



Senior Design Project

“Bumpster”

Final Report

Supervisor: İbrahim Körpeoğlu

Jury Members: Bülent Özgüç
Uğur Güdükbay

Innovation Expert: Armağan Yavuz

Project Website: www.bumpster.me

Group Members: Duygu Durmuş
Gülsüm Güdükbay
Doğukan Yiğit Polat
Muhammed Çavuşoğlu
Bora Bardük

May 3, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

Introduction	3
System Overview	4
Bumpster Algorithmic Design	5
Map Generation System	5
Socket Programming	5
Final Architecture and Design	6
Subsystem Decomposition	7
Presentation Layer	7
Application Logic Layer	8
Network and Data Management Layer	8
Subsystem Services	9
Client	9
User Interface Subsystem	9
Server	10
Deep Learning Subsystem	10
Deep Learning Preparation Subsystem	11
Networking Subsystem	12
Data Management Subsystem	12
Controller	13
User Authentication Control Subsystem	13
Impact of Engineering Solutions	14
Global Impact	14
Economic Impact	14
Environmental Impact	15
Societal Impact	15
Engineering Solutions and Contemporary Issues	15
Tools and Techniques Used	15
Libraries	15
Keras	15
Tensorflow	16
SDWebImage 3.8.1	16
Frameworks	16
ARKit	16
CUDA	16
APIs	16
Blender Python API	17
SceneKit	17
Internet Resources	18
Blender Stack Exchange	18
Usage of bisi for Neural Network	18
Software, Tools and Services Used	18
Amazon Elastic Compute Cloud(EC2)	18
AWS Lambda	19
AWS Cognito	19
Amazon S3	19
AWS Mobile Hub	19
Xcode	19
Swift	20
Usage of Blender for Training Data	20
Rendering Maps	21

Normal Map	21
Metalness Map	21
Roughness Map	21
Ambient Map	21
Albedo Map	21
Rendering Scene	21
Normal Map	21
Metalness Map	21
Roughness Map	22
Ambient Map	22
Albedo Map	22
User Manual	23
Installing Instructions	23
Home Screen (Scene View)	24
New Photo Selection View	25
Photo Taking View	26
Photo Edit View	27
Upload Progress View	28
Share (Export) View	31
Class Diagrams	32
References	32

1. Introduction

There are several types of mapping targeting to different aspects of material. Bump mapping is one of these techniques used in computer graphics which enables simulating bumps and wrinkles on the surface of a model. It is possible to obtain the appearance of textured surface by simply using RGB or grayscale values to define dark parts as deepened and bright parts as heightened areas [1]. The shading effect and tiny details on the surface creates an illusion of depth which is useful for graphic designers and game artists to get realistic surfaces of their models.

The existing systems based on texturing software such as CrazyBump are good at exporting different kinds of maps from photographs or images [2]. Getting good results is possible, but the accuracy is not high. The map types that will be generated include bump, metalness, roughness, gloss or ambient maps. The results can be customized with photoshop features of these texturing software programs, but it also takes time to get realistic textured surface of a model. In addition, existing texture software programs are not available in mobile platforms so reaching wide range of customers and accessing benefits of these programs are impossible. Therefore, there is a need for creating bump maps in different platforms with higher accuracy which also reduces the necessity of additional customization.

Bumpster generates bump and physically based rendering maps using a specifically trained neural network. It enables to identify the topological properties of the surface, as well as the material properties using a single photograph. Our product is used in the server to make the required operations and communicate with the front-end applications on various platforms. Bumpster provides rather large benefits for the industry, as the previous solution to this kind of topographical analysis required expensive methods which require personnel with expertise, expensive equipment, and time [3].

Since there are no other tools that extract physically based rendering maps with one or more photos using a neural network, this application will be beneficial in terms of accuracy and usability. After the extraction of the user selected mapping, the user obtains the resulting map and fit it onto the original photo to see the material properties or bump to create a realistic

image. Then, the artists can tweak the chrominance, roughness and other properties of the final product and save it for later usage.

In this report, our aim is to provide an introduction and system overview of our system. Firstly, final algorithmic, architecture and design are described. Then, impact of engineering solutions and engineering solutions and contemporary issues are given. Afterwards, tools and techniques used in our system are described in details. It is followed by user manual and class diagrams. Therefore, the clarification of functionalities in our software is provided.

2. System Overview

There are several tools for different mapping types based on different aspects of material. These tools mostly target to the game developers and computer graphic designers and they are expensive. Simulating realistic bumps and wrinkles on the surface of a mode enables generating realistic scenes in the games. Since these tools are mostly used at workplace by specific job areas, only desktop is supported as a platform. Thus, these tools are not flexible in terms of customer range, scope of applicability areas and platform support. They also need topological analysis requiring time and expensive methods, expertise, equipment and they do not have share options which prevent any global impact.

There are also free mobile applications for interior design such as Marshall's application for trying out new wall paint colors on your own walls. However, these kind of applications are not accurate enough. They can only target to specific customers of a brand. The mobile apps are based on augmented reality and they are not capable of generating maps and 3D models.. Therefore, these tools are not realistic enough and scope of applicability areas and customer range are restricted. They do not have share options which prevent any global impact.

Bumpster promotes 3D surface modeling and simulation which is powered by augmented reality and neural network. It serves freely wide range of customers by combining the professional side of texture software with better accuracy with the mobile application. The companies benefit from Bumpster can return profit as they do not pay high prices to professional tools. It has an easy to learn user interface. The user just uploads photograph(s) to the application to obtain a 3D model with realistic rendering of the interaction of light with the plane surface in the most accurate way to simulate bumps, wrinkles, metal and rough

areas. These models can be shared and viewed in augmented reality and a feed based on user's request. Since users can also view other shared models in this feed, users can follow different and popular textures and models around the world. Professional tools based on texturing software only serve to texture need of game artists and developers and mobile apps only target to specific customers of a brand. However, our application targets to a wide range of customers including interior, exterior and landscape designers, generic and independent customers, hobbyists, computer graphic designers which also extends scope of applicability areas of Bumpster. It serves many areas such as art, game scenes, interior, exterior and exterior design and to the wide range of customers. Wide range of customers sharing different models also have many engineering impacts. Since it is an iOS application, it is portable unlike other applications. In addition, the output of our application is supported with rendering maps using a specifically trained neural network which provides a better accuracy than professional tools with less topological analysis required. Therefore, it is more reliable than most of professional tools in terms of accuracy and precision.

3. Bumpster Internal Design

3.1. Machine Learning Based Generation of PBR Maps

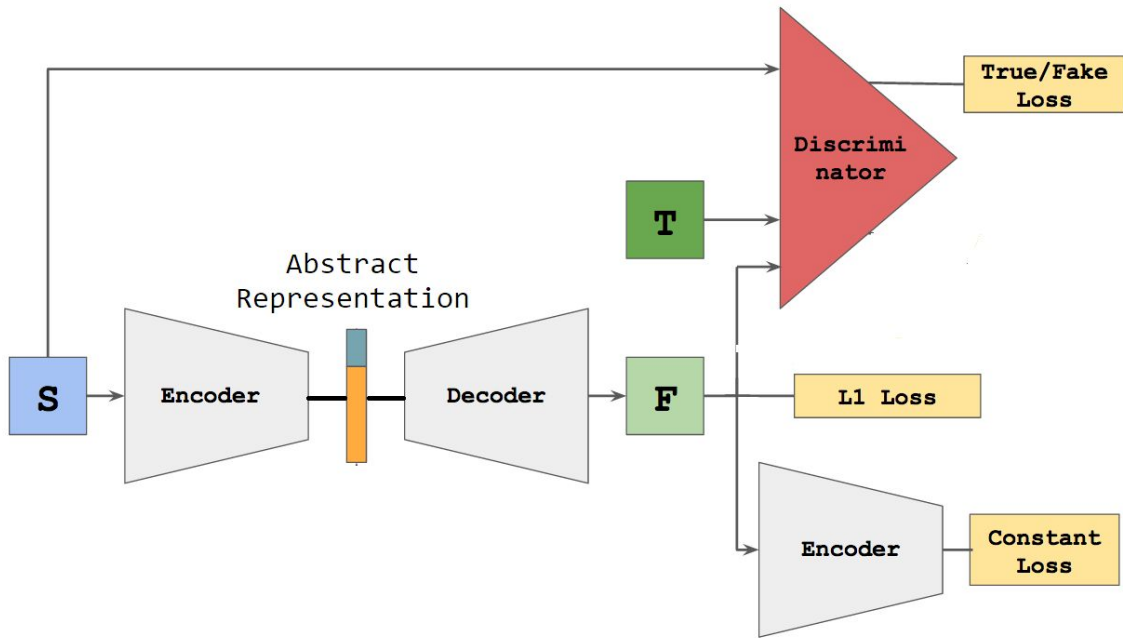
Bumpster uses a deep learning based image-to-image predictor to generate PBR maps. Deep learning models are basically multidimensional regressors. Their very high number (usually in millions) of trainable parameters enable effective detection of complex nonlinear correlations between datasets. Training process is based on a special algorithm called back-propagation which propagates the error (loss) back to the preceding layer to calculate the gradient vector of loss function and updates the trainable parameters in the negative direction of error gradient (error minimization). Loss function is a representation of prediction fault on the training set.

With the rising popularity of deep learning models, convolutional neural networks (CNN) became very effective and popular in modern image analysis/processing applications. CNNs introduced superior performance compared to any other conventional machine learning methods in image classification. However, CNNs usage scope is not limited only to classification applications. Popularity of Generative Adversarial Networks (GAN) brought

excitement to the possibility of using CNNs in generative models. GANs consist of two main parts: Generator and Discriminator. They both are regular neural networks but only difference is that the generator's output is generally a very high dimensional vector while discriminator output just provides a binary prediction regarding the quality of generator output. Discriminator and generator compete throughout the training process. Discriminator receives loss when it fails to recognize fake generator output from the original ground truth. On the other hand, generator receives loss from correct discriminator predictions; with other words, when it fails to fool the discriminator, it gets punished. They incrementally improve each others performances and generator turns out generating accurate data which discriminator (thus, people too) fails to differentiate from truth. GANs that consist of CNNs are called DCGANs (Deep Convolutional Generative Adversarial Networks) and are intensely used in signal generation applications. For example, generator of a DCGAN which performs handwritten digit generation takes 10 inputs representing which digit to be generated and returns a 2 dimensional pixel array (an image) of the chosen handwritten digit. The model we chose to perform the training of Bumpster models is a special kind of DCGAN. Its generator input is not a simple vector that consists of a few decision parameters but also an image just like its output. In order to get something meaningful out of its huge input, generator consists of an "encoder" (like discriminator) which determines an abstract representation of output that it has to generate. This abstract representation is then converted to the actual output by the "decoder" of our generator. Discriminator is not much different than regular DCGANs.

3.1.1. Training

We implemented a slightly modified version of the model described in "Image-to-Image Translation with Conditional Adversarial Networks" paper by Phillip Isola, Jun-Yan Zhu, Tinghui Zhou and Alexei A. Efros; which is also known as "pix2pix". Our Python implementation of the model uses Keras running on Tensorflow. Our training system runs Tensorflow on CUDA framework so we use NVIDIA's CUDA enabled GPUs to perform our training. Below is a figure that shows the dataflow of our DCGAN implementation, where S denotes the input image, T denotes the ground truth and F denotes the "Fake" generated output. L1 Loss is used to introduce stronger consistency with the actual ground truth and True/Fake Loss is the Discriminator loss explained above.



3.1.2. Generation

Despite training phase taking hours and even days, generation is done in seconds once we have the proper weights. We simply save trained weights for each model and load them onto a freshly initialized Generator, feed our inputs to it and use the generated output directly. Discriminator is not involved in generation phase, as expected.

4. Final Architecture and Design

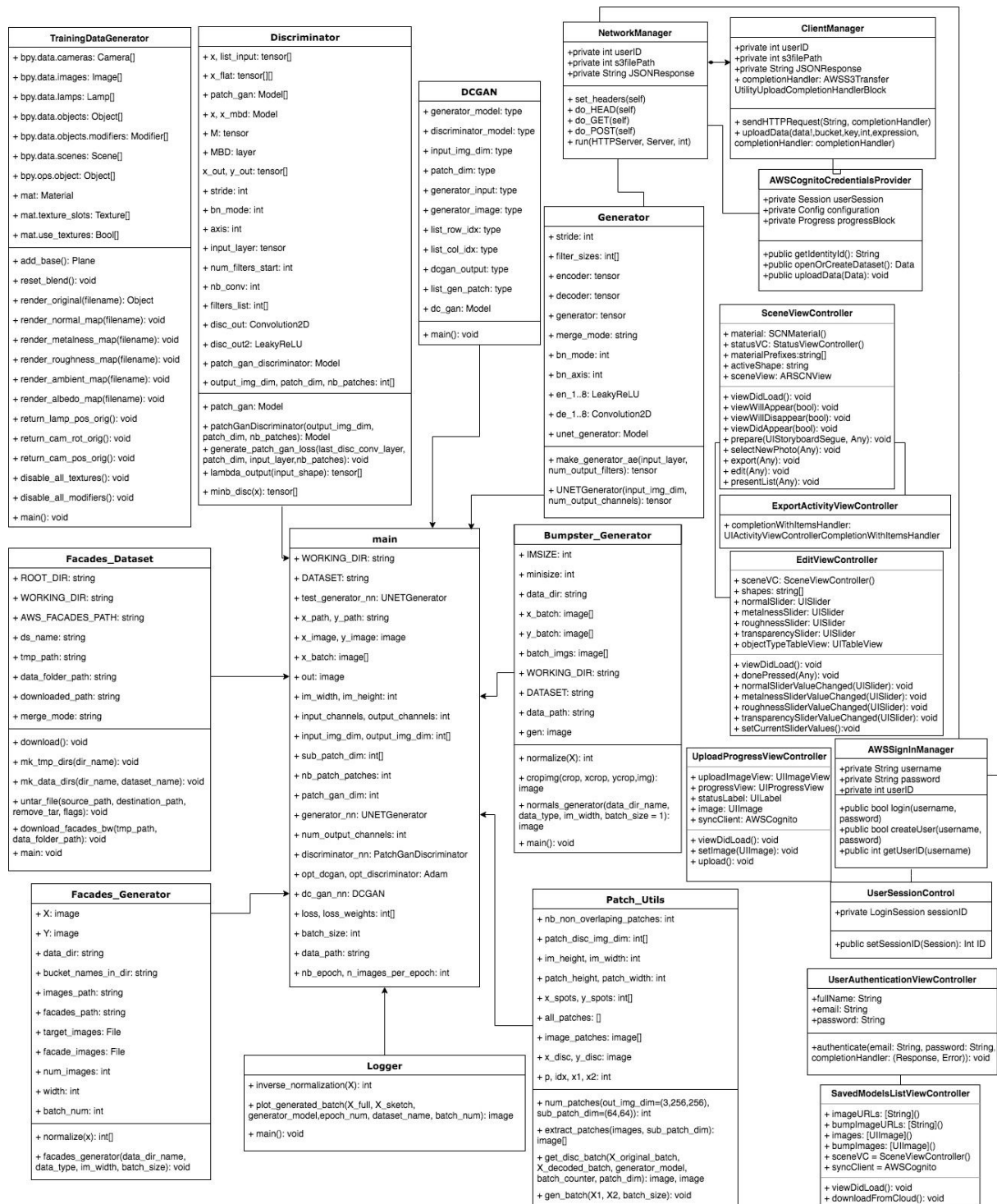


Figure 1: The final uml diagram of the project

4.1. Subsystem Decomposition

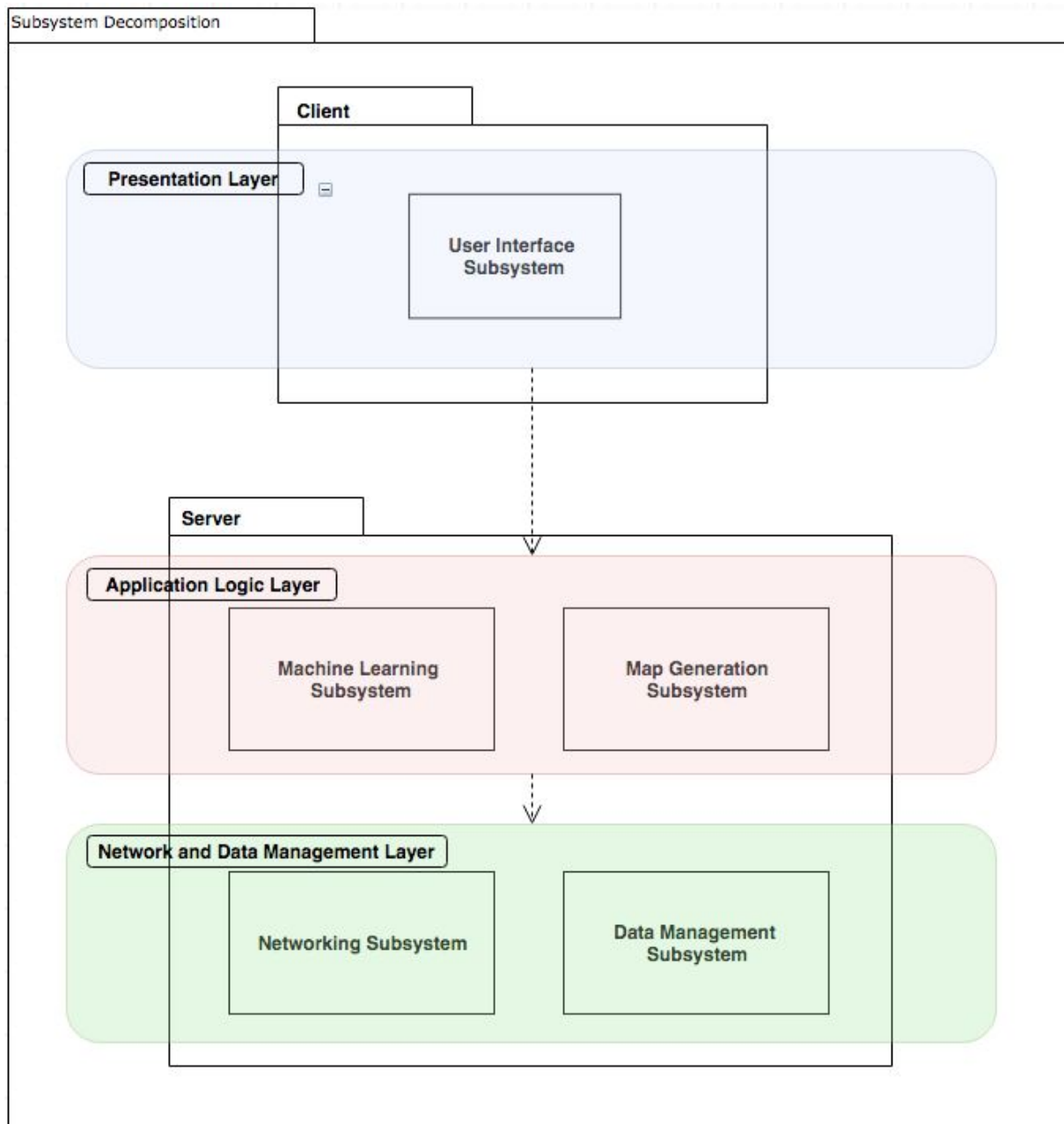


Figure 2 : Subsystem Decomposition

4.1.1. Presentation Layer

The presentation layer contains user interface subsystem. It is the layer which responds to the request of users. Hence, presentation layer is the layer which interacts with the users. This layer is also a client since it requests services from machine learning subsystem and map generation subsystem found in application logic layer.

4.1.2. Application Logic Layer

The application logic layer consists of machine learning subsystem and map generation subsystem. Map generation subsystem benefits from machine learning subsystem for generating several types of maps. Therefore placing these subsystems into the same layer is logical. Both of subsystems provide services to the presentation layer so application logic layer can also be seen as a server. Also, this layer uses network subsystem and data management subsystem found in the network and data management layer.

4.1.3. Network and Data Management Layer

Network and data management layer consists of networking subsystem related to issues such as user account management and client-server communication and data management subsystem related to manager and model classes for insertion, editing and deletion of users' models. This layer is also a part of server because it provides these services explained above to clients.

4.2. Subsystem Services

4.2.1. Client

4.2.1.1. User Interface Subsystem

User interface subsystem will provide user a set of convenient and easy-to-use tools for pre-processing the input data and post-processing the output data. It is placed in presentation layer to interact with users. It displays available data to the users. In order to make the system less complex, this subsystem is separated from other subsystems. User interface provides a connection between computer graphics, image processing tools and graphic designers. This subsystem will ensure that an easy and understandable user interface for user-friendliness. Hence, users can edit generated maps and view them from different angles easily.

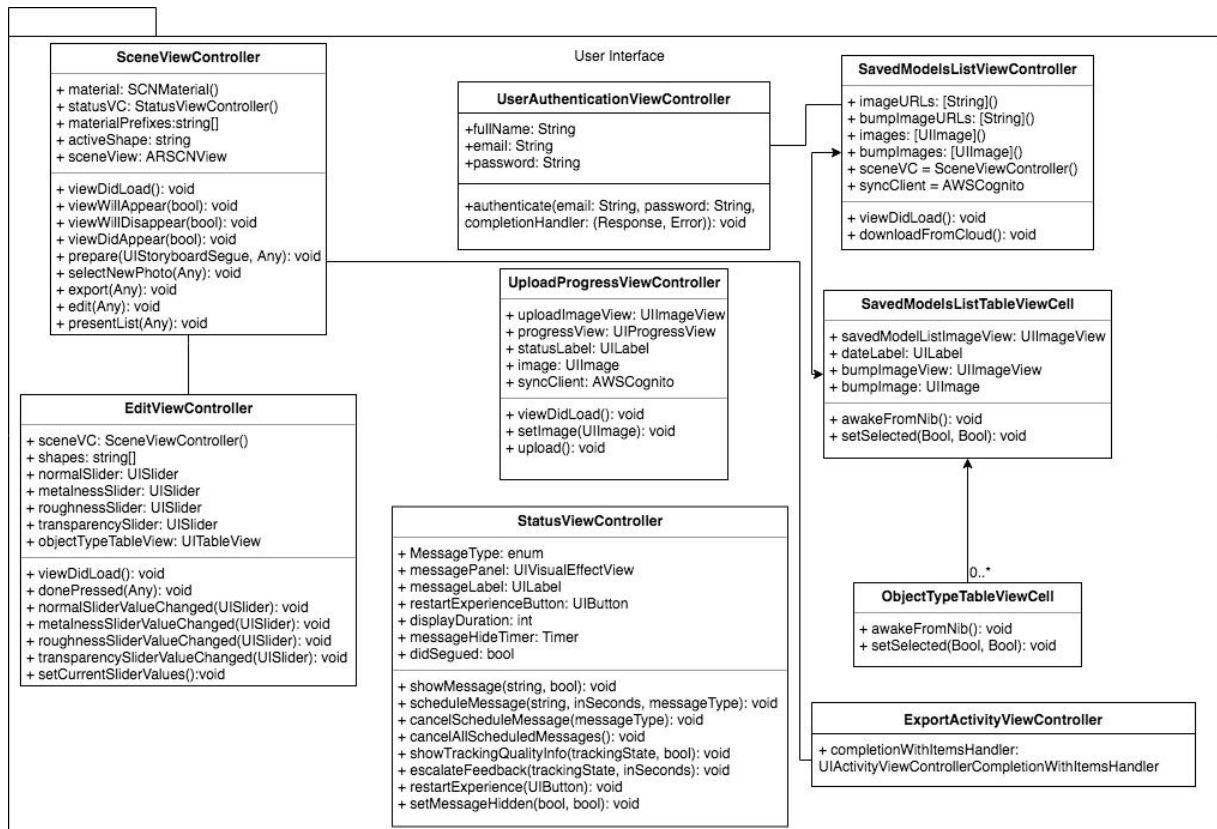


Figure 3 : The uml diagram for user interface

4.2.2. Server

4.2.2.1. Deep Learning Subsystem

This subsystem will not run regularly together with other subsystems and will be used to train new models or tweak the existing models to power the map generation subsystem. This subsystem is separated from other subsystems to avoid complex system structure. Hence, for the sake of separation of concerns, it is better to separate machine learning subsystem.

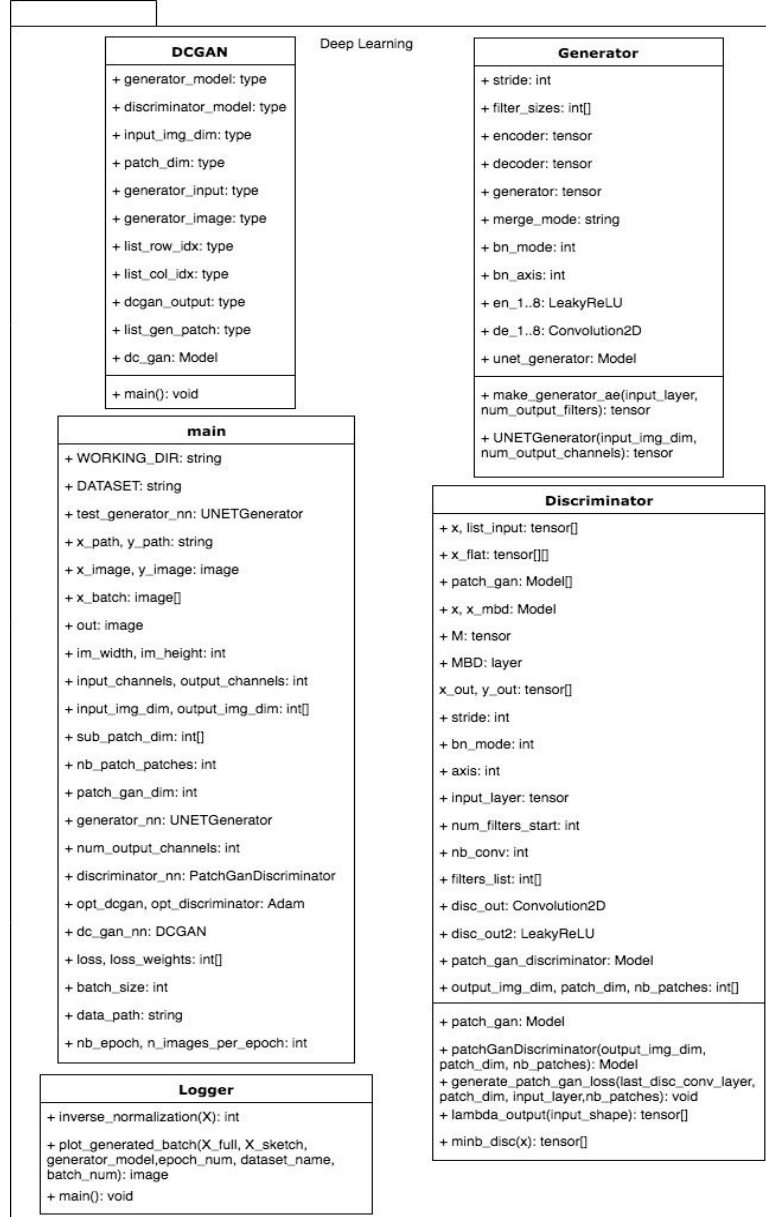


Figure 4: The uml diagram for deep learning

4.2.2.2. Deep Learning Preparation Subsystem

Deep Learning Preparation subsystem is able to wrap the input and output maps of the model to objects and files for usage in Deep Learning Subsystem and in the User Interface Subsystem. The subsystem itself is also complex, so it is logical to consider map generation subsystem as a separate subsystem.

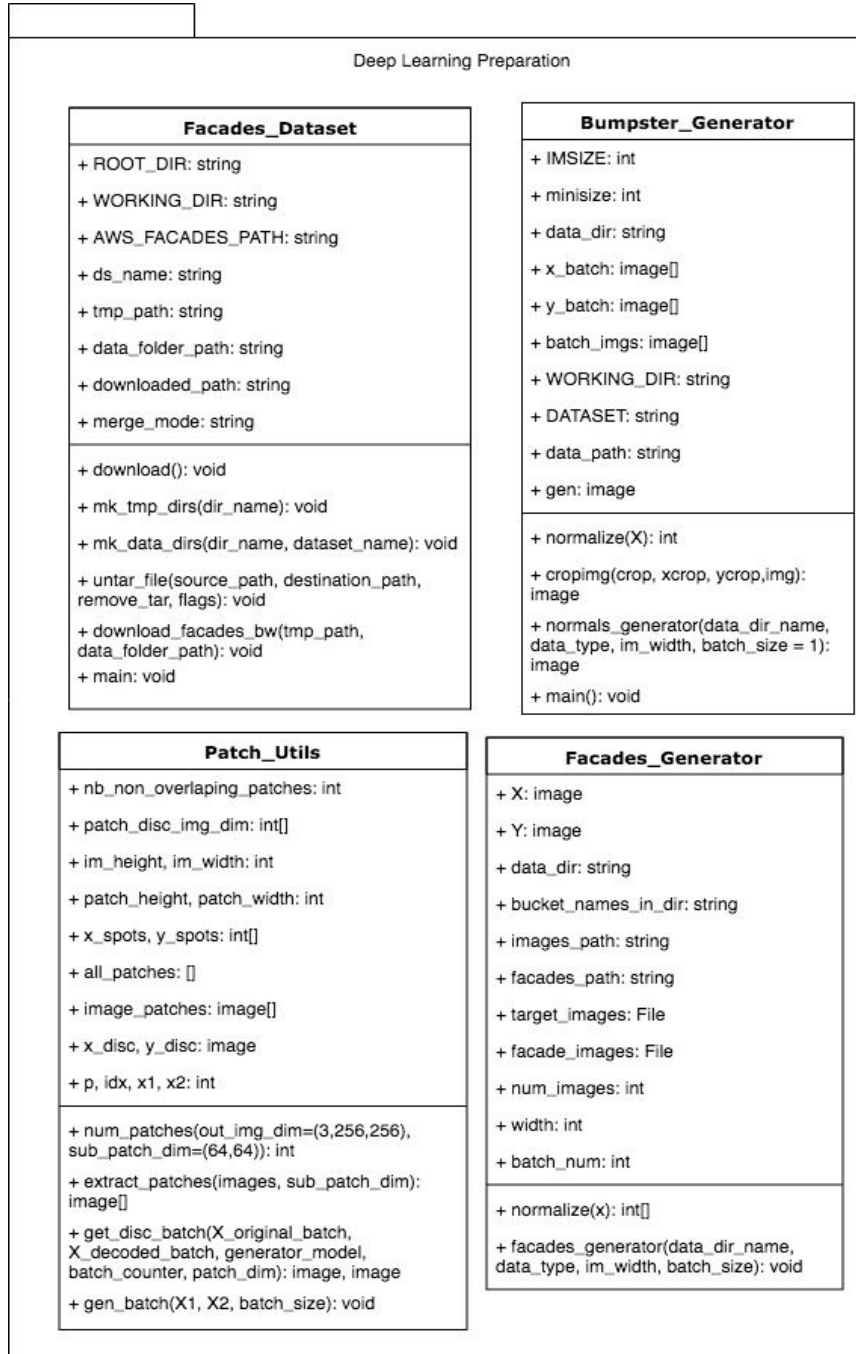


Figure 5: The uml diagram for deep learning preparation

4.2.2.3. Networking Subsystem

This subsystem will handle user account management and client-server communication related issues. The user will send their input photo(s) of a surface and the photo(s) will be sent to the server for map generation. The server will then output the generated map(s) to the user. The access to the network is done via HTTP requests such as GET and POST.

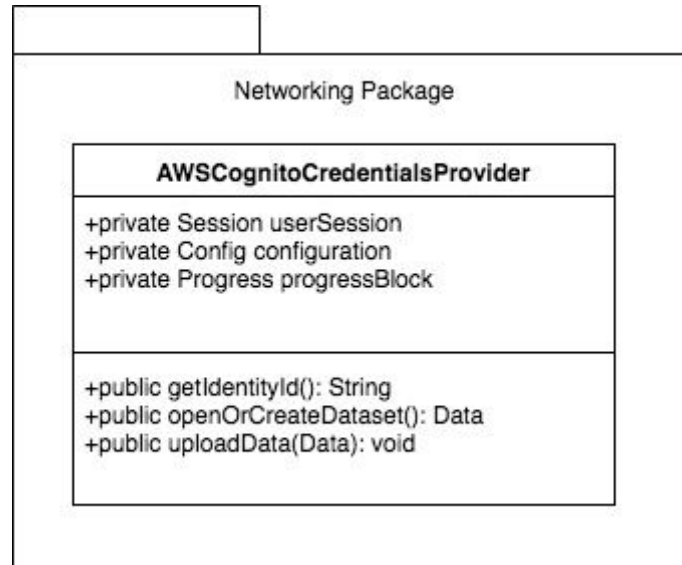


Figure 6: The uml diagram for networking

4.2.2.4. Data Management Subsystem

This subsystem will provide data access management connecting the GUI with the AWS S3 storage. It will include data access functions to AWS via the wrapped functions for the API. The access to the network is done via HTTP requests such as GET and POST.

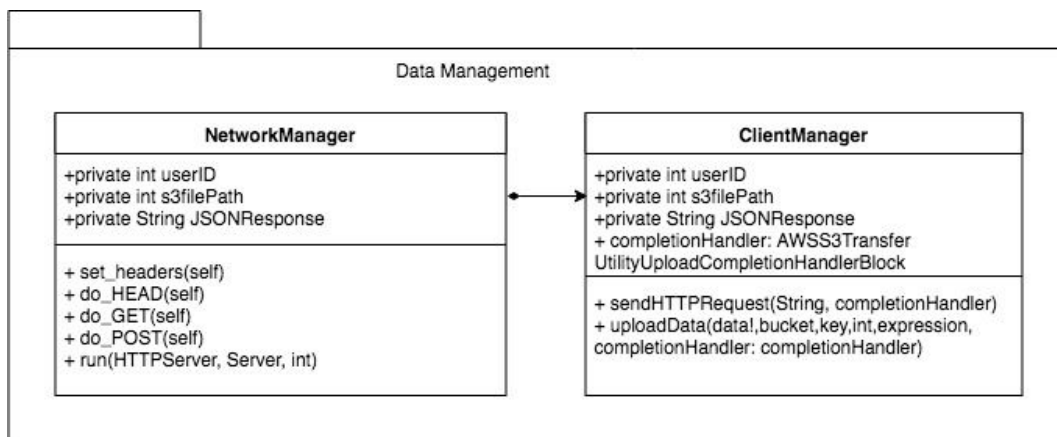


Figure 7: The uml diagram for data management

4.2.3. Controller

4.2.3.1. User Authentication Control Subsystem

This package includes all of the classes that act as a bridge between the Data Management Package and the User Authentication View Package, which is in the Client Subsystem. In the User Authentication View, there will be calls to functions AWS. After the authentication is performed, login session will be held in the UserSessionControl class, which is located in the Networking Package.

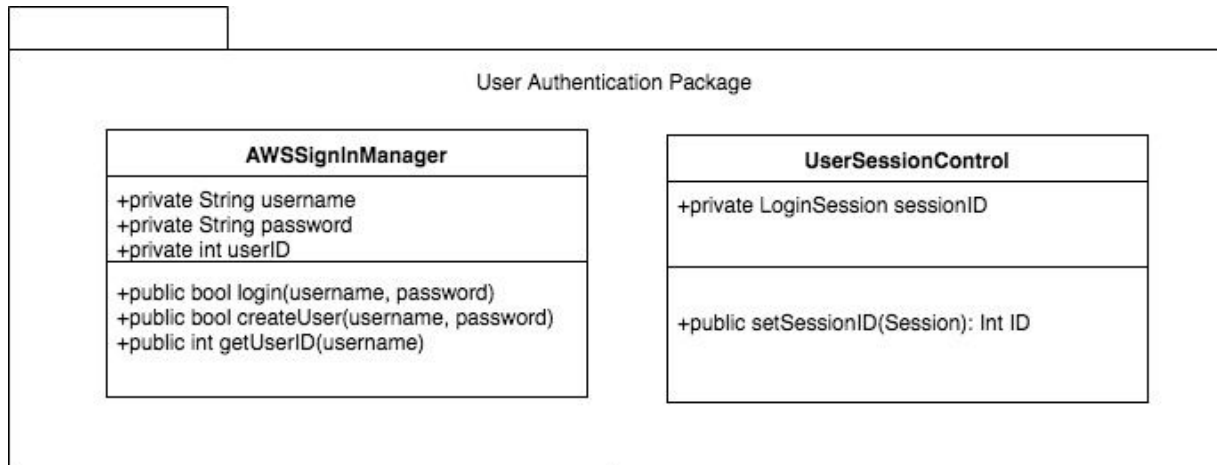


Figure 8: The uml diagram for user authentication

5. Impact of Engineering Solutions

5.1. Global Impact

There are several tools for different mapping types based on different aspects of material. These tools mostly target to the game developers and computer graphic designers. Since these tools are mostly used at workplace by specific job areas, only desktop is supported as a platform. Thus, these tools are not flexible in terms of customer range, scope of applicability areas and platform support. They do not have share options which prevent any global impact. There are also mobile applications for interior design such as Marshall's application for trying out new wall paint colors on your own walls. These kind of applications have social media share options.

Bumpster has a global impact by involving different cultures from the world. These models can be viewed in augmented reality. Then, the model can be shared in the feed or social media based on user's request. Since users can also view other shared models in this feed, users can follow different and popular textures and models around the world. Furthermore, our application targets to a wide range of customers including interior, exterior and landscape designers, generic and independent customers, hobbyists, computer graphic designers which also extends scope of applicability areas of Bumpster. Wide range of customers sharing different models also have a global impact.

5.2. Economic Impact

Existing professional tools such as CrazyBump and nDo2 related to texturing software are good at exporting different kinds of maps from photographs are expensive for generic customers and the ones which are not working on generating maps such as graphic designers and game artists as a job. The free mobile apps are based on augmented reality and they are not capable of generating maps and 3D models in augmented reality. Bumpster promotes 3D surface modeling and simulation which is powered by augmented reality and neural network. Therefore, Bumpster is more reliable than most of professional tools in terms of accuracy and precision. In addition, since Bumpster is available in app store freely for wide customer range. Furthermore, the companies benefit from the usage of Bumpster can return profit as they do not pay high prices to professional tools.

5.3. Environmental Impact

The environmental impact of Bumpster lies in its ability to enhance the process of topographical map generation. As known using traditional methods to conduct topographical analysis, the surface which will be modeled had to be inside the studio environment. If this texture existed in the nature and was an integral part of the ecosystem, it can be said that traditional methods harm the environment. With Bumpster since we do not need to change the location of the material of interest, we would be leaving the environment as is, and use deep learning techniques to obtain studio-like results.

Another environmental impact Bumpster positively has is the amount of physical models required to be generated. As artists or developers needs to visualize the product in the environment they desire, they are inclined to physically generate the model using plastic and use the physical model to derive their conclusions. Since Bumpster supports the model to be placed in the environment the user is in using Augmented Reality, the need to use plastic and generate physical models is minimized if not eliminated.

5.4. Societal Impact

Professional tools based on texturing software only serve to texture need of game artists and developers and mobile apps only target to specific customers of a brand. Bumpster serve many areas such as art, game scenes, interior, exterior and exterior design and to the wide range of customers. For interior design, Normally, a customer searching for the right carpet for his/her sitting room might need to go to carpet seller more than once and get some limited carpet sample from the store to try out at the home to find the correct carpet. With Bumpster, the customer can just take a photo of as many carpets as he/she likes to and then try out at home or the carpet seller can send the photos of desired carpets so the customer even does not need to go to the store. Then, the customer can put the carpet wherever he/she wants in the sitting room with the advantage of augmented reality and accuracy provided by neural network. This situation is similar for exterior and landscape design for customers, companies and authorities trying out different products. Therefore, the carpet example can also be thought as a vase, painting, a texture to cover a furniture etc. In addition, for game scene design, the game artists need to work on producing realistic textures for hours and they can

only work on desktop. With Bumpster, the game artist can discover new textures and generate more accurate maps using their mobile phones and waste less time to make realistic scenes for the games.

6. Engineering Solutions and Contemporary Issues

The language of Bumpster is selected as English since both the world and specifically the industry this product is aimed uses English more frequently than other languages. But since Bumpster is built to be as improvable as possible, we plan to support more native languages by using the phone's language.

In today's world the issue of security is crucial. Because of this, Bumpster needs to keep respect the security of customers and provide full mutual exclusivity between the models and data of the users. Because of this, Bumpster uses the id of the device it is running on to generate secret credentials, and uses the secret credentials to access the database, upload data. The database the data is stored is also maintained at highest security. The database provider Amazon Web Services provide the interface to securely upload/download our data to Amazon S3 via SSL endpoints using the HTTPS protocol. We also plan to use the Server-Side Encryption (SSE) option to encrypt data stored in the future [16].

Because of the nature of our application, a stable internet connection during the uploads/downloads is required by default. To increase security and provide the best results possible, we use a powerful server to store and use the deep learning model we implemented. To exchange data with this server and store data as an individual user, the device which the application is running on needs to have an internet connection. Thanks to the current standards of internet connection globally available, this decision was made with confidence.

One of the main issues we faced was the unavailability of data we could find to train the deep learning model. We needed the pair of image vs relevant topographical maps to train and validate the model. Because a data like this is almost non-existent to be found and utilised, we generated our own data using automated processes. We established this by generating a random physical texture using Blender's Python API, and generate results using different ambient lighting to achieve results as natural as possible.

We also found it hard to work with large amounts of data being processed on our workspaces. Since the files we used to train and get results from the deep learning model were big and large in amount, our computers became unresponsive and the process of uploading/downloading became a bottleneck in our workflow. While the problem of working with these data on our computers were not solvable, we managed to solve the problem of data transferring by changing the resolution of images where applicable, and using download/upload management APIs to handle larger files successfully.

Another engineering problem we faced as Bumpster was the amount of budget and support available to the development team. As known, Bumpster rents powerful computer services in the cloud to provide the best, most secure, and most reliable system possible. Because the amount of processes conducted in the system, the rather large amount of requests made by users, and data stored in the cloud storage, the costs of maintaining the system increased exponentially. We believe that because the budget was very tight in the development stage of our project, the accuracy and speed of our system took a lot of time and work to live up to its potential.

7. Tools and Techniques Used

7.1. Libraries

7.1.1. Keras

Keras is a high-level neural networks API written in Python which can run on top of Tensorflow[12]. The focus of Keras is fast experimentation, easy and fast prototyping. It can run continuously on CPU and GPU. Both convolutional, recurrent networks and combination of them are supported. Keras is user-friendly, extensible and modular.

7.1.2. Tensorflow

TensorFlow is a cross-platform open source software library for numerical computation using data-flow graphs developed by the Google Brain Team [13]. It is compatible with higher-level APIs such as Keras which makes training and evaluating models easily.

7.1.3. SDWebImage 3.8.1

It is an asynchronous image downloader with cache support with an UIImageView category [14].

7.2. Frameworks

7.2.1. ARKit

ARKit is an Augmented Reality framework for iOS that provides a platform for developing Augmented Reality (AR) apps for iPhone and iPad. Augmented Reality simulates the user experiences that add 2D or 3D elements to the live view of a device's camera in a way that makes the added elements appear to be in the real world. ARKit enables device motion tracking, advanced scene processing, display conveniences and camera scene capture to make the experience of building an Augmented Reality application simple. ARKit creates a correspondence between the real-world space and the virtual space in which a visual content can be modelled. ARKit uses world and camera coordinate systems which follow the right-handed convention. It uses a technique called visual-inertial odometry that enables to create a correspondence between real and virtual spaces and this technique combines information from the device's motion sensing hardware with computer vision analysis of the scene that is visible to the device's camera. These techniques create a high-precision model of the device's position and motion. We used ARKit in Bumpster to view the output 3D model which has the albedo, ambient, metalness, roughness and normal maps that are generated by the neural network, mapped. We also added gesture recognizing features to our app to enable interaction with 3D objects such as rotating and scaling.

7.2.2. CUDA

CUDA is a parallel computing platform and a programming model that is developed by NVIDIA for computing tasks run on graphical processing units. CUDA enables a dramatic speed up for the computing applications by using the power of GPUs. In GPU accelerated programs, the sequential part of the program runs on the CPU that has an optimal performance for single-threaded performance. However, the compute intensive part of the programs run on many GPU cores in parallel. CUDA enables developers to program in several languages such as C, C++, Python and etc. These languages have their own ways to express parallelism by the aid of basic keywords. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime [1].

7.3. APIs

This section describes all of the APIs used in the development of Bumpster.

7.3.1. Blender Python API

Blender API is used in Blender Software for Python for scripting to customize the application and write specialized automated programs. The Blender/Python API can edit any data the user interface can (such as scenes, meshes, particles, objects etc.), modify user preferences, keymaps and themes, run tools with own settings, create user interface elements like headers, panels and menus, create new tools, create interactive tools, create new rendering engines that can integrate with Blender, define new settings in Blender data, draw in the 3D view using OpenGL commands from Python. We used Blender Python API to render our own training data for the neural network model to generate physically based rendering maps to be further used in the Bumpster application (especially in the Augmented Reality mode for more realistic previews of image-mapped planes) [2]. The main advantage of the Blender Python API is that the entire Blender GUI is built with the Python API so we can do anything that the development team of Blender can do, therefore it is fully interactive.

7.3.2. SceneKit

SceneKit is a framework that enables creating 3D games and adding 3D content to applications using high level scene descriptions. It easily enables users to add animations, physics simulations, realistic physically based rendering and particle effects. This framework combines a descriptive API with a high-performance rendering engine for rendering, manipulation and importing of 3D assets. SceneKit is easy to use, because it only requires the descriptions of the scene's contents and the actions you want the scene to perform along with its animations [3]. SceneKit acts like a layer over the lower level libraries like OpenGL and Metal. SceneKit uses a node-based content management to model the hierarchy. We used SceneKit to model and manipulate our 3D objects in augmented reality environment. Integration between SceneKit and ARKit made is easier for us to place, model, modify and interact with 3D objects in AR.

7.4. Internet Resources

7.4.1. Blender Stack Exchange

Stack Overflow is the largest, most trusted online community for developers, professional and enthusiast programmers to learn, share their programming knowledge, and build their careers [8]. Blender Stack Exchange is a question and answer site for people who use Blender to create 3D graphics, animations and games [7] . Since generating training data for Bumpster requires the usage of Blender and internet lacks of Blender related problems and solutions, Blender Stack Exchange is the most reliable and largest site to solve our problems based on Python and Blender usage.

7.5. Software, Tools and Services Used

7.5.1. Amazon Elastic Compute Cloud(EC2)

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that ensures secure, resizable compute capacity in the cloud [9]. Amazon EC2 aims to make web-scale cloud computing easier for developers with its simple web service interface enabling to obtain and configure capacity with minimizing friction. It provides developers complete control of their computing resources and let them run on Amazon's computing environment. It has the advantage of reducing the time required to obtain and boot new server instances to minutes, providing developers the tools to build failure resilient applications and isolate them from common failure scenarios and allowing developers to quickly scale capacity, both up and down, as their computing requirements change [9]. In addition, it changes the economics of computing by enabling only the payment of the capacity that developer actually uses. It has the benefits of elastic web-scale computing, complete control of instances, flexible cloud hosting services, integration with most AWS services, reliable environment, highest priority, low financial rate for the compute capacity developer actually consume and easy start [9].

7.5.2. AWS Lambda

AWS Lambda lets you run code without providing or managing servers. Similar to Amazon EC2, developers only pay for the compute time they consume [10]. With AWS Lambda, developers can run their code for virtually any type of application or backend service with zero administration. It takes care of everything needed to run and scale code with high availability after uploading code. Developers can also set up their code to automatically trigger from other AWS services or call it directly from any web or mobile app [10]. There are no servers to manage and it automatically scales application by running code in return for each trigger.

7.5.3. AWS Cognito

Amazon Cognito handles authentication and enables to add user sign-up, sign-in and access control to your web and mobile apps quickly and easily [15].

7.5.4. Amazon S3

Amazon S3 (Amazon Simple Storage Service) is a storage service by Amazon. It is designed to make web-scale computing easy for the developers. It has simple web services interface that allows the developers to store and retrieve data at any time and from anywhere. The storage is provided through web services interfaces such as REST and SOAP.

7.5.5. AWS Mobile Hub

AWS Mobile is a fast way to build apps that scale [11]. It has benefits such free tier, adding cloud services like storage, bots, database, authentication fast and delivering quality apps.

7.5.6. Xcode

We've used Xcode 9.3 IDE as our development environment for our application. Throughout development, we've deployed our application to a testing device using Xcode's run on device feature.

7.5.6.1. Swift

We've used Swift 4.1 by Apple as our programming language for the iOS application.

7.5.7. Usage of Blender for Training Data

Blender is a free and open source 3D model creation software which supports all 3D pipeline steps, which are modelling, simulation, rendering, motion tracking, game creation, animation and so on. Advanced users use scripting via Python and Blender's API for customizing the application and automating tasks to be done (rendering scenes in our case). Blender has a responsive development feature, which gives instant previews. Blender is a cross-platform software and runs well in Linux, Windows and MacOSx. We used Blender in all of these operating systems. Also, it has a Python development environment and Python console, which indicates that it has a built in integrator for Blender API used via Python. Blender is an open-source project under the GNU General Public License and it is completely free. It is being actively developed by many people from around the world and used by animators, artists, hobbyists, students and much more [4].

Blender is a software that is easy to learn and easy to use, since there are many tutorials and other references available [5]. There are many views and render modes to see materials, objects and light conditions in the scene for easy debugging and easy view. Blender's embedded Python interpreter is always active while Blender is running and this interpreter runs scripts to draw the user interface and it is also utilised in its internal tools too. Codes from external tutorials can be run with Blender's interpreter. Blender also provides its Python modules like *bpy* and *mathutils* to the interpreter so imports can be performed and access to Blender's data, classes and functions is given. When the Python scripts that modify Blender's internal data directly in the interactive console, the 3D viewport update is seen [6].

7.5.7.1. Rendering Maps

7.5.7.1.1. Normal Map

In order to render the normal map, we used the normal channel of material's first texture slot.

7.5.7.1.2. Metalness Map

In order to render the metalness map, we used the color diffuse channel of material's second texture slot.

7.5.7.1.3. Roughness Map

In order to render the roughness map, we used the color diffuse channel of material's third texture slot.

7.5.7.1.4. *Ambient Map*

In order to render the ambient map, we used diffuse and color diffuse channels of material's fourth texture slot.

7.5.7.1.5. *Albedo Map*

In order to render the albedo map, we used specular and color diffuse channels of material's fifth texture slot.

7.5.7.2. *Rendering Scene*

The following sections describe how each map is mapped onto the plane in Blender Python. For each texture, single texture slot of the material is used.

7.5.7.2.1. *Normal Map*

In order to map the normal map onto the plane, we assigned normal factor of material's first texture slot to 0.2. We changed the texture coordinates to ORCO which is generated coordinates. In addition, we used map normal of material's first texture slot and normal map of the normal texture.

7.5.7.2.2. *Metalness Map*

In order to map the metalness map onto the plane, we gave metalness information to plane by the specular and color specular channels because the interaction of the light with the plane surface is represented in most accurate way. Also, we changed the hardness factor to 1 so as to give the illusion of a more pointy reflection of light. We changed the texture coordinates to ORCO which is generated coordinates. We changed the blend type to MIX to mix the light information of metalness with roughness since both maps were given from specular channel. In addition, we used map normal of material's second texture slot.

7.5.7.2.3. *Roughness Map*

In order to map the roughness map onto the plane, we gave roughness information to plane by the specular and color specular channels with negative specular color factor because the interaction of the light with the plane surface is represented in most accurate way. We changed the texture coordinates to ORCO which is generated coordinates. We changed the blend type to MIX to mix the light information of roughness with metalness since both maps were given from specular channel.

7.5.7.2.4. Ambient Map

In order to map the ambient map onto the plane, we gave ambient information to plane with the diffuse and color diffuse channels because the simulation of global illumination shadows can only be assured by the occluded ambient light map pixels (dark ones). We changed the blend type to MIX which allowed us to use albedo map with ambient map through the diffuse channel. In addition, we used map normal of material's fourth texture slot.

7.5.7.2.5. Albedo Map

In order to map the albedo map onto the plane, we gave albedo information to plane with the diffuse and color diffuse channels because the real base color information can only be given with this way.

8. User Manual

8.1. Installing Instructions

In order to install and run Bumpster, open Bumpster.xcworkspace file in Xcode. Make sure your testing device is connected to you Mac with a USB cable. In Xcode, using the devices menu, select your device and run (⌘+R) the application. Make sure your device is unlocked to enable Bumpster to launch.

8.2. Sign In Screen

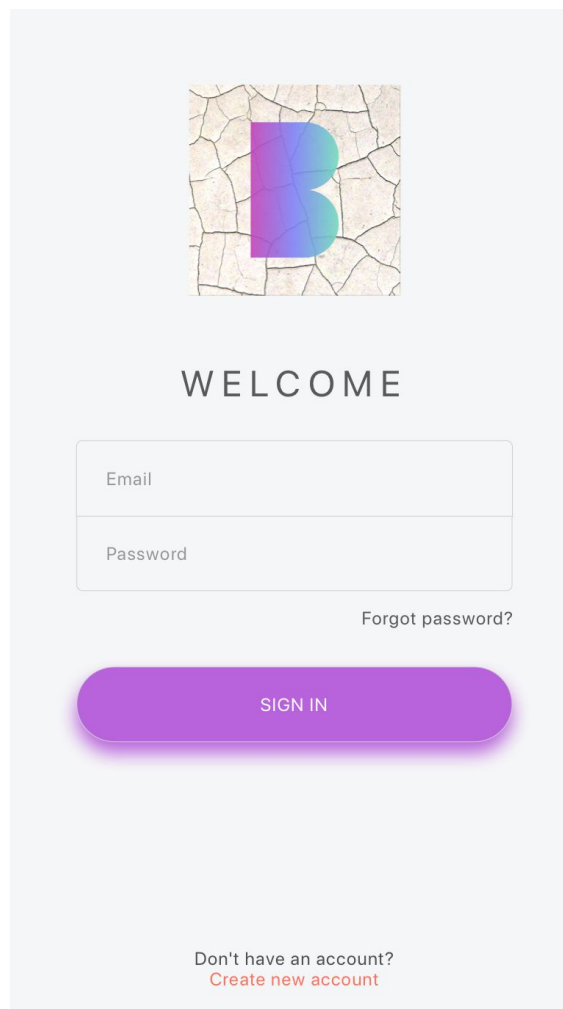
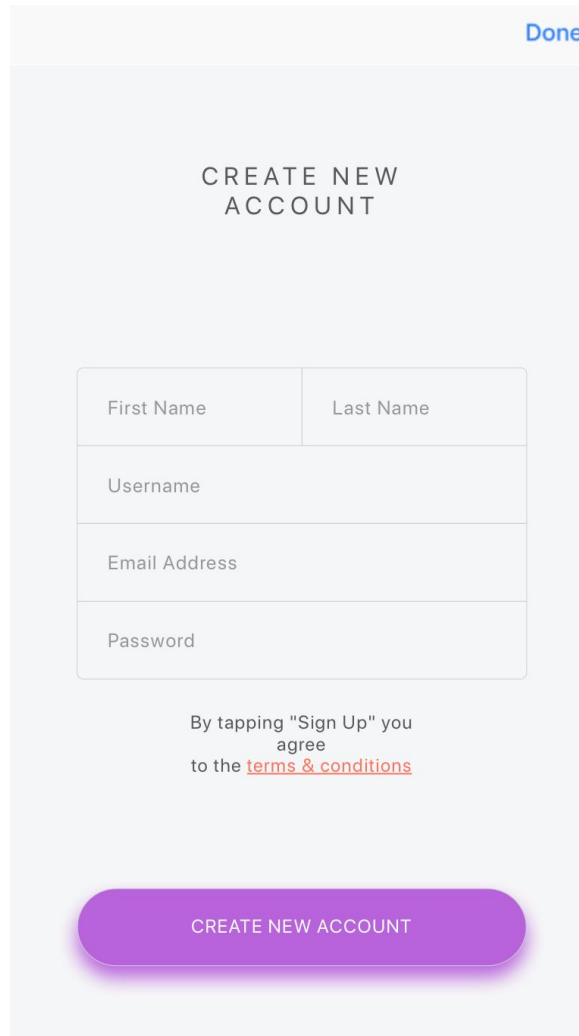


Figure 9:Sign In

The users are greeted with a sign in screen if it is the first time they are using the app. The view consists of an email and password input fields. When the user provides the correct

credentials and clicks "*Sign in*", the view navigates to the home screen. The user can also navigate to sign up screen by clicking "*Create new account*" below.

8.3. Sign Up Screen



The image shows a mobile app screen for creating a new account. At the top right, there is a blue "Done" link. The main heading is "CREATE NEW ACCOUNT" in all caps. Below the heading is a form with five input fields: "First Name" and "Last Name" (side-by-side), "Username", "Email Address", and "Password". Below the form, there is a line of text: "By tapping 'Sign Up' you agree to the [terms & conditions](#)". At the bottom, there is a large purple rounded rectangle button with the text "CREATE NEW ACCOUNT" in white.

Figure 10:Sign Up

If the user doesn't have an account, a new account can be created at this screen. To create an account, the user needs to fill First Name, Last Name, Username, Email Address, and Password fields. When the user enters the correct credentials, they can click "*Create new account*" to register themselves and navigate to home screen. If the user remembers their credentials, they can click "*Done*" and return to the Sign In screen.

8.4. Home Screen (Scene View)



Figure 11: Home Screen

This view is the main view of the application. Export, edit, uploading new photos, displaying list actions are accessible from this view. In upper left part of the screen we have a status view. It displays instructions to the user such as “Tap to place an object”. Being an augmented reality application, the main view functions as a camera. The perspective of the AR view is automatically updated as the user moves around the object. This view has different gesture recognizers to allow user interaction. In order to add an object, tap once to the AR view. In order to remove an object, tap once on the object. Use two fingers to rotate the object. Use two fingers and pinch to scale up or down the object.

8.5. New Photo Selection View

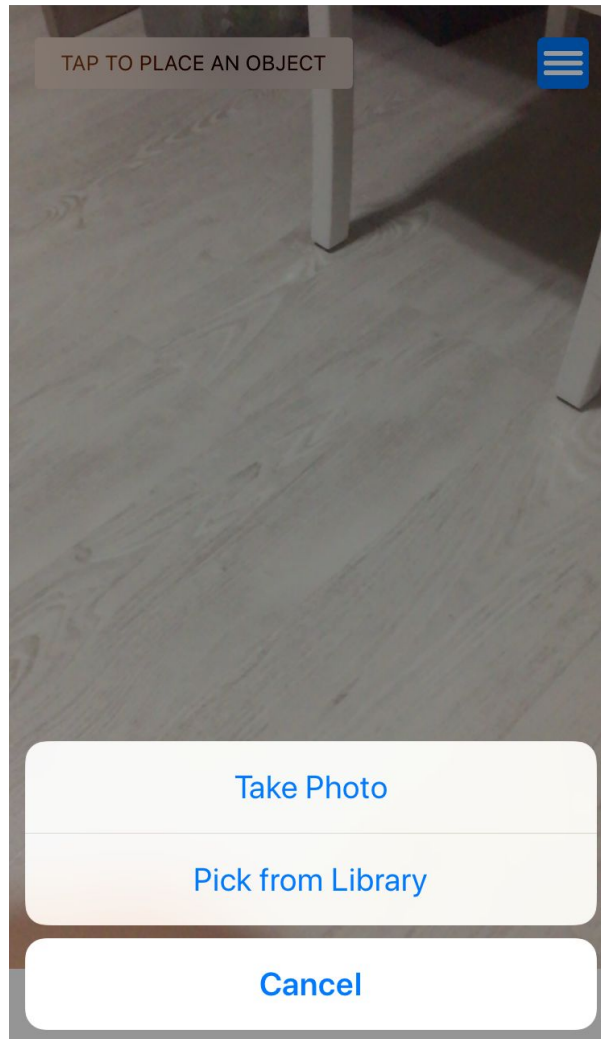


Figure 12: New Photo Selection View

When camera icon in the Home Screen is pressed, this view is presented. In order to upload a new image, users can take a photo directly from the application using “Take Photo” button. Users can also pick an image from their photo library using “Pick from Library” button. Users can press “Cancel” button to dismiss this view without any action.

8.6. Photo Taking View



Figure 13: Photo Taking View

This view is used to take photos within our application. Users can focus by tapping the camera view, adjust flash settings using the flash button in the top-left corner. Users can switch between front and main cameras using the button in the bottom-right corner. Users can take a photo using the circle-shaped button and cancel this action using the Cancel button.

8.7. Photo Edit View

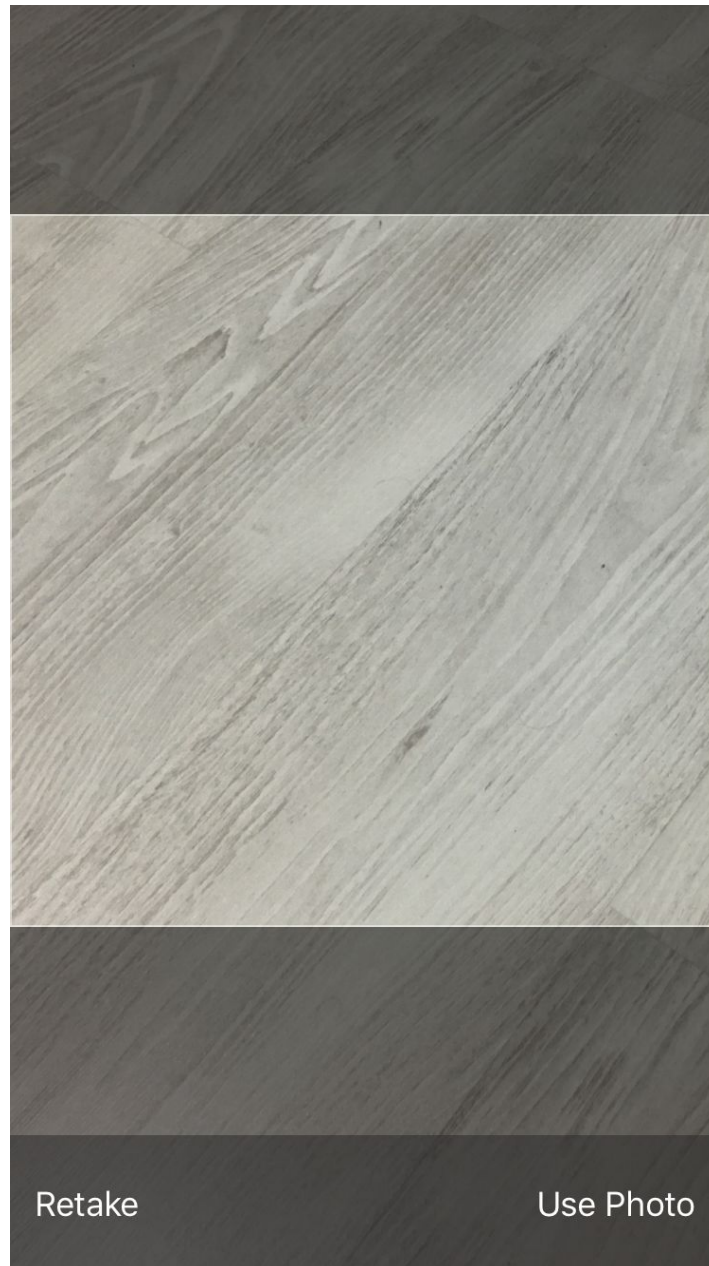


Figure 14: Photo Edit View

Users can crop and select specific parts of the photo using this view. Users can press Retake button to take another photo, or press Use Photo to send this photo to cloud.

8.8. Upload Progress View



Figure 15: Upload Progress View



Finishing up...

Figure 16: Upload Progress View (Finishing up)



Please check your internet connection

Figure 17: Upload Progress View (Please check your internet connection)

Users can view the progress of upload using the increments in the blue progress bar and the status label. During the upload process, the progress bar is updated incrementally. Once the progress bar is loaded and the photo is uploaded to the cloud, we run code on EC2. To indicate that this action is in progress, we use a status label titled “Finishing up...”. If this action takes more than 20 seconds, we update this status label to a red warning label titled “Please check your internet connection” to give users more insight on this process, and dismiss this view automatically.

8.9. List Models View

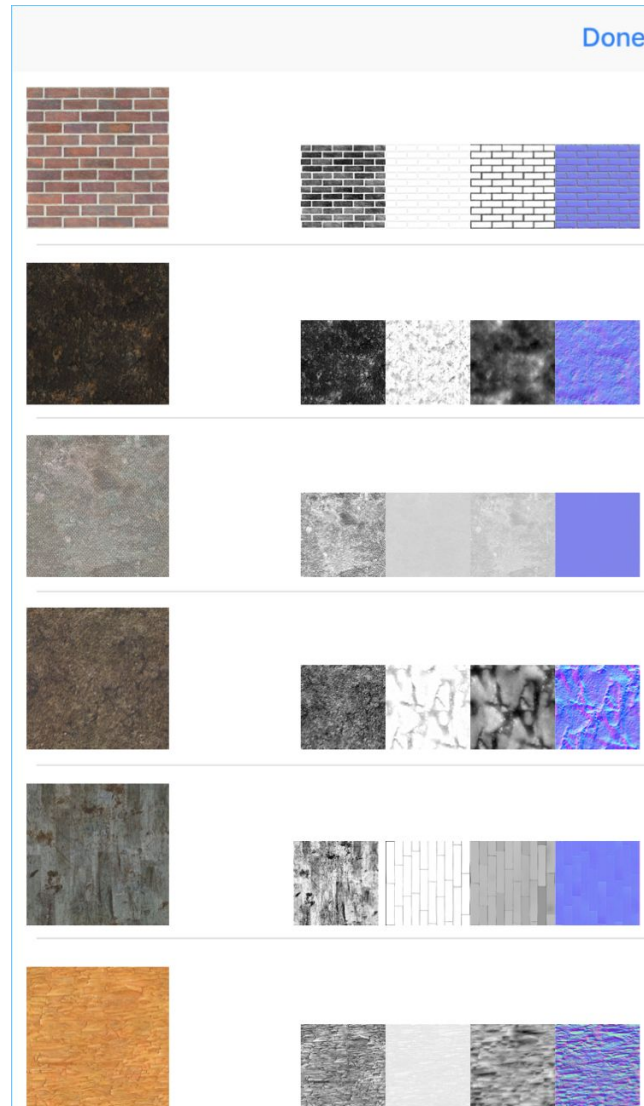


Figure 18:List View

Users can select the pre-saved model from the table view, and select the data desired to be viewed, edited, or shared. A single cell consists of an input image on the left (taken or uploaded by the user), and generated maps on the right as four smaller images. The users can click to select the image, or click "*done*" to dismiss the view into the home screen.

8.10. Share Export View

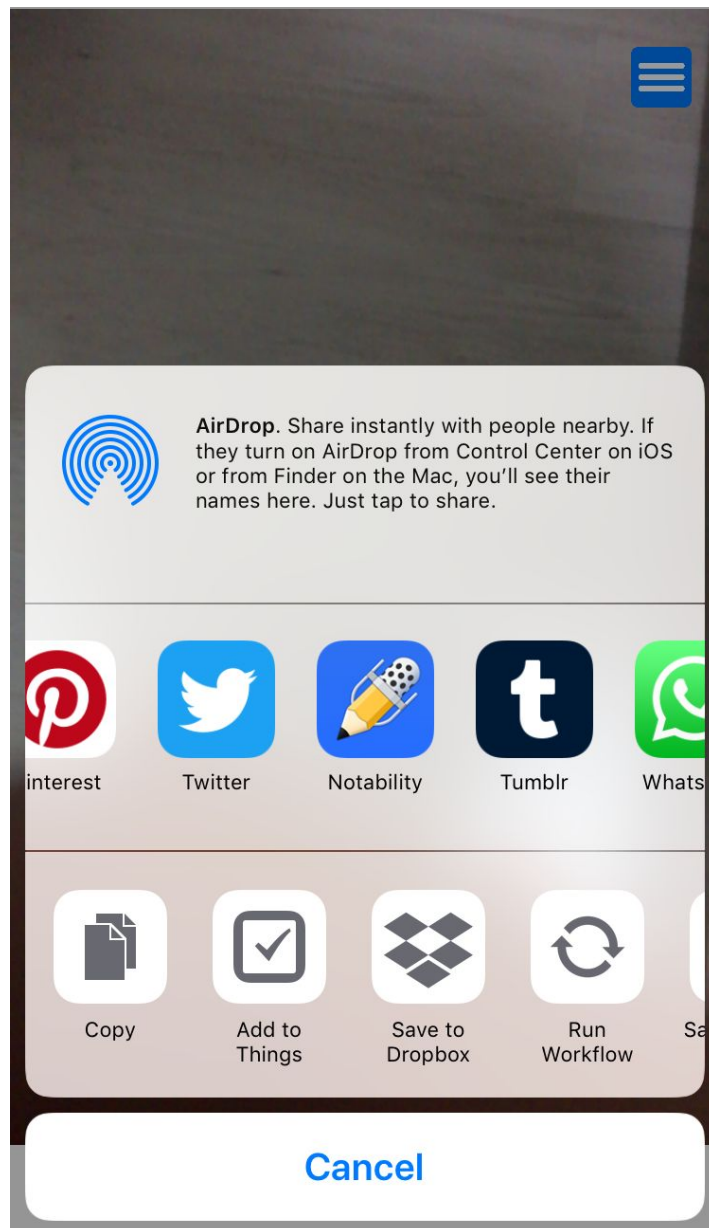
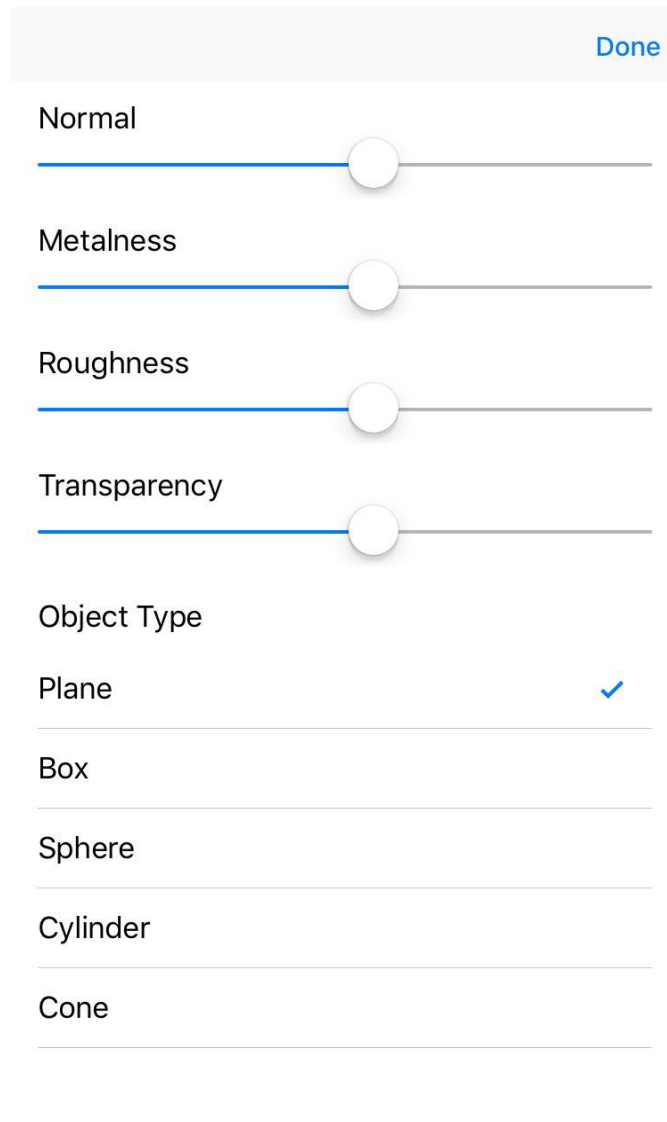


Figure 19: Share (Export) View

Users can share their generated maps by pressing the Export button in Home Screen. This export view supports various platforms for sharing including messaging apps like WhatsApp and iMessage, social media platforms like Twitter, Pinterest and Facebook, and file management services like Dropbox and iCloud.

8.11. Edit Model View



The image shows a user interface for editing a model. At the top right is a blue 'Done' button. Below it are four horizontal sliders, each with a white circular knob. The sliders are labeled 'Normal', 'Metalness', 'Roughness', and 'Transparency'. Below the sliders is a section titled 'Object Type' followed by a list of options: 'Plane', 'Box', 'Sphere', 'Cylinder', and 'Cone'. The 'Plane' option is selected, indicated by a blue checkmark to its right.

Figure 20:Edit Model

With the desired model selected from the list, user can now edit the model as desired using the properties: Normal, Metalness, Roughness, Transparency. These values are edited with a slider for a more natural and responsive experience for the user. The user can also change the object which the texture will be modeled on. The user can select from the preset objects: Plane, box, sphere, cylinder, and cone. When the user clicks "*done*", the view will dismiss into the model view.

9. Class Diagrams

TrainingDataGenerator
+ bpy.data.cameras: Camera[] + bpy.data.images: Image[] + bpy.data.lamps: Lamp[] + bpy.data.objects: Object[] + bpy.data.objects.modifiers: Modifier[] + bpy.data.scenes: Scene[] + bpy.ops.object: Object[] + mat: Material + mat.texture_slots: Texture[] + mat.use_textures: Bool[]
+ add_base(): Plane + reset_blend(): void + render_original(filename): Object + render_normal_map(filename): void + render_metalness_map(filename): void + render_roughness_map(filename): void + render_ambient_map(filename): void + render_albedo_map(filename): void + return_lamp_pos_orig(): void + return_cam_rot_orig(): void + return_cam_pos_orig(): void + disable_all_textures(): void + disable_all_modifiers(): void + main(): void

DCGAN
+ generator_model: type + discriminator_model: type + input_img_dim: type + patch_dim: type + generator_input: type + generator_image: type + list_row_idx: type + list_col_idx: type + dcgan_output: type + list_gen_patch: type + dc_gan: Model
+ main(): void

Generator
+ stride: int + filter_sizes: int[] + encoder: tensor + decoder: tensor + generator: tensor + merge_mode: string + bn_mode: int + bn_axis: int + en_1..8: LeakyReLU + de_1..8: Convolution2D + unet_generator: Model
+ make_generator_ae(input_layer, num_output_filters): tensor + UNETGenerator(input_img_dim, num_output_channels): tensor

Discriminator
+ x, list_input: tensor[] + x_flat: tensor[][] + patch_gan: Model[] + x, x_mbd: Model + M: tensor + MBD: layer x_out, y_out: tensor[] + stride: int + bn_mode: int + axis: int + input_layer: tensor + num_filters_start: int + nb_conv: int + filters_list: int[] + disc_out: Convolution2D + disc_out2: LeakyReLU + patch_gan_discriminator: Model + output_img_dim, patch_dim, nb_patches: int[]
+ patch_gan: Model + patchGanDiscriminator(output_img_dim, patch_dim, nb_patches): Model + generate_patch_gan_loss(last_disc_conv_layer, patch_dim, input_layer, nb_patches): void + lambda_output(input_shape): tensor[] + minb_disc(x): tensor[]

main
+ WORKING_DIR: string
+ DATASET: string
+ test_generator_nn: UNETGenerator
+ x_path, y_path: string
+ x_image, y_image: image
+ x_batch: image[]
+ out: image
+ im_width, im_height: int
+ input_channels, output_channels: int
+ input_img_dim, output_img_dim: int[]
+ sub_patch_dim: int[]
+ nb_patch_patches: int
+ patch_gan_dim: int
+ generator_nn: UNETGenerator
+ num_output_channels: int
+ discriminator_nn: PatchGanDiscriminator
+ opt_dcgan, opt_discriminator: Adam
+ dc_gan_nn: DCGAN
+ loss, loss_weights: int[]
+ batch_size: int
+ data_path: string
+ nb_epoch, n_images_per_epoch: int

Facades_Dataset
+ ROOT_DIR: string
+ WORKING_DIR: string
+ AWS_FACADES_PATH: string
+ ds_name: string
+ tmp_path: string
+ data_folder_path: string
+ downloaded_path: string
+ merge_mode: string
+ download(): void
+ mk_tmp_dirs(dir_name): void
+ mk_data_dirs(dir_name, dataset_name): void
+ untar_file(source_path, destination_path, remove_tar, flags): void
+ download_facades_bw(tmp_path, data_folder_path): void
+ main: void

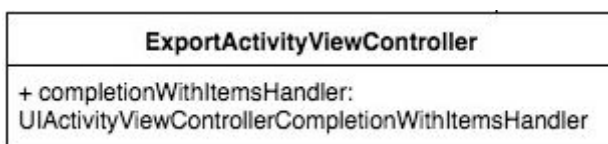
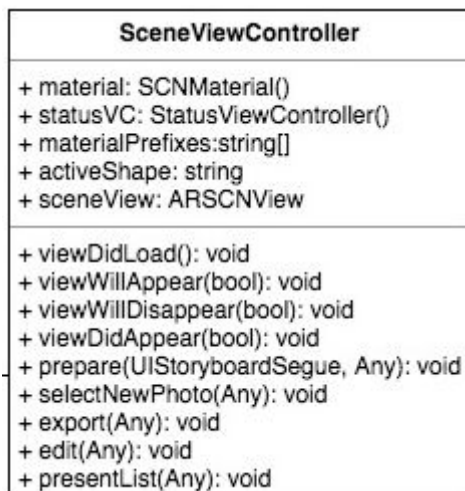
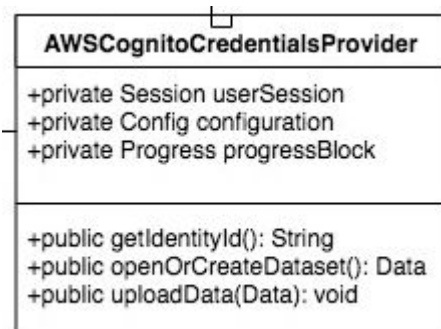
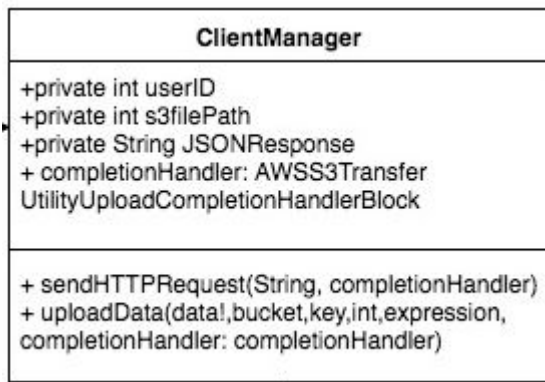
Bumpster_Generator
+ IMSIZE: int + minisize: int + data_dir: string + x_batch: image[] + y_batch: image[] + batch_imgs: image[] + WORKING_DIR: string + DATASET: string + data_path: string + gen: image
+ normalize(X): int + cropimg(crop, xcrop, ycrop,img): image + normals_generator(data_dir_name, data_type, im_width, batch_size = 1): image + main(): void

Patch_Utils
+ nb_non_overlapping_patches: int + patch_disc_img_dim: int[] + im_height, im_width: int + patch_height, patch_width: int + x_spots, y_spots: int[] + all_patches: [] + image_patches: image[] + x_disc, y_disc: image + p, idx, x1, x2: int
+ num_patches(out_img_dim=(3,256,256), sub_patch_dim=(64,64)): int + extract_patches(images, sub_patch_dim): image[] + get_disc_batch(X_original_batch, X_decoded_batch, generator_model, batch_counter, patch_dim): image, image + gen_batch(X1, X2, batch_size): void

Facades_Generator
+ X: image + Y: image + data_dir: string + bucket_names_in_dir: string + images_path: string + facades_path: string + target_images: File + facade_images: File + num_images: int + width: int + batch_num: int
+ normalize(x): int[] + facades_generator(data_dir_name, data_type, im_width, batch_size): void

Logger
+ inverse_normalization(X): int + plot_generated_batch(X_full, X_sketch, generator_model, epoch_num, dataset_name, batch_num): image + main(): void

NetworkManager
+private int userID +private int s3filePath +private String JSONResponse
+ set_headers(self) + do_HEAD(self) + do_GET(self) + do_POST(self) + run(HTTPTServer, Server, int)



EditViewController
+ sceneVC: SceneViewController() + shapes: string[] + normalSlider: UISlider + metalnessSlider: UISlider + roughnessSlider: UISlider + transparencySlider: UISlider + objectTypeTableView: UITableView
+ viewDidLoad(): void + donePressed(Any): void + normalSliderValueChanged(UISlider): void + metalnessSliderValueChanged(UISlider): void + roughnessSliderValueChanged(UISlider): void + transparencySliderValueChanged(UISlider): void + setCurrentSliderValues():void

UploadProgressViewController
+ uploadImageView: UIImageView + progressView: UIProgressView + statusLabel: UILabel + image: UIImage + syncClient: AWSCognito
+ viewDidLoad(): void + setImage(UIImage): void + upload(): void

AWSSignInManager
+private String username +private String password +private int userID
+public bool login(username, password) +public bool createUser(username, password) +public int getUserID(username)

UserSessionControl
+private LoginSession sessionID
+public setSessionID(Session): Int ID

UserAuthenticationViewController
+fullName: String +email: String +password: String
+authenticate(email: String, password: String, completionHandler: (Response, Error)): void

SavedModelsListViewController
+ imageURLs: [String]() + bumpImageURLs: [String]() + images: [UIImage]() + bumpImages: [UIImage]() + sceneVC = SceneViewController() + syncClient = AWSCognito
+ viewDidLoad(): void + downloadFromCloud(): void

10. References

- [1] <https://developer.nvidia.com/cuda-zone>
- [2] https://docs.blender.org/api/2.79/info_quickstart.html
- [3] <https://developer.apple.com/documentation/scenekit>
- [4] <https://www.blender.org/about/>
- [5] <https://code.blender.org/2013/10/redefining-blender/>
- [6] https://docs.blender.org/api/2.79/info_overview.html
- [7] <https://blender.stackexchange.com/>
- [8] <https://stackoverflow.com/>
- [9] https://aws.amazon.com/ec2/?sc_channel=PS&sc_campaign=acquisition_TR&sc_publisher=google&sc_medium=ec2_b&sc_content=ec2_e&sc_detail=amazon%20ec2&sc_category=ec2&sc_segment=177549435029&sc_matchtype=e&sc_country=TR&sc_kwid=AL!4422!3!177549435029!e!!g!!amazon%20ec2&ef_id=V@VRDwAABWKD76KK:20180429141207:s
- [10] <https://aws.amazon.com/lambda/>
- [11] <https://aws.amazon.com/mobile/>
- [12] <https://keras.io/>
- [13] <https://opensource.com/article/17/11/intro-tensorflow>
- [14] <https://libraries.io/cocoapods/SDWebImage/3.8.1>
- [15] <https://aws.amazon.com/cognito/>
- [16] <https://aws.amazon.com/s3/>