3D Asset Generation

CS256 Group A: Aaron Mandeville, Justin Chang, Niraj Pandkar, Ji An Lee, Joshua Benz, Archit Jain

I. Abstract

Inspired by the shortcomings of Atlas, we propose to use a 3D-Recurrent Reconstruction Neural Network (3D-R2N2) to reconstruct 3D models that can be made into video game assets [1][9]. This model leverages locality by using an attention mechanism on the probabilistic voxel occupancy map to overcome the problems of occlusion. The model learns shape-priors and tries to leverage the attention mechanism to generate the other pieces by selectively updating what should be reconstructed. This means that we only need a series of RGB images from different viewpoints as input. Therefore we no longer need corresponding pose information or to use any SLAM methods. We include segmentation in the preprocessing stage to extract what we want to reconstruct. The model performs well on images of objects classes that are contained in the ShapeNet and Pascal3D datasets [4][6].

II. Introduction

The goal of the project is to create 3D models of objects which can be imported as assets into software pertaining to 3D graphics. Specifically, we would like to create 3D model files that can be imported into the Unity game engine. Through this project, we hope to demonstrate an understanding of 3D object reconstruction and explore the different methods and techniques one can use to achieve this goal.

Using the 3D-R2N2 [1] as the backbone of the project, 3D models of objects are created from a sparse number of views. The required input for the model is a series of images of the target object from various angles (the more angles, the better). The output is a 3D rendered object in a .obj file format. Image segmentation is used to refine the image before feeding it into the model, resulting in a cleaner object model. The project team created a web application to allow users to upload videos of an object to run through the model and return an object file.

Prior to using the 3D-R2N2 model, the project team attempted to use the Atlas 3D Scene Reconstruction [9] model and DeepVoxels: Learning Persistent 3D Feature Embeddings [2] for 3D object reconstruction. The technical challenges faced during the implementation of these models have been explained in the shortcomings section of this report.

III. Literature Survey

a. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction [1]

This paper takes the problem of object 3D reconstruction from a single or series of images. It does not rely on a dense number of viewpoints to account for any self occluded surfaces. It also does not require objects having a Lambertian surface. Many modern 3D object reconstruction models rely on SLAM, which requires those two constraints. This model uses a shape-priors method. This method infers small geometric shapes that make the building blocks of an object [1].

The novelty of this model is that it uses networks to infer the mapping of the geometry of the primitive shapes that make up the object to the images of the object. Based on what the model has learned from training it is able to do this. The model requires very little supervision and overhead because it doesn't utilize keypoints, segmentation, viewpoint labels, class labels, or camera calibration. The resulting model is a fast network that outperforms other single-view reconstruction methods and enables 3D reconstruction where traditional SLAM methods fail.

The model is divided into three parts: an encoder, 3D convolutional LSTM, and a decoder. The principle behind the architecture design is to utilize LSTM to remember previous observations and use those observations to improve 3D reconstruction. The encoder component is a 2D-CNN tasked with the downsampling of the input 127x127 image into lower-dimensional features. The 3D-Convolutional LSTM learns how to construct each subvoxel space using the loss between predicted reconstruction and the ground truth data. The decoder uses the hidden states of the LSTM for the final 3D voxel reconstruction.

b. ATLAS: 3D Scene Reconstruction [9]

This paper predicts a full 3D model from a range of images. The input for the method described in the paper is a monocular video. Unlike popular methods, the authors decided to take an arbitrary sequence of frames from the video as an input. This eliminates the need for frame selection, which otherwise would have increased the complexity. The model presented in this paper is trained for the 3D reconstruction of indoor scenes.

First, 2D features are extracted from each image in the sequence using a 2D Convolutional Neural Network, specifically using a RESNET50-FPN architecture. Second, These extracted 2D features are then back-projected into a canonical voxel volume similar to DeepVoxels - all the pixels get mapped onto a ray in the volume using the known camera intrinsics and extrinsics. Third, A 3D Convolutional encoder-decoder architecture is further used to refine the accumulated features. Finally, a 1x1x1 convolution is used to regress the TSDF output.

IV. Methodology

Our method is to take a video as input to our application, process that video, and return the resulting 3D model to the user. When the video is uploaded to our server, we select a number of frames, usually around four to six frames, and process them. The selected frames are segmented to extract the desired object and converted to an input that the model can understand. This involves resizing the image, padding it with white pixels to make it square with dimensions 127x127 and giving it to the model as input. The model extracts features from the input image and places them into a 3D grid. The grid provides locality through an attention mechanism that is leveraged by using the model. We can use this to reconstruct visible parts of the voxel space while maintaining what was learned about the rest of the model even if it is occluded in the image. This is done for each input image in the sequence and the loss is computed after the model has processed all of the images in the sequence. Finally, the information learned goes through the decoding stage which contains various unpooling and convolution layers to produce a larger grid called a probabilistic voxel occupancy map [1]. We then set a threshold and use the map of probabilities to produce a binary occupancy grid. This binary grid is saved into a wavefront 3D object file which can be imported into most 3D viewers or game engines for use. Our application sends that file back to the user and is displayed to observe the result.

Datasets

The 3D-R2N2 model was trained on two different datasets:

a. ShapeNet [6]
Contains detailed 3D models of various algorithmically and manually annotated objects. A subset of 50k images called ShapeNetCore was used in the training of this model. It has 12 object categories in common with another popular benchmark data set called PASCAL 3D+

b. PASCAL 3D [4]

Contains images of a wide variety of objects ranging in size and background. It is built upon the PASCAL VOC 2012 dataset and contains corresponding 3D CAD models for training [11]. There are 12 categories with over 3,000 images per category. This was used for fine-tuning the model to decide how many iterations should be done as well as find other hyperparameters.

Our project was tested on the following datasets:

a. Custom Dataset:
Any user of the web app can also upload their own videos to test the model and obtain object files. Given a video, we would select a number of frames relative to the length of the video, which usually ended up being 4 to 6 images. An important note here is that the more frames we selected, the longer the inference time was, but if those frames were from varying viewpoints and at various different angles, the model generally does better. In fact, the authors of 3D R2N2 used up to 30 images and came to a similar conclusion [1]. After selecting the frames we would resize, pad, and segment the images to the appropriate requirements for the model.

V. Implementation Details

We created a simple web application to show our progress end-to-end. It starts with a video input and eventually serves the user an object file (.obj) which they can import into any kind of graphics engine or a game engine for further processing.

The architecture of the application includes the following major components -

a. **Flask Web Server**
We decided to go with the Flask web server because it’s a popular choice for prototyping ideas as well as because of its ease of development. The frontend of the web application is served using the Flask web server which is hosted on an Amazon EC2 instance. Once the web server receives a user input video, it processes and runs inference on the selected frames and sends an object file as a response.

b. **Frame selection from the input video**
Since the 3D-R2N2 model requires images as input, it was necessary to either ask the user for multiple individual images or just a simple video which can then be further processed to extract frames. We went with the latter choice since it involves a fewer number of clicks.

To obtain the images, a popular library called “ffmpeg” has been used. It allows us to extract random frames from the video. Based on the duration of the video, the model receives a varying number of images. We have set the defaults to 1 frame per 2 seconds.

c. **Removing the background of the object in the image**
In addition to the input for the deep learning model being images, the object in the images has to be segmented out and pasted on a white background. This is one of the additional tasks to be undertaken for the successful creation of the 3D object.

It involves making an external request to a well-established API called “remove.bg”. It perfectly segments out a wide variety of objects and worked miraculously for our use case. It can also take care of awkward angles/perspectives of the object in question.

d. **Dynamic input to the deep learning model**
The deep learning framework 3D-R2N2 was hardcoded to receive 3 images as an input. Our experimentation concluded that more images reinforce the model and provide an accurate 3D

representation of the object in the video. As mentioned in step “b”, the varying number of frames depends on the duration of the video.

Besides the number of images, the model also requires the images to be 127x127 in dimensions. So resizing the original frames to the above dimensions is mandatory.

e. Serving the object on the frontend

While we provide a download button for the user to download the generated object, we decided that rendering the object would be convenient for the user. This was achieved by using a popular Javascript 3D library called Three.js. It allowed us to scale and position the 3D object on our canvas as well as animate the object.

We are serving the deep learning model using GPUs which fasten the inference process and provide reliable response times to the user. All of our application’s dependencies are handled by Anaconda which allows us to save our environment and introduce portability.

VI. Shortcomings and challenges

Models

a. DeepVoxel

The project team initially chose DeepVoxels to be our base model. After setting up the environment on our EC2 instance, we found that the model does not generate a 3D model. Since the youtube video that summarized the paper that introduced the model shows a rotation of view around the object, we falsely assumed that it was a 3D model and not a series of images. The model takes images and creates new views in between angles of input views.

b. ATLAS

The ATLAS model was trained on the Scannet dataset and we needed to know how it was formatted so we could format our custom dataset to be compatible with the ATLAS model for transfer learning. However, ATLAS was designed to run off of a tab-separated values (TSV) file that contained metadata for each scene. We were unable to discover how that file was generated and how to replicate that format for our new dataset. Therefore we were unable to apply transfer learning to the final checkpoint of the ATLAS model.

Datasets

a. Generating Dataset for Atlas Model

Initially, we opted to generate data for our own inferences by taking camera frames and calculating the associated pose information for those frames using the phone’s IMU. The pose was initialized at zero at the time of frame capture, and from there, the gyroscope and accelerometer data were used to calculate camera position. Theoretically, due to the high sample rate of the IMU, the process of integrating rotation rates and translational acceleration works great, and accurate pose data can be calculated. Using this method of integration along with fusing the gravity vector measurement and the magnetic heading measurement, we were able to generate very accurate phone rotation data. However, in practice, integrating the accelerometer data twice to calculate the position results in exploding errors which increase unboundedly. Naturally, this is a big issue since the input of the ATLAS model requires accurate 3D pose information of the camera [9]. To resolve this issue, we needed to implement some sort of data fusion that would pull the accelerometer errors to within reason.To do this, we opted to utilize the phone camera to track movement using optical flow and simple feature trackers, such as a Kanade–Lucas–Tomasi feature tracker which just tracks edges and corners [8].

While we were initially able to get the visual odometry system to work, we noticed it failed with small movement and rotations due to the global scale ambiguity problem that is inherent with optical feature tracking. To combat this, we looked into using IMU measurement to calculate a global scale that could be applied to the position estimates generated by the visual odometry system. This new method of tracking position using IMU measurement and optical flow calculations is called visual-inertial odometry, and it is the backbone of SLAM (simultaneous localization and mapping).

Instead of building our own SLAM system, we chose to try and utilize an off the shelf system, namely VINS-Mono [7]. VINS-Mono is a monocular visual-inertial SLAM system that takes in video and IMU readings as input and outputs accurate and robust position and rotation estimates. This system showed extremely promising results for the purposes of generating data for our model, however, we failed to figure out a way to run the system. VINS-Mono runs on ubuntu ROS (robot operating system) and is tailored to take in the EuRoC MAV dataset as input [7][10]. We attempted to convert the VINS-Mono input to take in a live phone video feed as well as a live IMU datastream, however, learning ROS proved to be difficult and we were not able to implement this system in the time allotted.

b. Birbird Dataset Conversion for Atlas [3]

The dataset generally used in tandem with the ATLAS model is associated with indoor house scenes [8]. In our case, we want to model individual objects and reconstruct them to make them easily importable in game engines. For this purpose, we found an individual object instances dataset open-sourced by researchers from Berkeley. The authors of the BigBird dataset have provided a large scale 3D database of object instances.

The database consists of 125 objects each of which has 600 images from 5 different viewpoints. The camera on top, NP5 is the only reference camera for which pose information is provided. In addition to the camera intrinsics and extrinsics, the authors have also provided segmentation masks, merged point clouds, and reconstructed meshes of the objects. To obtain the pose information for the other cameras, relative transformations between all cameras and reference cameras are provided in a calibration file.

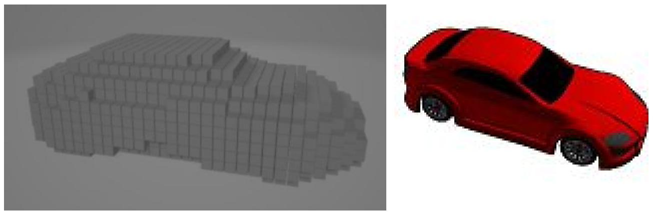
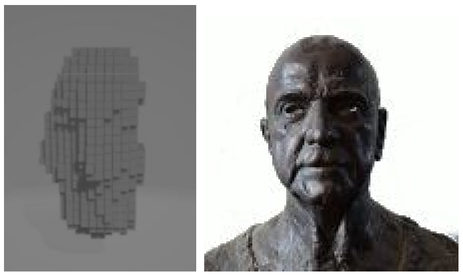
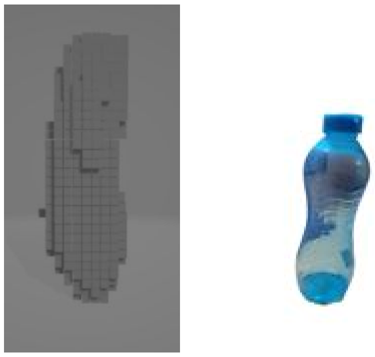
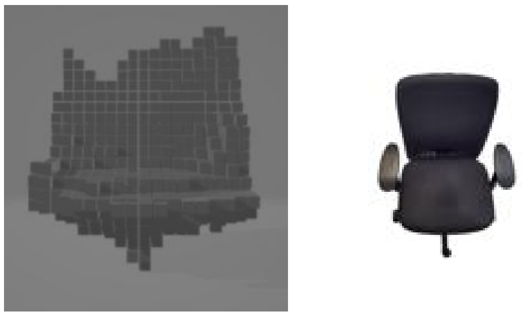
Unlike the Scannet dataset which provides the pose information in text (.txt) files, the BigBird dataset provides it in HDF5 format [3][6]. So the first step was to convert these .h5 files into properly formatted text files as required by ATLAS. We needed to convert them into a text file representing a 4x4 matrix containing the rotation and translation matrices. A python script to handle the generation of pose text files from the given series of h5 files was written.

The rotation matrix was only given for one reference camera. The reference camera is one that views a direct overhead shot of the object. Using transformation matrices given in the calibration.h5 file, we can create the pose matrices for the other four cameras. However, we were not able to successfully convert pose data to the desired format. The reference due to the ambiguity of which keys we should use. The makers of the dataset did not respond to us about how to convert the pose information. This resulted in our team having to abandon the dataset.

c. Scannet Dataset [6]

We contacted the creators of the ScanNet for permission to download their dataset. This dataset was 1.3TB and could not be downloaded to our instance due to space limitations. We could not analyze how the training occurred for all of the scenes due to this limitation. We downloaded the first scene and attempted to train using that one scene. However, that required modifications to metadata files that were unintuitive. This resulted in us reaching a roadblock.

VII. Results

From our results, we can see that the model attempts to generate pieces that are occluded during the inference process. It also appears to struggle with images that have little variation in its shape such as the water bottle. Another thing that we noticed is that the model has a difficult time reconstructing images when the views in the images are too similar. The model appears to need images that cover many viewpoints from different angles.

VIII. Discussion

Conclusion

The first milestone led us in the direction of looking for an idea that combined segmentation and depth. We decided on working on a project involving the 3D reconstruction of objects from 2D videos. Research was done towards calculating pose through the integration of sensor data and calculating depth from TSDFs. Due to unforeseen complications with the ATLAS model, we chose a model that used shape priors instead. A segmentation preprocessing step was added to our models to improve the end result of 3d rendering. Thus, we achieved our goal of generating a 3D model from a video.

Future Scope

In the future, we would like to work more directly with depth maps and augmenting layers in NN models for training. We would like to utilize the custom dataset we generated containing pose data for more accurate 3D reconstruction. While our current model is able to infer rough 3D shapes from a small set of 2D images, our initial goal was to create accurate 3D assets from many images, and methods that involve SLAM and TSDF would most likely provide more accurate results.

References

[1] C. Choy, D. Xu, J. Gwak, K. Chen, S. Savarese, “3D-R2N2: a unified approach for single and multi-view 3D object reconstruction”, Apr. 2016. [Online]. Available: arXiv:1604.00449v1

[2] Sitzmann, Vincent, et al. “DeepVoxels: Learning Persistent 3D Feature Embeddings.” *ArXiv:1812.01024 [Cs]*, Apr. 2019. *arXiv.org*, http://arxiv.org/abs/1812.01024.

[3] A. Singh, J. Sha, K. S. Narayan, T. Achim and P. Abbeel, "BigBIRD: A large-scale 3D database of object instances," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, pp. 509-516, doi: 10.1109/ICRA.2014.6906903.

[4] Y. Xiang, R. Mottaghi, and S. Savarese, “Beyond PASCAL: A benchmark for 3D object detection in the wild,” *IEEE Winter Conference on Applications of Computer Vision*, 2014.

[5] Google-Research-Datasets, “google-research-datasets/Objectron,” *GitHub*. [Online]. Available: https://github.com/google-research-datasets/Objectron/. [Accessed: 28-Nov-2020].

[6] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Niessner, “ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.*arXiv.org,* arXiv:1702.04405

[7] T. Qin, P. Li and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," in *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004-1020, Aug. 2018, doi: 10.1109/TRO.2018.2853729.

[8] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. International Joint Conference on Artificial Intelligence, pages 674-679, 1981.

[9] Murez, Zak, et al. “Atlas: End-to-End 3D Scene Reconstruction from Posed Images.” *ArXiv:2003.10432 [Cs]*, Aug. 2020. *arXiv.org*, http://arxiv.org/abs/2003.10432.

[10] Burri, Michael & Nikolic, Janosch & Gohl, Pascal & Schneider, Thomas & Rehder, Joern & Omari, Sammy & Achtelik, Markus & Siegwart, Roland. (2016). The EuRoC micro aerial vehicle datasets. The International Journal of Robotics Research. 35. 10.1177/0278364915620033.

[11] Everingham, M., Van~Gool, L., Williams, C., Winn, J., & Zisserman, A.. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.