Ray Tracing Learned Deformed SDFs

Update 2 // Ryan Zesch // Nov 21

1 Summary of Work Done

Since my last update, I first focused on the performance of the ray tracer I had written. I first profiled my existing code and made flamegraphs to see where to focus my efforts. Unfortunately, the slowest portions of my code all related to the time it took to evaluate my neural net, and to compute normals of the net. I was able to make improvements, however, by taking more aggressive steps in my sphere marching algorithm. I made additional improvements by batching finite difference normals when analytical derivatives were degenerate. Additionally, I made small adjustments to multithreading in order to decrease runtime.

After making these improvements, I took inspiration from Nvidia's Level of Detail network [1] and implemented data structures in order to speed up the ray tracing process. In particular, I implemented a preprocessing step to convert my neural net to a grid of distance values which can be trilinearly interpolated between. Normals, however, are still computed on the fly via network back propagation, as they yielded poor results. This resulted in a massive speedup of nearly 10 times. Preprocessing, however, can be relatively time consuming and takes $O(n^3)$ space.

Closer to this paper, I have implemented a sparse octree based structure in order to gain further speedup. This will allow for fewer steps during sphere tracing once completed, and will allow for finer grids for trilinear interpolation. It is currently slow, however, due to potentially needing

multiple on the fly normal computations per ray. I intend to implement normal interpolation for this method.

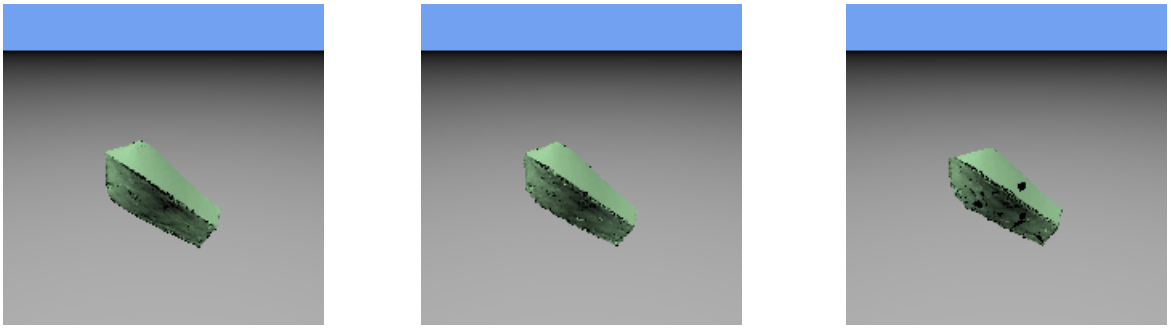

Figure 1: 200x200 renders of a deformed mesh using the net directly (1.5m), a full trilinear interpolation grid (12s rt + 5s preprocessing), and octree trilinear interpolation (4.5min, almost all time spent computing gradients)

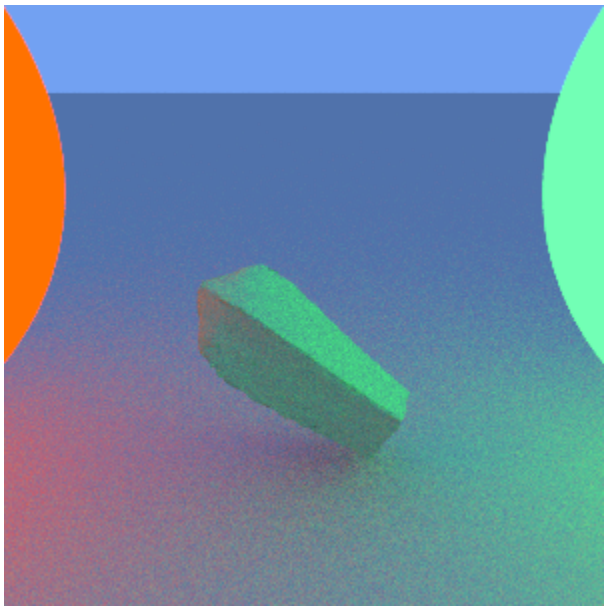

Figure 2: A 300x300 render of the deformed mesh, using a 64^3 grid for trilinear interpolation. This used standard raytraced lighting, with 50 rays per pixel and max recursion of 10. This render took 40 minutes.

2 Analysis of Work Done

I have taken a slightly different direction than expected compared to my stated goals for update 2.

- Instead of implementing a network based on [1], I instead used similar data structures and ray tracing algorithms to those presented in the paper. The network in the paper would not have been able to handle deformable SDFs without huge changes, so it was not a feasible goal.
- As expected, I profiled and optimized by base ray tracer and improved normal calculations in some cases. I achieved a much bigger speedup than expected.
- I am currently implementing algorithms for further speeding up ray tracing on my sparse octree based SDF

3 Plan for Completion

- By the Final Presentation (Dec. 15), I intend to
  - I hope to have optimized my ray tracer further and adapted other known algorithms to speed up ray tracing my network.
  - I hope to have trained my network on multiple scenes to render.
  - I will compare performance/runtimes for different approaches I have implemented.

References

[1] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. 2021.