BILKENT UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

CS491 – SENIOR DESIGN PROJECT

PROJECT *inlight*

HIGH LEVEL DESIGN REPORT

07.01.2015

Ahmet Yavuz Karacabey

Latif Uysal

Yiğit Özen

Süleyman Kılınç

TABLE OF CONTENTS

1. INTRODUCTION

Recent developments in technology made it possible to combine virtual and real world together. As the definition suggest, this is called augmented reality. Augmented reality is used to create a wide range of applications and services for almost everyone today. These include head mounted displays that perform like a smart phone (Google Glass for example) or gaming devices such as the Oculus Rift that provides the users with stereoscopic view as well as motion tracking. However these examples merely cover a fraction of Augmented Reality applications today.

Our project is an example of such case. We are creating an Android application for photorealistic rendering of different materials on phone or tablet screen. The application will display fabric samples on the screen and replace the actual fabric samples given at furniture sellers. The target is to make the difference so seamless that the displayed material under the same lighting conditions will look greatly alike.

The project is composed of modules, each tasked with a certain role in the flow of data. There are roughly 3 modules in the current system. Each of them will be thoroughly analyzed in the following parts. However the workflow in general is as follows. First module captures images from the front facing camera of device through a fisheye lens (for a field of view) and sends it to the second module. Second module processes the image captured with fisheye lens using the camera profile provided. Third module then uses the computed information to render the light on the material to be displayed. All of these are to be done in real time with minimal delay to achieve maximum usability.

The range of materials that can be rendered on the device is marginally infinite. So for this reason it is crucial to put our best effort to cover every possible case as best as possible. For example in case the device is required the render a fabric with engravings, this will require more than just 2D computation of light effect but also taking into consideration of these high and low spots in the material. In such a case, in order not to fall short of providing a photorealistic render, the application needs to support bump mapping or another such technique to handle the material. In the following sections these issues will be explained and analyzed in more detail.

1.1 PURPOSE OF THE SYSTEM

The project is meant to simulate realistic lighting effects on the various kinds of materials to be displayed on the device's screen. This is called photorealism. Photorealism is the key ingredient in order to achieve a computer generated image that resembles to the real one. Thus, the primary purpose of our project is to generate photorealistic images of different materials by calculating the light sources present at the location of the device.

Once able to produce accurate renders of the materials to be simulated, this application then can be used by the decorators and furnishing companies. As stated earlier, the application is to replace the actual fabric samples given to customers at furniture shops and such. So, in this manner, the long term purpose of the system is to succeed at taking the place of a fabric sample.

1.2 DESIGN GOALS

Fast Response: As stated earlier, this application is meant to replace actual fabric samples given out to customers at furniture shops, therefore it needs to feel like an actual fabric sample. For this, all of the computations must be done in real time and any change in lighting should be reflected immediately.

Portability: The application needs to perform accurately and smoothly on different Android devices. Device-dependent factors should therefore be handled in the program code accordingly. Additionally, different Android versions should be supported to a reasonable range as well.

Robustness: Due to its nature, the application will have to handle countless cases of lighting and material simulations. The underlying logic and application itself needs to be robust for all these cases and should output accurate results.

Manageable: The materials (fabric samples) that are going to be displayed in the application are subject to change. Therefore the application needs to employ a design that is compatible with such change. It should be trivial task for the application to load and use the new materials.

Modularity: The application is composed of various modules. Each of these modules needs to be working on their own independent of each other to allow reusability and ease of modification. This will also allow extensibility since each module can be modified as wished to employ new features.

Ease of use: The application is targeted at the average smart phone / tablet user. So the application should not be complicated and counterintuitive. Menus, layout and the execution cycle of the program should therefore be easy to understand and use.

Extensibility: The application we are developing needs to be compatible with future extensions. These future extensions are likely to be added to the underlying logic of the application, so it is important to design the code in such a way to allow improvements.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Fisheye Lens: A camera lens type that increases the coverage of the camera through optical manipulation.

Real-time process: A process that produces output at the same time of execution or with very minimal delay.

Fabric Sample: Piece of fabric that is given to customers at furniture shops to allow the customers to try out and see.

Photorealism: Rendering of computer generated images in a realistic fashion by making use of lighting in the environment.

Bump Mapping: A technique for creating 3D bumps on a surface by distorting the normal vectors on the surface.[7]

AR: Augmented Reality.

1.4 OVERVIEW

The primary goal of our application is to create photorealistic renders of various types of materials by computing the lighting effects in the environment. In order to achieve this, the device makes use of the front facing camera and a fisheye lens for a greater field of view. All of the computations are done in real time, so moving the device results in immediate and accurate changes in the render produced. See Figure 1 for an illustration of how the application works.

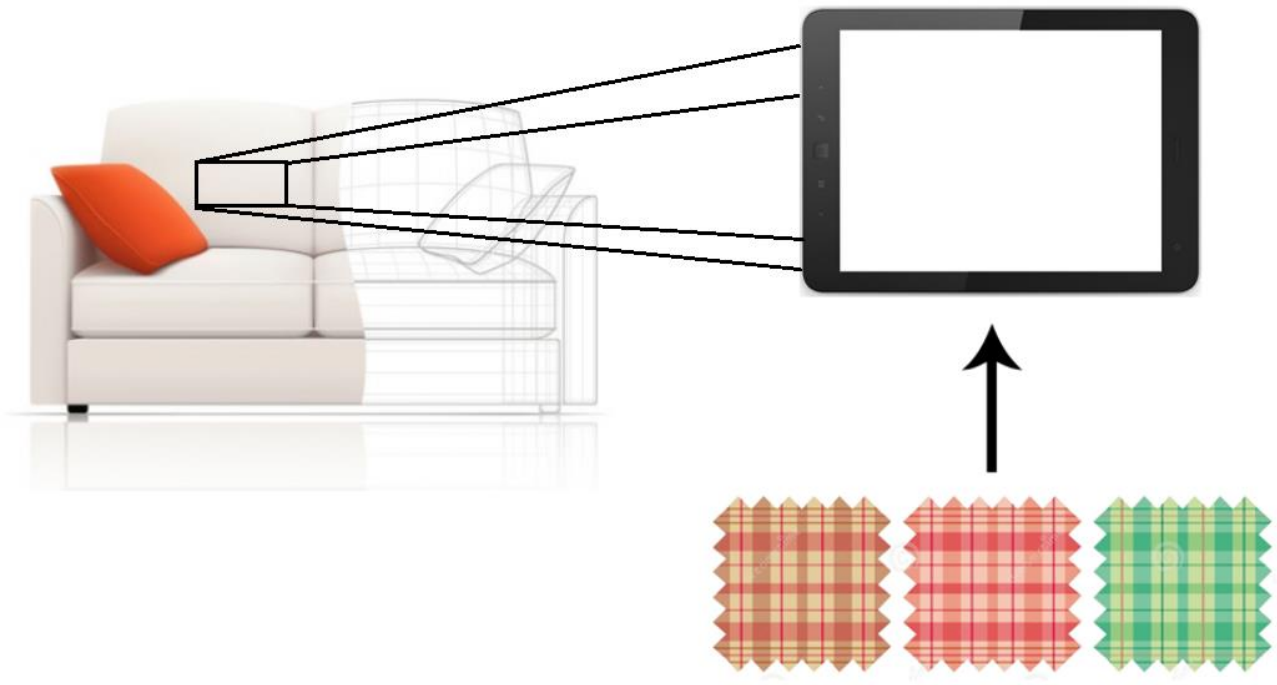

Figure 1 - System at Work

The final product is intended to replace fabric samples given at furniture shops. The user will be able to test and see many different fabric samples on their Android device. Menus, functions and the general layout of the program will be designed by considering the average smart device user, so the application can be used intuitively and easily. Figure 2 depicts the user – system interaction in Use Case diagram.

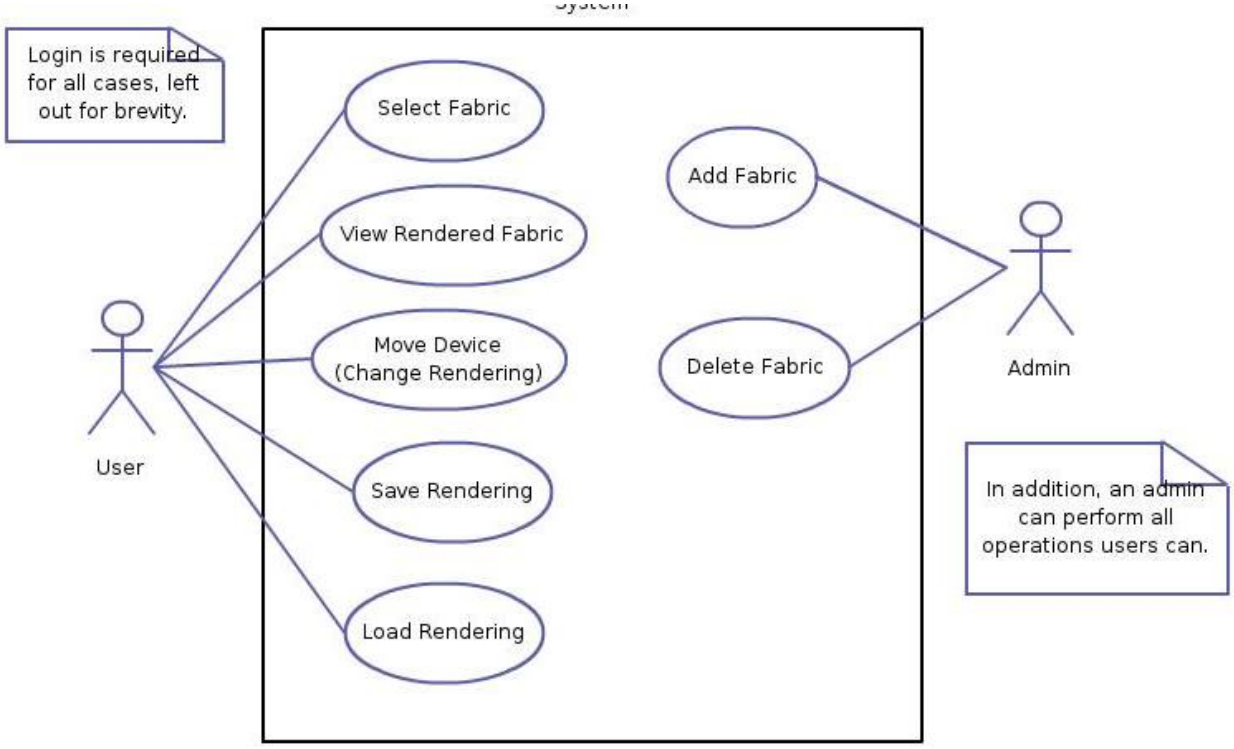

Figure 2 - User - System Interaction

The application needs to be compatible with different Android devices and versions. Additionally, it should be easy to manage, that is, it should be easy to load and use new materials and samples in the program. The code is going to be organized into modules to allow extensibility, ease of implementation and modification. The organization of Problem Domain classes and the relationship between different components are shown in Figure 3.

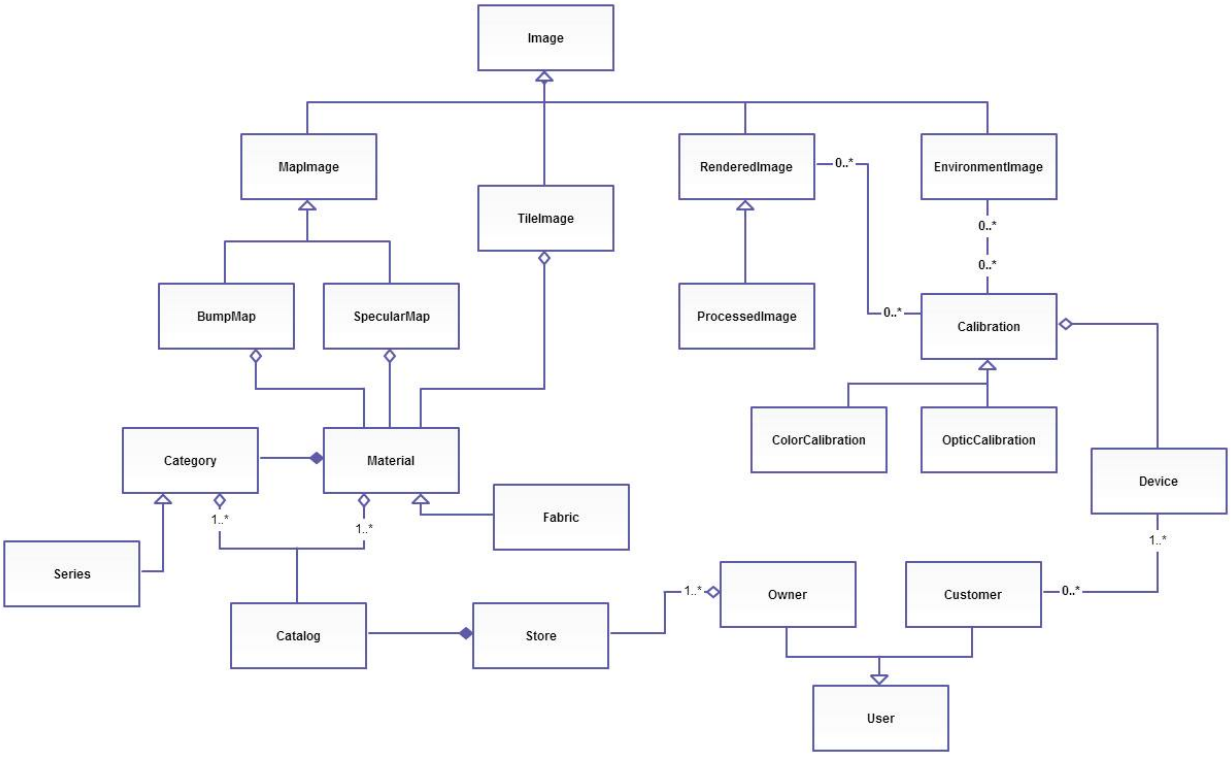

Figure 3 - Problem Domain Classes

2. PROPOSED SOFTWARE ARCHITECTURE

2.1 ARCHITECTURE OVERVIEW

Client-Server Architecture

The materials need to be stored in a central database and published to the clients since the database is dynamic and can grow larger than the client can store. Therefore we implement a client-server architecture involving the main client application and a central database server. The server publishes the data with a RESTful API, and the client and the server talks through HTTP. Apache HTTP Server answers to requests on the server side, and a single subsystem manages the communication with the server on the client side.

Model-View-Controller Architecture

The user interface follows Model-View-Controller (MVC) architecture for separation of concerns and easier implementation. The Model component interacts with the data-producing or

data-publishing parts of the application.[9] The View component handles displaying content and user interactions. The Controller component supervises the model and the view, manages the communication between them and dispatches the user actions to the responsible subsystems.

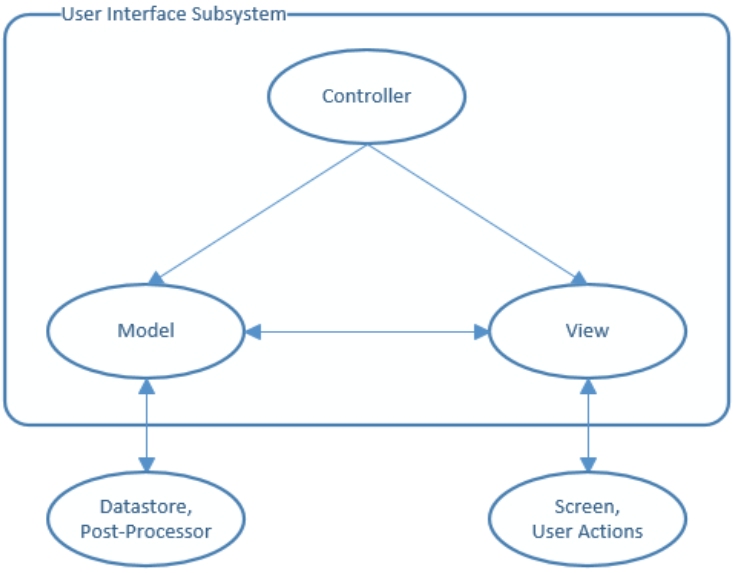

Figure 4 - MVC Architecture of the System

Three-Tier Architecture

The subsystem decomposition can be easily fit to Three-Tier Architecture with *Presentation Tier*, *Logic Tier* and *Data Tier*. Logic Tier stays between the other two. The business logic involving photo shooting, rendering and post-processing happens in the Logic Tier. The material data is handled in the Data Tier, as well as the rendering and calibration data that needs to be stored.[10] The Presentation Tier is responsible for displaying information which has its own MVC architecture. Its model component communicates with the Logic Tier, and the View component communicates with the device screen.

2.2 SUBSYSTEM DECOMPOSITION

Subsystem decomposition of the project is depicted in Figure 5. Each of these subsystems are explained in detail in following parts.

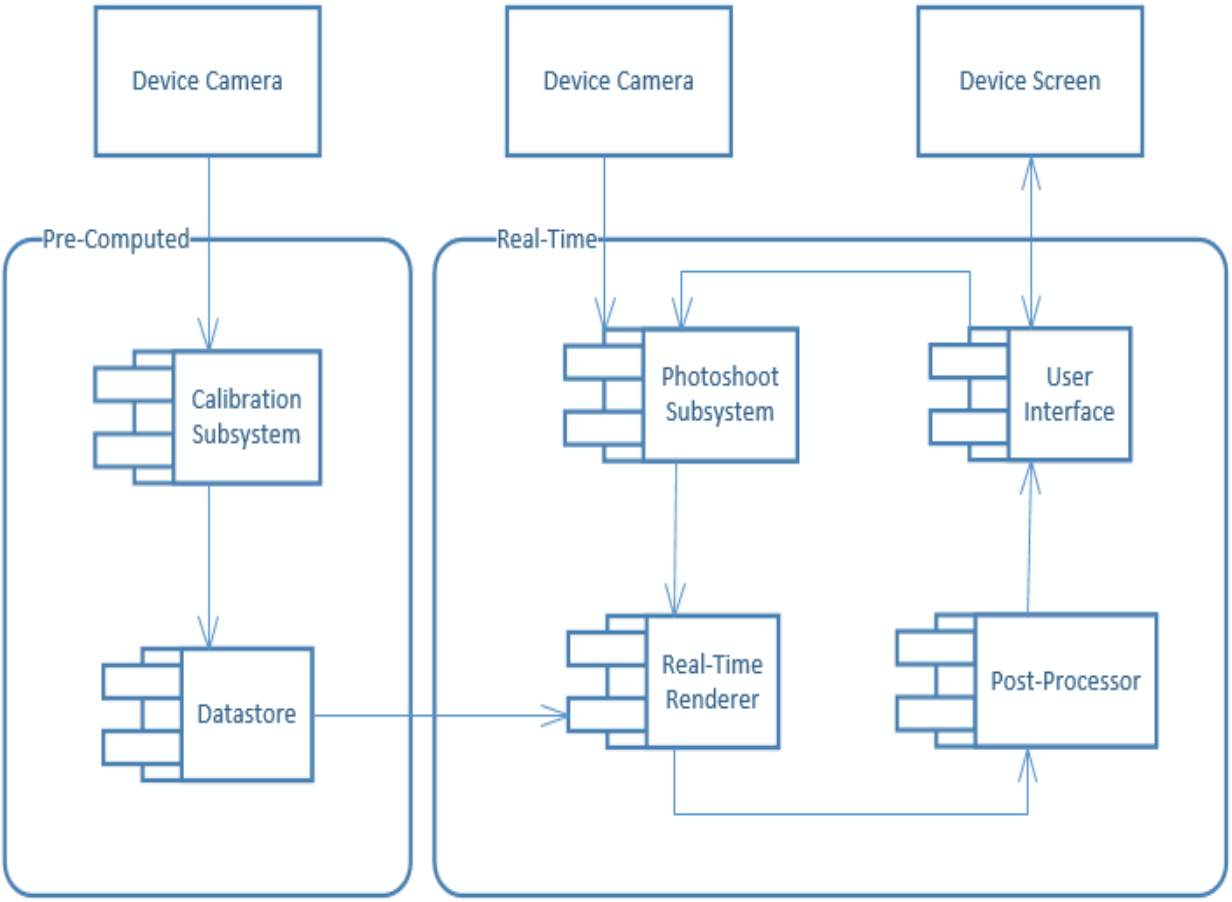

Figure 5 - Subsystem Decomposition

Calibration Subsystem: The mapping from 2D coordinates of the image data coming from the camera to 3D world coordinates is required by the renderer to calculate illumination since the illumination formulas include angles. This mapping does not change for a device, so that it is sufficient to make the calculations just once for one device. Calibration Subsystem implements the methods for calculating the mapping. It generates the values that can be plugged in the formulas during real-time rendering.[1] Other invariant values that are processed along with the

mapping during rendering are also pre-calculated by the Calibration Subsystem.[2] Calibration Subsystem sends the resulting data to the Datastore for storing.

Photoshoot Subsystem: Continuous photo shoots that are required for real-time identification of the surrounding environment are handled by the Photoshoot Subsystem.[3] Shutter speed, ISO, High Dynamic Range (HDR) and image resolution are adjusted by the Photoshoot Subsystem according to the requirements of the renderer. Photoshoot Subsystem may control the device's camera and/or post-process the photo to produce an image that satisfies the requirements. The final image is directly sent to the Real-Time Renderer.

Real-Time Renderer: Real-Time Renderer produces an image of the digital material as how it should appear in the surrounding environment. It focusses on illumination and shading.[4][5] The data required to render the image is retrieved from the Datastore and the Photoshoot Subsystem. The rendered image is sent to the Post-Processor.

Post-Processor: Post-Processor retrieves a rendered image from the Real-Time Renderer and makes the last modifications on it, such as color profiling. The modifications done by the Post-Processor are generally device-specific. The properties of the device that affect the appearance of the image on the screen are eliminated by the Post-Processor. Then, the image is sent to the User Interface Subsystem to display it.

User Interface Subsystem: Displaying the catalog of materials to the user, capturing user actions that trigger the renderer, displaying rendered images and actions such as saving and loading rendered images are handled by the UI Subsystem. The UI subsystem communicates with almost all other subsystems as it dispatches user actions to the responsible components.

Datastore: Datastore is responsible for storing and providing material data such as tile images and spectrophotometric properties, and the mappings and transformation matrices that are pre-computed and used by the renderer for illumination calculations.[6] Datastore handles all communication with the database on the server side. It also directly works on device's memory to place the data needed for real-time calculation in an effective manner.

2.3 HARDWARE/SOFTWARE MAPPING

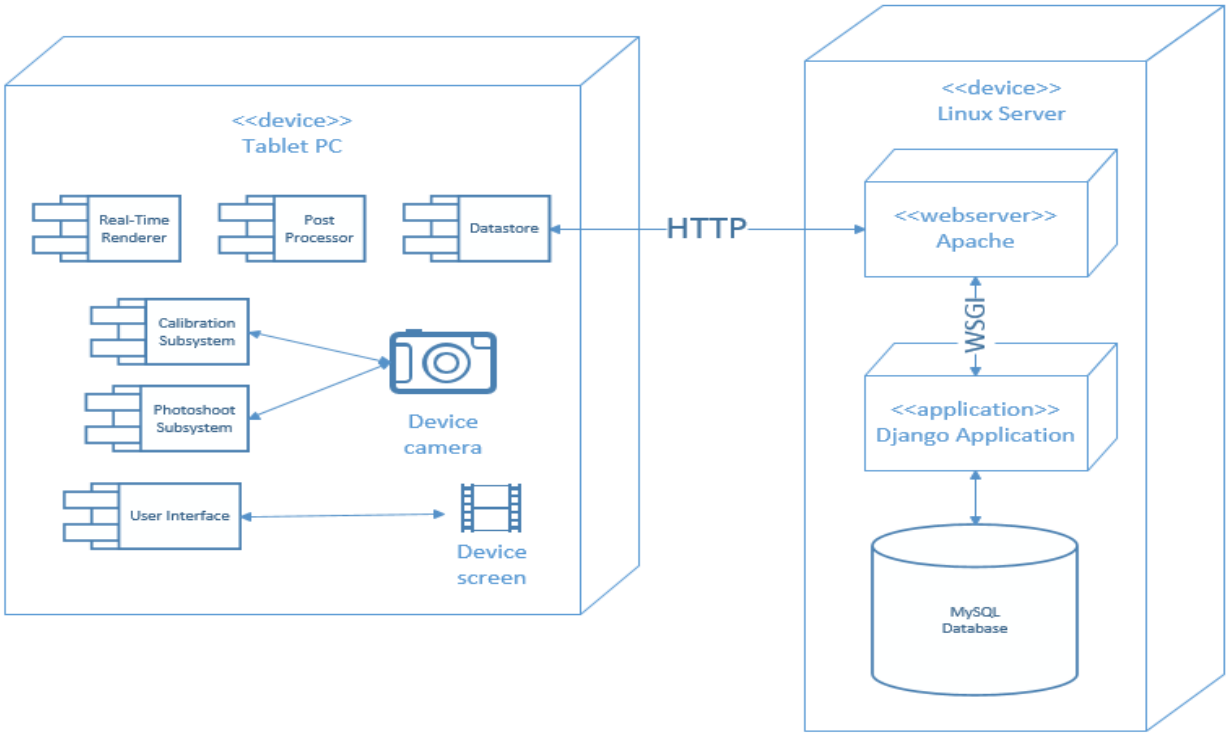

Figure 6 - Hardware/Software Mapping

Django Framework using a MySQL database will be used for its Object-Relational Mapper (ORM), RESTful API provider and built-in admin interface.[8] The database model can be defined using Python classes and the data can be modified using the easy-to-use admin interface. The data is published using RESTful API with JSON or XML serialization, which can be retrieved using HTTP and parsed easily by the client. The application will be running by the Apache HTTP Server using Web Server Gateway Interface (WSGI). The server will be running a Ubuntu operating system which is a Linux distribution, for its efficiency and stability. Also, Python interpreter, MySQL and Apache HTTP Server are either already installed or is easy to install on Ubuntu. The host machine will be a virtual server on Amazon Web Services (AWS) cloud platform. See Figure 6 for Hardware/Software mapping of the system.

The client application resides in a tablet PC. The first target operating system will be Android; iOS support will be considered later. The application will utilize the device's screen, camera and possibly, gyroscope. The application will require an Internet connection to connect to the server

in order to retrieve materials etc. But, using the application offline will also be possible if there exists some materials that were cached in the device, since the rendering process does not require an active Internet connection.

2.4 PERSISTENT DATA MANAGEMENT

Django Framework and MySQL will be used for persistent data as explained in the previous section. Possible tables for the data model are material, category, color calibration, optic calibration, device and user. The Material table will be the most comprehensive table we will use as it will contain the tile image, specular and bump mapping and spectrophotometric properties of the material. Categories will be used to categorize materials, for example as series of products. The User table is required for access control as will be explained in the next section. The Device table will be used to store device specific properties that affect photoshooting, rendering or post-processing such as camera lens or screen properties.

2.5 ACCESS CONTROL AND SECURITY

In the table below, the user types and associated control rights are given.

Table 1 - User Types and Roles in MySQL DB

Role	Fabric
User	Select View Save
Admin	Select View Save Add Delete Modify

2.6 GLOBAL SOFTWARE CONTROL

The control flow of the software is quite simple because the bulk of the application code consists of intensive computations codes. User interface interactions are handled in an event-driven fashion. When the user selects a particular fabric, its specific texture is fetched from the server. The heart of the application is viewing the selected fabric. At the preprocessing phase, we transform image coordinates to the corresponding world coordinates.

Every time an image is received from the camera, using the data from calibration, the coordinates of the light sources are computed. So the fabric texture is modified according to the spectral and diffuse lighting effects of these perceived light sources.[11] Then the view is updated with correct light and shading. We plan to use 30fps for this viewing loop. The user can opt to save a particular view any time on their device and review it later.

The admin can upload new fabrics onto the server, or modify or delete existing fabrics. This is similarly handled in event driven fashion with asynchronous communications with the server.

2.7 BOUNDARY CONDITIONS

Startup and Shutdown

During startup of clients, the device is checked if internet connection is available and the server is connected. Shutdown of clients requires no special action.

Exception Handling

Exception handling is quite simple. Hardware malfunctions are checked when camera the used. Images are also checked for compliance with particular image formats.

3. SUBSYSTEM SERVICES

Calibration Subsystem: Provides pre-computed data for Datastore.

Photoshoot Subsystem: Provide HDRI image for real Real-Time Renderer subsystem.

Real-Time Renderer: Provides rendered image for Post-Processor.

Post-Processor: Provide ready-to-display image for User Interface subsystem.

User Interface Subsystem: Provides user input for other subsystems.

DataStore: Provides static data for Real-Time Renderer subsystem.

4. REFERENCES

[1]: Kautz, J., Vazquez, P., Heidrich, W., Seidel, H.P.: A unified approach to prefiltered environment maps. In: EuroGraphics Rendering Workshop, pp. 185–196 (2000).

[2]: Ramamoorthi, R., Hanrahan, P.: An efficient representation for irradiance environment maps. In: Proc. of SIGGRAPH, pp. 497–500 (2001).

[3]: Hajisharify S., Kronanderz J., Miandjix E., Unger J. Real-time image based lighting with streaming HDR-light probe sequences. SIGRAD (2012).

[4]: Kautz, J., Daubert, K., Seidel, H.P.: Advanced environment mapping in VR applications. Computers & Graphics 28, 99–104 (2004).

[5]: King, G.: Real-time computation of dynamic irradiance environment maps; GPU Germs, vol. 2(4), pp. 98–105. Addison-Wesley Professional, London, UK (2005).

[6]: Kim J., Hwang Y., Hong H. Using Irradiance Environment Map on GPU for Real-Time Composition. 2007.

[7]: Bump Mapping: Blinn, James F. Simulation of Wrinkled Surfaces Computer Graphics, Vol. 12 (3), pp. 286-292 (1978)

[8]: Django Web Development Framework: Overview, Django, https://www.djangoproject.com/start/overview/ (Accessed: 30 December 2014)

[9]: MVC: Bruegge, B., Dutoit A.H., System Design: Decomposing the System in Object Oriented Software Engineering Using UML Patterns, Boston, Prentice Hall, pp. 240. (2010)

[10]: Calibration: Brant, S., Kannala, J., A Generic Camera Calibration Method for Fish-Eye Lenses, Helsinki, Finland, Helsinki University of Technology Laboratory of Computational Engineering, (2004)

[11]: Texture mapping: McReynolds, T., Blythe, D., Advenced Rendering in Programming with OpenGL, ACM Siggraph, (1997)