Пока что есть 4 варианта вокселизаторов:

1. Obj to obj\objvox
    - Что такое objvox
    - Работает с CUDA
    - Все в одном
2. Stl to obj, но работает криво, лучше забить Voxelizer-master
3. Stl to vox, surface VoxSurf
    - Нет оптимизации
    - Вроде как все юзер-фрэндли
    - Только модели из папки, мб можно настроить
4. Просто библиотека HeaderOnly
    - Дб все суперпросто и быстро
    - Надо разобраться что да как
5. Матлаб
    - Непонятно как использовать на практике
    - Отвратительно медленный
6. Второй матлаб
    - В целом неплохой вариант, если необходимо получить сетку

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.

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 или готового кода в программе Matlab. Было найдено два примера: в первом случае для входа и выхода используются файлы в формате .obj, вот втором- при подаче на вход файла в формате .stl в результате получается сетка, которую в дальнейшем можно использовать для визуализации.

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.

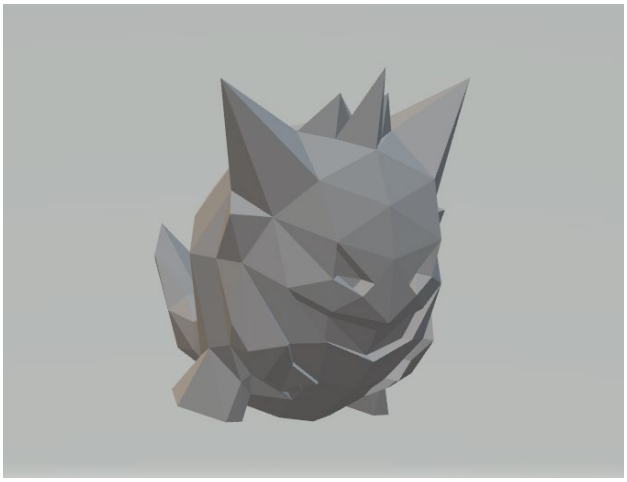

Рисунок 1- Модель для тестирования скорости

Результаты тестирования представлены в таблице 1.

Таблица 1- Сравнение скоростей обработки модели

Алгоритм	Скорость обработки одной модели
obj	1436 мс
stl	508 мс
Matlab1	>>1 с
Matlab2	966 мс

В итоге был выбран алгоритм VoxSurf, основываясь на следующих преимуществах: удобство настройки- для большинства параметров-переменных легко определить нужное значение и поменять его при необходимости, скорость работы, в виду того, что результатом является модель в формате .vox, предназначенном для хранения вокселей, а также из-за того, что .stl модели легко получать в САПР Solidworks, с помощью которого в дальнейшем и будут сформированы модели для датасета.

Алгоритм /мб использовать их оригнальные названия?/ рассматривается как альтернативный, и будет использоваться при необходимости. Также возможно использование кода в Matlab, чтобы в итоге получить сетку, в дальнейшем подаваемую в нейросеть.

Кроме того, для подготовки датасета алгоритм VoxSurf был доработан. Было настроено разрешение получаемой модели, то есть максимальное количество вокселей по любой из координат, а также доработан метод подачи

входных данных таким образом, чтобы на вход подавался путь до папки с файлами, путь до папки для хранения результата и класс изделия- строка, использующаяся в дальнейшем формировании имения файлов и по которой будет определяться класс изделия при обучении нейросети.

1. Fundamentals of Voxelization // IEEE Computer, Vol. 26, No. 7.
2. /ссылки на репозитории?/