Senior Design Project

“Bumpster”

Low-Level Design 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

February 12, 2017

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

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 overview of low-level architecture and design of our system. Firstly, object design trade-offs of our system and engineering standards are described. Then, interface documentation guidelines are given as an outline for class descriptions. Afterwards, packages with their functionalities in our system are described with class diagrams in detail. It is followed by interfaces of classes in all packages. Therefore, the clarification of functionalities in our software is provided.

1.1 Object Design Trade-Offs

1.1.1 Accuracy vs. Performance

Accuracy and Performance are important aspects of the system. In the case of performance, the system should be efficient and give fast response to user request. Software should give answer to peer request as fast as possible to provide best possible response and move on to process future requests in time. In addition, the database will be used efficiently which enables easy inserting, deleting and updating in terms of time cost. In the case of accuracy, existing systems based on texturing software such as CrazyBump are good at exporting different kinds of maps from photographs or scanned images, but the accuracy is not high. 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. Creating different kind of maps with higher accuracy is main aim of the project which also reduces the necessity of additional customization. However, creating accurate maps lowers the performance of system. If we are not able to generate maps accurately, there will be no use of performance in the system.Since our main goal is to generate accurate maps, we value accuracy over performance in the system. Thus, we aim to obtain accurate maps while keeping the performance as high as possible.

1.1.2 Functionality vs. Usability

The system should be able to offer necessary functionalities and user should be able to make use of the application easily. Functionality and usability are the core goals of our system. Bumpster offers functionalities such as generating physically based rendering maps based on specified photo(s), viewing end product(s) in many different angles in the mobile, saving extracted different kind of maps etc. Therefore, functionality is very significant for the system. Also, the system should be usable and user-friendly to allow users to benefit from functionalities easily. A simple and understandable user interface is required in order to enable user to access many functionalities offered by the system. The user may not use all functionalities if user interface is not user-friendly. Therefore, we value usability over functionality in the system. Our aim is to provide all functionalities while keeping the user interface usable. Consequently, creation of an user-interface that allows to access the functionalities of the system easily is important.

1.1.3 Compatibility vs. Extensibility

Extensibility of the system is important for handling upgrades, extensions and future versions of application. Bumpster is open to development and it should be easy to add and enhance features and functionalities. On the other hand, our desire is to make the system as compatible as possible and works in harmony with various API levels. Since Bumpster is dependent on many platforms such as iOS, Android and desktop, we value compatibility over extensibility in the system. Our aim is to make the application work efficiently in various platforms.

1.1.4 Security vs. Cost

Bumpster generates physically based rendering maps from uploaded photo(s). It is important to secure these photos, generated maps and user information. For security, we rely on Amazon S3. It is secure by default. It also supports user authentication to control access to data. Also, securely upload and download user data to Amazon S3 via SSL endpoints using the HTTPS protocol is possible [4]. If extra security is required, we might use Server-Side Encryption(SSE) option to encrypt user data stored. Providing secure environment to the system results in monetary, labor cost and time.

1.2 Interface Documentation Guidelines

In the documentation, the form of class name is ‘ClassName’ and class names are in singular form. The form of variable and method is ‘variableName’ and ‘methodName()’ respectively similar to form of class name. For each class, class name is given with its description and function. After the description and function of class, properties and then methods of class are listed. Properties are listed with their names and types. While defining methods, return values and parameters with their types (if any) are indicated. The outline explained above is given as a summary below:

Class Name
Class Description and Function
Properties(type of property, name)
Methods (return value, name, parameters)

1.3 Engineering Standards

UML is a powerful visual modeling language used for description of components and processes in software which allows an understandable and simple version of system structure, software components and functionalities. It can develop the quality of analysis and design of systems. It supports higher-level development concepts such as collaborations, frameworks, patterns and components. [5] Therefore, in the reports, UML principles are preferred for describing class interface and diagrams, use-case, sequence, activity, package and state diagrams, scenarios, subsystem decomposition, hardware/software mapping and software architecture. IEEE plays an important role in publishing technical works and sponsoring conferences and seminars. [6] It also produces some widely used standards which are also preferred to be used by engineers. The reports also follow IEEE citation guidelines for the references.

1.4 Definitions, Acronyms, and Abbreviations

Amazon S3: Amazon Simple Storage Service

API: Application Programming Interface

UI: User Interface

Authentication: Confirming users' identity

Client: Part of the application that users interact with

IEEE: Institute of Electrical and Electronics Engineers

MVC: Model View Controller

Server: Part of the application that controls data management, computational and logical operations.

UML: Unified Modeling Language

2. Packages

Our system will maintain 6 packages under three subsystems. Client Subsystem: User Interface Package; Server Subsystem: Machine learning package, Map generation package; Network and Data Management Subsystem: Networking Package, Data Management Package.

2.1 Client Subsystem

Client subsystem corresponds to the mobile application of Bumpster. It maintains the presentation layer. The client allows user to fulfill four roles: Create user, view models, take photo of surfaces, and editing model. During all these roles, client subsystem will allow user to access database and store models in database.

The presentations also include the augmented reality experience provided by packages in our choice of mobile development platform. The model which will be retrieved from the Server subsystem will be displayed.

Client subsystem has presentation layer which encompasses the user interface package. The package is the only package in the subsystem, and is responsible from the visualization of all these use cases mentioned.

2.1.1 Presentation Layer

2.1.1.1 User Interface Package

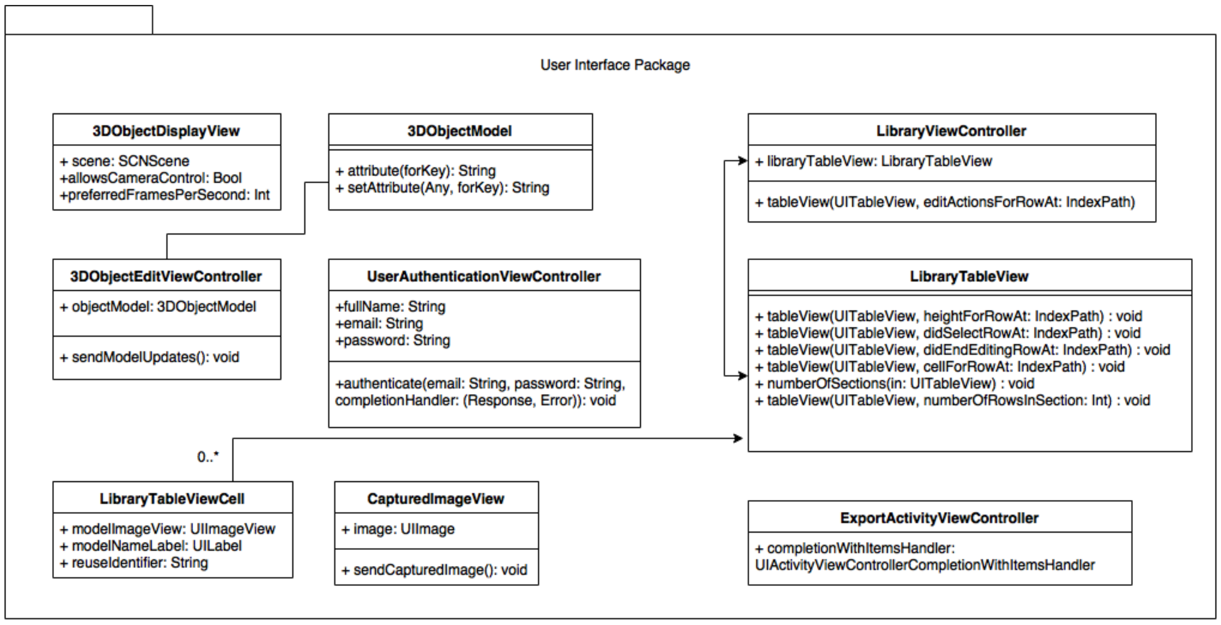

User Interface Package
3DObjectDisplayView
+ scene: SCNScene
+allowsCameraControl: Bool
+preferredFramesPerSecond: Int
3DObjectModel
+ attribute(forKey): String
+ setAttribute(Any, forKey): String
LibraryViewController
+ libraryTableView: LibraryTableView
+ tableView(UITableView, editActionsForRowAt: IndexPath)
3DObjectEditViewController
+ objectModel: 3DObjectModel
+ sendModelUpdates(): void
UserAuthenticationViewController
+fullName: String
+email: String
+password: String
+authenticate(email: String, password: String, completionHandler: (Response, Error)): void
LibraryTableView
+ tableView(UITableView, heightForRowAt: IndexPath) : void
+ tableView(UITableView, didSelectRowAt: IndexPath) : void
+ tableView(UITableView, didEndEditingRowAt: IndexPath) : void
+ tableView(UITableView, cellForRowAt: IndexPath) : void
+ numberOfSections(in: UITableView) : void
+ tableView(UITableView, numberOfRowsInSection: Int) : void
0..*
LibraryTableViewCell
+ modelImageView: UIImageView
+ modelNameLabel: UILabel
+ reuseIdentifier: String
CapturedImageView
+ image: UIImage
+ sendCapturedImage(): void
ExportActivityViewController
+ completionWithItemsHandler: UIActivityViewControllerCompletionWithItemsHandler

Figure 1: User Interface Package

User Interface package contains necessary view classes, models and methods related to all user interactions. Classes below that end with ‘Controller’ are classes of UIKit, they are related to UI part of the application, not the Controller part. These are named this way due to UIKit naming conventions.

3DObjectDisplayView: This class is a child of SceneKit’s SCNView class. This view is used to display 3D object, and it enables user interaction with the 3D object to view it in 360 degrees.

3DObjectModel: This class is a child of SceneKit’s SCNScene class. It represents a hierarchy of nodes with geometries, lights and cameras that form a displayable 3D scene.

3DObjectEditViewController: This class is a child of UIKit’s UIViewController class. It includes sliders for changing properties of the 3DObjectModel. It sends updated model to back-end when user taps the ‘Done’ button, and this button also dismisses this view.

CapturedImageView: This class is a child of UIKit’s UIImageView class. The image represented in this class is sent to back-end to generate a new model.

ExportActivityViewController: This class is a child of UIKit's UIActivityViewController class. It is used for copying or sharing the model.

UserAuthenticationViewController: This class is a child of UIKit's UIViewController class. Credentials from UITextFields are sent to back-end for authentication.

LibraryViewController: This class is a child of UIKit's UIViewController class. User's list of models are managed in this class. When the user deletes a model, necessary API calls are made here to update the list in the back-end as well.

LibraryTableView: This class is a child of UIKit's UITableView class. User's saved models are represented as a list using this class.

LibraryTableViewCell: This class is a child of UIKit's UITableViewCell class. Each model in LibraryTableView is displayed as a cell using this class.

2.2 Server Subsystem

Server subsystem is a part of our subsystem where critical activities such as the generation/storage and retrieval of models, and the registration of users are handled. Server contains the data of all models stored in the system and their relationship with all the users stored in the database. There are two layers included in this subsystem: Application logic, network and data management layer.

Some important functions to the network and data management layer include checking the credential of the user, retrieving all the models associated with the user, and storing the model generated by the neural network, or modified by the user.

The application layer subsystem will allow our system to generate neural network models. The data which will be sent by the client subsystem will be processed, and send back to the user and the network and data management layer.

Our server subsystem will maintain 5 packages under two layers. Application Logic Layer: Machine learning package, Map generation package; Network and Data Management Layer: Networking Package, Data Management Package.

2.2.1 Application Logic Layer

2.2.1.1 Machine Learning Package

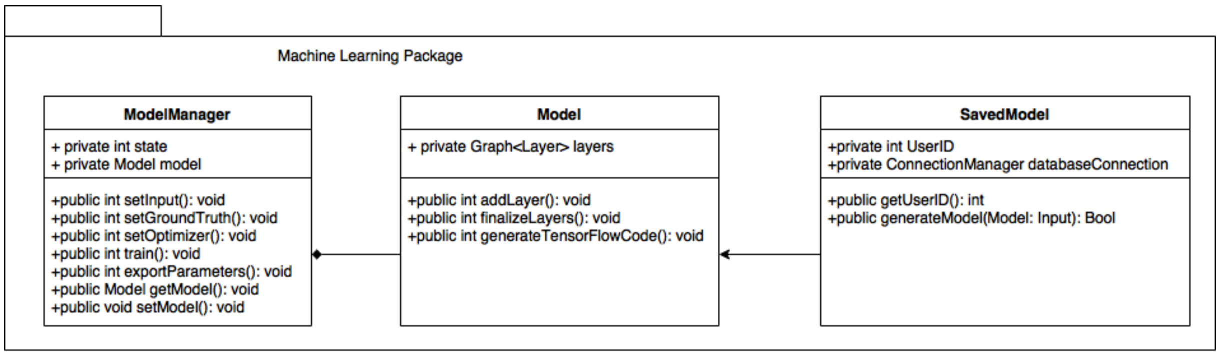

Machine Learning Package
ModelManager
+ private int state
+ private Model model
+public int setInput(): void
+public int setGroundTruth(): void
+public int setOptimizer(): void
+public int train(): void
+public int exportParameters(): void
+public Model getModel(): void
+public void setModel(): void
Model
+ private Graph<Layer> layers
+public int addLayer(): void
+public int finalizeLayers(): void
+public int generateTensorFlowCode(): void
SavedModel
+private int UserID
+private ConnectionManager databaseConnection
+public getUserID(): int
+public generateModel(Model: Input): Bool

Figure 2: Machine Learning Package

This package will behave as a helper package and will not run regularly together with other packages at all times. It will be used to train new models or tweak the existing models to power the map generation package. Models must be compatible with the interface that Map Generation Package uses.

2.2.1.2 Map Generation Package

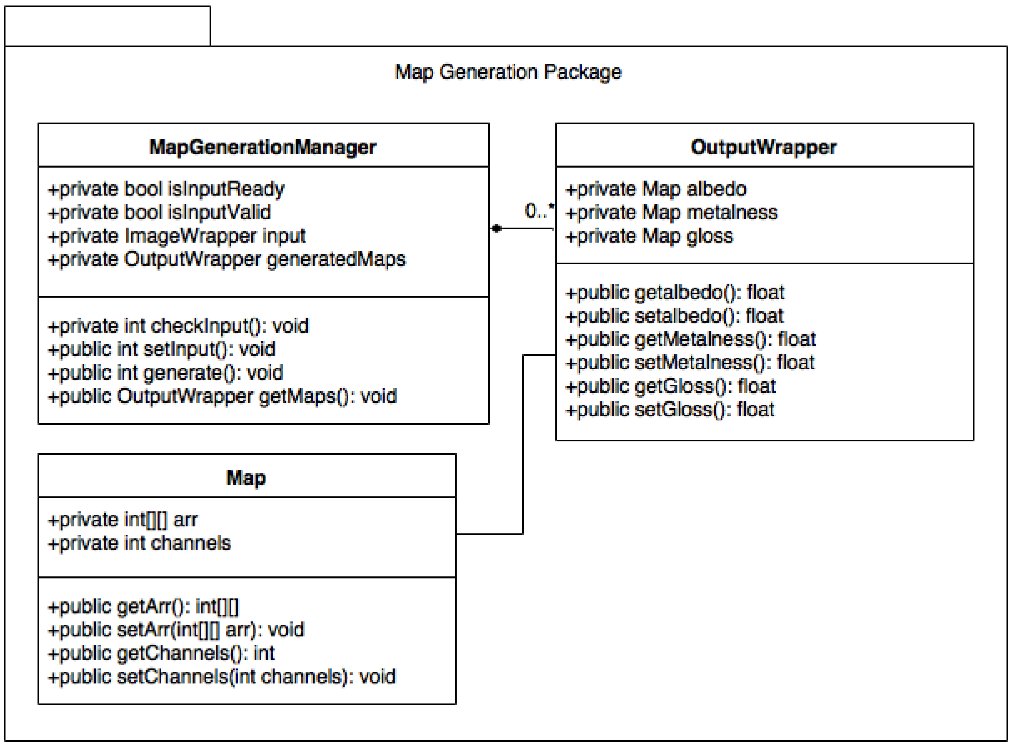
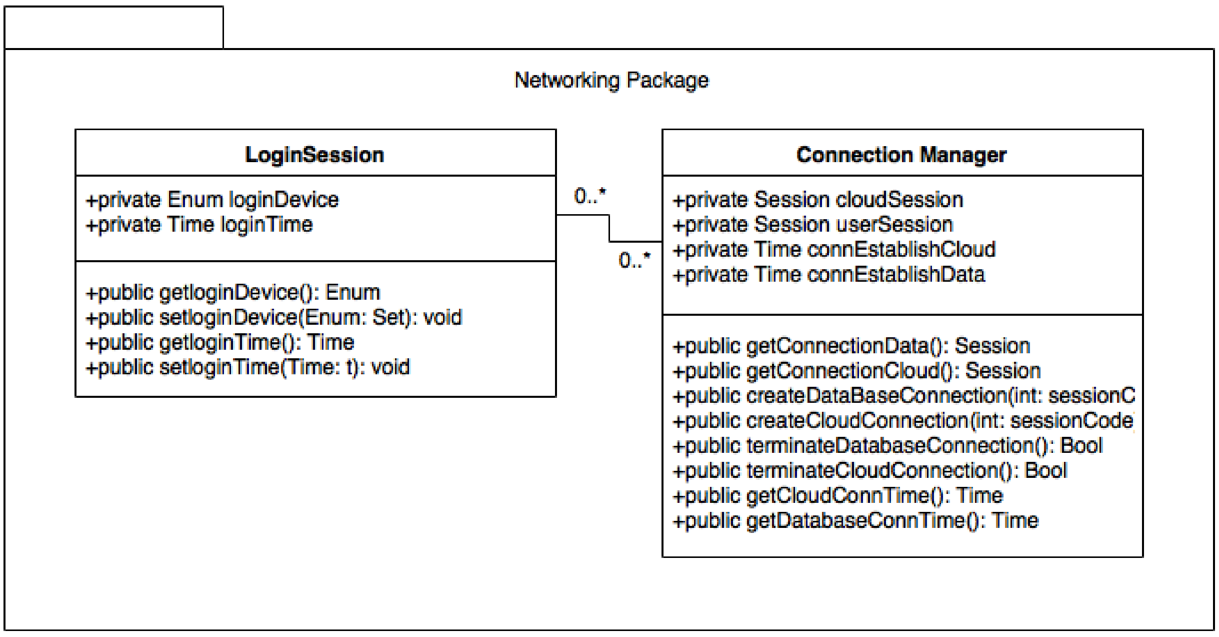

Map Generation Package
MapGenerationManager
+private bool isInputReady
+private bool isInputValid
+private ImageWrapper input
+private OutputWrapper generatedMaps
+private int checkInput(): void
+public int setInput(): void
+public int generate(): void
+public OutputWrapper getMaps(): void
0..*
OutputWrapper
+private Map albedo
+private Map metalness
+private Map gloss
+public getalbedo(): float
+public setalbedo(): float
+public getMetalness(): float
+public setMetalness(): float
+public getGloss(): float
+public setGloss(): float
Map
+private int[][] arr
+private int channels
+public getArr(): int[][]
+public setArr(int[][] arr): void
+public getChannels(): int
+public setChannels(int channels): void

Figure 3: Map Generation Package

Map generation package will be able to generate the PBR maps using models provided by the Machine Learning Package through a common interface. This package is responsible for generation of maps which are then displayed to users and edited by them. There are several types of maps generated for each upload of photograph(s) or scanned images. The package itself is also complex, so it is logical to consider map generation subsystem as a separate subsystem.

2.2.2 Network and Data Management Layer

2.2.2.1 Networking Package



Networking Package
LoginSession
+private Enum loginDevice
+private Time loginTime
+public getloginDevice(): Enum
+public setloginDevice(Enum: Set): void
+public getloginTime(): Time
+public setloginTime(Time: t): void
0..*
0..*
Connection Manager
+private Session cloudSession
+private Session userSession
+private Time connEstablishCloud
+private Time connEstablishData
+public getConnectionData(): Session
+public getConnectionCloud(): Session
+public createDataBaseConnection(int: sessionC
+public createCloudConnection(int: sessionCode
+public terminateDatabaseConnection(): Bool
+public terminateCloudConnection(): Bool
+public getCloudConnTime(): Time
+public getDatabaseConnTime(): Time

Figure 4: Networking Package

Networking package consists of classes related to the management of connections established both with cloud servers, and with database of our system. Networking package is primarily used by our system, and when user requires a connection to Bumpster's database, it can be used interchangeably.

2.2.2.2 Data Management Package

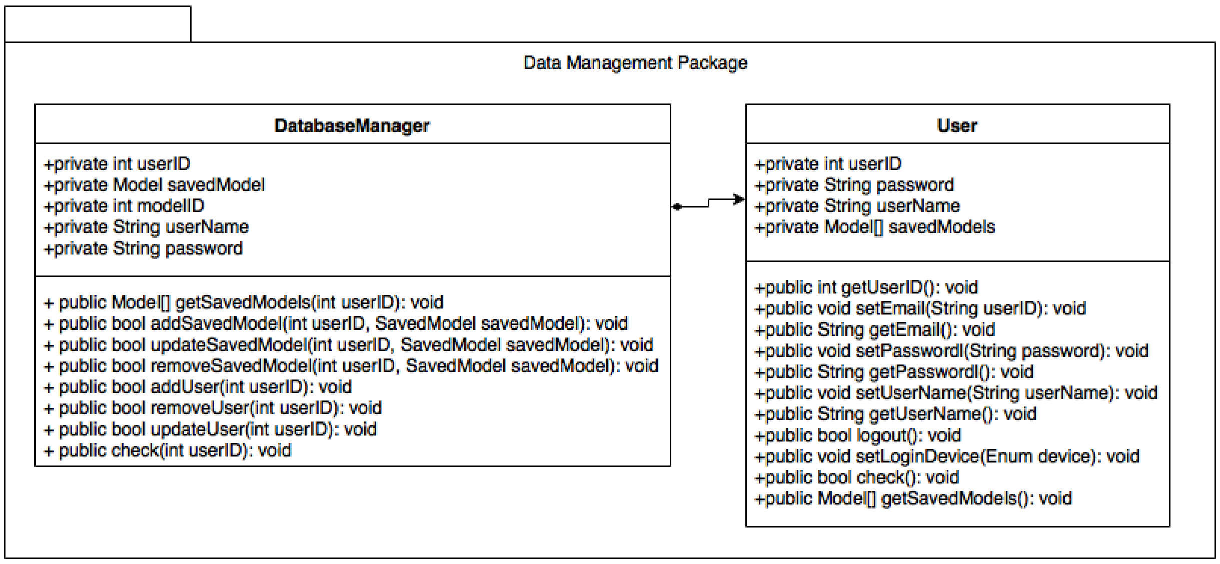

Data Management Package
DatabaseManager
+private int userID
+private Model savedModel
+private int modelID
+private String userName
+private String password
+ public Model[] getSavedModels(int userID): void
+ public bool addSavedModel(int userID, SavedModel savedModel): void
+ public bool updateSavedModel(int userID, SavedModel savedModel): void
+ public bool removeSavedModel(int userID, SavedModel savedModel): void
+ public bool addUser(int userID): void
+ public bool removeUser(int userID): void
+ public bool updateUser(int userID): void
+ public check(int userID): void
User
+private int userID
+private String password
+private String userName
+private Model[] savedModels
+public int getUserID(): void
+public void setEmail(String userID): void
+public String getEmail(): void
+public void setPasswordl(String password): void
+public String getPasswordl(): void
+public void setUserName(String userName): void
+public String getUserName(): void
+public bool logout(): void
+public void setLoginDevice(Enum device): void
+public bool check(): void
+public Model[] getSavedModels(): void

Figure 5: Data Management Package

Data management package will be able to manage user information, saved models and users. This package is specifically responsible for addition, deletion, update of users and save models for each user. Also, it controls user information such as password, username, saved models and ID.

2.3 Controller Subsystem

The controller subsystem includes three packages that are all in one layer: User Authentication Package, Workbench Package and Library Package. This subsystem is a bridge between the Client Subsystem and Server Subsystem, and is equivalent to the Controller in the MVC design pattern. The functions called from this subsystem's classes will call the Server Subsystem's classes, therefore it is a wall between the server and the client.

2.3.1 User Authentication Control Package

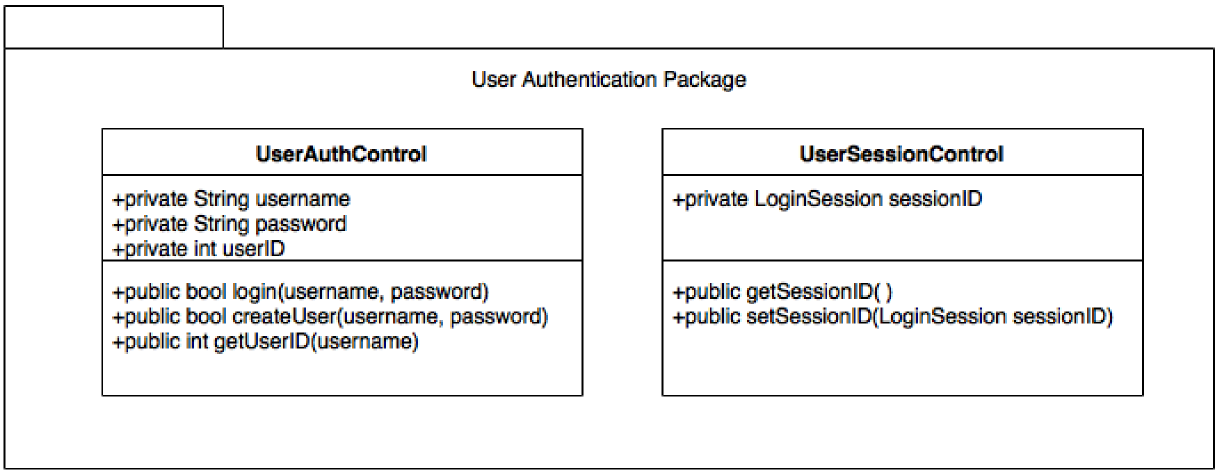

User Authentication Package
UserAuthControl
+private String username
+private String password
+private int userID
+public bool login(username, password)
+public bool createUser(username, password)
+public int getUserID(username)
UserSessionControl
+private LoginSession sessionID
+public getSessionID( )
+public setSessionID(LoginSession sessionID)

Figure 6: User Authentication Control Package

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 of classes of this package in order to access the Data Management Package classes and therefore the database itself. After the authentication is performed, login session will be held in the LoginSession class, which is located in the Networking Package.

2.3.2 Workbench Control Package

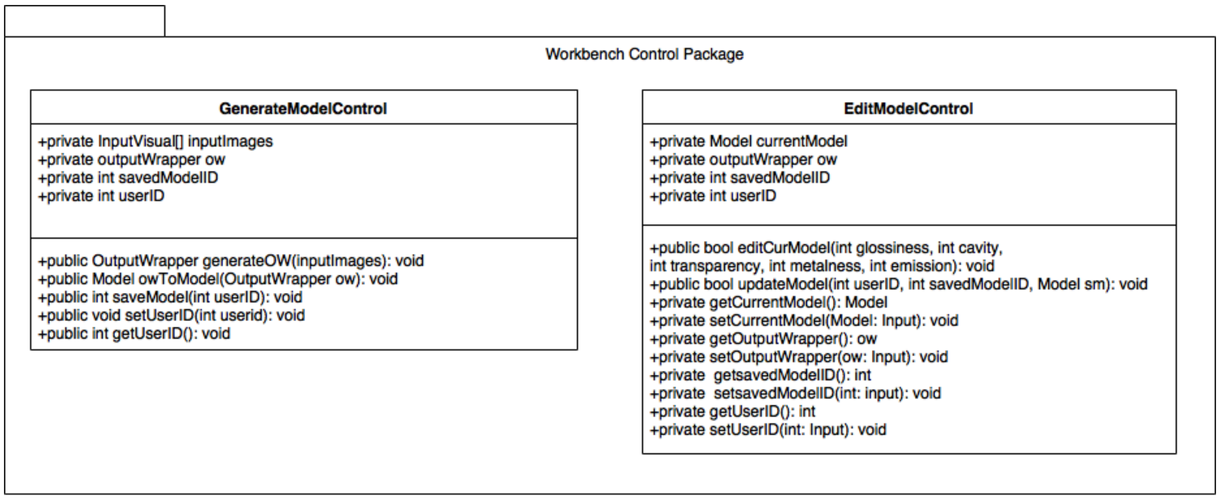
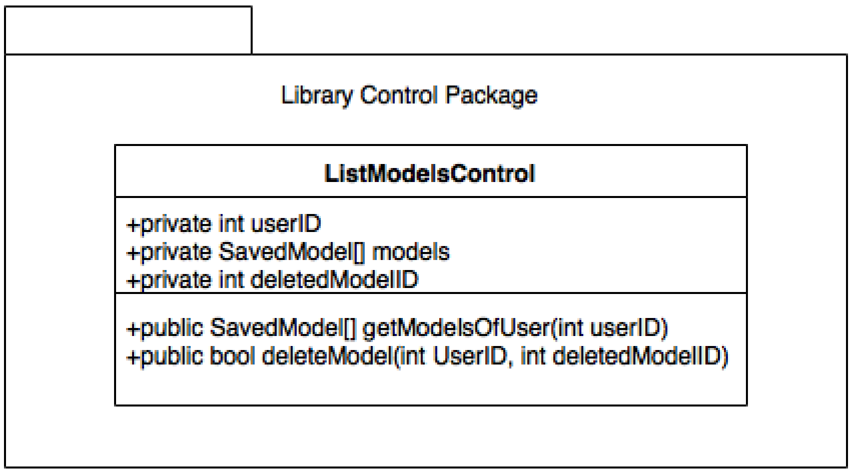

Workbench Control Package
GenerateModelControl
+private InputVisual[] inputImages
+private outputWrapper ow
+private int savedModelID
+private int userID
+public OutputWrapper generateOW(inputImages): void
+public Model owToModel(OutputWrapper ow): void
+public int saveModel(int userID): void
+public void setUserID(int userid): void
+public int getUserID(): void
EditModelControl
+private Model currentModel
+private outputWrapper ow
+private int savedModelID
+private int userID
+public bool editCurModel(int glossiness, int cavity, int transparency, int metalness, int emission): void
+public bool updateModel(int userID, int savedModelID, Model sm): void
+private getCurrentModel(): Model
+private setCurrentModel(Model: Input): void
+private getOutputWrapper(): ow
+private setOutputWrapper(ow: Input): void
+private getsavedModelID(): int
+private setsavedModelID(int: input): void
+private getUserID(): int
+private setUserID(int: Input): void

Figure 7: Workbench Control Package

The Workbench Control Package will include all of the Controller classes that will allow the functionalities of generating the models by taking photos, receiving the 3D models from the Server Subsystem's Map Generation Package and editing them. After the models are saved after creation or editing, this package will be in communication with the Data Management Package to access the storage and perform the suitable update operations on models and will be in communication with the OutputWrapper and SavedModel classes of the Map Generation Package to save the generated or updated model to the storage.

2.3.3 Library Control Package



Library Control Package
ListModelsControl
+private int userID
+private SavedModel[] models
+private int deletedModelID
+public SavedModel[] getModelsOfUser(int userID)
+public bool deleteModel(int UserID, int deletedModelID)

Figure 8: Library Control Package

This package will include all of the controller classes that will be a bridge between the Library View Package and the functionalities of viewing the current user's previously generated models and deleting them if desired. It will be in communication with the DatabaseManager class of the Data Management Package to access the storage and perform deletion and querying.

3. Class Interfaces

3.1 Client

3.1.1 Presentation Layer

3.1.1.1 User Interface Package

class 3DObjectDisplayView extends SCNView
This class is a child of SceneKit's SCNView class. This view is used to display 3D object, and it enables user interaction with the 3D object to view it in 360 degrees.
Properties
var scene: SCNScene var allowsCameraControl: Bool var preferredFramesPerSecond: Int

class 3DObjectModel extends SCNScene
This class is a child of SceneKit's SCNScene class. It represents a hierarchy of nodes with geometries, lights and cameras that form a displayable 3D scene.
Methods
func attribute(forKey: String) func setAttribute(Any?, forKey): String)

class 3DObjectEditViewController extends UIViewController
This class is a child of UIKit's UIViewController class. It includes sliders for changing properties of the 3DObjectModel. It sends updated model to back-end when user taps the 'Done' button, and this button also dismisses this view.
Properties
var objectModel: 3DObjectModel
Methods
func sendModelUpdates()

class CapturedImageView extends UIImageView
This class is a child of UIKit's UIImageView class. The image represented in this class is sent to back-end to generate a new model.
Properties
var image: UIImage
Methods
func sendCapturedImage()

class ExportActivityViewController extends UIActivityViewController
This class is a child of UIKit's UIActivityViewController class. It is used for copying or sharing the model.
Properties
var completionWithItemsHandler: UIActivityViewControllerCompletionWithItemsHandler

class UserAuthenticationViewController extends UIViewController
This class is a child of UIKit's UIViewController class. Credentials from UITextFields are sent to back-end for authentication.
Properties
let fullName: String let email: String let password: String
Methods
func authenticate(with email: String, password: String, completionHandler: (Response?, Error?) -> Swift.Void)

class LibraryViewController extends UIViewController
This class is a child of UIKit's UIViewController class. User's list of models are managed in this class. When the user deletes a model, necessary API calls are made here to update the list in the back-end as well.
Properties
let libraryTableView: LibraryTableView
Methods
func tableView(UITableView, editActionsForRowAt: IndexPath)

class LibraryTableView extends UITableView
This class is a child of UIKit's UITableView class. User's saved models are represented as a list using this class.
Methods
func tableView(UITableView, heightForRowAt: IndexPath) func tableView(UITableView, didSelectRowAt: IndexPath) func tableView(UITableView, didEndEditingRowAt: IndexPath) func tableView(UITableView, cellForRowAt: IndexPath) func numberOfSections(in: UITableView) func tableView(UITableView, numberOfRowsInSection: Int)

class LibraryTableViewCell extends UITableViewCell
This class is a child of UIKit's UITableViewCell class. Each model in LibraryTableView is displayed as a cell using this class.
Properties
var modelImageView: UIImageView var modelNameLabel: UILabel var reuseIdentifier: String

3.2 Server Subsystem

3.2.1 Application Logic Layer

3.2.1.1 Machine Learning Package

class ModelManager
this class provides an interface to TensorFlow to initialize and update model parameters.
Properties

private int state;
private Model model;

Methods

public int setInput();
public int setGroundTruth();
public int setOptimizer();
public int train();
public int exportParameters();
public Model getModel();
public void setModel();

class Model
this class provides a Java descriptor that generates a Keras model.
Properties
private Graph<Layer> layers;
Methods

public int addLayer();
public int finalizeLayers();
public int generateTensorFlowCode();

class SavedModel
this class provides the query generation related to database operations
Properties

private int UserID;
private ConnectionManager databaseConnection;

Methods

public getUserID(): int;
public generateModel(Model: Input): Bool;

3.2.1.2 Map Generation Package

class MapGenerationManager
this class enables straightforward TensorFlow library calls to generate PBR maps from a given Input
Properties

private bool isInputReady;
private bool isInputValid;
private ImageWrapper input;
private OutputWrapper generatedMaps;

Methods

private int checkInput():
public int setInput():
public int generate():
public OutputWrapper getMaps():

class OutputWrapper
this class encapsulates PBR maps
Properties

private Map albedo;
private Map metalness;
private Map gloss

Methods
getter, setter

class Map
this class wraps 2d bitmap encoding
Properties

private int[][] arr;
private int channels;

Methods
getter, setter

3.2.2 Network and Data Management Layer

3.2.2.1 Networking Package

Class ConnectionManager
this class manages the connections established to the cloud servers and database for our system
Properties

private Session cloudSession;
private Session userSession;
private Time connEstablishCloud;
private Time connEstablishData;

Methods

public getConnectionData(): Session;
public getConnectionCloud(): Session;
public createDataBaseConnection(int: sessionCode): Bool;
public createCloudConnection(int: sessionCode): Bool;
public terminateDatabaseConnection(): Bool;
public terminateCloudConnection(): Bool;
public getCloudConnTime(): Time;
public getDatabaseConnTime(): Time;

Class LoginSession
this class saves login session of user.
Properties

private Enum loginDevice;
private Time loginTime;

Methods
getters, setters

3.2.2.2 Data Management Package

Class DatabaseManager
this class is related to addition, deletion, update of models and users
Properties

private int userID;
private Model savedModel;
private int modelID;
private String userName;
private String password;

Methods

public Model[] getSavedModels(int userID);
public bool addSavedModel(int userID, SavedModel savedModel);
public bool updateSavedModel(int userID, SavedModel savedModel);
public bool removeSavedModel(int userID, SavedModel savedModel);
public bool addUser(int userID);
public bool removeUser(int userID);
public bool updateUser(int userID);
public check(int userID);

Class User
this class represents a user of the application, a user is identified with username and password
Properties

private int userID;
private String password;
private String userName;
private Model[] savedModels;

Methods

public int getUserID();
public void setEmail(String userID);
public String getEmail();
public void setPassword(String password);
public String getPassword();
public void setUserName(String userName);
public String getUserName();
public bool logout();
public void setLoginDevice(Enum device);
public bool check();
public Model[] getSavedModels();

3.3 Controller Subsystem

3.3.1 User Authentication Control Package

class UserAuthControl
this class provides wrapper functions for login and create user functionalities
Properties

private String username;
private String password;
private int userID;

Methods

public bool login(username, password);
public bool createUser(username, password);
public int getUserID(username);

class UserSessionControl
this class is to get and set the login session number
Properties
private LoginSession sessionID;
Methods
getter, setter

3.3.2 Workbench Control Package

class GenerateModelControl
this class is a bridge between the UI and the main functionality, which is generating maps
Properties

private InputVisual[] inputImages;
private outputWrapper ow;
private int savedModelID;
private int userID;

Methods

public OutputWrapper generateOW(inputImages);
public Model owToModel(OutputWrapper ow);
public int saveModel(int userID);
public void setUserID(int userid);
public int getUserID();

class EditModelControl
this class is a bridge between the UI and the edit functionality
Properties

private Model currentModel;
private outputWrapper ow;
private int savedModelID;
private int userID;

Methods

public bool editCurModel(int glossiness, int cavity, int transparency, int metalness, int emission);
public bool updateModel(int userID, int savedModelID, Model sm);
getters and setters

3.3.3 Library Control Package

class ListModelsControl
this class is a bridge between the UI and the Data Management Package to list the models and delete chosen ones
Properties

private int userID;
private SavedModel[] models;
private int deletedModelID;

Methods

public SavedModel[] getModelsOfUser(int userID);
public bool deleteModel(int UserID, int deletedModelID);

4. Glossary

Library: Collection of models generated by a specific user.

Workbench: Operations related to editing a specified model.

5. References

[1] ”Eliminate Texture Confusion: Bump, Normal and Displacement Maps”. https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps. [Accessed: Oct. 7, 2017].

[2] "CrazyBump - CrazyBump Professional”. https://www.creativetools.se/software/3d-software/crazybump-crazybump-professional. [Accessed: Oct. 7, 2017].

[3] Armagan Yavuz. Founder & CEO of TaleWorlds Entertainment. Interview. Oct. 6, 2017.

[4] “Amazon Simple Storage Service”. https://aws.amazon.com/s3/?hp=tile&so-exp=below. [Accessed: Feb. 02, 2018].

[5] "What is Unified Modeling Language(UML)?". https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: Feb. 02, 2018].

[6] "Overview of the IEEE 802.11 Standard". http://www.informit.com/articles/article.aspx?p=24411.[Accessed: Feb. 01, 2018].