3D Shape Retrieval and Sketched Scene Reconstruction using Triplet Embedding

Student Name: M. J. Redmond

Supervisor Name: F. Li

Submitted as part of the degree of MEng Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

Context/Background

The 3D Modelling industry has a potential bottleneck in the translation between concept art and fully realised 3D scenes. We investigate whether this process could be, at least partially, automated using modern machine learning techniques.

Aims

To create a machine learning system that can process 2D sketches and retrieve similar-looking 3D models from a database, and to design a user interface which implements this system in order to re-construct 3D scenes from a user's sketches.

Method

A simple classifier and an encoder using Triplet Embedding are employed to map the domain space of models and sketches onto each other. We then use Python's Tkinter and VPython packages to implement the UI.

Results

Our system produces, on average, comparable results to other systems. We identify several improvements that could be made with more advanced machine learning techniques, as well as better datasets.

Conclusions

We successfully prove that, even with simple machine learning systems, there is the possibility for automation within the 2D to 3D modelling process that thousands of 3D artists go through every day. This is an unexplored source of potential optimisation for the industry at large.

Keywords — Encoding, Feature Map, Domain Space, Sketch-based, Shape Retrieval

I INTRODUCTION

In 2018, the global 3D mapping and modelling industry was valued at $2.8 billion and is forecasted to rise to $6.5 billion by 2023 (Markets & Markets 2018). It is essential for a large range of industries, as everything from 3D maps to Visual Effects require 3D models in order to function. Typically 3D models are created manually, where concept artists draw the abstract design for the model or scene that they need to construct. This "concept art" can be from one or many viewports and can be coloured, or simple line drawings. It will then be "translated" by a 3D artist constructing the required 3D models.

Machine Learning, especially where it automates repetitive activity, has been a rapidly developing field in the subject of Computer Science. It concerns creating algorithms that can learn and adapt, similar to the human mind. This extends employing data structures that simulate the neurons inside of a human brain, called Neural Networks. These algorithms are very effective at

automating mundane and repetitive tasks but struggle when asked to be creative by themselves or when they do not have enough reference data to make decisions.

Convolutional Neural Networks (CNN) were first popularised by Kunihiko Fukushima (Fukushima 1980) and attempt to simulate the human visual cortex. Each layer in the network moves several neurons (often called filters or kernels) across the height and width of an n-dimensional tensor. A neuron will sum all the multiplications from its neighbourhood allowing it to produce a new tensor by collecting values at every position in the tensor. Many neurons work together to produce tensors that act as feature maps of the previous layer. With an image as an input, these feature maps can detect edges, colour changes and curves. Several of these layers stacked together can detect entire shapes, by mapping them to a one-dimensional tensor of all features that an image contains or partially contains. It is these features that we can use to classify objects or compare how similar objects from different domains are.

User Interfaces are, by far, the most common way for creating 3D content, with popular software being 3DS Max, Maya or the free and open source, Blender3D. These software applications allow the user to create and manipulate the various vertices, edges and faces that make up 3D models, as well as handling texture mapping and animation rigging. Other potential methods for creating content include 3D scanning (Levoy et al. 2000), photogrammetry (Wolf & Dewitt 2000) and digital sculpting (Perry & Frisken 2001).

We believe that the process of creating 3D models from 2D reference images is one that can be automated. There are many machine learning techniques that create a function mapping one domain to another; in this case, an image to a 3D model, and said function should perform well enough to be used practically. This would be saving both time and money within the industry.

In order to prove this, we shall develop a simple Machine Learning network that can retrieve 3D models from a database using 2D images and will work for simple line drawings. This network will be able to train on any dataset that contains a classified set of sketches and models. This network will then be implemented into a piece of software, with a User Interface that allows an artist to draw sketches and recreate a 3D scene through just sketches alone to demonstrate that the process of automation can be combined with already established methods.

Our aims are as follows:

1. Basic Objectives
    - Find and alter two datasets for training the Machine Learning system.
    - Design and create specific loaders for those datasets that correctly map the domain space.
2. Intermediate Objectives
    - Design and train a CNN that classifies 2D sketches.
    - Design and train a CNN that can embed said sketches, in order to compare image to model similarities.
3. Advanced Objectives
    - Create a UI that will allow the user to draw different objects and have their models retrieved.
    - Create a system that will re-integrate the retrieved models into the scene.

Over the course of this project, we have completed all of these aims, to varying degrees.

The basic objectives are simple tasks to prepare pre-existing datasets for training our networks. We will use these benchmarks to compare performance as well as find methods for using these datasets for our purposes. As there is lack of specific research to do with this problem, the datasets that exist are rare and sparse. They do not contain data that would be useful for practical applications, nor is the data they do carry evenly distributed within their classes. However, through altering these datasets and writing programs to manage and convert the data into the required format, we will be able to test and evaluate our system, to a much higher standard.

The intermediate objectives are focused around the creation of the machine learning system that can map the sketches to their respective models in the database. We have split the system into two networks. One carries out the simple function of classifying objects, to reduce processing time. The other focuses on the core feature embedding for image similarity. As the second network will need to run through many models in the database, classifying a sketch first decreases the set of models that will need to be processed.

The advanced objectives are to implement the network into the UI and show its practical applications. These are advanced because they have the potential to require very complex UX design, Computer Graphics and 3D Geometric calculations in order to be completed in their entirety. We will unlikely be able to reach the high standards of current 3D modelling software on the market. However, as long a basic system is in place, it can show its potential as well as room for growth.

II RELATED WORK

Classifying and embedding hand-drawn sketches has been a significant area of research in Computer Graphics for over a decade (Kara & Stahovich 2005). In (Wang et al. 2015), the authors provided one of the most comprehensive and best performing methods for Sketch-Based Shape Retrieval using machine learning. Their method involved using a Siamese network and a loss function that compared both with-domain and cross-domain similarities. This method outperformed every state-of-the-art method at the time and is the basis for our machine learning implementation in this project. A Siamese network is a network architecture created by Bromley (Bromley et al. 1994) and popularised for use in similarity metrics by Chopra et al. (Chopra et al. 2005). It uses two convolution neural networks with shared parameters in order to map two input domains into the same output encoding. As well as this, the work implements the common technique in shape retrieval of view generation into order to query the model database, though with only two views per model.

Our evolution on (Wang et al. 2015) is to implement Triplet Encoding (Hoffer & Ailon 2015), which uses a triplet loss function between three encoding networks. This is to gauge both positive and negative comparisons in data, and to compare between similar and dissimilar sketches and models. A potential improvement on this technique would be to use Lifted Structured Feature Embedding which has already been used to compare similarity between views of real-life objects (Song et al. 2015). It could be possible to make it work for the views of a 3D model. Implementing ”dropout” techniques to increase input noise and produce more reliable results is also a tried and tested method, that has long existed with image classification techniques (Krizhevsky et al. 2012).

The datasets we will implement are SHREC13’ (Li et al. 2013) which is based on the SBSR dataset (Eitz et al. 2012) and the further improved SHREC14’ (Li et al. 2014) which contains

more evenly distributed models. Both datasets contain a series of sketches and CAD models that are divided up into 90 classes collected from user queries. As there is an uneven distribution between the number of models per class in SHREC13', SHREC14' aims to increase the number of models using other datasets, increasing model variety in the process.

Fine Pose Estimation has been shown as a valid method for re-integrating 3D models into a scene (Lim et al. 2014). This method has also been proven to work with the IKEA model database, as well as CAD models (Lim et al. 2013). It creates a model that is then trained on a parts-based system, taking pre-determined sections of the model and identifying their locations within an input image, thus allowing for efficient location and translation. Each model has to be segmented, with each segment having a weighted "importance". The more advanced work here could help to make our results more accurate but we choose to take a simpler approach, as the sketches will be abstract and not perfectly resemble the view of a 3D model.

Further development on using cross-domain neural networks has been done, by using recognised 3D object features, rather than projecting 3D models into 2D images (Zhu et al. 2016). This can be further improved by implementing a pyramid hierarchical structure in order to combat the problems of view variance and deformation between sketches and models.

The practical use of a sketch-based system using machine learning has already been implemented in a simple context, namely designing clothes onto 3D human models (De Paoli & Singh 2015). A lot of research has gone into generative models, ones that can take sketches and create 3D models from them. These use variations of 3D-GAN network; a new type of network structure that uses volumetric convolutional layers in order to handle 3D data as opposed to 2D and uses the same Generator and Discriminator training system (Wu et al. 2016). This has the ability to generate high quality 3D models from a pre-determined domain space. The 3D VAE-GAN architecture, on the other hand, uses an Encoder for 2D images and a Decoder into 3D images, in order to extend the architecture for use in 2D images. If you can discern the normals, depth and silhouette of an object, you can use a 3D-VAE-GAN to generate it (Wu et al. 2017).

Other studies focus on the sketching side of sketch-based model generation, aligning the outlines of different sketches into order to calculate the proposed 3D model (Li et al. 2016), or segmenting and identifying areas of architectural designs to create a 3D model of a building (Yin et al. 2008).

In order to create large-scale object detection and model re-integration datasets, the technique of mapping a set of features and using a distance metric to compare similarities between them in order to find the most similar looking results is a potential method for shape retrieval. This was shown in the creation of the dataset ObjectNet3D (Xiang et al. 2016). While this dataset is used for real-life shape retrieval, we aim to implement its core concept into our sketch-based shape retrieval system.

Using human participants to evaluate the performance of large-scale image retrieval systems has also been explored (Eitz et al. 2011). Here they created a benchmark using a user study of 28 participants and calculating their correlation coefficients. While this is for image-based retrieval and not 3D models, we can use this study as a basis for implementing and dissecting a user study of our own.

III SOLUTION

A Structure Overview

Our structure to solve this is to adapt Wang's original work (Wang et al. 2015). We will have a Comparator network that uses a Siamese Network with Triplet Loss. When a sketch is given to the system, it will iterate over the models in the database, calculating the similarity between the sketch and each model. Whatever has the highest similarity is retrieved from the database and returned. In order to reduce the amount of computation time, we have implemented a Classifier network, that will classify the sketch, selecting a subsection of the database to iterate over.

This system will then be implemented into a software allowing the user to sketch out a scene and have it be recreated as a 3D model. The basic premise allows for the software to be initialised with the background of a cube-shaped room. The user will be prompted to draw one or several objects onto the screen, each on separate into their own layer. Then each layer will be ran through the network to produce the model that is the closest in similarity to each sketch. The model's position and rotation will be calculated given the sketch's position and similarity. Finally, the model of each layer will be re-integrated into a model of the room, which will then be displayed on a 3D viewer.

B Datasets

Two different datasets were used for experimentation. Each of these datasets contained a series of sketches each with their own class, and a series of 3D models. In order for our network to work for practical purposes, we used both the image and the model classifications provided. Once classified, the Comparator would only have to iterate over models within a certain class.

B.1 SHREC13

This dataset was created for the Shape Retrieval Competition 2013 (Li et al. 2013). It addressed concerns about the lack of proportionality between classes in the previously used SBSR dataset and has been frequently used for sketch-based shape retrieval research (Wang et al. 2015). SHREC13 originally had 7200 sketches and 1258 models. However this has been significantly reduced for practical purposes as several of the classes, such as "Skyscraper" or "Airplane", would not be used for room reconstruction. Our system could easily be expanded to use all 90 classes, but for efficiency and practicality, we have reduced this number to 17 classes.

However, this dataset does have the disadvantage of a lack of 3D models per class, with some classes containing no more than 5 models.

B.2 SHREC14

To address these concerns, SHREC14 was created, which added significantly more 3D models to the classes raising the model count to 8987 (Li et al. 2014). However, due to source of these models originating from outside the SBSR dataset, there is a greater variety in models and the models are oriented in different ways. This makes the objective of shape retrieval considerably more difficult and is what the Classifier system hopes to alleviate.

B.3 Other

Other potential datasets we could have used include the SBSR dataset which we elected not use, due to the SHREC13 and SHREC14 datasets both being major improvements to them. As well as this, the benchmark created for Yoon et al.'s work, wherein they reduce the PBR dataset to 13 classes and 250 sketches, could be integrated into the project (Yoon et al. 2010). However, this benchmark had considerably fewer sketches and models and we felt that it would not produce as reliable results with our system.

B.4 3D Database Creation

Another consideration is the processing of 3D models. The 3D models need to be converted into a format that can act as an input for the network. Previous papers have compared the use of standardized 3D vectors against the new "Voxel" technique (Wu et al. 2016) (Wu et al. 2017), however in (Eitz et al. 2010), the authors used "views" to render a 3D model into several 2D images. This way the Comparator will have the same input for the 2D sketches and the 3D models.

We automated this process by using the free, open source 3D modeller, Blender3D. Blender3D allows for scripting in Python, therefore we can code the several stage process to turn each model into a database of 2D images. Every model is imported into a Blender3D "scene", which contains a camera that acts as a viewport for rendering the model, and a light that gives the model its shading. Using Blender3D's own library of methods, the origin point of the imported model is translated to the median point of its vertices, with the models position being set to $\vec{0}$. This means that if the camera is pointed towards the centre and rotated around the image, the model is always in view, with its vertices evenly distributed around the image.

As well as this, the "shader" for the rendering needed to be altered from it's default diffuse setting. As the authors suggest in (Wang et al. 2015), the networks are most effective when the sketches are similar to the views. Therefore, some post-processing was added to the shader to give each model an outline, with their surfaces being rendered white with light shading. This gave each view the appearance of a line drawing, while their shape could still be easily identified.

C Network Structure

Each network was carefully designed for its purpose. We took into account the processing power each one would require, the amount of times one would need to use it in a practical sense and how accurate it would need to be in order to remain as effective. These factors determined how many features each network would generate as well as how many layers the data would need to be processed through, accounting for time and memory usage.

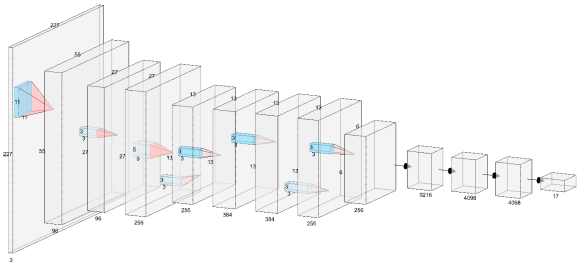

Figure 1: Classifier Diagram

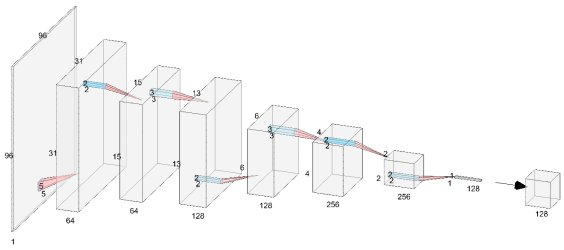

Figure 2: Comparator Diagram

C.1 Classifier

We based this network on the AlexNet architecture which won the ImageNet Large Scale Visual Recognition Challenge 2012. It contains several convolutional layers, each one followed by a pooling layer, ending with a series of a fully connected layers. Its output is a 1x17 tensor with each element describing the weight towards the same number of classes. In order to find which class the sketch belongs to, you have to select the highest value. We found that a large improvement to this architecture is to remove normalisation and add Dropout Layers. These give each connection a probability of 60% to being zeroed when forward-passing through the network, making the process noisy and allowing the network to train around the noise and produce more accurate results. This is especially helpful given the abstract nature of the problem. This network takes a tensor input of 1x227x227, a normalised greyscale image.

C.2 Comparator

The Comparator has less layers and a larger output than its counterpart, but still has convolutional layers followed by pooling layers. This structure and its dimensions are taken directly from Hoffer's paper as they are optimum for triplet loss in that case. This network produces a 128-dimensional vector that acts as our feature map.

This network takes a tensor input of 1x96x96, which is also normalised and greyscale. The sketches and model views are all black and white so, in order to increase performance, we decided on 1 channel images instead of 3.

We also trained a network to take this feature map and produce a 3-dimensional tensor that can be act as a point in 3D space. This allows us to visualise the domain space.

D Network Implementation

The networks were designed, trained and tested using PyTorch. PyTorch is a free and open source machine learning library for Python, based on the Torch library. It contains all the systems required to create a fully functioning Machine Learning system, as well as several preset datasets, optimisers and loss functions, that are commonly used in the industry. It is robust and easy to implement, which allows us to experiment and quickly find the right techniques required to solve the problem. This has an advantage over PyTorch's alternative, TensorFlow, which is slightly more complex and much harder to change and re-implement architectures. As well as this, PyTorch is becoming more prevalent in modern day machine learning programs. A

more commonly used library legitimises our demonstration of the applicability of automation in modern day 3D modelling and mapping.

Both networks are stored on a python file. Each training system, imports the networks and required libraries, defines the dataloaders for supplying the networks with data during training. Next they initialise the network, optimiser and loss function and iterate the network through the dataset, tracking and plotting the loss as it progresses. It will repeat and run through the training data several times, each one known as an "Epoch". The dataset is randomly shuffled every time. The python files are designed to take advantage of PyTorch's built in compatibility with NVidia's CUDA framework. This allows the python files to easily use parallel processing when training the network through multiple "workers" to fetch data and run it through the data. With the CUDA framework, training can be processed much faster, and uses the Graphical Processing Unit (GPU) instead of the core Central Processing Unit (CPU) of a system. Several GPUs can run in parallel to allow use of more workers.

D.1 Dataloaders

In order for networks to be trained, they need to receive the data from datasets through dataloaders. While the Classifier's data was simple sketches with their labels being their class, the Siamese Comparator which used triplet embedding was much more complicated.

The Network requires three images while training: an anchor, a positive and a negative. However, the relationship between these three images changes depending on what we are trying to teach the network. The network needs to learn three metrics in order to successfully map the sketch domain to the model domain. First, it needs to learn which models are different from each other via class. In this scenario for each anchor image, the positive returned would be a random image of the *same class*, and the negative would be a random image of a *different class*. Second, the network needs to learn which models are different from each other, with the positive returning a random image that is a view of the *same model* as the anchor and the negative returning a random view of a *different model but within the same class*. This one is key to mapping the similarity between models. Finally, throughout both of these, the network needs to retain the domain space of sketches, ensuring that all metrics are allowing for the abstract nature of sketches. In this scenario, each anchor image is a sketch, with its positive being a random view from the same class and its negative being a random view from a different class.

Each random image is produced using a hash function, meaning the results are entirely reproducible while avoiding "collisions" where the same triplet is processed more than once. Each image in the dataset is used as an anchor 20 times, giving a large dataset for training and allowing the networks to have a greater degree of accuracy.

D.2 Training

In order to train the networks, each one was given an optimiser and a loss function. An optimiser is a class in PyTorch that automatically updates the weights of network to minimise the output of the loss function. Both networks were given the Stochastic Gradient Descent (SGD) optimiser. Every iteration it uses stochastic methods (ones that randomly determine new solutions) to estimate the negative gradient of the system at that time from a random section of the whole network's weights. This helps find the local minimum, by scaling steps by the current gradient and optimising the network. This allows the network to be trained much faster, as the entire

network is not being taken into account in every iteration, with the trade-off being that it takes a while for the loss to converge.

The Classifier uses the Cross-Entropy loss function, a logarithmic function that increases the further the classification is to the correct value. This is a regular loss function for classifiers and allows the optimizer to quickly reduce the loss at the initial stage of training. The Comparator, on the other hand, uses the Triplet loss function. This function takes the exponentials of the difference in magnitudes between the network results, with loss tending towards zero as distance between the anchor and the positive decreases and the distance between the anchor and the negative increases.

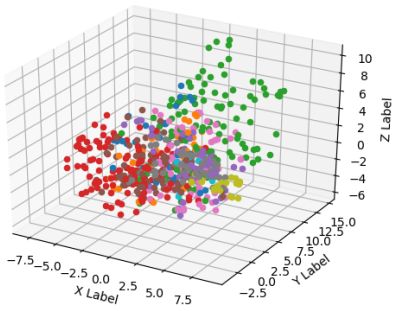

Figure 3: Plotting the generated points for each model view in the dataset. The colour of each point indicates its class.

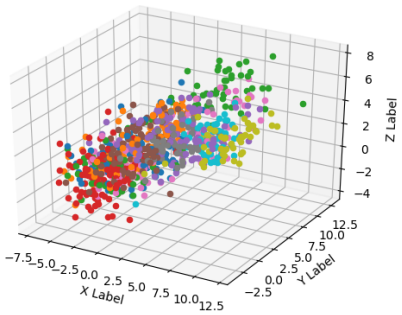

Figure 4: Plotting the generated points for each sketch in the dataset. The classes are the same as in Figure 3.

Figure 3 and Figure 4 show the distribution of points generated by the model from the dataset. As you can see, the various classes have been organised into their own areas of 3-dimensional space, however they have been drawn apart internally to detect reduce distance between similar-looking views. As well as this, the sketches and views of the models occupy similar areas of 3D space, which allows the Comparator to detect similarities between sketches and models.

E Software Development

In order to implement these networks in a practical setting, a piece of software needed to be developed. We decided to keep the software as rudimentary as possible, so there was enough to prove our methods but give room for further development and growth.

E.1 UI Overview

The UI was designed using TKinter. TKinter is one of the oldest Python GUI libraries, with a simple, singular action loop and a wide variety of classes to construct and properly format a User Interface layout. The UI consists of a main drawing window; with the background of the room you insert the models into. There is a list of layers on the side which can be added to or taken away from, allowing the user complete control of the separation of their sketches. Each layer has its own buttons in order to classify and retrieve its models, with buttons at the top of the list to perform group-wide actions.

We could have used further machine learning to take a single sketch and differentiate between

objects within the sketch, much like the YOLO network (Redmon et al. 2016) or the experiments conducted on the ObjectNet3D dataset (Xiang et al. 2016). However, as this was not core to our project nor completing our aims, we opted for giving the user more control, with the trade-off that human error can occur and produce strange results. To minimise these, the user interface tries to be very clear how each sketch is being fed into the network, highlighting bounding boxes and declaring how much each sketch will be rescaled for the network.

E.2 Image Processing

Several layers of image processing had to be taken in order to give good results when classifying and comparing the sketches given by the user. Each image is given a border of white space, so that it is more comparable to the training sketches. As well as this the images are given further borders to ensure their shape is square, so that there is no warping of proportions, altering the input and increasing the likelihood of false results.

E.3 3D Rendering

The 3D rendering of the system was implemented using VPython. In a more advanced system, or one integrated with already existing software, a more hard-coded approach with OpenGL or WebGL would be more effective and optimised. However, creating a 3D renderer wasn't the purpose of our project, so we decided to opt for a simpler method that would demonstrate the effectiveness of the system.

VPython does not have built in support for 3D meshes, nor does have any methods to import OFF files which is the format the 3D models were in. We opted to designing our own import function, that would read the file and translate it into a list of triangles of positions. We would then use this to make many instances of VPython's Triangle class, compounding them together for each object so they could be translated and rotated. Our initial issue was that OFF files do not store normal data, and the algorithms for creating normals are very complex. Therefore, this was automated with another Blender3D script, that would iterate through the models, calculating their normals and converting it into the OBJ using Blender3D's own library. Then instead of using the OFF files, the OBJ files would be loaded using another hand-written import function, taking in normals as well.

There are several areas of potential further development. The possibility of integrating textures, detecting which texture the model has, using UV coordinates to map said texture to the model. However, these were not necessary to meet our objectives.

F Scene Reconstruction

Once the images were selected, and the models retrieved, the last stage of our project was to reconstruct the models into the scene. In order to do this, we had to take information from the sketch and use it to create a position, rotation and scale in the scene.

F.1 Position

When translating the sketches position on the screen into a 3D position in the scene, we opted to using simple 3D line intersection. As there are 5 walls in the scene that the object could possibly

be on (the Top, Bottom, Left, Right and Back walls), we first found the midpoint of the sketch, determining which portion of the screen it was on, with each portion assigned to a different wall. Once the wall was selected, its position was then calculated as the intersection between a line originating at the camera and passing through the object, and a plane with the wall's location and normal. To get the direction of the line, we used the screen space location of the midpoint as well as the field of view of the camera to create a unit vector in the direction through the object's midpoint. Using linear algebra to solve the intersection, this resulted with a 3-dimensional vector for the positions.

F.2 Scale

While scaling the object seemed simple at first, seeing as we could get the size of sketch on, a problem arose in the matter of perspective. In order to remain the same size on the screen, objects which are further away would have to be scaled up and vice versa. Therefore, we measured the distance of the object from the camera and used that value in a formula to linearly scale it, factoring in the original size of the import model to reduce distortions due to the database. The resulting size was then multiplied by the normal vector of the surface it was located on and added to the final position vector, so the model didn't clip through the walls. The models were all scaled uniformly to prevent distortion.

F.3 Rotation

Matching the rotation ended up being the most difficult part of reconstruction. Without Fine Pose Estimation (Lim et al. 2014), a process that would require perfect sketches of the models, there was no way to accurately estimate the orientation of the object without a separate system. Therefore, we opted to using the Identifier to compare the similarities between the sketch and different orientations of the same model. Whichever orientation it was closest to, it was given. While imperfect, this method did at least attempt to solve the problem, leaving the opportunity to further improve it.

With all three transformations calculated, the objects could be re-integrated into the system and displayed through VPython's WebGL client.

G Testing

In order to fully evaluate our project, it was necessary to design tests that would cover the effectiveness of the system. We split this into three parts to cover various aspects of the system, from the Networks themselves to the UI we had implemented.

Most of our results will be presented using the SHREC13 dataset, specifically when it comes Human Subjective training, but performance and precision-recall will use both datasets to compare.

G.1 Network Effectiveness

A simple accuracy test was sufficient for the Classifier over the dataset's test set of images. For the Comparator, a precision-recall curve needed to be implemented. A precision-recall curve is used to measure the precision (the likelihood that the Comparator will return correct results)

against the recall (the number of results the Comparator has been asked to return). We measured this by writing a program that would take each sketch and iterate through all models, rather than a subsection using the Classifier. It would then calculate which models are closest to the sketch in the feature map for each step of the recall.

G.2 Subjective Quality

In order to judge the subjective quality of how similar the retrieved models are to their sketches, we elicited the help of human subjects. We created an anonymous questionnaire using Google Forms, which contained pairs of sketches and the models they had retrieved. Subjects then went through each pair of images and rated how similar they were to each other. Each participant was given the option of explaining their choice so that we would get some insight into how they made their decisions. The images were evenly distributed among classes and were randomly selected to ensure that both positive and negative results were included.

IV RESULTS

Here are the results presented from the variety of tests.

A Experimental Settings

All testing was done on a single GPU system with an Nvidia Geforce GTX 1080 Graphics Processing Unit and an Intel i7 8700K Central Processor Unit. The GPU has 8GB of VRAM and operating at a frequency of 1607 MHz while the CPU has 6 cores with clockspeeds of 3.7GHz. The system has 32GB of Random Access Memory and was operating on 64-bit Windows 10. This setup is considered mid-tier in professional terms of processing power and affordability, meaning that the system could be used and trained by a user with any budget but also showing that the system could be scaled up to train even faster. Due to this, the evaluation will focus on the results of the training, not the efficiency. However, to show the scalability, the training process was optimised using PyTorch's library to allow for data to be loaded using "workers" where the data would be assigned threads. These threads would load data while other data was being processed and the data would be loaded in "batches" where a thread would process several elements of data at a time with the network designed around a higher functionality.

An "epoch" is full iteration of the dataset. A network is usually trained using many epochs to iterate through the full dataset. The Classifier was trained using 500 epochs, due to the size of the dataset being relatively low compared to the original Alexnet problem. This took 1 hour for both datasets. The Comparator training ran for a much lower 15 epochs on each dataset, taking 3 hours to complete for SHREC'13 and 12 hours to complete for SHREC'14. It took much longer for SHREC'14 due to the sheer size of the model dataset, which was over 4 times the number of models. These training times were found by monitoring the average loss of intervals of 1000 forward passes of the network over time. All losses plateaued somewhere between 0.02 and 0.05, except for training the Comparator on the SHREC'14 dataset which plateaued around 0.2. This could be due to the large variety of the models in the dataset, as many have askewed sizes and rotations compared to the rest of the models, causing outliers and less similarities within the views. Further revisions should be made to the dataset and the method in which views were generated in order to improve these results.

B Classifier Accuracy

The Classifier was run through the testing set of sketches. This would test how accurate it was in correctly classifying the objects. Table 1 show the breakdown of the accuracy for each class that was tested.

	Accuracy	
Class	**SHREC'13**	**SHREC'14**
bed	100%	63.33%
beer-mug	100%	73.33%
bench	100%	50%
bicycle	100%	83.33%
cabinet	100%	30.00%
chair	96%	76.66%
couch	100%	50%
door	100%	70%
floor_lamp	100%	70%
ladder	100%	93.33%
laptop	73.33%	90%
piano	98%	46.66%
potted_plant	100%	80%
table	98%	70%
tablelamp	98%	66.66%
tv	98%	60%
vase	100%	56.66%

Table 1: CLASS BREAKDOWN OF CLASSIFICATION ACCURACY

The total accuracy of the Classifier was 98.55% on the SHREC'13 dataset and 66.86% on the SHREC'14.

C Comparator Accuracy

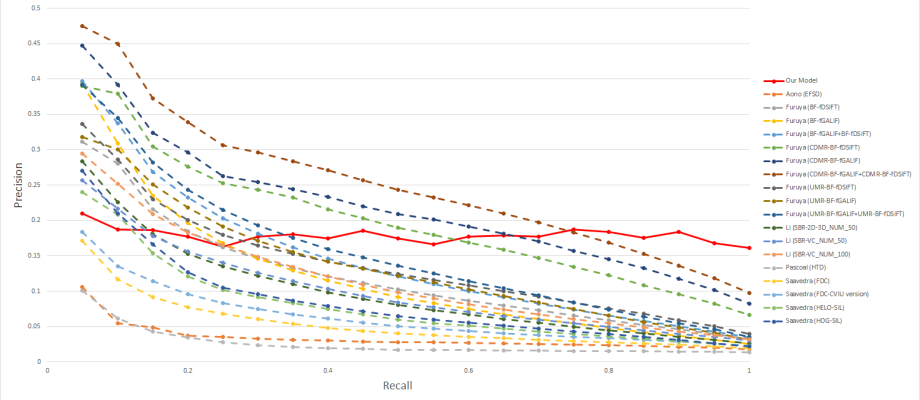

Figure 5: The plotted Precision-Recall curve against the SHREC'13 Benchmark.

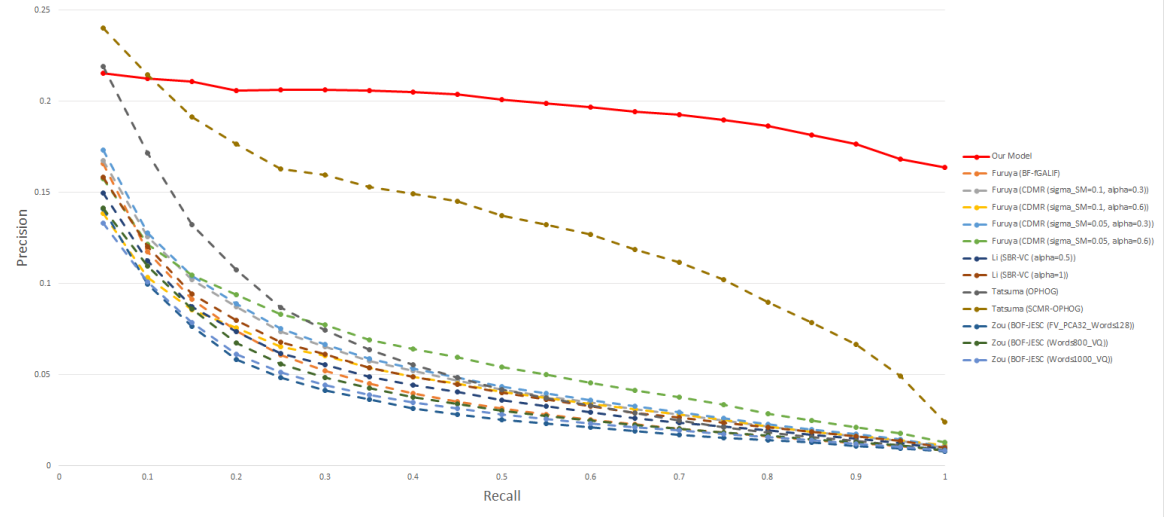

Figure 6: The plotted Precision-Recall curve against the SHREC'14 Benchmark.

To measure the accuracy of the Comparator, we plotted a Precision-Recall curve for the network. If you retrieve the K models from the database from a sketch where $1 <= K <= n$, n being the total number of models in the database, then the Precision is the percentage of models in K that are the same class of the sketch and the Recall is the percentage of the same class of models that have been retrieved out of all models that are the same class (Powers 2011).

The curve was plotted by going through the testing set of sketches, retrieving models by using the Comparator. Models were retrieved using a distance function between feature maps of the models and the sketch. Feature maps that are closer to the sketch's are retrieved first. Each model retrieved for each sketch, plotted a precision-recall value onto a scatter graph. Figures 5 and 6 show the calculated curve based on that scatter graph, taking averages at intervals of 0.05 recall with a moving average of 3 applied to smooth it out. This is then plotted against the results given by each dataset's benchmark. The solid red line represents our model's curve in both.

D Subjective Accuracy

These results were taken from an online questionnaire that was distributed randomly on the internet. 20 people took the survey. It contained a series of pairs of images, each pair had a 2D sketch and the 3D model it retrieved through the system. For each pair of images, the subject was asked to rate how similar the two images were, on a scale of 1 to 9. Only a subset of 8 classes where shown to the user so as to keep the question count low and not affect the results.

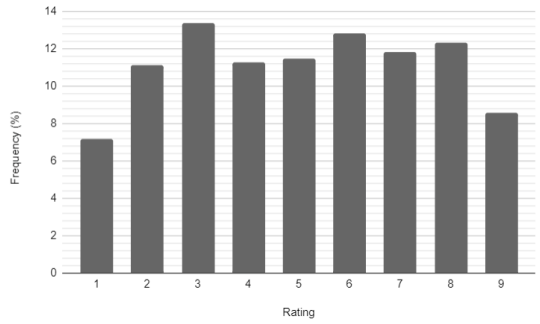

Figure 7: The frequency of ratings give for the entire questionnaire.

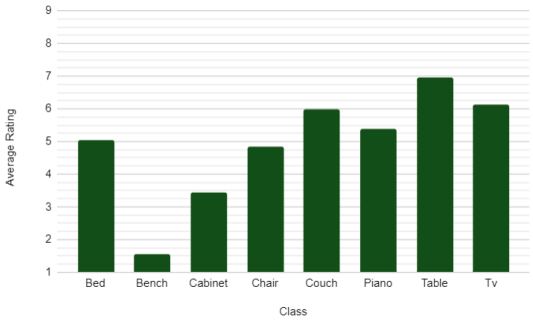

Figure 8: The per-class averages of ratings in the questionnaire.

Figure 7 shows the frequency of ratings over all the results in percentages. Each rating was tallied and then divided by the total number of ratings to reach this number. Figure 8 shows the average rating for each class of retrievals that were shown to the subject.

With each question, the subjects were given an optional text field to give their reasoning as to how they came to the conclusion of their rating. Many commented on being able to recognise the class of an object but noticed the inherent differences on the actual retrieved model and ended up giving more neutral scores. Some noticing differences between subsets of classes, some of the images underneath the ”piano” class would be technically classed as a keyboard in real life. Some commented on the angles obscuring the differences between the two images. However, when there was a false positive given in classification, many were able to identify that something was wrong and used that in their reasoning.

V EVALUATION

There are many components to this project, each of which should be evaluated in turn. There is the general performance of the networks implemented, and whether or not they demonstrate an effective machine learning algorithm. Next there is the robustness of the software and how it functions. Finally, there is how these elements come together to prove the viability of this system being used in a practical sense.

A Network Effectiveness

As can be seen from Table 1, the Classifier performed considerably better on the SHREC’13 Dataset. This would be due to the sketches being more evenly distributed in the SHREC’14 Dataset, meaning that sketches were removed from certain classes. Using the same reduced subset of classes and training for the same number of iterations produces less accurate results. However, the fact that it is more evenly distributed means that classes that were proven difficult to classify in the SHREC’13 dataset. For example, the Laptop class can actually have improved results but the classes of Bench, Cabinet and Couch are considerably less accurate, likely due to their similarities to one another. Overall, it is shown that even with difficult benchmarks, the Classifier has the capacity to perform accurately.

Comparing Figure 5 against Figure 6 gives us an interesting evaluation of the Comparator. Both curves have average performance within their respective benchmarks though comparable to

higher methods, when you consider how difficult the task has been made by reducing the dataset. However, the drop-off of the curve is considerable smaller than most other systems, practically non-existent. This shows even while it may not be as precise as other methods, in earlier recalls, the system is much more precise when all members of a class have been recalled. This can demonstrate a considerably more consistent mapping between classes overall, which proves the benefit of the Classifier, allowing us to narrow down the database to small subsets based on the classification.

Another interesting result is that the curves differ very little between datasets, despite being trained for the same amount of time and one having a considerably higher loss when plateaued. This is a testament to the scalability of the network, allowing it to give as precise results as when the benchmark is considerably harder, with much more varied models. While it would be preferable for the network to perform better on an easier problem, further improvements made to the system in order to do so, will undoubtedly carry over to harder benchmarks.

Even then, the results still underperform Wang's work (Wang et al. 2015), showing that perhaps a simpler loss function and Siamese network can produce more precise retrievals. However, the fact that there is room for improvement in our system, despite it still functioning and performing to a good standard, shows that there is room for further research and development. If this system can still be implemented practically, then in the future it can be improved for further professional use.

B Subjective Quality

As shown in Figure 7, the overall results from the questionnaire were varied. Some retrievals were given very low scores, and some given very high. The aim of the questionnaire was to qualitatively measure how well the system was at producing similar-looking models, while there were certainly less low scores (1-2) than average-high scores (6-8), it is not by a wide enough margin to make any conclusions. The abstract nature of the sketches, combined with very nature of subjective quality, has likely caused this variance. Each different subject is going to have a different frame of reference for how similar the sketches could be as well as an understanding that models are not being generated but rather retrieved. In order to gain further understanding, a smaller survey pool should be tested, potentially a group of 3D modelling artists, who are informed on the nature of the project, as they would then be able to compare the performance of the system against their own experiences with the 2D to 3D process.

Breaking down the average results in Figure 8 gives us further understanding. There were certainly classes that gained much higher ratings than others. "Table", "TV" and "Couch" retrievals received good results, likely due to their classes being simple and very recognisable shapes, allowing there to be little possible variation between the sketch and the retrieved model. Classes like "Cabinet" and "Chair" received lower ratings due to their more complicated shape and potentially less accurate classification. We consider the "Bench" result an outlier, due to it containing a correct classification, but its sketch also an incredibly abstract drawing, leaving almost no similarity between the sketch and the model that was retrieved.

While the written user feedback was sparse, with a retrieval getting at most 6 comments, the comments that were given, shone a light on how subjects were making their comparisons. There were certainly irregularities in the ways subjects were comparing the two images. The given set of retrievals could have benefited from being more curated, specifically giving only true positives in order to properly judge how effective the Comparator is. Given that the subjects

were able to easily identify the false positives there were no faults in their immediate reasonings of similarity. It is far more likely that the varied results were due to different levels of scrutiny between subjects.

C Software Robustness

The software implemented is robust and does not crash. All of its features perform correctly and once a scene has been drawn out with several objects, it will produce a fully reconstructed 3D scene. The versatility and usability of this software will be demonstrated in the Oral Examination.

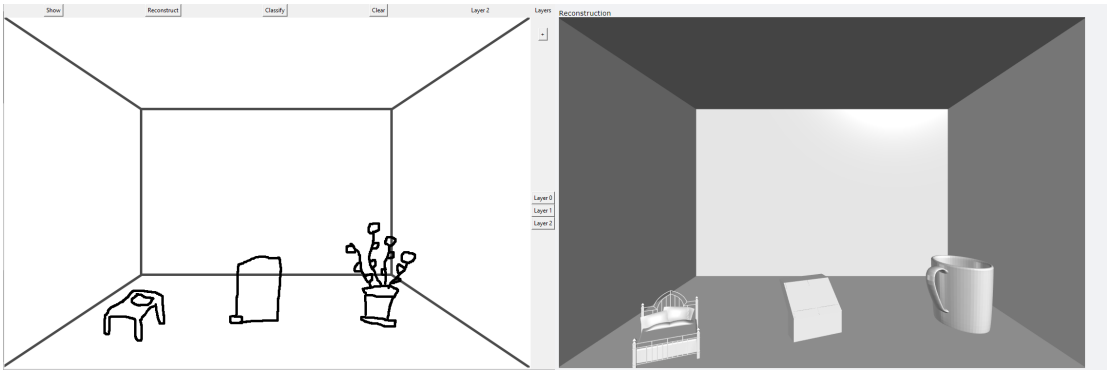

Figure 9: An example reconstructed scene using the software, show both positive and negative results.

D Overall Viability

The largest factor in how viable this system is to be used practically, or even professionally, is its simplicity. You can only draw from one perspective, you can only reconstruct certain classes of models and they will be reconstructed on calculated but predetermined locations. As well as this, the system is not generative, it does not create models from the sketches but rather maps their abstract nature to actual features in a database of models. You cannot take a sketch of several objects as an input; each object must be sketched on separate layers in order for them to be individually reconstructed.

It is for these reasons that we believe this system would not be used professionally in its current state. However, it is still viable to be used practically, and speeds up production on smaller scale projects. The simplicity prevents it for being used on 3D modelling projects that require a large variety of models, from many perspectives that are highly detailed and textured. However, this software would significantly speed up the production of 3D scenes that use prefabricated assets that need to be reorganised in the same setting every time (e.g. a simple puzzle video game with many levels). This way, it would become a separate tool among many in production.

VI CONCLUSIONS

In conclusion, we have successfully proven that machine learning can be used to automate the procedure of turning 2D sketches into 3D scenes. We completed all of our aims for the project and its performance has produced interesting results. While the system is not precise nor varied enough for professional use, it can certainly be both used for smaller-scale projects and be improved in several aspects, increasing its effectiveness.

Thanks to Machine Learning, AI automation has become a huge industry in itself, as it helps reduce costs in multiple areas of production. In 2014, it was reported that 77% of factory workforce for Apple and Samsung's biggest supplier had been replaced with robotics (Wakefield 2016). The use of this technology has the power to streamline processes and reduce the need for manual labour but there is the danger of removing jobs from this industry entirely (Wang & Siau 2019). Our project avoids this issue in two regards. Firstly, this is "weak" AI, such that this AI has been specifically trained for this purpose and to re-purpose it would take time and engineers. The process in creating 3D models is something people are heavily educated on in order to operate in the 3D modelling field. Therefore, to be as versatile as a 3D artist, a computer system is going to need a lot more multipurpose and adaptive AI than this relatively simple system. Secondly, nothing is being generated in this program. It is taking established creations from the 2-D designers as the users and the 3-D designers as the ones who design the models in the database and bridging the gap between them. While generative models are certainly possible and should be explored, they were not necessary to show the applications of AI here.

An AI system is only as good as the data we feed it. As of the current date, there is still much to be desired in publicly available datasets for sketch-based shape retrieval. The human brain's abstract interpretations of shapes are so varied person-to-person that we need much more data in order to map it completely, and successfully test those mappings.

Irregardless of the project's viability, throughout the project we have proved that there are many ways of advancing the system. From using a YOLO approach to detect and separate objects, to using Fine Pose Estimation to have higher accuracy in re-integrating objects. Not only would a few improvements in any of these systems make it much more viable for professional use, but it proves overall that machine learning can be used to automate this process. If Shape Retrieval is already possible, then generative models can be developed. Further research and development into this area of automation could produce a variety of new systems that would greatly increase the efficiency we turn 2D sketches, that are potentially abstract, into fully functional 3D models.

With these first steps, we have proven that automation can be easily integrated into the 3D Mapping and Modelling Industry and there is huge potential for development here that has not been fully explored. In addition, we have the shown the variety of techniques that could be implemented, as well as simple systems that make use some of the industry's most popular frameworks and languages. Improvements and areas for further research have been outlined throughout this project. The potential for future development of robust and generative systems is unbounded.

References

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E. & Shah, R. (1994), Signature verification using a" siamese" time delay neural network, *in* 'Advances in neural information processing systems', pp. 737–744.

Chopra, S., Hadsell, R. & LeCun, Y. (2005), Learning a similarity metric discriminatively, with application to face verification, *in* '2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)', Vol. 1, IEEE, pp. 539–546.

De Paoli, C. & Singh, K. (2015), 'Secondskin: Sketch-based construction of layered 3d models', *ACM Trans. Graph.* **34**(4).

Eitz, M., Hildebrand, K., Boubekeur, T. & Alexa, M. (2010), Sketch-based 3d shape retrieval, *in* 'ACM SIGGRAPH 2010 Talks', pp. 1–1.

Eitz, M., Hildebrand, K., Boubekeur, T. & Alexa, M. (2011), 'Sketch-based image retrieval: Benchmark and bag-of-features descriptors', *IEEE Transactions on Visualization and Computer Graphics* **17**(11), 1624–1636.

Eitz, M., Richter, R., Boubekeur, T., Hildebrand, K. & Alexa, M. (2012), 'Sketch-based shape retrieval', *ACM Trans. Graph. (Proc. SIGGRAPH)* **31**(4), 31:1–31:10.

Fukushima, K. (1980), 'Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position', *Biological cybernetics* **36**(4), 193–202.

Hoffer, E. & Ailon, N. (2015), Deep metric learning using triplet network, *in* 'International Workshop on Similarity-Based Pattern Recognition', Springer, pp. 84–92.

Kara, L. & Stahovich, T. (2005), 'An image-based, trainable symbol recognizer for hand-drawn sketches', *Computers Graphics* **29**, 501–517.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* F. Pereira, C. J. C. Burges, L. Bottou & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 25', Curran Associates, Inc., pp. 1097–1105.

Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J. et al. (2000), The digital michelangelo project: 3d scanning of large statues, *in* 'Proceedings of the 27th annual conference on Computer graphics and interactive techniques', pp. 131–144.

Li, B., Lu, Y., Godil, A., Schreck, T., Aono, M., Johan, H., Saavedra, J. M. & Tashiro, S. (2013), SHREC'13 Track: Large Scale Sketch-Based 3D Shape Retrieval, *in* U. Castellani, T. Schreck, S. Biasotti, I. Pratikakis, A. Godil & R. Veltkamp, eds, 'Eurographics Workshop on 3D Object Retrieval', The Eurographics Association.

Li, B., Lu, Y., Li, C., Godil, A., Schreck, T., Aono, M., Chen, Q., Chowdhury, N. K., Fang, B., Furuya, T. et al. (2014), Shrec'14 track: Large scale comprehensive 3d shape retrieval, *in* 'Eurographics Workshop on 3D Object Retrieval', Vol. 2, .

Li, C., Lee, H., Zhang, D. & Jiang, H. (2016), 'Sketch-based 3d modeling by aligning outlines of an image', *Journal of Computational Design and Engineering* **3**(3), 286–294.

Lim, J. J., Khosla, A. & Torralba, A. (2014), Fpm: Fine pose parts-based model with 3d cad models, *in* 'European conference on computer vision', Springer, pp. 478–493.

Lim, J., Pirsiavash, H. & Torralba, A. (2013), Parsing ikea objects: Fine pose estimation, pp. 2992–2999.

Markets & Markets (2018), '3d mapping and modeling market worth $6.5 billion by 2023'.
URL: *https://www.marketsandmarkets.com/PressReleases/3d-mapping.asp*

Perry, R. N. & Frisken, S. F. (2001), Kizamu: A system for sculpting digital characters, *in* 'Proceedings of the 28th annual conference on Computer graphics and interactive techniques', pp. 47–56.

Powers, D. M. (2011), 'Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation'.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788.

Song, H. O., Xiang, Y., Jegelka, S. & Savarese, S. (2015), 'Deep metric learning via lifted structured feature embedding', *CoRR* **abs/1511.06452**.

Wakefield, J. (2016), 'Foxconn replaces 60,000 factory workers with robots.'.
URL: *https://www.bbc.co.uk/news/technology-36376966*

Wang, F., Kang, L. & Li, Y. (2015), Sketch-based 3d shape retrieval using convolutional neural networks, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1875–1883.

Wang, W. & Siau, K. (2019), 'Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda', *Journal of Database Management (JDM)* **30**(1), 61–79.

Wolf, P. R. & Dewitt, B. A. (2000), *Elements of photogrammetry: with applications in GIS*, Vol. 3, McGraw-Hill New York.

Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, B. & Tenenbaum, J. (2017), Marrnet: 3d shape reconstruction via 2.5 d sketches, *in* 'Advances in neural information processing systems', pp. 540–550.

Wu, J., Zhang, C., Xue, T., Freeman, B. & Tenenbaum, J. (2016), Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling, *in* 'Advances in neural information processing systems', pp. 82–90.

Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., Mottaghi, R., Guibas, L. & Savarese, S. (2016), Objectnet3d: A large scale database for 3d object recognition, *in* 'European Conference Computer Vision (ECCV)'.

Yin, X., Wonka, P. & Razdan, A. (2008), 'Generating 3d building models from architectural drawings: A survey', *IEEE computer graphics and applications* **29**(1), 20–30.

Yoon, S. M., Scherer, M., Schreck, T. & Kuijper, A. (2010), Sketch-based 3d model retrieval using diffusion tensor fields of suggestive contours, *in* 'Proceedings of the 18th ACM international conference on Multimedia', pp. 193–200.

Zhu, F., Xie, J. & Fang, Y. (2016), Learning cross-domain neural networks for sketch-based 3d shape retrieval, *in* 'Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence', AAAI'16, AAAI Press, p. 3683–3689.