DEEP LEARNING BASED LARGE SCALE INVERSE KINEMATICS ACCELERATED BY INTEL® OPENVINO™ TOOLKIT

Yoon Dongwon (Game AI Team, Game AI Lab, AI Center, NCSoft)
Jang Hanyoung (Motion AI Team, Game AI Lab, AI Center, NCSoft)
Agenda

• Introduction
• Deep learning based Inverse Kinematics
• Optimization using Intel® OpenVINO™ toolkit
• Result
• Conclusion
The Team

Game AI Lab

- Motion AI Team
- Game AI Team
- Reinforcement Learning Team
Inverse Kinematics (IK)

- Compute joint angle that makes end-effector move to the target.
Inverse Kinematics in Games

- Generate animations that interact with the surrounding environment

  ex) Solving the foot skating problem

  ex) Generate animation to catch some object
Goal

• Large number of characters climb rugged cliff with IK
Challenges

Traditional methods are not suitable for the objective

<table>
<thead>
<tr>
<th></th>
<th>Numerical approach</th>
<th>Analytical approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>😊 Good</td>
<td>Bad</td>
</tr>
<tr>
<td>Performance</td>
<td>Slow</td>
<td>😊 Fast</td>
</tr>
</tbody>
</table>

Solution

<table>
<thead>
<tr>
<th></th>
<th>Numerical approach</th>
<th>Analytical approach</th>
<th>Deep Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>😊 Good</td>
<td>Bad</td>
<td>😊 Good</td>
</tr>
<tr>
<td>Performance</td>
<td>Slow</td>
<td>😊 Fast</td>
<td>😊 Fast</td>
</tr>
</tbody>
</table>
Approach

“Input”
- Current Hands and Feet Positions
- Target Hands and Feet Positions

“Output”
- Climbing motion
  - Number of Frames: 60
  - Length: 2 seconds

Deep Learning based IK Solver
Architecture [1/2]

Deep Learning based IK Solver

Trajectory Network ➔ 5 Trajectories ➔ 60 Pose Networks

Deep Learning based IK Solver
Architecture [2/2]

Deep Learning based IK Solver

Trajectory Network

5 Trajectories

Curve fitting

Smoothened 5 Trajectories

60 Pose Networks

Deep Learning based IK Solver
Gathering Training Data

1. Produce manually created “reference motion”.
2. Then randomize “start and end targets” based on “reference motion”.
3. Finally for each “start and end targets”, generate motion data using “numerical approach”.

Random targets
Reference motion
Generated motion using numerical approach
OPTIMIZATION USING INTEL® OPENVINO™
Overview

- The process of Deep-learning based IK

Training server

Deploy
Trained Network

Infers IK Animation

Client

“How to optimize client-side inference?”
CPU vs GPU [1/3] : Requirements

① Small-sized tasks
- Small-sized neural networks
- Tasks are performed for each game loop
- Small batch size

② Inference on Game-Client
- Require quick response (low-latency)
- GPU is busy rendering.
CPU vs GPU [2/3]: ① Small-sized tasks

Small sized neural networks and small batch size

Large sized neural networks and large batch size
### CPU vs GPU [3/3]: Inference overhead on Game-Client

**CPU:** Intel i7-6700K  
**GPU:** Nvidia GTX 1080  

<table>
<thead>
<tr>
<th>Without Inference</th>
<th>CPU usage</th>
<th>GPU usage</th>
<th>Game frame rate</th>
<th>Inference latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Game</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Overwatch</td>
<td>22 %</td>
<td>97 %</td>
<td>146 fps</td>
<td>-</td>
</tr>
<tr>
<td>PUBG</td>
<td>38 %</td>
<td>97 %</td>
<td>141 fps</td>
<td>-</td>
</tr>
<tr>
<td>Assassin Creed : Odyssey</td>
<td>53 %</td>
<td>96 %</td>
<td>69 fps</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inference on CPU using Openvino</th>
<th>CPU usage</th>
<th>GPU usage</th>
<th>Game frame rate</th>
<th>Inference latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Game</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>50.63 ms</td>
</tr>
<tr>
<td>Overwatch</td>
<td>68 % (309.1 %)</td>
<td>98 % (101.0 %)</td>
<td>145 fps (99.3 %)</td>
<td>57.97 ms (114 %)</td>
</tr>
<tr>
<td>PUBG</td>
<td>75 % (197.4 %)</td>
<td>96 % (99.0 %)</td>
<td>137 fps (97.2 %)</td>
<td>59.92 ms (118 %)</td>
</tr>
<tr>
<td>Assassin Creed : Odyssey</td>
<td>84 % (158.5 %)</td>
<td>96 % (100.0 %)</td>
<td>69 fps (100.0 %)</td>
<td>82.87 ms (164 %)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inference on GPU using Openvino</th>
<th>CPU usage</th>
<th>GPU usage</th>
<th>Game frame rate</th>
<th>Inference latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Game</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>43.87 ms</td>
</tr>
<tr>
<td>Overwatch</td>
<td>38 % (172.7 %)</td>
<td>97 % (100.0 %)</td>
<td>119 fps (81.5 %)</td>
<td>77.51 ms (177 %)</td>
</tr>
<tr>
<td>PUBG</td>
<td>66 % (173.7 %)</td>
<td>97 % (100.0 %)</td>
<td>113 fps (80.1 %)</td>
<td>76.01 ms (173 %)</td>
</tr>
<tr>
<td>Assassin Creed : Odyssey</td>
<td>66 % (124.5 %)</td>
<td>96 % (100.0 %)</td>
<td>58 fps (147 %)</td>
<td>88.93 ms (203 %)</td>
</tr>
</tbody>
</table>
Choosing Solution for Deep Learning Inference

Average Latency on CPU
(Lower numbers indicate better performance)

<table>
<thead>
<tr>
<th>Method</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive cpp</td>
<td>423.48</td>
</tr>
<tr>
<td>Numpy</td>
<td>34.75</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>20.34</td>
</tr>
<tr>
<td>DL Inference Engine</td>
<td>2.21</td>
</tr>
</tbody>
</table>

12X faster
20X faster
191X faster
Introduction to Intel OpenVINO

Train a model

Model Optimizer

Deep Learning Deployment Toolkit

Deploy Pre-Trained Model

IR

Game Client

IK Solver

Inference engine

MKL-DNN Plug-in

CPU

Tensorflow, Caffe, MXNet, ONNX
Optimization: Cubic Hermite Spline [1/2]

\[ p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1 \]
Optimization : Cubic Hermite Spline [2/2]

\[ p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1 \]

in matrix form

\[
\begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_0 \\
p_1 \\
m_0 \\
m_1
\end{bmatrix}
\]

Matrix calculations can be accelerated by Intel Openvino .
Optimization: Batch inference

Serial computation

Parallel computation
Optimization for multiple characters [1/2]

Character
Character
Character
Character
Character

Next motion frame
Motion timer updates frame number.

Request next frame motion data

Batch Manager

IK Solver

Trajectory Network
Pose Network

Next frame motion data
Frequency of used batch size

- (10, 14.5)
- (20, 15.3)
- (30, 9.0)
- (50, 7.7)
- (70, 5.5)
Finding optimal batch size that achieves maximum throughput while maintaining limited latency

Set $bs$ to initial batch size

Repeat until we obtain a reliable max latency

Start measuring latency

If number of inference $> bs$
  infer multiple times

Else If number of inference $\leq bs$
  infer once and ignore unused space

Finish measuring latency

Update max latency

End
Quality Comparison

Numerical Approach

Our Approach
CONCLUSION
Deep Learning based Inverse Kinematics

- Character animation becomes more realistic by solving IK.
- Conventional approaches for full-body IK are complex or computationally expensive.
- Solving full-body IK using Intel OpenVino is cost efficient and also flexible.
- Process of “Climbing IK Solver”
  1. Trajectory network
  2. Curve fitting
  3. Pose network
- Gathered training data from manually crafted reference motion.
Optimization using Intel® OpenVINO™ toolkit

- CPU is suitable for computing inference on Game-Client.
- Deep Learning Deployment Toolkit is high performance framework for computing inference on a CPU.
- Model Optimizer converts and optimizes the trained model to IR. On Game-Client, Inference Engine loads this IR and it can now run inference on CPU using optimized hardware plugin (MKL-DNN plugin).
- Improved curve fitting performance by computing matrix form of cubic Hermit spline with Inference Engine.
- Batching inference task improves performance by parallelizing inference task by the Inference Engine.
- Finding optimal batch size to meet limited latency
QUESTIONS?

Yoon Dongwon
E-mail: dongwonyoon@ncsoft.com
Reference

https://intel.github.io/mkl-dnn/understanding_memory_formats.html