Пока что есть 4 варианта вокселизаторов:

1. Obj to obj\objvox

* Что такое objvox
* Работает с CUDA
* Все в одном

1. Stl to obj, но работает криво, лучше забить Voxelizer-master
2. Stl to vox, surface VoxSurf

* Нет оптимизации
* Вроде как все юзер-фрэндли
* Только модели из папки, мб можно настроить

1. Просто библиотека HeaderOnly
   * Дб все суперпросто и быстро
   * Надо разобраться что да как
2. Матлаб
   * Непонятно как использовать на практике
   * Отвратительно медленный

Остаются варианты 1, 3, 4.

1- ~~работает с obj, будет сложно делать датасет, но оно работает 100% (в папке E:\Soft\voxelizer-master\out\build\x64-Debug (по умолчанию)\bin):~~

~~./voxelizer -f obj E:\Soft\charizard.obj E:\Soft\voxecharizard.obj~~

~~Могут быть проблемы с работой с stl, переведенными в obj~~

3- все шикарно, работает с stl, но надо научиться менять размерность (а надо ли?) + ~~запускает только файлы из папки~~

~~4- насчет работы хз, но будет легко включить в проект, плюс вроде как проще~~

~~В итоге есть исполняемый файл, надо посмотреть на вариантивность размерностей и научиться запускать сразу для множества файлов~~

Остается проблема с числом пикселей, мое решение- оставить разрешение 128 и расширять модели до квадрата по 128 пикселей- потому что стороны не могут быть больше 128 (он вписывает модель максимально эффективно в квадрат, одна из сторон которого 128, остальные минимальные)- возможно стоит просто забить, потом будет траить значение пикселя в пределах 128.

Следующая задача- сделать анализ, почему был выбран именно это вокселизатор и его метод работы.

Общий план:

1. Нир- доработать, привести в человеческий дипломный вид
2. Вокселизатор- см. ниже
3. Датасет- как формировался (папки, разделение, можно описать методику 5 на 2), описать modelNet10
4. Нейросеть- варианты, почему выбрали именно эту
5. Тестирование- мб описать метод (5 на 2), результаты?
6. Вывод- все что сделали, в какую сторону можно двигаться

Первая задача при обработке моделей- подготовка входных данных для нейросетей. Проблема возникает при неоднородности входных данных, то есть при отличающихся размерах подаваемых моделей. Чтобы избежать ошибок, решено было подготовить алгоритм, который будет вокселизировать модели, ограничиваясь настраиваемым размером. Для разработки в качестве основы были выбраны пять уже готовых алгоритмов, и из них выбран подходящий для текущей задачи. Далее будут описаны минусы и плюсы каждого варианта и обоснование выбора итогового. Также будут приведены результаты сравнения скоростей.

Прежде чем переходит к алгоритмам, стоит определить, что из себя представляют воксели и как они хранятся в памяти. Воксели, также как и пиксели, не содержат в себе информации о положении в пространстве. Их координаты вычисляются из трехмерной матрицы, которая моделирует модель. Но, при преобразовании из воксельной модели в .obj, воксели преобразуются в типовую для этого формата модель (к примеру, вершины и грани), что увеличивает общее время работы программы, так как ресурсы тратятся на хранение объекта в памяти и преобразование. Формат .vox разработан для использования в программе *magicavoxel* и предназначен для хранения вокселей, то есть реализует трехмерную матрицу.

Алгоритмы ниже были выбраны с учетом применимости, на вход они принимают файлы в форматах .obj или .stl.

Изначально все найденные алгоритмы можно поделить на 3 группы: первая- на входе получает модель в формате .obj, результат на выходе может варьироваться. Для работы выбран алгоритм /ссылка на гит?/. Ключевым преимуществом этого алгоритма стоит выделить использование архитектуры параллельных вычислений CUDA, что в теории должно ускорить обработку. На выходе создается файл в формате .obj или .objvox.

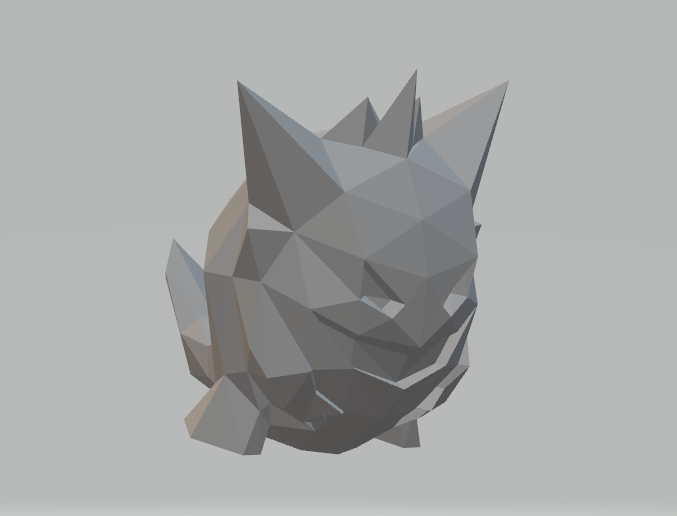
Второй вариант- алгоритмы, использующие в качестве входа файлы в формате .stl, что упрощает их практическое использование, так как файлы в этом формате занимают значительно меньше памяти и проще в строении, а также их можно получить в различных САПР. В этой работе тестировались 2 алгоритма: /ссылка на гит?/ и /ссылка на гит?/. Основными плюсами были простота реализации и настройки, так как алгоритмы используют один файл и стороннюю библиотеку в первом варианте и один файл во втором. На выходе в первом варианте получается файл .vox, во втором- матрица координат вокселей, которую можно в дальнейшем использовать для создания любых файлов.

Третий вариант- использование библиотеки в программе Matlab. Таких алгоритмов было найдено 2: в первом случае на вход подается модель в формате .obj, на выходе получается трехмерная матрица из bool-переменных. Во втором случае на вход подается .stl модель, а выход- также трехмерная матрица, но в формате int\_8, т.е. целые числа размером не больше 8 бит.

Для выбора итогового варианта использовались следующие критерии: во-первых, их должно быть удобно подстраивать под текущую задачу обработки большого числа моделей. Также, на выходе должны получаться удобные для обработки модели, которые занимают минимум места и хранят минимально необходимый объем информации. Третьим критерием является скорость работы.

Алгоритм, на выходе которого получается матрица вокселей, дает возможность встраивать его в различные программы, но в этой задаче получение сетки недостаточно и его необходимо дорабатывать, в дальнейшем он рассматриваться не будет.

Скорость работы алгоритмов была протестирована на модели, показанной на рисунке 1. Она была выбрана из-за своей сложной структуры, что позволит точнее определить разницу в скорости алгоритмов, которая будет увеличиваться с увеличением сложности модели.

Рисунок 1- Модель для тестирования скорости

Результаты тестирования представлены в таблице 1.

Таблица 1- Сравнение скоростей обработки модели

|  |  |
| --- | --- |
| Алгоритм | Скорость обработки одной модели |
| obj | 1436 мс |
| stl | 508 мс |
| Matlab (obj) | >>1 c |
| Matlab (stl) | 539 мс |

Очевидно преимущество в скорости алгоритмов, основанных на .stl моделях. Также эти алгоритмы удобно использовать для составления датасета, методика подготовки которого будет описана далее.

За основы был взять алгоритм из matlab ввиду легкой реализуемости и изменения, а также из-за понятного кода. Также был подготовлен альтернативный вариант- алгоритм VoxSurf.

Для составления датасета оба алгоритма были изменены. Далее улучшение алгоритма будет показано на примере алгоритма для matlab, улучшение VoxSurf производилось аналогично.

На вход подается папка с файлами (dirData), и далее для каждого файла в ней проводится операция вокселизации и записи в файл. В итоге для каждого класса получается набор файлов, представляющих собой сохраненную переменную Matlab, из которых в дальнейшем формируется датасет.

|  |
| --- |
| dirData = dir('75/\*.stl'); |
| for index = 1:58 |
| filename1 = convertStringsToChars(string(dirData(index).folder) + '\' + string(dirData(index).name)); |
| [OUTPUTgrid] = VOXELISE(32,32,32,filename1,'xyz'); |
| OUTPUTgrid = int8(OUTPUTgrid); |
| filename = '75\_mat/' + string(index) + '.mat'; |
| save(filename, 'OUTPUTgrid'); |
| end |

* [Fundamentals of Voxelization](http://www.cs.sunysb.edu/~vislab/projects/volume/Papers/Voxel/index.html) (англ.). IEEE Computer, Vol. 26, No. 7 (July 1993). Дата обращения: 9 июля 2010. [Архивировано](https://www.webcitation.org/65XTHidqM?url=http://www.cs.sunysb.edu/vislab/wordpress//projects/volume/Papers/Voxel/index.html) 18 февраля 2012 года.
* /ссылки на репозитории?/