Задание 3. Neural SDF

Цель задания – на практике познакомиться с использованием нейросетей для представления объектов в компьютерной графике.
Основные задачи:

- *Реализация прямого и обратного прохода нейронной сети (Multi-Layered Perceptron) без использования сторонних библиотек;*
- *Использование нейросети для приближения функции дистанции до произвольного объекта.*

Требования к оформлению

Требуется написать консольное приложение, работающее в 2 режимах - обучения и рендера.

- В режиме обучения программе подается 1 аргумент - **file.obj** путь до файла с моделью в формате wavefront .obj (https://en.wikipedia.org/wiki/Wavefront_.obj_file). Гарантируется, что объект состоит только из треугольников и координаты его точек находятся в интервале [-1, 1]. Требуется обучить нейросеть предсказывать функцию дистанции для данного объекта (гиперпараметры выбираются по вашему усмотрению), писать в консоль, как изменяется функция потерь во время обучения, после чего сохранить веса в файл weights.bin и отрендерить полученную SDF. Также необходимо написать время, потраченное на обучение. Если вы реализовали обучение на GPU, то используйте только этот вариант, это сократит время проверки.
- В режиме рендера программа вызвается с 3 аргументами **N_layers, layer_size, weights.bin.** Это количество скрытых слоев, размер скрытых слоев и файл с весами соответственно (см. дальше про архитектуру нейросети и соглашение о хранении весов). Здесь необходимо инициализировать нейросеть с указанными параметрами и весами и отрендерить с ее помощью две картинки с результатами работы CPU и GPU алгоритмов в файлы с именами **out_cpu.png**, **out_gpu.png.** Также написать время, потраченное на рендер каждой из картинок.

Решение нужно прислать в виде zip архива с названием следующего формата:

<номер группы>_<инициалы>_<фамилия>.zip

В корень проекта необходимо приложить **readme.txt** файл, содержащий:

- детали сборки проекта (используемое API, набор библиотек, порядок сборки и т.д.)
- аргументы командной строки для запуска приложения (если есть дополнительные)
- спецификацию вашего ПК и результаты замеров времени на нём (см. секцию 3. перенос на GPU)

Основное правило: сборка и запуск вашего проекта должны быть тривиальны.

Все файлы с весами и тестами, которые упоминаются в тексте, можно найти в репозитории https://github.com/SammaelA/NAIR_2024

Вступление

Треугольные меши остаются самым популярным способом представления 3D контента в компьютерной графике, по многом благодаря созданным эффективным алгоритмам рендера, которые получили повсеместную аппаратную поддержку. Аппаратно поддерживаемая растеризация на протяжении многих лет ограничивала исследователей и разработчиков, поскольку без ее использования было почти невозможно добиться приемлемой производительности. Однако в последние годы все новые видеокарты имеют аппаратную поддержку трассировки лучей, которая более не привязывает разработчиков к полигональным мешам как единственно возможному представлению модели. В первом задании вы уже познакомились с одним из альтернативных представлений - SDF, функции дистанции со знаком. Однако их возможности на первый взгляд сильно ограничены тем набором примитивов, которые удается записать в виде явных функций. При этом нетрудно заметить, что любой меш, имеющий однозначное разделение на внутренний и внешний объем (closed или watertight mesh) порождает свою уникальную функцию дистанции. Более того, мы можем ее вычислить - в каждой точке пространства функция дистанции до меша будет равна расстоянию до ближайшего его треугольника. Определить знак, то есть находимся ли мы внутри или снаружи, также нетрудно. На первый взгляд получение SDF из меша может казаться бесполезной задачей, но это не так. Те из вас, кто в прошлом году проходил наш курс FAIR, помнят задание по построению 2D SDF из монохромного изображения (https://drive.google.com/file/d/1Ti9U3fEkRbu5nkLozKRjJT_LeAu2WxqA/view?usp=sharing) . Использование функции дистанции позволяло сохранить плавность линий при значительно меньшем разрешении итоговой SDF-картинки. Заметим также, что предложенный там метод был универсальным, его можно применить к любой монохромной картинке. Функция дистанции в том примере приближалась набором своих значений на двумерной сетке. Аналогичную идею можно применить и в 3D - такая SDF на сетке будет соотноситься с воксельным массивом также, как 2D SDF - с картинкой. Такое приближение функций дистанции было реализовано и используется, например, в одной из лучших на текущий момент работ по трехмерной реконструкции. Однако в этом задании мы предлагаем вам пойти по другому пути. Известно, что достаточно большая нейросеть может хорошо приблизить даже очень сложную функцию. Это верно и для SDF, тем более что это в конечном счёте просто функция R^3 -> R, заданная на всем пространстве.
В данном задании вам предлагается реализовать простую архитектуру нейронной сети SIREN и использовать ее для приближения произвольных функций дистанции. Задание разбито на несколько этапов, мы настоятельно рекомендуем выполнять их именно в таком порядке, так вы повышаете вероятность успеха и гарантируете себе часть баллов даже если не удастся дойти до конца.

Для выполнения задания вы не можете использовать никакие сторонние библиотеки, кроме стандартной библиотеки C++, stbi_image, CUDA и библиотек kernel slicer'a, (в том случае если вы используете его для переноса на GPU). Вы можете использовать другие технологии для переноса на GPU по согласованию с организаторами курса, при этом нельзя пользоваться существующими библиотеками для тензорных вычислений или работы с нейронными сетями (Numpy или PyTorch в Python например).

Основная часть

Задание состоит из нескольких взаимозависимых шагов, которые имеет смысл выполнять в указанном тут порядке. За каждый основной шаг будут начисляться баллы. Задание объемное и выполнение всех основных шагов даст достаточное количество баллов для успешного прохождения курса. Если вам по каким-то причинам нужны дополнительные задания - пишите в группу Телеграм, мы назначим их и бонусные баллы индивидуально.

Шаг 1. SIREN (10 баллов)
Умножение матриц - базовая операция, лежащая в основе работы нейросетей. Сначала вам необходимо добавить операцию умножения произвольных матриц. Для выполнения дальнейших шагов необходимо выбрать правильное представление - лучше всего хранить матрицу как один непрерывный участок памяти, где матрица лежит по строкам. Вы можете заметить, что умножение матриц C = A*B предполагает непоследовательный доступ к памяти B, т.е. плохое использование кеша. Это можно решить, если проводить операцию с транспонированной второй матрицей C = A*(B^T). Тогда нужно добавить еще отдельно операцию транспонирования и считать C = A*((B^T)^T).

SIREN (https://arxiv.org/abs/2006.09661) - это простая нейросеть, состоящая из нескольких слоев полносвязных слоев, между которыми к данным применяется функция синуса с масштабированием. output = sin(w0*input). Формально w0 является гиперпараметром слоя, но по ходу задания можно считать его константой w0 = 30. На вход SIREN принимает 3 числа - x, y, z координаты точки, на выходе - 1 число, которое в нашем случае имеет смысл функции дистанции в этой точке. Все веса нейросети хранятся в полносвязных слоях (осуществляют операцию output = A_i*input + b_i). Каждый полносвязный слой может иметь свой размер (разумеется, размер выхода слоя N должен совпадать с размером входа слоя N+1). На этом этапе вам нужно реализовать прямой проход этой нейросети. Вам необходимо проверить это на подготовленном тестах. В файле sdf<n>_arch.txt написано, из каких слоев какого размера состоит нейросеть, а файл sdf<n>_weights.bin содержит веса обученной нами нейросети.
Наконец файл sdf<n>_test.bin имеет следующую структуру
(N, pos_1_x, pos_1_y, pos_1_z, …, pos_N_x, pos_N_y, pos_N_z, dist_1, …, dist_N), т.е. содержит сначала число точек N, потом массив этих точек, потом массив эталонных дистанций. Посчитайте выход вашей нейросети на точках из этого массива и сравните с эталонными значениями - ошибка <1e-5 говорит о том, что вы все сделали правильно.

! Формат файлов с весами нейросети !
Бинарные файлы с весами, которые вы используете для проверки, и которые вы будете сами сохранять после обучения, должны удовлетворять следующему соглашению: Веса хранятся в массиве float подряд по слоям (A_1, b_1, A_2, b_2, …., A_k, b_k), матрицы хранятся по строкам. Бинарый файл не содержит в себе размер массива, т.к. он однозначно определяется архитектурой сети.

Шаг 2. Визуализация (5 баллов)
Теперь у вас есть реализация нейросети SIREN и веса из файла, но непонятно какую модель они задают. Чтобы это узнать, необходимо применить алгоритм sphere tracing, но уже для нейронных SDF. Позиции камер и направление источника света можете выбирать на свое усмотрение.
Возможно, при первой отрисовке вы получили неадекватный результат или просто однотонно серую картинку. Это связано с тем, что нейросеть хорошо приближает исходную SDF только там, где находились точки из обучающей выборки, а они находились в единичном кубе ([-1,1] по всем осям). Модифицируйте sphere tracing так, чтобы рассматривать только тот участок луча, который лежит в этом кубе.
Нейронная сеть требует на несколько порядков больше операций для вычисления SDF, чем заданная явными функциями сцена. Чтобы рендеринг не был мучительно долгим, вы можете уменьшить разрешение (512*512 должно хватить) и удалить дополнительные эффекты, такие как ambient occlusion или тени.

Шаг 3. Обучение (10 баллов)
Умение вычислять нейронные SDF с уже рассчитанными весами - это отлично, но бесполезно, если непонятно, откуда эти веса получать. Очевидно, для этого придется обучить SIREN. Делается это стандартным образом - стохастическим градиентным спуском.Файл sdf<n>_points.bin содержит обучающую выборку и имеет следующую структуру
(N, pos_1_x, pos_1_y, pos_1_z, …, pos_N_x, pos_N_y, pos_N_z, dist_1, …, dist_N), т.е. содержит сначала число точек N, потом массив этих точек, потом массив эталонных дистанций. Для обучения SIREN вам необходимо реализовать:

- вычисление loss функции (MSE) и ее градиента
- вычисление градиентов по весам и input полносвязных и sin слоев
- метод обратного распространения ошибки по внутренним слоям нейросети
- оптимизатор Adam
- инициализацию весов (см. формулу в оригинальной статье, хотя кажется He initialization тоже прокатит)

Вы можете экспериментировать с архитектурой нейросети или взять ее из sdf<n>_arch.txt из первого шага.
Рекомендуемые гиперпараметры: размер батча 512, w0 = 30, Adam learning rate = 0.00005f, число итераций - 15000.
При таких параметрах вы должны получить результат, достаточно похожий на то, что получилось с нашими весами на шаге 2.

Если нейросеть не обучается, то возможно вы неправильно считаете градиенты. Если не удается решить проблему методом пристального взгляда, то можно упростить задачу. Сделайте нейросеть с одним полносвязным слоем, N входами и одним выходом. Так вы фактически будете решать задачу линейной регрессии. Обучающую

выборку, где результат - линейная комбинация элементов входного вектора (типа y_i = x1_i + x2_i - 2*x3_i). Если градиенты правильные, то вы сможете быстро обучить сеть до околонулевого loss. Когда это получится, добавьте еще один слой и повторите - нейросеть снова должна обучиться идеально.

Шаг 4. Перенос прямого прохода на GPU (5 баллов)

Отрисовка изображения нейронной SDF занимает весьма много времени и перенос вычислений нейросети на GPU - отличный способ ее ускорить. Используйте подход, аналогичный первому заданию, чтобы заставить ваш код выполняться параллельно на GPU.

Шаг 5. Перенос обратного прохода на GPU (10 баллов)

Обучение нейросети также занимает много времени и может быть значительно ускорено за счет GPU, однако сделать это несколько сложнее. Рекомендуется следующий подход - каждая итерация обучения должна состоять из нескольких вызовов kernel'ов. Например, первый kernel вычисляет прямой проход нейросети, попутно сохраняя промежуточные выходы слоев в отдельный буфер (они нужны для обратного прохода). Второй kernel вычисляет функцию потерь и производит обратный проход, вычисляя градиенты функции потерь по весам нейросети. Наконец третий kernel производит одну итерацию алгоритма градиентного спуска.

В случае, если ваша реализация на GPU будет показывать ускорение менее, чем в 2 раза, вы получите на 40% меньше баллов за этот пункт.
Если ваша реализация окажется значительно быстрее эталонной, то вы получите на 40% больше баллов за этот пункт. Эталонная реализация сделана криво и у вас есть хороший шанс ее превзойти.

Удачи!