Final Goal Calibration

- Team SegFault
- Byungju, Manoj, Mateusz

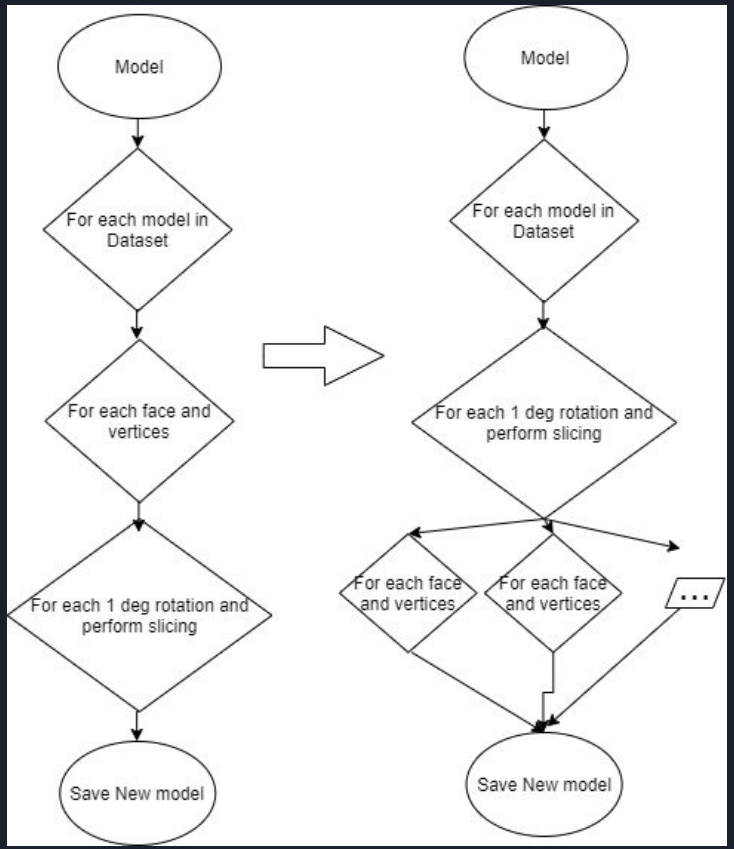
Initial Objective

- We set out to implement the methodology described in the work Generative and Discriminative Voxel Modeling with Convolutional Neural Networks.
- A dataloader system was needed in order to convert the database's .off files into workable voxel objects.
- This would consist of the design of a machine learning model (a variational autoencoder) to train on the Princeton ModelNet database.
- The model would then be able to generate voxel objects given a random seed. We would use voxel-based convnets for object classification.
- A GUI would be developed that displays generated objects as well as providing a user interface to dynamically interpolate between models.

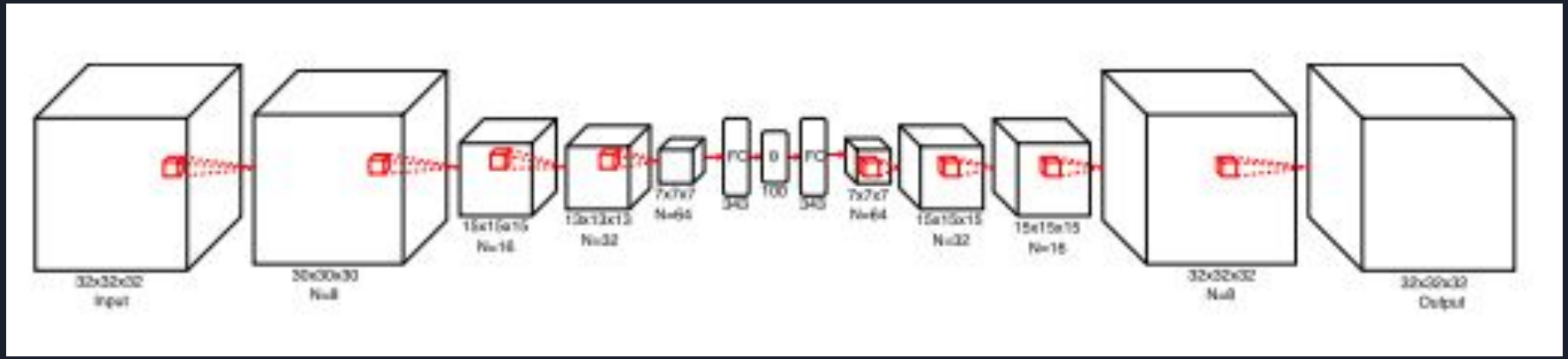
Dataloader (Feed Preprocessing)

- Code conversion improved performance in space complexity from ~10GB RAM occupancy as mentioned in the matlab code to ~2GB RAM
- Using OpenMP to Muti-thread the matrix rotation and slicing.

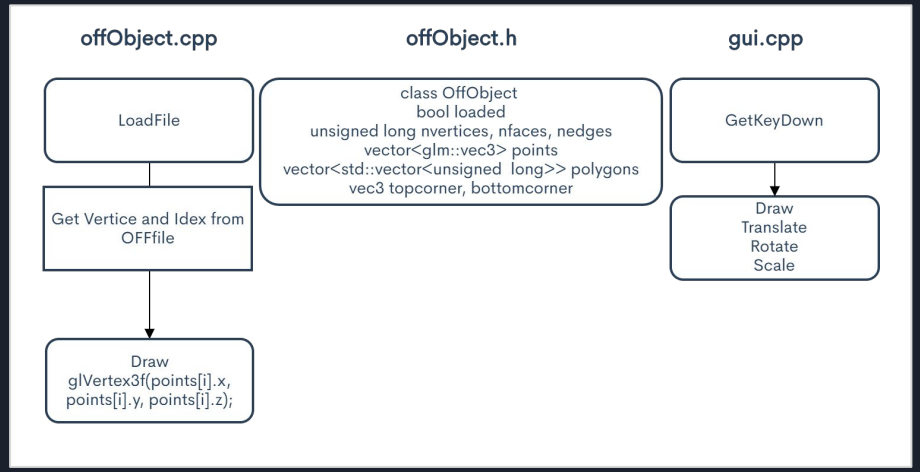
- Now we could do it online in training and testing.
- (Can be used for Real-time)



Variational Autoencoder Architecture



GUI code

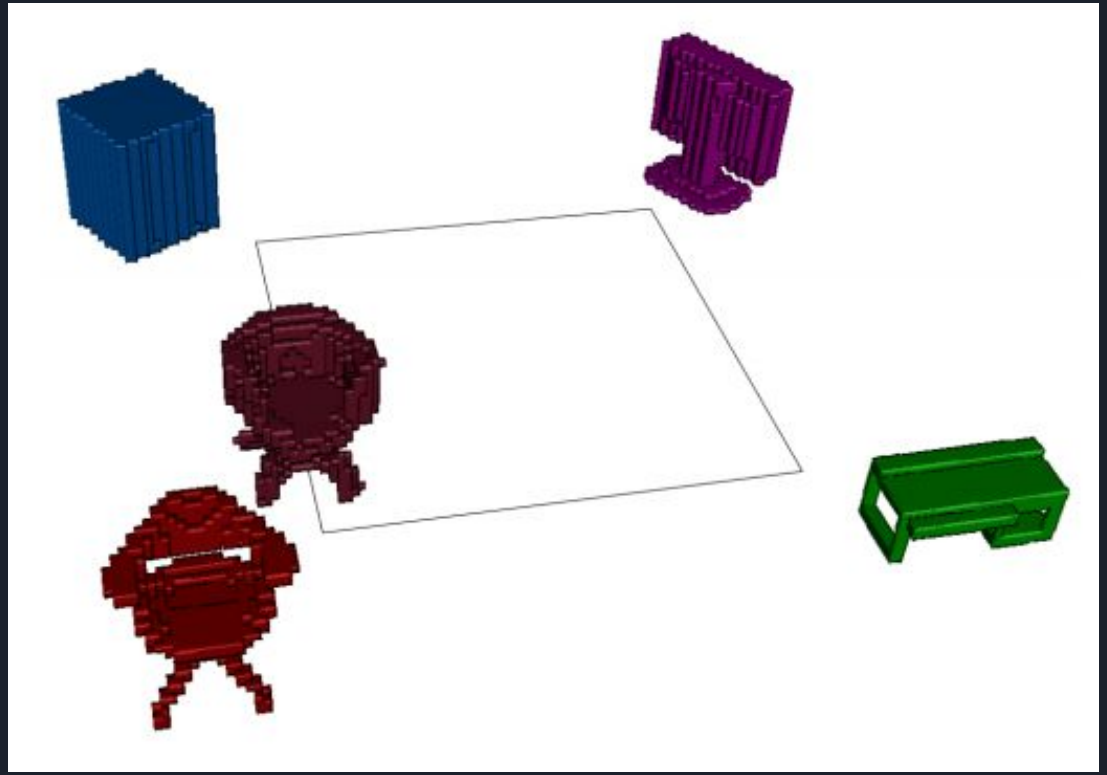


offObject.cpp* | offObject.h | gui.cpp

Miscellaneous Files | OffObject

```
59      polygons.reserve(nfaces);
60      for (auto i = 0; i < nfaces; ++i) {
61          if (!get_next_uncommented_line(infile, info)) {
62              return false;
63          }
64          std::istringstream info_stream(info);
65          unsigned long n;
66          unsigned long index;
67          info_stream >> n;
68          polygons.emplace_back(n);
69          for (auto j = 0; j < n; ++j) {
70              info_stream >> index;
71              polygons[i][j] = index;
72          }
73      }
74
75      infile.close();
76      return (loaded = true);
77  }
78  void OffObject::draw(DISPLAY_TYPE displayType) const {
79      if (displayType == DISPLAY_TYPE::POINTS) {
80          glBegin(GL_POINTS);
81          std::for_each(points.begin(), points.end(),
82              [](const auto& point) { glVertex3f(point.x, point.y, point.z); });
83          glEnd();
84      }
85      else {
86          GLenum mode = (displayType == DISPLAY_TYPE::POLYGONS) ? GL_POLYGON : GL_LINE_LOOP;
87          for (const auto& indexes : polygons) {
88              glBegin(mode);
89              for (const auto& i : indexes) {
90                  glVertex3f(points[i].x, points[i].y, points[i].z);
91              }
92              glEnd();
93          }
94      }
95  }
```

GUI Display



Concerns to Address

- **C++ Tensorflow API** does not natively possess certain functionalities (such as 3D batch normalization), thus requiring some workarounds.
- Processing the database proved to be more involved than initially thought.
- Data loading in C++ requires O(n^2 + k) (n>k) based on batchsize. Reduction in time complexity with rotating first and voxelizing to O(k^2 + n).
- Advanced features of GUI, such as interpolation between objects, as well as voxception based classifiers, have proven to be rather complex and time-consuming to implement.

Project Adjustment and Final Deliverables

- We will have a functioning dataloader and VAE model, capable both of training on and generating these voxel objects.
- We intend to table the implementation of Voxception classifiers, though we may given time availability. (According to time available)
- Our GUI will display models of the training set and generated objects, as well as the progress of the VAE, but we do not intend to featured

Any Questions?