Rendering with Neural Intersection Functions

Personal Information

Name: Jin Ke

University: Tongji University

Email: rainy.autumn.fall@gmail.com

Time zone: GMT+8

github: Rainy-fall-end

Project Information

Project Title

Rendering with Neural Intersection Functions

Brief Summary of Project

Ray tracing involves computationally expensive calculations and geometry with varying degrees of complexity. According to Fujieda et. al[1], a novel Neural Intersection Function can be used for ray intersection queries. This method has better efficiency while ensuring image quality. The objective of this project is to implement and rigorously evaluate this Neural Intersection Function against the industry-standard bounding volume hierarchy (BVH) algorithm, providing a comprehensive comparative analysis of their respective performance characteristics.

Previous work

Brl-cad Team have tried a project BRL-CAD Neural Rendering which aimed to encode BRL-CAD geometry into a machine learning model to develop three-dimensional renderings.

In the `DATA COLLECTION METHODS section`, they attempted three methods: the mixed bounding box approach, the pure bounding box approach, and the grid approach.

In the encoder section, they primarily explored the `TRADITIONAL ENCODING APPROACH`, which can be further divided into `SINGLE MODEL ARCHITECTURE` and `DUAL MODEL ARCHITECTURE`. The former employs a single network architecture for training, while the latter utilizes two networks: one for determining if the ray hits the wire and the other for determining the distance between the hit rays and hit points.

Subsequently, they also attempted the `GRID ENCODING APPROACH`, a method proposed by Fujieda et al[1].

They utilized the PyTorch C++ API to accomplish this project and rewrote the `art::raytrace` method, in `work.c`:

```
if(neural_rendering != 1) {
        (void)rt_shootray(&a); // This is the call to rt_shootray we need to edit
    }
    else {  // Doing neural rendering
   ...
    }
```

Users can train the model beforehand by using `bin/rt_trainneural *.g Body1.2.4.b.c.s` and then define `neural_rendering` to determine whether to use the trained neural network for rendering.

Howerver, there are still several shortcomings in this work:

- The current data collection methods exhibit significant randomness. In the conclusion section of their report, it is mentioned that, "Variations in the collection of training data, as well as potential tweaks to the model, may lead to significant improvements in the performance of the system."
- They did not fully reproduce the method mentioned in the NIF paper; they only implemented part of the **NIF outer network** and did not implement the **NIF inner network**.
- The neural grid structure they used is too simple：

```
self.model = nn.Sequential(
            nn.Linear(5, 120),
            nn.ReLU(),
            nn.Linear(120, 80),
            nn.ReLU(),
            nn.Linear(80, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1),
            nn.Sigmoid()
        )
```

- During the rendering process, the camera is static, and dynamic camera rendering has not been implemented.

Detailed project description

This project aims to utilize NIF to accelerate the ray tracing process and explore whether this algorithm can be extended to arbitrary camera viewpoints. I am currently planning to complete the following four three of the content:

Parts 1 : Data Collection and Dataset Construction.

Due to the fact that the NIF method involves two networks, namely the outer network and the inner network, with the demarcation point being the Axis Aligned Bounding Box (AABB).

In the process of **data selection**, I will first use #method1 to randomly obtain some points. Then, I plan to build a neural network based on active learning. This neural network can help me filter out boundary points lying between 0 and 1, which are more conducive to training.

In **selecting methods**, I will primarily focus on the boundary-based active learning approach, as proposed by Jiang et al. in Minimum-Margin Active Learning[2]. I hope to employ this method to identify the following type of datas:

$$
\begin{gathered}
model(X_1) = 1 \\
model(X_2) = 0 \\
where, ||X_1 - X_2|| -> 0
\end{gathered}
$$

Finally, I aim to perform density-differentiated sampling on both boundary and non-boundary data, using these data to train the model.

Parts 2 : Finish the Neural Network Framework.

In previous work, the `GRID ENCODING APPROACH` has been finished. The next step was to finish the outer network and inner network.

I will continue development using PyTorch. In [1],the outer network in NIF contains two hidden lay- ers with 64 nodes in each of them, whereas the inner network comprises three hidden layers with 48 nodes each. Every hidden layer is followed by a leaky ReLU activation function except for the last one. To meet user demands, we will incorporate the number of hidden layers and the number of nodes as parameters, catering to models of varying sizes. Additionally, we can consider making neural network hyperparameters adjustable, allowing users to create different networks and train them using their preferred methods.

In the conclusion of the previous work, the existing network structure is too simple. Consider the following improvements:

- **Add residual modules[3]**. Residual modules can increase the depth and expressive power of the network. By introducing skip connections, residual modules allow the network to learn residual functions, thereby propagating gradients more effectively and alleviating the vanishing gradient problem.
- **Quantify Classification Uncertainty[4]** This enables the model to make more informed decisions, especially in situations where the confidence of the prediction is low. I hope to hand over those uncertain rays to traditional methods to ensure the final quality of the generated image, striking a balance between quality and efficiency, with the specific threshold being determined by the user.

About Code Implementation:

```
int
rt_shootray(register struct application *ap)
```

A feasible approach is to rewrite this method:

```
int
rt_shootrayNeural(register struct application *ap)
```

But I hope to go further on this basis. GPUs have stronger computational capabilities compared to CPUs. It's wasteful to calculate only one ray at a time on the GPU. Therefore, I hope to be able to calculate multiple rays at once.But this involves too many functions, so I may first consider modifying `do_pixel` to achieve this:

```
void do_pixel(int cpu, int pat_num, int pixelnum)
void do_pixel(int *pat_num, int *pixelnum)
```

Parts 3 Rendering Efficiency and Quality Testing

To assess the feasibility of neural network-accelerated ray tracing, testing will involve multiple models and can be broken down into the following points:

- **Rendering Speed Comparison**: Compare the rendering times between traditional ray tracing methods and neural network-accelerated ray tracing for each model.
- **Image Quality Assessment**: Evaluate the quality of rendered images produced by neural network-accelerated ray tracing compared to traditional methods.
- **Scalability Testing**: Assess the scalability of the neural network approach by increasing the complexity of the scenes or the size of the dataset.
- **Increase testing in complex scenarios**: Testing rendering effects with multiple entities present.

*Parts 4 Extend NIF to arbitrary camera viewpoints

Currently, the model still renders on a fixed camera. This section will explore whether NIF can achieve rendering on a dynamic camera.

To achieve rendering on a dynamic camera, it is necessary to use deep neural networks. Otherwise, it would be impossible to cover such a large sample space. In NIF, the model's input is:

$$T_{in} = \{(p, d)|p, d \in R^3\}$$

To extend NIF to arbitrary camera viewpoints, we can add camera parameters:

$$T_{in} = \{(p, d, c_p, c_d) | p, d, c_p, c_d \in R^3\}$$

The key issue lies in how to encode these parameters to facilitate training by the neural network.

Improtance of the Project

The importance of the project lies in its potential to revolutionize the field of ray tracing by leveraging neural networks to significantly improve rendering efficiency while maintaining or even enhancing image quality

Deliverables

- A new interface for:

```
int
rt_shootray(register struct application *ap)
```

- test interface for this method, from speed and quality
- Comparison report between the NN interface and traditional methods, based on comprehensive testing with various models.
- Report for extend NIF to arbitrary camera viewpoints
- Wiki page update with detailed information about Appleseed integration.

Development Schedule

- **Community Bonding Period**
    - Familiarizing with previous work
    - Read related paper[1-4]
    - Ask for other cleaning works from the mentors, if required
- **Week 1(27 May)**
    - Familiarizing rt project in brl-cad
    - Familiarizing details NIF papers

- **Week 2(3 June)**
    - Build database from simple to complex
    - Build active learning network
- **Week 3-4(10 June)**
    - Build outer NN
    - Do some test for outer NN
- **Week 5(24 June)**
    - Build inner NN
    - Do some test for inner NN
- **Week 6(1 July)**
    - Merge two networks
    - Add Quantify Classification Uncertainty to both network, finish rendering with both neural network and traditional method.
- **Week 7(8 July)**
    - Test method
    - Build more test models,especially complex models.
    - Improve network structure based on test results
- **Week 8(15 July)**
    - Integrate NN framework with existing rendering pipeline in BRL-CAD.
- **Week 9(22 July)**
    - Extend NIF to arbitrary camera viewpoints(finish framework)
- **Week 10(29 July)**
    - Construct test cases for rendering from arbitrary camera viewpoints
    - Extend NIF to arbitrary camera viewpoints(test framework)
- **Week 11-12(5 August)**
    - Write report
    - Write wiki
- **Final week**: Submit Final Evaluation and Code to the Melange Home

My preparation for the Project

- I have successfully build brl-cad with appleseed from source code
- I have read the whole report about "BRL-CAD Neural Rendering" and familiarized with their code
- I have found some errors when build brl-cad with appleseed version 2.0+ and I

have submitted a pull request[4]
- I have submitted a pull request to fix bugs and finish "to-do" in rt project.[5]

Why BRL-CAD?

During a rewarding four-month internship at a geometry engine company, I honed my skills in b-rep modeling and developed a profound interest in computational mathematics. My passion lies at the intersection of computational geometry and artificial intelligence, which motivated my pursuit of this program.

Why me?

I have previously interned at a company, so I believe I have decent C++ skills. In the AI field, I have participated in several projects, although most of them were done using Python. I begin with this book [Neural Networks and Deep Learning] and implemented most of the models inside.(http://neuralnetworksanddeeplearning.com/).
Here are some projects which I involved in:

DDPM-cherry
I created a dataset of cherry blossoms and used DDPM to generate related data.

welding-prediction
Using neural networks to predict welding deviations.

Reference

[1]Neural Intersection Functions
[2]Minimum-Margin Active Learning
[3]Resnet in resnet: Generalizing residual architectures
[4]https://github.com/BRL-CAD/brlcad/pull/117
[5]https://github.com/BRL-CAD/brlcad/pull/120