unreal Engine. In *SIGGRAPH Asia 2020 (SA '20 Posters), December 04-13, 2020.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3415264.3425474

1 INTRODUCTION

Implementing the Motion Matching technique in real-time systems leads to realistic blending between different character animations, while eliminating the tedious process of setting up multi-dimensional blend-spaces [1]. A typical Motion Matching system includes a cost function that takes into account trajectory, velocity and pose data to identify the ideal pose with the minimal error between the desired and current inputs. Motion Matching has been shown to work well with one character; however, the computational requirements of this technique do not scale well as the number of characters and the size of the animation database increase. Recent works have been described to speed up Motion Matching using techniques like k-d trees [5], voxel-based tables [4] and trajectory clustering [6].

A learned alternative to Motion Matching using neural-network-based generative models [2] was also recently proposed which can replace each step in the Motion Matching process. Implementing this requires knowledge of neural networks and other machine learning algorithms. In the next section, we describe our approach which can be easily integrated into a Motion Matching system available on the Unreal Engine marketplace [3].

2 APPROACH

2.1 Game Thread and The Render Thread

The Unreal Engine divides work into game and render threads. The Motion Matching plugin [3] performs all of it's calculations on the game thread. For our Motion Matching implementation, the game thread is responsible for sending the inputs to the render thread and playing the selected animation. The render thread performs the animation cost calculations. Since the render thread trails the game thread, results from the render thread are not available to the game thread in the same frame. This is a major limitation as Motion Matching requires current animation inputs every frame and cannot accommodate for this one frame delay. In the next section we describe modifications to the Motion Matching algorithm in order to overcome this limitation.

2.2 GPU Motion Matching Implementation

GPU-based Motion Matching is performed by a compute shader. The first step is to store the data derived from the motion captured animations into a structured data buffer during initialization. This

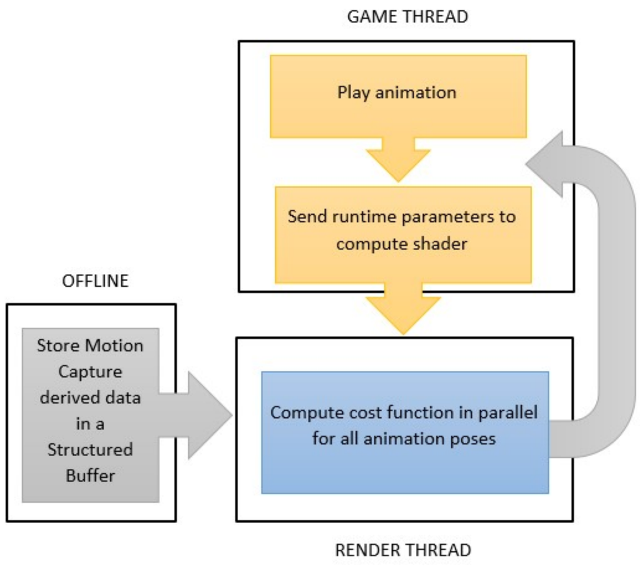
GAME THREAD
Play animation
Send runtime parameters to compute shader
OFFLINE
Store Motion Capture derived data in a Structured Buffer
Compute cost function in parallel for all animation poses
RENDER THREAD

Fig. 2. Inter-thread communication pipeline.

data includes root bone trajectory, pose positions and velocities. Corresponding run-time parameters are sent to the GPU as shader parameters. The compute shader is dispatched with the updated parameters. The computer shader performs calculations using thread groups, each of which has multiple threads. The number of thread groups and threads is specified based on the number of animation poses within the motion captured data. This enables the cost of every pose to be computed in parallel.

The shader code is similar to the original Motion Matching system for calculating the cost function based on velocity, trajectory and poses. Instead of performing these calculations linearly on the CPU, each thread uses its thread-ID to process a specific animation pose. The compute shader outputs the computed cost values and the animation corresponding to the minimal cost value is played. The cost function requires the current animation and trajectory information. This information is available to a CPU-based Motion Matching system as the results are processed within the same frame. In our approach, the processed results are not available in the same frame as the render thread executes behind the game thread. Thus, the input trajectory information required by the GPU-based animation database search is not available within the same frame. To overcome this limitation, the current trajectory was substituted by the current animation's predicted trajectory as an input to the cost function. The future trajectory of a given pose is comprised of 10 samples in the future of the current pose. By utilizing an earlier sample (e.g. 9th sample) we were able to obtain the required trajectory information. The results were consistent when compared against the original Motion Matching algorithm with a one frame delay.

3 RESULTS

We tested our approach on a machine with NVIDIA GeForce GTX-980 CPU and an Intel Core i7-5690X CPU, on multiple characters and compared our results using Unreal Engine's profiling system. Typically in Motion Matching, velocity and desired trajectory inputs are taken from the user inputs at run-time. To implement the motion matching algorithm on large sets of crowd characters, We developed a simple AI network that moves characters to different way-points within our game level. The desired trajectory is derived from the AI path and avoidance forces computed by the detour crowd system.

Time taken to perform calculations is shown for upto 200 characters which use the Filmstorm plugin [3], trajectory clustering [6] and our approach. The results given for [6] are based on our implementation of the method within the Unreal Engine. An error threshold value of 900 was used to perform trajectory clustering, which led to character animations consistent with the Filmstorm Motion Matching algorithm with the same inputs. Compared to the Filmstorm implementation, our approach decreased computation times up-to 95%.

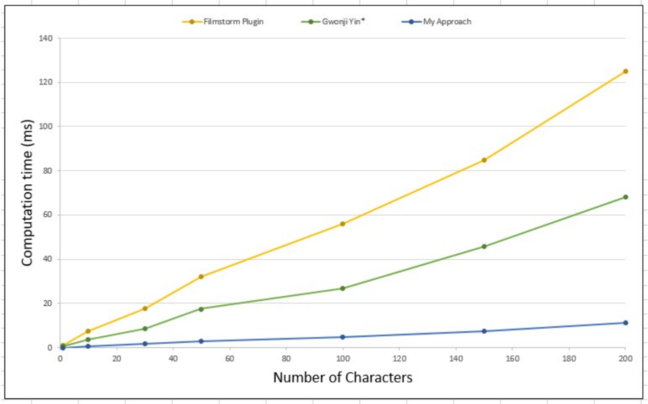

Fig. 3. Time taken (in ms) to perform calculations.

4 FUTURE WORK

Our current implementation dispatches a unique shader for each character and could benefit from dispatching one shader for all the characters to minimize inter thread communication. We are also investigating techniques to move the entire Motion Matching system to the GPU.

REFERENCES

[1] Michael Buttner. 2015. Motion Matching - The Road to Next - Gen Animation. *Nucl.ai Conference 2015* (2015). https://youtu.be/z_wpgHFSWss
[2] Maksym Perepichka Daniel Holden, Oussama Kanoun and Tiberiu Popa. 2020. Learned Motion Matching. *ACM Trans. Graph.*, Article 1 (2020), 13 pages. https://doi.org/10.1145/3386569.3392440
[3] Filmstorm. 2019. Motion Matching System. (2019). https://www.unrealengine.com/marketplace/en-US/product/motion-matching-system
[4] David Hunt Michael Buttner and Richard Lico. 2018. Topics in Real-time Animation. *ACM SIGGRAPH 2018 Courses (SIGGRAPH '18). ACM, New York, NY, USA*, Article 17 (2018), 1 pages. https://doi.org/10.1145/3214834.3214882
[5] Katsuhiro Nomura. 2017. Motion Matching no Tsukurikata. *Computer Entertainment Developers Conference 2017* (2017). https://cedil.cesa.or.jp/cedil_sessions/download?file_name=C17_85.pdf&path=2017%2Fcedec2017%2FC17_85.pdf
[6] Gwonjin Yi and Junghoon Jee. 2019. Search Space Reduction In Motion Matching by Trajectory Clustering. *Proceedings of SA '19 Posters. ACM, New York, NY, USA* (2019), 2. https://doi.org/10.1145/3355056.3364558