Optimizing the Performance of Computer Vision Application

Caleb Brohman, UNC Charlotte

Erik Saule, College of Computing and Informatics

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Introduction

Introduction:

- Computer vision is a field that enables machines to interpret and analyze visual data, allowing us to demonstrate various applications and optimalizations through live demos.

Challenges:

- Performance is often limited by computational resources, processing speed, and accuracy, which can hinder real-time processing and effectiveness.

Research Focus:

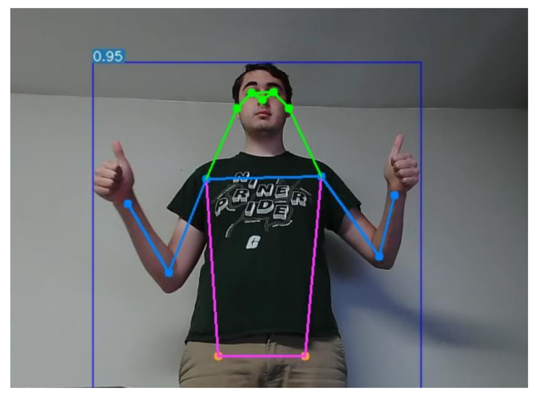
- Our research aims to optimize performance by improving algorithm efficiency and resource management, enhancing speed and accuracy.

How they work:

- Both computer vision applications follow the same structure of capturing a frame, processing it with a machine learning model, rendering it, and looping until completion.

Importance:

- Computer vision uses complex algorithms to help machines interpret and respond to visual data, making them increasingly important in everyday life.



Webcam output from application 1

Application 1: Position Estimation

Objective:

- To increase performance of a live webcam feed and make the application estimate position in real-time.

Challenges:

- Ensuring real-time image processing
- Providing accurate position estimations

Methods:

- Utilize modern hardware to do complex calculations faster.
- Use efficient and effective pretrained models

Application 2: Action Recognition

Objective:

- To identify and classify human actions from a live webcam feed in real-time and overlay the predictions.

Challenges:

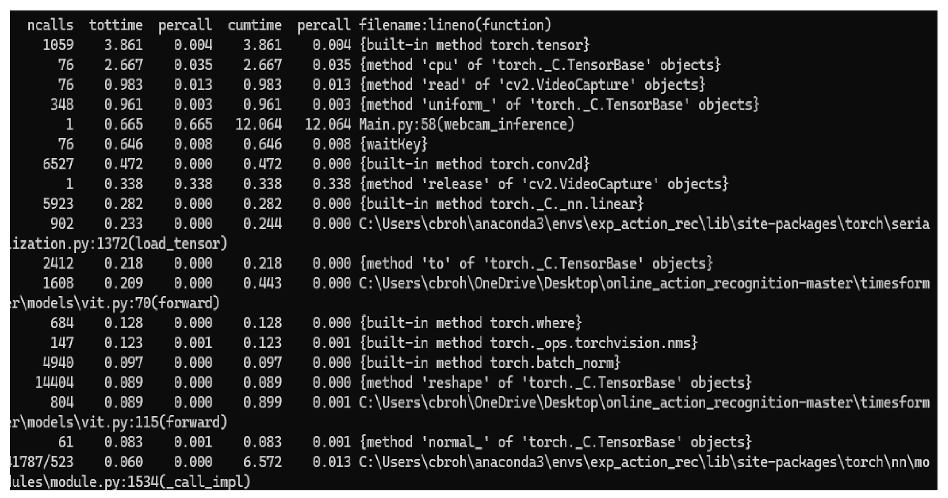
- Ensuring real-time action recognition in a live demonstration
- Recognizing and processing multiple actions for display

Methods:

- Using a profiler to find bottlenecks in the code
- Using basic strategies to decouple the rendering and analysis sections.

Profiling

- A code profiler analyzes performance by measuring the execution time for each function.
- Profilers are used to identify inefficient parts of code.
- By utilizing a profiler, we were able to identify bottlenecks in our second application to optimize it for performance.



```
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1059    3.861    0.004    3.861    0.004 {built-in method torch.tensor}
      76    2.667    0.035    2.667    0.035 {method 'cpu' of 'torch._C.TensorBase' objects}
      76    0.983    0.013    0.983    0.013 {method 'read' of 'cv2.VideoCapture' objects}
     348    0.961    0.003    0.961    0.003 {method 'uniform_' of 'torch._C.TensorBase' objects}
       1    0.665    0.665   12.064   12.064 Main.py:58(webcam_inference)
      76    0.646    0.008    0.646    0.008 {waitKey}
    6527    0.472    0.000    0.472    0.000 {built-in method torch.conv2d}
       1    0.338    0.338    0.338    0.338 {method 'release' of 'cv2.VideoCapture' objects}
    5923    0.282    0.000    0.282    0.000 {built-in method torch._C._nn.linear}
     902    0.233    0.000    0.244    0.000 C:\Users\cbroh\anaconda3\envs\exp_action_rec\lib\site-packages\torch\seria
ization.py:1372(load_tensor)
    2412    0.218    0.000    0.218    0.000 {method 'to' of 'torch._C.TensorBase' objects}
    1608    0.209    0.000    0.443    0.000 C:\Users\cbroh\OneDrive\Desktop\online_action_recognition-master\timesform
r\models\vit.py:70(forward)
     684    0.128    0.000    0.128    0.000 {built-in method torch.where}
     147    0.123    0.001    0.123    0.001 {built-in method torch._ops.torchvision.nms}
    4940    0.097    0.000    0.097    0.000 {built-in method torch.batch_norm}
   14404    0.089    0.000    0.089    0.000 {method 'reshape' of 'torch._C.TensorBase' objects}
     804    0.089    0.000    0.899    0.001 C:\Users\cbroh\OneDrive\Desktop\online_action_recognition-master\timesform
r\models\vit.py:115(forward)
      61    0.083    0.001    0.083    0.001 {method 'normal_' of 'torch._C.TensorBase' objects}
1787/523    0.060    0.000    6.572    0.013 C:\Users\cbroh\anaconda3\envs\exp_action_rec\lib\site-packages\torch\nn\mo
ules\module.py:1534(_call_impl)
```

Output of profiler from application 2

Results

Performance Improvements:

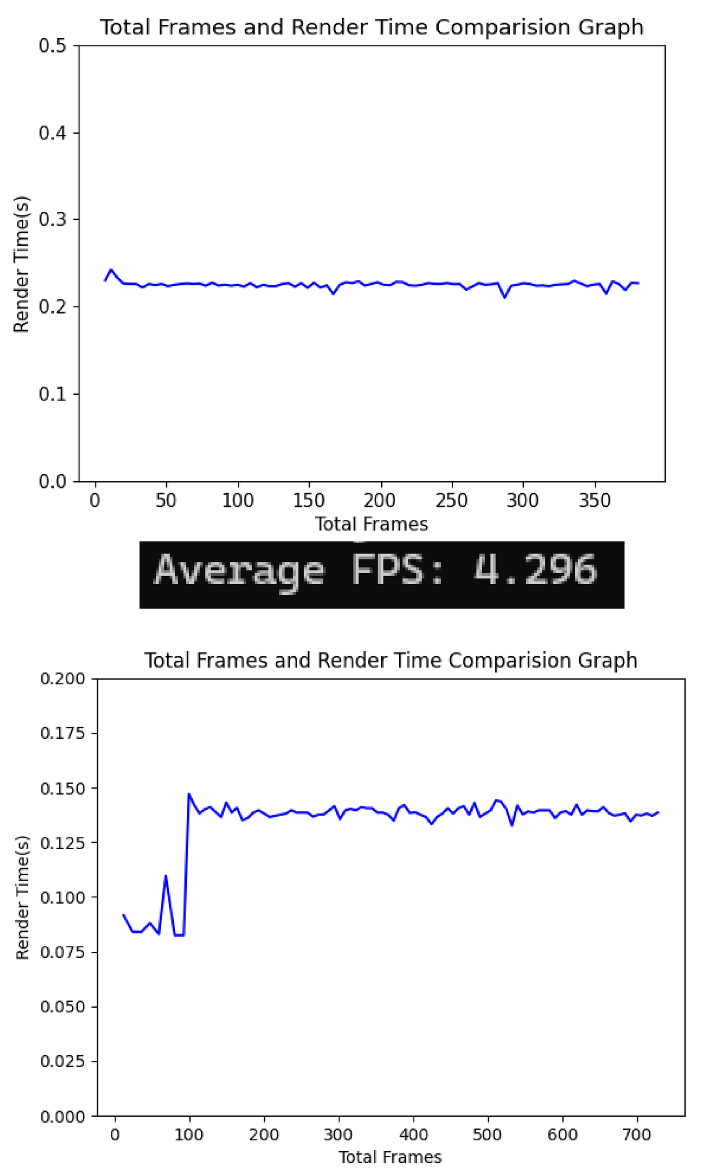
- Overall gain with significant increase in frames per second (FPS) for both applications after code modifications.

Application 1:

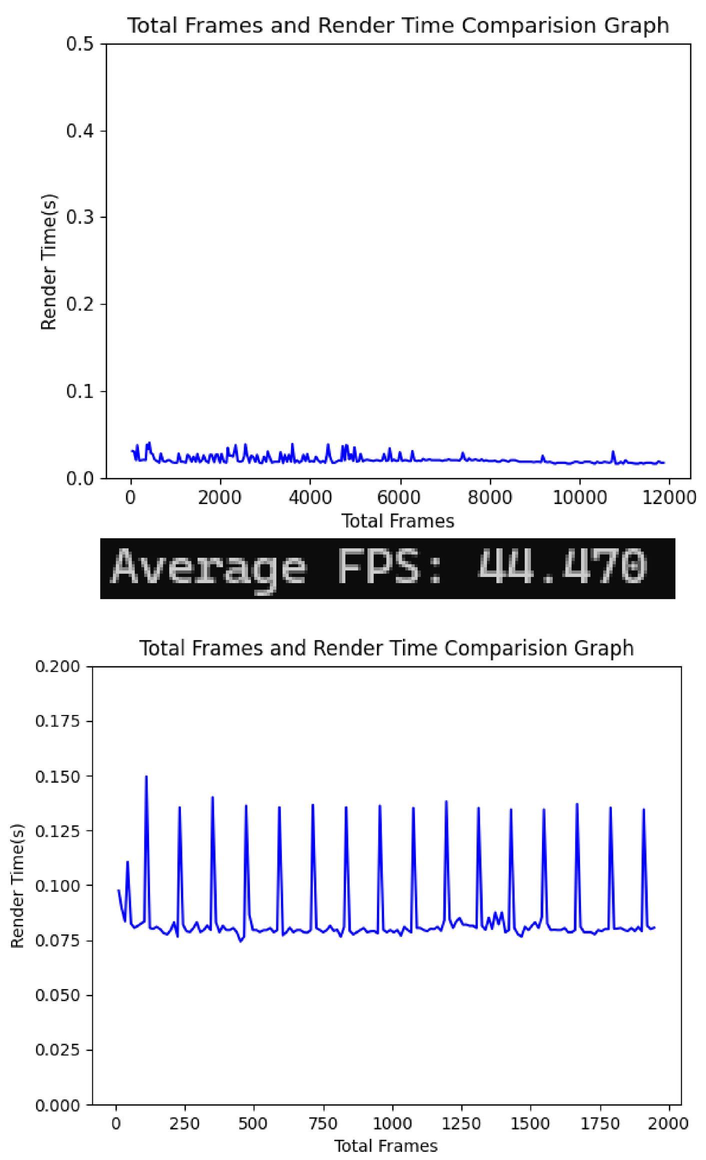
- Utilizing a GPU for video rendering, frame rate improved by 10 times its original value.

Application 2:

- Large predictions are processed every few frames instead of every frame to maintain real-time look and accuracy.



Applications before optimizations



Applications after optimizations

Conclusions

- Utilizing modern hardware for complex computations is key for real-time image processing.
- By leveraging modern hardware and efficient software, noticeable improvements in real-time performance have been observed on both applications.

Future Works

- Implementing threading in application 2 to further separate rendering from analysis.
- Making the applications compatible on different machines regardless of hardware limitations.