

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

К. Ма, В. Хегде, Л. Йольан

Трехмерное глубокое обучение на Python

УДК 004.04
ББК 32.372
М12

Ма К., Хегде В., Йольян Л.
М12 Трехмерное глубокое обучение на Python / пер. с англ. А. В. Логунова. – М.: ДМК Пресс, 2023. – 226 с.: ил.

ISBN 978-5-93700-202-0

В этом руководстве исследуется современное трехмерное глубокое обучение: приводятся пошаговые объяснения базовых понятий и концепций, а также практические примеры, на основе которых вы сможете создавать собственные модели. Вы научитесь обрабатывать 3D-данные с использованием облаков точек, полигональных сеток; работать с 3D-геометрией, моделями камеры, системами координат; разбираться в понятиях отрисовки, затенения и др.; применять современные продвинутые модели трехмерного глубокого обучения, такие как NeRF, SynSin, Mesh R-CNN.

Издание предназначено для практиков машинного обучения от начального до среднего уровня, исследователей данных, а также инженеров машинного и глубокого обучения, которые хотят изучить и применять методы трехмерного компьютерного зрения.

УДК 004.04
ББК 32.372

First published in the English language under the title '3D Deep Learning with Python' – (9781803247823).

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-80324-782-3 (англ.)
ISBN 978-5-93700-202-0 (рус.)

© 2022 Packt Publishing
© Перевод, оформление, издание, ДМК Пресс, 2023

Содержание

От издательства 10

Об авторах 11

О рецензентах 12

Предисловие 14

Часть I. Основы обработки 3D-данных 18

Глава 1. Введение в обработку 3D-данных 19

Технические требования 20
Настройка среды разработки 20
Представление 3D-данных 21
  Представление в виде облака точек 22
  Представление в виде полигональной сетки 22
  Представление в виде воксела 23
Формат файла 3D-данных – файлы PLY 24
Формат файла 3D-данных – файлы OBJ 29
Понятие системы 3D-координат 37
Понятие модели камеры 39
Пример программирования моделей камеры и систем координат 40
Резюме 43

Глава 2. Введение в трехмерное компьютерное зрение и геометрию 44

Технические требования 45
Ознакомление с базовыми понятиями отрисовки, растеризации и затенения 45
  Понятие барицентрических координат 47
  Модели источника света 48
  Концепция модели затенения по Ламберту 48
  Концепция модели освещения по Фонгу 49

Пример программирования 3D-отрисовки 50
Использование разнородных пакетов данных в библиотеке PyTorch3D и оптимизаторов PyTorch 57
Пример программирования разнородных мини-пакетов 59
Понятия трансформации и поворота 63
Примеры программирования трансформации и поворота 65
Резюме 66

Часть II. Трехмерное глубокое обучение с использованием библиотеки PyTorch3D 68

Глава 3. Подгонка деформируемых сеточных моделей к необработанным облакам точек 69

Технические требования 70
Задача подгонки полигональных сеток к облакам точек 70
Формулирование задачи подгонки деформируемой полигональной сетки в задачу оптимизации 73
Функции потери для регуляризации 74
Функция потери с учетом лапласианова сглаживания полигональной сетки 74
Функция потери с учетом согласованности нормалей полигональной сетки 75
Функция потери с учетом длин ребер полигональной сетки 75
Реализация подгонки полигональной сетки с помощью библиотеки PyTorch3D 76
Эксперимент без использования каких-либо регуляризационных функций потери 80
Эксперимент с использованием только одной функции потери – потери с учетом длин ребер полигональной сетки 81
Резюме 82

Глава 4. Обнаружение и отслеживание позы объекта с помощью дифференцируемой отрисовки 83

Технические требования 85
Зачем нужна дифференцируемая отрисовка 85
Как сделать отрисовку дифференцируемой 86
Какие задачи можно решать с использованием дифференцируемой отрисовки 89
Задача оценивания поз объекта 90
Как это программируется 93
Пример оценивания позы объекта для подгонки силуэта и подгонки текстуры 100
Резюме 107

Глава 5. Понятие дифференцируемой объемометрической отрисовки 109

Технические требования 110
Общий обзор объемометрической отрисовки 110
Понятие отбора лучей 112
Применение отбора объемов 115
Обследование лучевого маршировщика 116
Дифференцируемая объемометрическая отрисовка 118
Реконструкция 3D-моделей по многоракурсным изображениям 118
Резюме 123

Глава 6. Обследование нейронных полей яркости излучения (NeRF) 124

Технические требования 125
Концепция нейронных полей яркости излучения (NeRF) 125
Что такое поле яркости излучения? 126
Представление полей яркости излучения с помощью нейронных сетей 127
Тренировка модели NeRF 128
Понимание архитектуры модели NeRF 136
Понимание объемной отрисовки с использованием полей яркости излучения 142
Проецирование лучей на сцену 143
Накопление цвета луча 143
Резюме 144

Часть III. Современное трехмерное глубокое обучение с использованием библиотеки PyTorch3D 145

Глава 7. Обследование контролируемых нейронных полей признаков 146

Технические требования 147
Концепция синтеза изображений на основе GAN-сети 147
Введение в композиционный 3D-информированный синтез изображений 149
Генерирование полей признаков 152
Отображение полей признаков в изображения 153
Обследование контролируемой генерации сцен 155
Обследование контролируемой генерации автомобилей 156
Обследование контролируемой генерации лиц 158
Тренировка модели GIRAFFE 160
Начальное расстояние Фреше 161
Тренировка модели 161
Резюме 162

Глава 8. Моделирование человеческого тела в 3D 164

Технические требования 165
Постановка задачи 3D-моделирования 165
Определение подходящего представления 166
Концепция техники линейно-переходного кожного покрова 168
Концепция модели SMPL 170
Определение модели SMPL 170
Форма и шаблонная полигональная сетка в зависимости от позы 171
Суставы в зависимости от формы 171
Применение модели SMPL 172
Оценивание позы и формы человека в 3D с помощью метода SMPLify 174
Определение целевой функции оптимизации 175
Обследование метода SMPLify 176
Выполнение исходного кода 177
Обследование исходного кода 178
Резюме 182

Глава 9. Сквозной синтез ракурсов с помощью модели SynSin 183

Технические требования 184
Общий обзор синтеза ракурсов 184
Сетевая архитектура модели SynSin 185
Сети пространственных признаков и глубин 186
Нейронный отрисовщик облака точек 187
Модуль уточнения и дискриминатор 190
Тренировка и тестирование модели на практике 191
Резюме 201

Глава 10. Модель Mesh R-CNN 202

Технические требования 203
Общий обзор полигональных сеток и вокселов 203
Архитектура модели Mesh R-CNN 204
Графовые свертки 207
Предсказатель полигональной сетки 209
Демонстрация модели Mesh R-CNN с помощью PyTorch3D 212
Демонстрационный пример 212
Резюме 220

Тематический указатель 221

Глава 1

Введение в обработку 3D-данных

В этой главе мы обсудим несколько базовых понятий, весьма существенных для трехмерного глубокого обучения, которые будут часто использоваться в последующих главах. Вы начнете со знакомства с наиболее часто используемыми форматами 3D-данных, а также многими способами манипулирования ими и конвертации их в разные форматы. Мы начнем с настройки среды разработки и установкой всех необходимых программных пакетов, включая Anaconda, Python, PyTorch и PyTorch3D. Затем мы поговорим о наиболее часто используемых способах представления 3D-данных – например, облаках точек, полигональных сетках и вокселах. Затем мы перейдем к форматам файлов 3D-данных, таким как файлы PLY и OBJ. Затем обсудим системы 3D-координат. Наконец, мы обсудим модели камеры, которые в основном связаны со способом отображения 3D-данных в 2D-изображения[1].

После прочтения этой главы вы сможете легко отлаживать алгоритмы трехмерного глубокого обучения, проводя инспекцию файлов выходных данных. Благодаря четкому пониманию систем координат и моделей камеры вы будете готовы опираться на эти знания и узнать о более продвинутых темах трехмерного глубокого обучения.

В данной главе будут охвачены следующие ниже главные темы:

- настройка среды разработки и установка дистрибутива Anaconda, библиотек PyTorch и PyTorch3D,
- представление 3D-данных,
- форматы 3D-данных – файлы PLY и OBJ,
- системы 3D-координат и конверсия между ними,
- модели камеры – перспективная и ортографическая камеры.

[1] Син. соотнесение 3D-данных с 2D-изображениями. – *Прим. перев.*

Технические требования

Для выполнения примеров исходного кода этой книги в идеале понадобится компьютер с графическим процессором. Тем не менее для выполнения фрагментов исходного кода вполне будет достаточно только центрального процессора(ов).

Рекомендуемая компьютерная конфигурация включает следующее:

- GPU, такой как серия GTX или серия RTX с не менее 8 Гб памяти,
- Python 3,
- библиотеки PyTorch и PyTorch3D.

Фрагменты исходного кода к этой главе находятся по адресу https://github.com/packtpublishing/3d-deep-learning-with-python.

Настройка среды разработки

Сначала давайте создадим среду разработки для всех прилагаемых к этой книге примеров программирования. Для всех примеров исходного кода Python в этой книге рекомендуется использовать машину Linux.

1. Сначала мы настроим широко используемый дистрибутив Python под названием Anaconda, который идет в комплекте с мощной реализацией CPYTHON. Одним из преимуществ использования дистрибутива Anaconda является его система управления пакетами, позволяющая пользователям легко создавать виртуальные среды. Индивидуальная редакция дистрибутива является бесплатной для одиночных практиков, студентов и исследователей. В целях установки дистрибутива мы рекомендуем посетить его веб-сайт anaconda.com, на котором можно получить подробные инструкции. Самым простым способом установки дистрибутива Anaconda, как правило, является выполнение скрипта, который нужно скачать с веб-сайта дистрибутива. После настройки дистрибутива выполните следующую ниже команду, чтобы создать виртуальную среду Python 3.7:

   ```
   $ conda create -n python3d python=3.7
   ```

   Эта команда создаст виртуальную среду Python версии 3.7. Для того чтобы использовать эту виртуальную среду, ее нужно сначала активировать.
2. Активируйте только что созданную виртуальную среду следующей ниже командой:

   ```
   $ source activate python3d
   ```

3. Установите библиотеку PyTorch. Подробные инструкции по установке PyTorch находятся на ее веб-странице по адресу www.pytorch.org/get-

started/locally/. Например, на своем рабочем столе Ubuntu с CUDA 11.1 я установлю PyTorch 1.9.1 следующим образом:

```
$ conda install pytorch torchvision torchaudio
cudatoolkit-11.1 -c pytorch -c nvidia
```

4. Установите библиотеку PyTorch3D. Это библиотека Python с открытым исходным кодом для трехмерного компьютерного зрения, недавно выпущенная исследовательской группой Facebook AI Research. Библиотека PyTorch3D предоставляет много функций-утилит, позволяющих с легкостью манипулировать 3D-данными. Будучи спроектированной специально для глубокого обучения, она позволяет обрабатывать почти все 3D-данные мини-пакетами, такими как камеры, облака точек и полигональные сетки. Еще одной ключевой особенностью библиотеки PyTorch3D является реализация очень важной техники трехмерного глубокого обучения, именуемой *дифференцируемой отрисовкой*[1]. Тем не менее самым большим преимуществом данной библиотеки трехмерного глубокого обучения является ее тесная связь с PyTorch.
   Библиотеке PyTorch3D могут понадобиться некоторые зависимости, и подробные инструкции по установке этих зависимостей находятся на домашней странице PyTorch3D на Github по адресу github.com/facebookresearch/pytorch3d. После установки всех зависимостей, если следовать инструкциям веб-сайта, установка PyTorch3D легко выполняется следующей ниже командой:

```
$ conda install pytorch3d -c pytorch3d
```

Теперь, когда мы создали среду разработки, давайте продолжим и займемся изучением представления данных.

Представление 3D-данных

В этом разделе вы познакомитесь с наиболее часто используемыми представлениями 3D-данных. Выбор представления данных является особенно важным конструктивным решением для многих систем трехмерного глубокого обучения. Например, облака точек не имеют решетчатых структур, поэтому свертки обычно невозможно использовать для них напрямую. Представления в виде вокселов имеют решетчатые структуры, однако они, как правило, потребляют большой объем компьютерной памяти. Мы обсудим плюсы и минусы этих 3D-представлений подробнее в этом разделе. Пред-

[1] Дифференцируемая отрисовка (differentiable rendering) – это относительно новая и захватывающая область исследований в компьютерном зрении, преодолевающая разрыв между 2D и 3D, позволяющая связывать пикселы 2D-изображения с 3D-свойствами сцены. – *Прим. перев.*

ставлениями 3D-данных, получившими наиболее широкое применение на практике, обычно являются облака точек, полигональные сетки и вокселы.

Представление в виде облака точек

Облако 3D-точек – это очень простое представление 3D-объектов, в котором каждое облако точек – это просто множество 3D-точек, и каждая 3D-точка представлена одним трехмерным кортежем (*x*, *y* и *z*). Сырые мерные данные многих камер глубины обычно представляют собой трехмерные облака точек.

С точки зрения глубокого обучения облака 3D-точек являются одним из неупорядоченных и нерегулярных типов данных. В отличие от регулярных изображений, в которых по каждому отдельному пикселу можно определить соседствующие ему пикселы, в облаке точек нет четких и регулярных определений соседних точек по каждой точке – т. е. применить свертки к облакам точек обычно невозможно. И поэтому для обработки облаков точек необходимо использовать специальные типы моделей глубокого обучения, такие как PointNet: https://arxiv.org/abs/1612.00593.

Еще одной проблемой облаков точек в качестве тренировочных данных для трехмерного глубокого обучения является проблема разнородности данных – т. е. по каждому тренировочному набору данных разные облака точек могут содержать разное число 3D-точек. Один из подходов к решению проблемы разнородности данных заключается в вынужденном назначении всем облакам точек одинакового числа точек. Однако это не всегда возможно – например, число возвращаемых камерами глубины точек может отличаться от кадра к кадру.

При тренировке моделей глубокого обучения разнородные данные могут создавать некоторые трудности для мини-пакетного градиентного спуска. В большинстве систем глубокого обучения подразумевается, что каждый мини-пакет содержит тренировочные примеры одинакового размера и мерности. Предпочитаются именно такие однородные данные, поскольку их можно обрабатывать на современном оборудовании для параллельной обработки наиболее эффективным образом, таком как графические процессоры. Эффективная обработка разнородных мини-пакетов требует дополнительной работы. К счастью, PyTorch3D предоставляет целый ряд способов эффективной обработки разнородных мини-пакетов, которые очень важны для трехмерного глубокого обучения.

Представление в виде полигональной сетки

Полигональные сетки, или меши, – это еще одно широко используемое представление 3D-данных. Как и точки в облаках точек, каждая полигональная сетка содержит множество 3D-точек, именуемых вершинами. Кроме того, каждая полигональная сетка содержит множество многоугольников, именуемых гранями, которые определены на вершинах.

В большинстве основанных на данных приложений полигональные сетки являются результатом постобработки сырых мерных данных камер глубины. Нередко они создаются вручную в процессе конструирования 3D-ресурсов. По сравнению с облаками точек полигональные сетки содержат дополнительную геометрическую информацию, кодируют топологию и имеют информацию о нормалях к поверхности. Эта дополнительная информация становится особенно полезной в тренировке обучающихся моделей. Например, графовые сверточные нейронные сети обычно трактуют полигональные сетки как графы и определяют сверточные операции, используя информацию о соседстве вершин.

Подобно облакам точек, полигональные сетки также имеют схожие проблемы разнородности данных. И снова PyTorch3D предоставляет эффективные способы оперирования разнородными мини-пакетами данных полигональной сетки, что делает трехмерное глубокое обучение весьма эффективным.

Представление в виде воксела

Еще одним важным представлением 3D-данных является представление в виде воксела. Воксел – это аналог пиксела в трехмерном компьютерном зрении. Пиксел определяется путем деления прямоугольника в 2D на меньшие прямоугольники, и каждый малый прямоугольник – это один пиксел. По аналогии с этим воксел определяется путем деления трехмерного куба на кубы меньшего размера, и каждый такой куб называется одним вокселом. Соответствующие процессы показаны на следующем ниже рисунке:

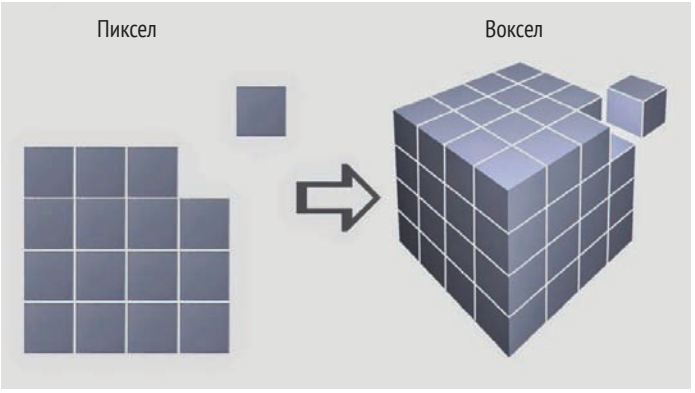


Рис. 1.1 ❖ Представление в виде воксела является трехмерным аналогом двумерного представления в виде пиксела, где кубическое пространство делится на малообъемные элементы

В представлениях в виде воксела для представления 3D-поверхностей обычно используются **функции усеченных расстояний со знаком** (**TSDF**)[1].

[1] От англ. *Truncated Signed Distance Function. – Прим. перев.*

Функция расстояния со знаком (**SDF**)[1] может быть определена на каждом вокселе в качестве расстояния (со знаком) между центром воксела до ближайшей точки на поверхности. Положительный знак в SDF указывает на то, что центр воксела находится вне объекта. Единственное различие между TSDF и SDF заключается в том, что значения TSDF усекаются, в результате чего значения TSDF всегда варьируются в интервале от –1 до +1.

В отличие от представлений в виде облаков точек и полигональных сеток представление в виде вокселов упорядочено и является регулярным. Это свойство похоже на пикселы в изображениях и позволяет использовать сверточные фильтры в моделях глубокого обучения. Одним из потенциальных недостатков представления в виде вокселов является то, что для них обычно требуется больше компьютерной памяти, но указанный недостаток можно уменьшить за счет таких методов, как хеширование. Тем не менее представление в виде вокселов является важным представлением 3D-данных.

Существуют представления 3D-данных, отличные от упомянутых выше. Например, в многоракурсных представлениях используется несколько изображений, взятых с разных точек зрения, чтобы представлять трехмерную сцену. В представлениях RGB-D используется дополнительный канал глубины, чтобы представлять 3D-сцену. Однако в этой книге мы не будем погружаться в эти 3D-представления слишком глубоко. Теперь, когда вы познакомились с основами представлений 3D-данных, самое время заняться несколькими форматами файлов, часто используемыми для облаков точек и полигональных сеток.

Формат файла 3D-данных – файлы PLY

Формат файла PLY[2] был разработан в середине 1990-х годов группой исследователей из Стэнфордского университета. С тех пор он превратился в один из наиболее широко используемых форматов файлов 3D-данных. Формат файла PLY имеет как ASCII-версию, так и двоичную версию. Двоичная версия предпочтительнее в тех случаях, когда необходимо минимизировать размеры файлов и обеспечить эффективность обработки. ASCII-версию легко отлаживать. Здесь мы обсудим базовый формат PLY-файлов и технику использования как пакета Open3D, так и библиотеки PyTorch3D для загрузки и визуализации 3D-данных из PLY-файлов.

В этом разделе мы собираемся обсудить два наиболее часто используемых формата файлов данных, чтобы представлять облака точек и полигональные сетки, формат PLY-файла и формат OBJ-файла. Мы обсудим сами форматы и способы загрузки и сохранения этих форматов файлов с помощью библиотеки PyTorch3D. Библиотека PyTorch3D предоставляет отличные функции-утилиты, поэтому с помощью этих утилит загрузка из этих форматов и сохранение в них проста и эффективна.

---

[1] От англ. *Signed Distance Function. – Прим. перев.*

[2] От англ. *Polygon File Format. – Прим. перев.*

Пример PLY-файла cube.ply показан в следующем ниже фрагменте исходного кода (данный файл находится в папке ply_io главы книги в репозитории на GitHub):

```
ply
format ascii 1.0
comment создан для книги Глубокое обучение в 3-D на Python
element vertex 8
property float32 x
property float32 y
property float32 z
element face 12
property list uint8 int32 vertex_indices
end_header
-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
3 0 1 2
3 5 4 7
3 6 2 1
3 3 7 4
3 7 3 2
3 5 1 0
3 0 2 3
3 5 7 6
3 6 1 5
3 3 4 0
3 7 2 6
3 5 0 4
```

Как видно из приведенного выше примера, каждый PLY-файл содержит секцию заголовка и секцию данных. Первая строка каждого PLY-файла в кодировке ASCII всегда содержит ключевое слово ply, указывая на то, что это PLY-файл. Вторая строка format ascii 1.0 показывает, что файл имеет кодировку ASCII с номером версии. Любые строки, начинающиеся с ключевого слова comment, будут рассматриваться как строка комментариев, и, следовательно, все, что следует за комментарием, будет игнорироваться при загрузке PLY-файла компьютером. Строка element vertex 8 означает, что первым типом данных в PLY-файле является вершина, и всего имеется восемь вершин. Выражение property float32 x означает, что в каждой вершине есть свойство x с типом float32. Аналогичным образом каждая вершина также имеет свойства y и z. Здесь каждая вершина – это одна 3D-точка. Строка element face 12 означает, что второй тип данных в указанном PLY-файле имеет тип face, и всего имеется 12 граней. Выражение property list uint8 int32 vertex_indices показывает, что каждая грань будет списком индексов вершин. Секция заголовка PLY-файла всегда заканчивается строкой end_header.

Первая часть секции данных PLY-файла состоит из восьми строк, каждая строка которой является записью одной вершины. Три числа в каждой строке представляют три свойства `x`, `y` и `z` вершины. Например, три числа `-1`, `-1`, `-1` указывают на то, что координата `x` вершины равна `-1`, координата `y` вершины равна `-1` и координата `z` вершины равна `-1`.

Вторая часть секции данных PLY-файла состоит из 12 строк, каждая строка которой является записью одной грани. Первое число в последовательности чисел указывает число имеющихся у грани вершин, а последующие числа являются индексами вершин. Индексы вершин определяются по порядку, в котором вершины объявлены в PLY-файле.

Для открытия приведенного выше файла можно использовать как пакет Open3D, так и библиотеку PyTorch3D. Пакет Python Open3D очень удобен для визуализации 3D-данных, а библиотека PyTorch3D удобна для применения этих данных в моделях глубокого обучения. Ниже приведен фрагмент исходного кода файла Python `ply_example1.py` в папке `ply_io` главы книги в репозитории на GitHub для визуализации полигональной сетки в PLY-файле `cube.ply` и загрузки вершин и полигональной сетки в виде тензоров PyTorch:

```
import open3d
from pytorch3d.io import load_ply

mesh_file = 'cube.ply'
print('Визуализация полигональной сетки с помощью Open3D')
mesh = open3d.io.read_triangle_mesh(mesh_file)
open3d.visualization.draw_geometries([mesh],
       mesh_show_wireframe = True,
       mesh_show_back_face = True)

print('Загрузка того же файла с помощью PyTorch3D')
vertices, faces = load_ply(mesh_file)
print('Тип vertices = ', type(vertices))
print('Тип faces = ', type(faces))
print('vertices = ', vertices)
print('faces = ', faces)
```

В приведенном выше фрагменте исходного кода Python PLY-файл полигональной сетки `cube.ply` сначала открывается с помощью функции `read_triangle_mesh`[1] пакета Open3D, и все 3D-данные читаются в переменную `mesh`. Затем полигональную сетку можно визуализировать, используя функцию `draw_geometries`[2] данного пакета. При выполнении этой функции пакет Open3D выведет на экран окно для интерактивной визуализации полигональной сетки, в котором ее можно интерактивно поворачивать, увеличивать и уменьшать ее масштаб, используя мышь. PLY-файл `cube.ply`, как можно догадаться, определяет полигональную сетку куба с восемью вершинами и шестью сторонами, каждая сторона которого покрыта двумя гранями.

---

[1] Прочитать треугольную сетку. – *Прим. перев.*

[2] Начертить геометрические объекты. – *Прим. перев.*

Для загрузки той же сетки также можно использовать библиотеку PyTorch3D. Однако на этот раз мы собираемся получить несколько тензоров PyTorch – например, один тензор для вершин и один тензор для граней. Эти тензоры можно напрямую водить в любую модель глубокого обучения PyTorch. В данном примере функция `load_ply` возвращает кортеж с вершинами и гранями, обе из которых обычно находятся в формате тензоров PyTorch. При выполнении исходного кода файла Python `ply_example1.py` возвращенные вершины должны быть тензором PyTorch с очертанием[1] `[8, 3]`, т. е. восемь вершин, и в каждой вершине по три координаты. Аналогичным образом возвращенные грани должны быть тензором PyTorch с очертанием `[12, 3]`, т. е. 12 граней, и у каждой грани по 3 индекса вершин.

В следующем ниже фрагменте исходного кода демонстрируется еще один пример PLY-файла, `parallel_plane_mono.ply`, который тоже можно скачать из репозитория книги на GitHub. Единственное различие между полигональной сеткой в этом примере и полигональной сеткой в PLY-файле `cube.ply` состоит в том, что теперь вместо шести сторон куба у нас только четыре грани, которые образуют две параллельные плоскости:

```
ply
format ascii 1.0
comment создан для книги Глубокое обучение в 3-D на Python
element vertex 8
property float32 x
property float32 y
property float32 z
element face 4
property list uint8 int32 vertex_indices
end_header
-1 -1 -1
1 -1 -1
1 1 -1
-1 1 -1
-1 -1 1
1 -1 1
1 1 1
-1 1 1
3 0 1 2
3 0 2 3
3 5 4 7
3 5 7 6
```

Полигональную сетку можно интерактивно визуализировать с помощью следующего ниже фрагмента исходного кода файла Python `ply_example2.py`.

1. Сначала импортируем все необходимые библиотеки Python:

```
import open3d
from pytorch3d.io import load_ply
```

[1] Также форма (англ. *shape*). – *Прим. перев.*

2. Затем, используя пакет Open3D, загружаем полигональную сетку:

```
mesh_file = 'parallel_plane_mono.ply'
print('Визуализация полигональной сетки с помощью Open3D')
mesh = open3d.io.read_triangle_mesh(mesh_file)
```

3. Далее применяем его метод `draw_geometries`, чтобы открыть окно для интерактивной визуализации полигональной сетки:

```
open3d.visualization.draw_geometries([mesh],
                                     mesh_show_wireframe=True,
                                     mesh_show_back_face=True)
```

4. Затем используем библиотеку PyTorch3D, чтобы открыть ту же полигональную сетку:

```
print('Загрузка того же файла с помощью PyTorch3D')
vertices, faces = load_ply(mesh_file)
```

5. И в конце распечатываем информацию о загруженных вершинах и гранях. На самом деле это просто обычные тензоры PyTorch3D:

```
print('Тип vertices = ', type(vertices),
      ', тип faces = ', type(faces))
print('vertices = ', vertices)
print('faces = ', faces)
```

Для каждой вершины также можно задать свойства, отличные от координат *x*, *y* и *z*. Например, можно задать цвета каждой вершины. Ниже приведен пример `parallel_plane_color.ply`:

```
ply
format ascii 1.0
comment создан для книги Глубокое обучение в 3-D на Python
element vertex 8
property float32 x
property float32 y
property float32 z
property uchar red
property uchar green
property uchar blue
element face 4
property list uint8 int32 vertex_indices
end_header
-1 -1 -1 255 0 0
1 -1 -1 255 0 0
1 1 -1 255 0 0
-1 1 -1 255 0 0
-1 -1 1 0 0 255
1 -1 1 0 0 255
1 1 1 0 0 255
-1 1 1 0 0 255
```

```
3 0 1 2
3 0 2 3
3 5 4 7
3 5 7 6
```

Обратите внимание, что в приведенном выше примере, наряду с *x*, *y* и *z*, мы также задаем несколько дополнительных свойств по каждой вершине, т. е. свойства `red`, `green` и `blue`, все это в типе данных `uchar`. Теперь каждая запись одной вершины представляет собой одну строку из шести чисел. Первые три числа – координаты *x*, *y* и *z*. Следующие три – значения RGB.

Полигональную сетку можно визуализировать с использованием файла Python `ply_example3.py` следующим образом:

```
import open3d
from pytorch3d.io import load_ply

mesh_file = 'parallel_plane_color.ply'
print('Визуализация полигональной сетки с помощью Open3D')

mesh = open3d.io.read_triangle_mesh(mesh_file)
open3d.visualization.draw_geometries([mesh],
                                     mesh_show_wireframe=True,
                                     mesh_show_back_face=True)
print('Загрузка того же файла с помощью PyTorch3D')
vertices, faces = load_ply(mesh_file)
print('Тип vertices = ', type(vertices),
      ', тип faces = ', type(faces))
print('vertices = ', vertices)
print('faces = ', faces)
```

В PLY-файле `cow.ply` мы также предоставляем реальный пример полигональной 3D-сетки. Читатели могут визуализировать указанную сетку с использованием файла Python `ply_example4.py`.

До сего момента мы говорили о базовых элементах формата файла 3D-данных PLY, таких как вершины и грани. Далее мы обсудим формат файла 3D-данных OBJ.

Формат файла 3D-данных – файлы OBJ

В этом разделе мы обсудим еще один широко используемый формат файла 3D-данных, формат файла OBJ. Формат файла OBJ был впервые разработан компанией Wavefront Technologies Inc. Как и формат файла PLY, формат OBJ также имеет как ASCII-версию, так и двоичную версию. Двоичная версия является проприетарной и незарегистрированной. И поэтому в данном разделе мы обсудим ASCII-версию.

Как и в предыдущем разделе, мы собираемся обследовать этот формат файла, посмотрев на примеры. Первый пример, `cube.obj`, приведен ниже. Как можно догадаться, в OBJ-файле определяется полигональная сетка куба.

```
mtllib ./cube.mtl
o cube
# Список вершин
v -0.5 -0.5 0.5
v -0.5 -0.5 -0.5
v -0.5 0.5 -0.5
v -0.5 0.5 0.5
v 0.5 -0.5 0.5
v 0.5 -0.5 -0.5
v 0.5 0.5 -0.5
v 0.5 0.5 0.5

# Список точек/линий/граней
usemtl Door

f 1 2 3
f 6 5 8
f 7 3 2
f 4 8 5
f 8 4 3
f 6 2 1
f 1 3 4
f 6 8 7
f 7 2 6
f 4 5 1
f 8 3 7
f 6 1 5
```

Первая строка, `mtlib ./cube.mtl`, объявляет сопровождающий файл библиотеки шаблонов материалов (**MTL**)[1].

MTL-файл описывает свойства затенения поверхности, которые будут объяснены в следующем далее фрагменте исходного кода.

В строке `o cube` начальная буква `o` указывает на то, что в строке задается объект, имя которого – `cube`. Любая строка, начинающаяся с `#`, является строкой комментариев, т. е. остальная часть строки будет компьютером проигнорирована. В каждой строке, начинающейся с `v`, `v` указывает на то, что в данной строке задается вершина. Например, строка `v -0.5 -0.5 0.5` задает вершину с *x*-координатой `0.5`, *y*-координатой `0.5` и *z*-координатой `0.5`. В каждой строке, начинающейся с `f`, `f` указывает на то, что в данной строке содержится определение одной грани. Например, строка `f 1 2 3` задает грань, причем три ее вершины являются вершинами с индексами 1, 2 и 3.

Строка `usemtl Door` объявляет, что объявленные после этой строки поверхности должны быть затенены с помощью определенного в MTL-файле материального свойства под именем `Door`.

Сопровождающий MTL-файл `cube.mtl` приведен ниже. Указанный файл задает материальное свойство, именуемое `Door`:

```
newmtl Door
Ka  0.8 0.6 0.4
```

[1] От англ. *Material Template Library. – Прим. перев.*

```
Kd  0.8 0.6 0.4
Ks  0.9 0.9 0.9
d   1.0
Ns  0.0
illum 2
```

Мы не будем подробно обсуждать эти материальные свойства, кроме `map_Kd`. Если вам интересно, то можете обратиться к стандартному учебнику по компьютерной графике, такому как «Компьютерная графика: принципы и практика»[1]. Мы перечислим некоторые грубые описания этих свойств следующим образом, просто ради полноты:

- `Ka`: задает окружающий цвет,
- `Kd`: задает рассеянный цвет,
- `Ks`: задает бликовый цвет,
- `Ns`: определяет фокус бликовых моментов,
- `Ni`: определяет оптическую плотность (он же индекс рефракции),
- `d`: задает коэффициент растворения,
- `illum`: задает модель освещения,
- `map_Kd`: задает файл цветной текстуры, который будет применен к диффузной отражательной способности материала.

OBJ-файл `cube.obj` можно открыть как с помощью пакета Open3D, так и с помощью библиотеки PyTorch3D. Следующий ниже фрагмент исходного кода, содержащийся в файле Python `obj_example1.py`, можно скачать из репозитория книги на GitHub:

```
import open3d
from pytorch3d.io import load_obj

mesh_file = 'cube.obj'
print('Визуализация полигональной сетки с помощью Open3D')
mesh = open3d.io.read_triangle_mesh(mesh_file)
open3d.visualization.draw_geometries([mesh],
                                     mesh_show_wireframe=True,
                                     mesh_show_back_face=True)

print('Загрузка того же файла с помощью PyTorch3D')
vertices, faces, aux = load_obj(mesh_file)
print('Тип vertices = ', type(vertices))
print('Type faces = ', type(faces))
print('Тип aux = ', type(aux))
print('vertices = ', vertices)
print('faces = ', faces)
print('aux = ', aux)
```

В приведенном выше фрагменте исходного кода заданная полигональная сетка куба интерактивно визуализируется с помощью функции Open3D `draw_geometries`. Указанная сетка будет показана в окне, и вы сможете ее поворачивать, увеличивать и уменьшать ее масштаб, используя мышь. Полигональную

[1] От англ. *Computer Graphics: Principles and Practice. – Прим. перев.*

сетку также можно загрузить с помощью функции PyTorch3D `load_obj`. Функция `load_obj` вернет переменные `vertices`[1], `faces`[2] и `aux`[3] в формате тензора PyTorch либо в формате кортежей тензоров PyTorch.

Пример результата работы фрагмента исходного кода `obj_example1.py` приводится ниже:

```
Визуализация полигональной сетки с помощью Open3D
Загрузка того же файла с помощью PyTorch3D
Тип vertices = <class 'torch.Tensor'>
Тип faces = <class 'pytorch3d.io.obj_io.Faces'>
Тип aux = <class 'pytorch3d.io.obj_io.Properties'>
vertices = tensor([[-0.5000, -0.5000,  0.5000],
                   [-0.5000, -0.5000, -0.5000],
                   [-0.5000, 0.5000, -0.5000],
                   [-0.5000, 0.5000,  0.5000],
                   [ 0.5000, -0.5000,  0.5000],
                   [ 0.5000, -0.5000, -0.5000],
                   [ 0.5000, 0.5000, -0.5000],
                   [ 0.5000, 0.5000,  0.5000]])
faces = Faces(verts_idx=tensor([
                   [0, 1, 2],
                   [5, 4, 7],
                   [6, 2, 1],
                   ...
                   [3, 4, 0],
                   [7, 2, 6],
                   [5, 0, 4]]),
               normals_idx=tensor([
                   [-1, -1, -1],
                   [-1, -1, -1],
                   [-1, -1, -1],
                   [-1, -1, -1],
                   ...
                   [-1, -1, -1],
                   [-1, -1, -1]]),
               textures_idx=tensor([
                   [-1, -1, -1],
                   [-1, -1, -1],
                   [-1, -1, -1],
                   ...
                   [-1, -1, -1],
                   [-1, -1, -1]]),
               materials_idx=tensor([0, 0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0, 0]))
aux = Properties(
           normals=None,
           verts_uvs=None,
```

[1] Вершины. – *Прим. перев.*

[2] Грани. – *Прим. перев.*

[3] Вспомогательная переменная. – *Прим. перев.*