BILKENT UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

SENIOR DESIGN PROJECT

inlight

04.05.2015

Project Members

Yiğit ÖZEN

Latif UYSAL

A. Yavuz KARACABEY

Süleyman KILINÇ

Supervisor

Uğur GÜDÜKBAY

Jury Members

Selim AKSOY

Özgür ULUSOY

Table of Contents

1. Introduction

1.1. Executive Summary

Project inlight is an Augmented Reality project targeted at Android smart phones and tablets. The project is going to be used by furniture and decoration industry, and their customers. Our final product is an Android application that simulates fabric samples given to customers at furniture shops on the device screen. The displayed material, that is, the fabric sample, will be rendered in a realistic manner by capturing the lighting in the rendered environment through the device's front facing camera.

The application works as a mobile catalogue for browsing and testing fabrics samples for a furniture shop or home decorator. For modular use and ease of control, image files for fabrics samples are kept in a server where the application will dynamically access during use. Most importantly, the application delivers fabric renders in real time, which contributes to simulate a realistic model of a fabric sample on device screen. Please refer to Appendix A for a graphical representation of the application at work.

1.2. Product Scope

The final product of the project is an Android application to render fabric samples on device screen in a photorealistic manner by making use of the front facing camera of the device for capturing lighting information in the environment. The image files for fabric samples are retrieved from a remote database and not stored locally due to space concerns. Therefore the products of this project include a DBMS to host the fabric image files for the application to access, and the mobile application powered by OpenGL that renders the fabric images on the screen.

The database side of the project hosts image files and any supplementary material such as bump maps and certain fabric related coefficients. The database is not directly accessible for anyone besides the development team. For this reason, any kind of updates, additions or removals

related to fabric images or any data residing in the database will be done in the supervision of development team.

The application side of the project is where the rendering takes places in. Retrieved fabric images from the database are processed in a pipelined manner by constantly capturing the lighting information in the environment using the device camera and then using this information to model how the lighting is going to act on the fabric and finally producing the render. The application features a very simple user interface as it is intended to be practical and straightforward, as close as possible to a real fabric catalog.

1.3. Project Purpose

Augmented Reality is a human-computer interaction format where the perceived reality through computer is augmented in certain ways such as visual and audial. Augmented Reality applications are becoming more and more popular every day. Most of these applications are targeted at mobile platforms such as smart phones, tablets and wearable devices like Google Glass. Augmented Reality, by definition, is intended to make human-machine interactions smoother and easier while delivering content richer in context.

Project inlight is an example of such a case. In our project we are targeting to develop an Android application where the displayed materials (fabric samples) on the device screen are augmented using the front facing camera of the device. The camera will constantly scan the environment to create a mapping of the light sources acting on the device's screen. This lighting map will then be passed to the graphical processing module to calculate how they would act on a real piece of fabric sample. Finally, what is displayed on the device screen will look greatly alike to the actual piece of fabric.

In this sense, the purpose behind our project is to create an application that will abolish the need for real fabric samples given to customers and furniture shops by incorporating the system described above. Our initial surveys with some well-known furniture shops displayed that each piece of fabric sample costs them around $1 depending on the fabric type. In a very simple manner, considering this, if a shop on average hands out 100 samples every day, it costs the company around $3000 every month. Additionally, handing out multiple samples per customer

increases the costs even more. On the customers' side, having access to not only a few but to a very large variety of different samples readily available on their tablets or smart phones come in very practical. This will also spare them the burden of having to depend on the actual physical entity of the fabric sample.

The first objective of our project was to develop a firm basis for photorealistic rendering on the device using the front facing camera. This is the most crucial part of the project as the final product would have to display a physically accurate and realistic representation of the fabric sample. Additionally, one particular reason why this rendering module is important is that this module can be reused easily in any Augmented Reality application for mobile devices. Considering that the application is to be used by the average smart phone or tablet user, interaction with application needs to be straightforward and easy. In the end, the ultimate objective of the project is to produce an easy to use Android application for furniture sellers that accurately render fabrics samples at any given environment.

Currently, there is not any commercially available application that serves as a mobile catalogue for furniture sellers. In this manner, our product is completely unique and has no competitors. However, photorealistic rendering in Augmented Reality is an issue many people and companies are interested in. The method that we are going to incorporate in our product, that is, using the front facing camera for creating photorealistic fabric renders has never been used in a commercial application before and has so far been limited to research use only. In terms of this, our project adapts this concept to an applicable format and imposes certain optimizations such as performance and visual quality on it.

1.4. Tools and Technologies Used

Renderer subsystem within Android application was implemented using Java programming language as a platform requirement and it uses OpenGL ES for hardware-accelerated rendering using the graphics processing unit (GPU). Android Studio IDE is used for developing the application. Android Studio provides single environment for developing, testing and debugging an Android application.

Calibration subsystem was implemented using Python language as a separate software. It utilizes OpenCV computer vision library. PyCharm IDE was used for development of the software.

Git was used for revision control along with GitHub for repository hosting. Both Android Studio and PyCharm have built-in support for both Git and GitHub, of which we also took advantage. Trello, which realizes Kanban paradigm in a web-based application, was used for project management. Google Drive, Google Docs and Microsoft Office Word were used for document creating, sharing and collaborative editing.

Google Nexus 10, Samsung S7580 and LG G2 mobile devices were used for testing and demonstrations. Sunvito universal magnetic detachable lens kit were used as the fisheye lens for mobile devices.

Figure 1: Lens Kit for Mobile Devices

1.5. Glossary

Radiant Flux: Radiant energy emitted, reflected, transmitted or received, per unit time.

Irradiance: Radiant flux received by a surface per unit area.

Radiance: Radiant flux emitted, reflected, transmitted or received by a surface, per unit solid angle per unit projected area.

Photorealism: Rendering that attempts to produce the scene as realistically as possible as in a photograph.

Camera Calibration: Estimating the parameters of a camera model, also called camera resectioning.

2. Technical Details

2.1. Summary

Inlight aims at photorealistic rendering in real-time, using mobile devices. Therefore, photorealism and performance to maintain real-time interaction are the main requirements.

There are two main components for photorealistic rendering:

1. Illumination Model
2. Shading Model

Illumination model consists of modeling the lights, their representation in the software and usage of the representation in rendering. For our purposes we use the technique called **Image-Based Lighting**, in which we use omnidirectional images of the environment to determine the illumination [1]. Shading model, also called reflection model, consists of altering the color of surfaces based on the angles of the incoming lights, angle of the viewer, surface normal and physical properties of the materials. We implement **Physically Based Rendering** which models how light behaves in reality based on principles of physics.

Image-Based Lighting and Physically Based Rendering together form our photorealistic rendering system. The other requirement is performing the rendering operations in real-time. For this purpose, we use **Spherical Harmonics Representation** for lights and reflections, and **Monte Carlo Integration** to calculate those representations [2][3].

Each of the techniques mentioned will be explained in the following subsections.

2.2. Image-Based Lighting (IBL)

Image-Based Lighting (IBL) is the process of illuminating scenes and objects using images of light from the real world. When used effectively, IBL can produce realistic rendered appearances of objects. The steps of IBL are:

1. Capturing real-world illumination as an omnidirectional image.
2. Mapping the illumination onto a representation of the environment.
3. Simulating the light coming from the environment in the scene.

IBL is one of the main components of inlight. A surface on the screen of a mobile device is illuminated by lights coming in front of the device. Therefore the illumination can be captured by using a front-facing camera on the device. Almost all mobile devices of today include a front-facing camera. The environment we are interested in is a hemisphere. This corresponds to 180-degree angle of view in the horizontal plane. Front-facing cameras on mobile devices today provides around 45-degree angle of view. Fish-eye lenses are convenient to increase the angle of view. Such lenses for mobile devices which can increase angle of view to around 165-degree can be found in electronic stores with around $5 price. However, fisheye lenses introduce a high distortion, which makes the calibration of the camera critically important.

2.2.1. Camera Calibration and Light Directions

Lenses usually have some radial distortion. Fisheye lenses have greater radial distortion. Distortion coefficients along with other intrinsic parameters of a camera depend on neither the scene viewed nor the captured image resolution. So that, all images from the camera of the device will have the same distortion effects. Intrinsic parameters of the camera can be calculated from several views of a known calibration pattern [4]. OpenCV library has a calibration module and built-in support for a chessboard as a calibration rig. We used OpenCV and a chessboard to calculate the intrinsic parameters of our camera. Since intrinsics do not change, for any image, 2D points in the image plane can be reprojected to 3D world using the intrinsics. Reprojection provides 3D coordinates in the camera frame, which can be considered as a vector whose direction corresponds to the direction of the light coming from that pixel of the image. We developed a Python application that takes several photoshoots of the chessboard viewed from different angles as the input, and produces the 2D directions array as the output. The directions are stored in a 2D array whose width and height are equal to the image resolution. The array is

downscaled using medians at the same ratio of the downscale of the image itself. Then the direction matrix is stored in a file to be used during rendering.

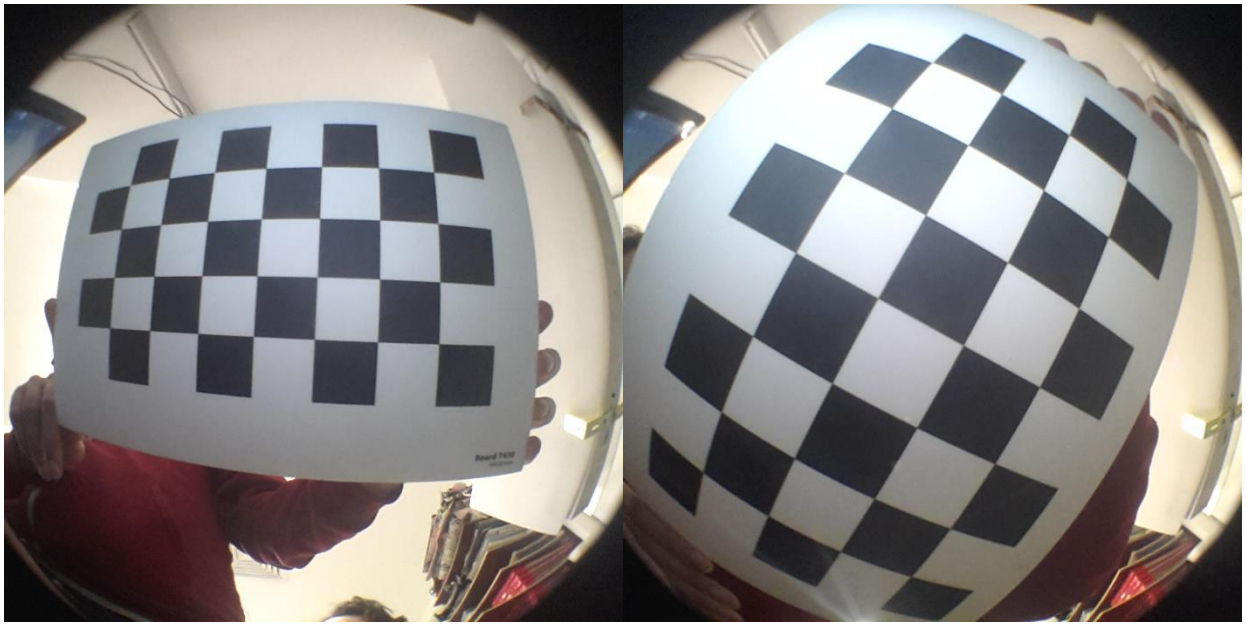

Figure 2: Two images of chessboard pattern during calibrating Nexus 10's camera with fisheye lens

Radiances in the form of RGB values of pixels of the image from the camera together with the directions corresponding to each pixel are sufficient to calculate irradiance, i.e. how much the surface is illuminated [5][6][7].

2.3. Physically Based Rendering (PBR)

Modeling light-matter interaction (shading model) is one of the main components in our framework, the other being modelling illumination. The idea behind Physically Based Rendering (PBR) is to create a user friendly way of achieving a consistent, plausible look under different lighting conditions. PBR models how light behaves in reality, without using multiple ad-hoc models that may or may not work. PBR follows principles of physics, including energy conservation, Fresnel reflections, and how surfaces occlude themselves [8].

2.3.1. Bidirectional Reflectance Distribution Function (BRDF)

Bidirectional Reflectance Distribution Function (BRDF) is a function that defines how light is reflected at an opaque surface. The function takes incoming light direction, and outgoing direction, and returns the ratio of reflected radiance exiting along outgoing direction to the irradiance incident on the surface from incoming direction. BRDFs can be measured directly from real objects using calibrated cameras, but models which makes it editable using a small number of intuitive parameters exist. Inlight makes it possible to use arbitrary BRDFs, which let complex models run in real-time with pre-calculation. In our implementation we use Cook–Torrance model for specular reflection [9]. Cook–Torrance model accurately models the Fresnel effect, increasing the specular reflectivity as the surface turns away from the viewing direction. It also models micro imperfections in the surface, taking into account how they add to and attenuate the specular reflections. In following formulas V is the vector to the viewer, L is the vector to the light, N is the surface normal vector, and H is the half-angle vector. We use the following formula for Cook–Torrance BRDF:

$$k_{spec} = \frac{F\ G\ D}{4 \times (V \cdot N) \times (N \cdot L)}$$

2.3.1.1. Fresnel, F

Fresnel equations describe the behavior of light when moving between media of different refractive indices in which both reflection and refraction of the light may occur. We are concerned with the Fresnel reflection. We use Schlick's approximation to approximate Fresnel reflection.

$$F = f + (1 - f)\ (\ 1 - (H \cdot V))^5$$

where f is the Fresnel parameter of the material.

2.3.1.2. Geometric Attenuation, G

The geometric attenuation factor, G, accounts for the occlusion and shadowing of one microfacet by another [10]. The formula for G is:

$$G = \min\left(1, \frac{2(H \cdot N)(V \cdot N)}{V \cdot H}, \frac{2(H \cdot N)(L \cdot N)}{V \cdot H}\right)$$

Shadowing and masking of a surface are illustrated in Figure 2. Shadowing affects incoming light, whereas masking affects reflected light.

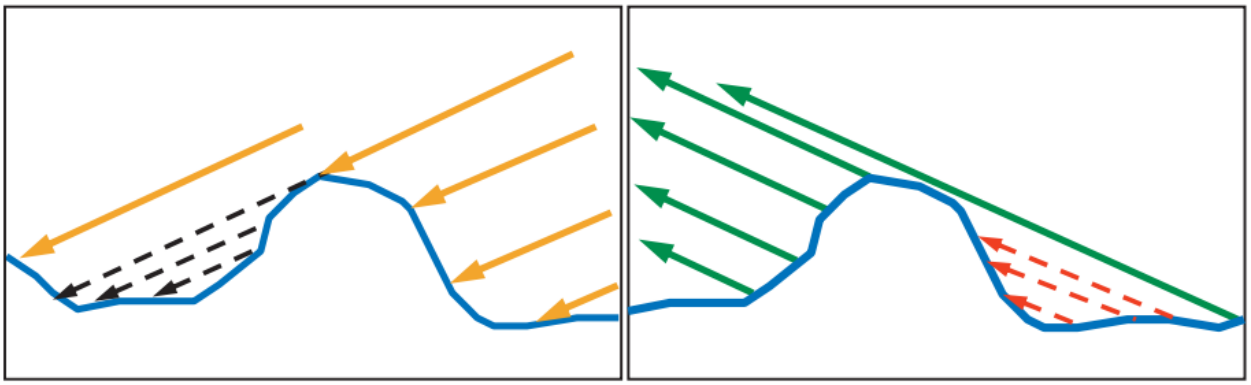

Figure 3: Self-Shadowing and Masking

2.3.1.3. Microfacet Slope Distribution, D

Microfacet slope distribution models the statistical probability that microfacets are oriented towards the specular half-angle vector, which maximizes the reflection. Beckmann distribution is a popular model for microfacet slope distribution, but we use a simpler formula:

$$D = \frac{r^2}{\pi\ (\ (N \cdot H)^2\ r^2\ +\ 1\ )^2}$$

where r is the roughness parameter of the material.

The effect of the roughness to the reflections is illustrated in Figure 3.

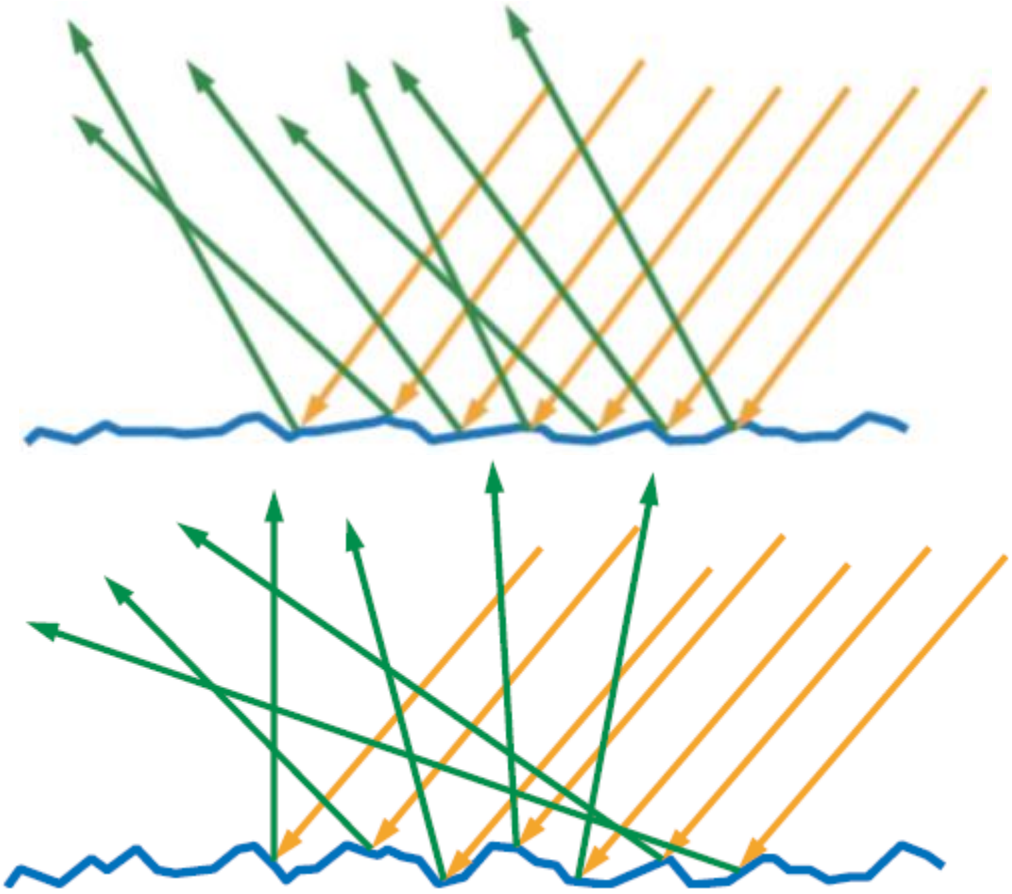

Figure 4: Two surfaces with different roughness. Rougher (second) surface causes blurrier reflections.

2.3.1.4. Tiled Textures and Normal Mapping

Textures of materials are stored as bitmap images. If the material has a tiled pattern, the texture bitmap consists of a tile. Surfaces are flat in our 3D models. We use normal mapping to simulate bumps and dents. Normal maps are stored as RGB images with the same resolution with the texture image, where the RGB components correspond to the Cartesian coordinates of the surface normal.

2.4. Spherical Harmonics Representation (SH)

Spherical Harmonics are a frequency-space basis for representing functions defined over the sphere. It is analogous to the Fourier transform but defined across the surface of a sphere instead of a circle. R. Ramamoorthi et al. proposed a method in 2001 which compresses the environment map into 9 spherical harmonics coefficients [2]. Using the spherical harmonic approximation allows to include complex shading models under environmental illumination with little pre-computation. Furthermore, because the irradiance can be calculated from only 9 coefficients, rendering is more efficient.

We can imagine a point on a surface as being surrounded by a hemisphere of light. A simplistic way to analyze the lighting is to model the hemisphere with only a number of point lights. We can map Cartesian coordinates of point lights to spherical coordinates. Point light representation is fast and easy to work with. However, we are concerned with extracting as much possible detail from the surrounding scene to capture the radiance received on the surface. Spherical Harmonics provides a fast way to compress the lighting data compactly.

Spherical harmonics are basis functions which are used to reconstruct a function defined on the sphere. In order to understand Spherical Harmonics representation we need a good bit of mathematical background. Intuitively though, it is similar to frequency domain representation signals. As the Fourier transform works over the unit circle for one dimensional functions, spherical harmonics work over the unit sphere for two dimensional functions. They are scaled with coefficients and combined just like sine and cosine functions of Fourier series.

First of all, we need to work with spherical coordinates which are defined as;

$$x = sin\theta\ cos\varphi$$

$$y = sin\theta\ sin\varphi$$

$$z = cos\theta$$

where θ is the angle from the z-axis, and φ is the angle of the projected point on the plane from the x-axis.

The general form of spherical harmonics is defined in the complex domain.

$$Y_l^m(\theta,\varphi) = K_l^m\, e^{im\varphi}\, P_l^{|m|}(cos\theta)$$

where $l \in \mathbb{N}, -l \le m \le l$

The real form is sufficient for our purposes, because we only approximate real functions. After taking the real part of the above, and simplifying, we end up with the below expanded form.

$$Y_l^m(\theta,\varphi) = \begin{cases} \sqrt{2}\, K_l^m \cos(m\varphi)\, P_l^m(\cos\theta), & m > 0 \\ \sqrt{2}\, K_l^m \sin(-m\varphi)\, P_l^{-m}(\cos\theta), & m < 0 \\ K_l^0\, P_l^0(\cos\theta), & m = 0 \end{cases}$$

P_l^m are the associated Legendre Polynomials. They are defined recursively over the range [-1,1] as follows.

$$(l-m)P_l^m(x) = x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x)$$
$$P_m^m(x) = (-1)^m(2m-1)!!\,(1-x^2)^{m/2}$$
$$P_{m+1}^m(x) = x(2m+1)P_m^m(x)$$

Note that double factorial $n!! = \prod_{i=0}^{k}(n-2i) \quad where\ k = \left\lceil \frac{n}{2} \right\rceil - 1$. In other words it is the product of all positive even integers less than or equal to n if n is even and the product of all odd integers less than or equal to n if n is odd.

The most important property of Legendre polynomials is that they are orthonormal. When the product of two of them is integrated, the result if 1 if they are the same, and 0 if they are different. The band index l takes nonnegative integer values, m takes signed integer values between -l and l. Thus for each band index l, there are 2l+1 polynomials. For example band 0 has only 1 polynomial P_0^0 whereas band 1 has three: $P_1^{-1}, P_1^0, and\ P_1^1$.

K_l^m are scaling constants which are used to normalize the functions.

$$K_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}$$

Putting all of these into action, we can derive the first three bands of spherical harmonics basis.

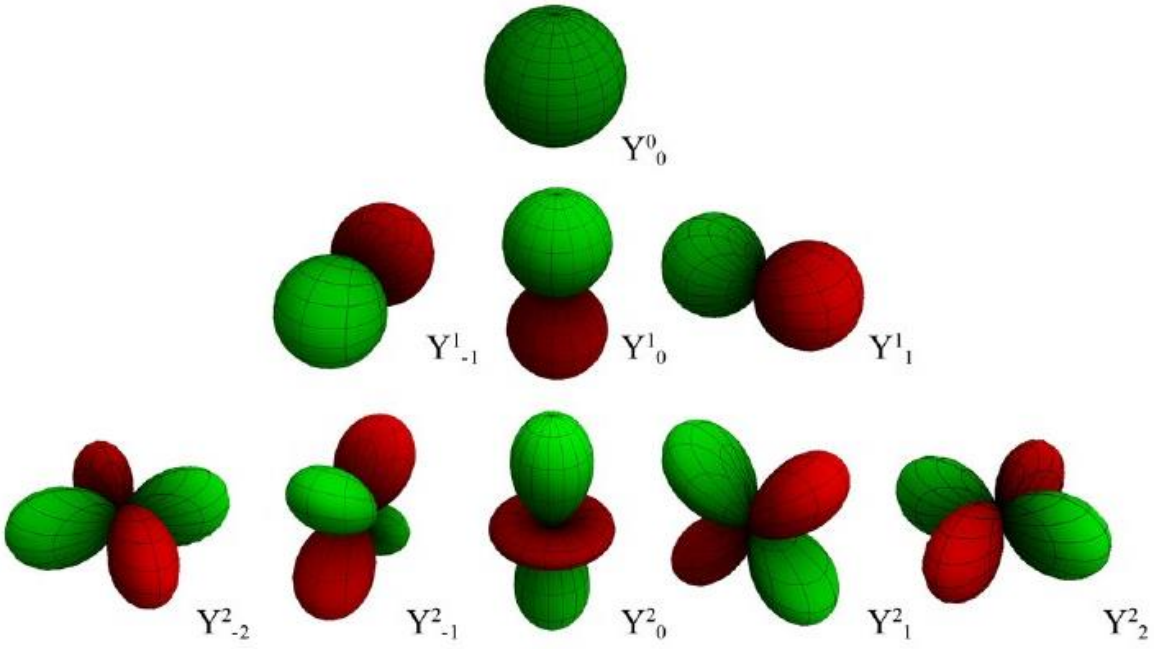

Figure 5: Graphical representation of first three level of Spherical Harmonics

$$Y_0^0(\theta,\varphi) = \sqrt{\frac{1}{4\pi}}$$

$$Y_1^{-1}(\theta,\varphi) = -\sqrt{\frac{3}{4\pi}}y$$

$$Y_1^0(\theta,\varphi) = \sqrt{\frac{3}{4\pi}}z$$

$$Y_1^1(\theta,\varphi) = -\sqrt{\frac{3}{4\pi}}x$$

$$Y_2^{-2}(\theta,\varphi) = \sqrt{\frac{15}{4\pi}}xy$$

$$Y_2^{-1}(\theta,\varphi) = -\sqrt{\frac{15}{4\pi}}yz$$

$$Y_2^0(\theta,\varphi) = \sqrt{\frac{5}{16\pi}}(3z^2-1)$$

$$Y_2^1(\theta,\varphi) = -\sqrt{\frac{15}{4\pi}}zx$$

$$Y_2^2(\theta,\varphi) = \sqrt{\frac{5}{16\pi}}(x^2-y^2)$$

Figure 6: Real-valued formulas of Spherical Harmonics

In order to approximate a two dimensional function f(s) defined over the unit sphere S, we need to calculate coefficients c_l^m. Then, we can write the n[th] order approximation $\tilde{f}(s)$ of a given function $f(s)$ as follows.

$$\tilde{f}(s) = \sum_{l=0}^{n} \sum_{m=-l}^{l} c_l^m Y_l^m(s)$$

This process is called *expansion*. The approximation gets more accurate as n increases. However, we opted to only use n=3 bands because of performance concerns. Our results, in general show that 9 spherical harmonics is enough for our purposes.

The coefficients c_l^m are computed by the following integral:

$$c_l^m = \int_s f(s) Y_l^m(s)\, ds$$

This process is called *projection*.

In practice, we parameterize the unit sphere in spherical coordinates. Therefore projecting incident radiance L into the spherical harmonic basis can be done via the following integral. L_l^m are the coefficients of the spherical harmonics for lighting function.

$$L_l^m = \int_0^{2\pi} \int_0^{\pi} L(\theta, \varphi)\, Y_l^m \sin(\theta)\, d\theta\, d\varphi$$

$$B_l^m(N) = \int_0^{2\pi} \int_0^{\pi} BRDF(N, \theta, \varphi)\, Y_l^m \sin(\theta)\, d\theta\, d\varphi$$

Viewer is fixed in our application, so that we omit it in BRDF parameters.

Light and BRDF are multiplied when calculating reflected light L_o .

$$L_o\,(V) = \int_0^{2\pi} \int_0^{\pi} L(\theta, \varphi)\, BRDF(N, \theta, \varphi)\; d\theta\, d\varphi$$

The orthonormality property provides for a very simple expression to compute the integrated product of two functions represented in the SH basis [11].

$$L_o\,(V) = \sum_i L_i B_i$$

Where L_i and B_i are all L_l^m and B_l^m coefficients we used in our application. Since we use 9 coefficients, the irradiance E can be calculated with: [2]

$$E(\mathbf{n}) = \mathbf{n}^t M \mathbf{n}$$

Where M is a matrix derived from coefficients:

$$M = \begin{bmatrix} c_1 L_2^2 B_2^2 & c_1 L_2^2 B_2^{-2} & c_1 L_2^1 B_2^1 & c_2 L_1^1 B_1^1 \\ c_1 L_2^{-2} B_2^{-2} & -c_1 L_2^2 B_2^2 & c_1 L_2^{-1} B_2^{-1} & c_2 L_1^{-1} B_1^{-1} \\ c_1 L_2^1 B_2^1 & c_1 L_2^{-1} B_2^{-1} & c_3 L_2^0 B_2^0 & c_2 L_1^0 B_1^0 \\ c_2 L_1^1 B_1^1 & c_2 L_1^{-1} B_1^{-1} & c_2 L_1^0 B_1^0 & c_4 L_0^0 B_0^0 - c_5 L_2^0 B_2^0 \end{bmatrix}$$

2.4.1. Computation of BRDF Projection into Spherical Harmonic Coefficients

Unlike lighting function, BRDF depends on the surface normal N as well as the light direction L. To be able to expand BRDF for an arbitrary N, L pair; we project BRDF into SH for L parameter separately for different normals. For expansion, we use bilinear interpolation to find SH coefficients for the surface normal. BRDF SH coefficients are not used for expanding BRDF for a particular light direction, but later multiplied by the Lighting Function SH coefficients to calculate irradiance on the surface.

BRDF values do not change with changing radiance, but they depend on the Fresnel and roughness parameters of the material. Therefore, we only need to compute BRDF once for a material. BRDF SH coefficients are pre-computed and stored to reduce real-time overhead in our application.

```
private static ArrayList<Vector3D> generateSampleVectors(int N) {
    ArrayList<Vector3D> sampleVectors = new ArrayList<Vector3D>();
    int sqN = (int) Math.sqrt(N);
    double oneover = 1.0 / sqN;
    for (int a = 0; a < sqN; a++) {
        for (int b = 0; b < sqN; b++) {
            double p = (a + Math.random()) * oneover;
            double q = (b + Math.random()) * oneover;
            double theta = 2.0 * Math.acos(Math.sqrt(1.0 - p));
            double phi = 2.0 * Math.PI * q;
            sampleVectors.add(new Vector3D(sph2cart(new double[]{theta, phi})));
        }
    }
    return sampleVectors;
}
```

Snippet 1: Java Method for Generating Random Vectors in 3D

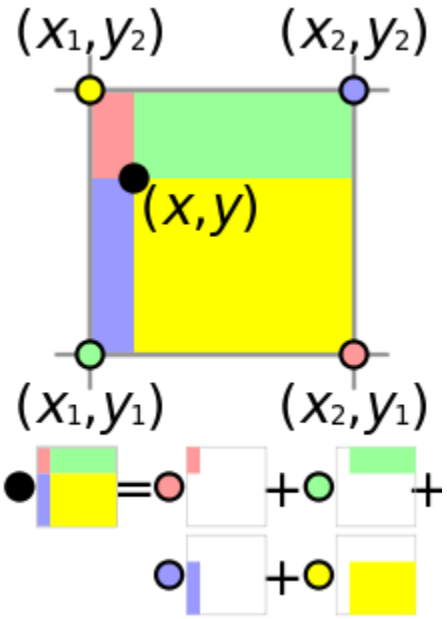

Figure 7: Graphical representation of bilinear Interpolation which is used for interpolating BRDF SH coefficients

2.5. Monte Carlo Integration

The given integrals must be calculated in order to project BRDF and Lighting functions into SH coefficient. We need a fast and numerical method for calculating the integrals. In our application we use Monte Carlo Integration for this purpose. It is based the approximate equivalence between the expected value of function of a random variable and the mean of large number of samples from the function.

$$E[f(x)] = \int f(x)p(x)dx$$

$$E[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Vector samples are generating using randomized azimuth angle and zenith angle for BRDF projection. The value of the function at the direction of each sample is calculated and summed up. Then the result is multiplied by a factor equal to $\frac{4\pi}{N}$ where N is the number of samples to find the mean value multiplied by the area of the sphere.

$$B_l^m = \frac{4\pi}{N} \sum_{j=1}^{N} BRDF(\theta,\varphi)\ Y_l^m(\theta,\varphi)$$

For lighting projection, we use the vectors in our environment directions matrix instead of random directions since the value of the function for outside of the area of the environment image will be equal to zero and have no effect on the sum. The sum is then multiplied by $\frac{4\pi}{NM}$ where N and M are the row and column sizes of the matrix.

$$L_l^m = \frac{4\pi}{NM} \sum_{i=1}^{N}\sum_{j=1}^{M} Light_{i,j}\ Y_l^m(\theta,\varphi)$$

Where Θ and ϕ are azimuth and zenith angle of the direction corresponding to the pixel i-j.

2.6. Shading Equation

We have two components in shading equation, diffuse and specular. Diffuse reflection have the color of the material, whereas specular component have the color of the light. At the fragment shader, we calculate total gray scale illumination from lighting coefficients and specular from lighting and BRDF coefficients multiplied. Gray scale illumination is multiplied by the texture color, which corresponds to the diffuse component. We get the color of a fragment by summing up its diffuse and specular components [13].

```
uniform sampler2D u_Texture;
uniform sampler2D u_Bump;
uniform mat4 u_IrradianceMatrix[3];
uniform float u_BRDFCoeffs[225];
varying vec2 v_TexCoord;

void main() {
    mat4 m_IrradianceMatrix[3];
    vec4 normal = vec4(normalize(texture2D(u_Bump, v_TexCoord).rgb*2.0-1.0), 1.0);

    float brdf[9];
    getBRDF(normal.xyz, brdf);

    // update irradiance matrix
    for(int band=0; band<3; band++){
        float c5L20 = (u_IrradianceMatrix[band][2][2] / 0.743125) * 0.247708;
        float c4L00 = u_IrradianceMatrix[band][3][3] + c5L20;

        m_IrradianceMatrix[band][0][0] = u_IrradianceMatrix[band][0][0] * brdf[8];
        m_IrradianceMatrix[band][0][1] = u_IrradianceMatrix[band][0][1] * brdf[4];
        m_IrradianceMatrix[band][0][2] = u_IrradianceMatrix[band][0][2] * brdf[7];
        m_IrradianceMatrix[band][0][3] = u_IrradianceMatrix[band][0][3] * brdf[3];
        m_IrradianceMatrix[band][1][0] = u_IrradianceMatrix[band][1][0] * brdf[4];
        m_IrradianceMatrix[band][1][1] = u_IrradianceMatrix[band][1][1] * brdf[8];
        m_IrradianceMatrix[band][1][2] = u_IrradianceMatrix[band][1][2] * brdf[5];
        m_IrradianceMatrix[band][1][3] = u_IrradianceMatrix[band][1][3] * brdf[1];
        m_IrradianceMatrix[band][2][0] = u_IrradianceMatrix[band][2][0] * brdf[7];
        m_IrradianceMatrix[band][2][1] = u_IrradianceMatrix[band][2][1] * brdf[5];
        m_IrradianceMatrix[band][2][2] = u_IrradianceMatrix[band][2][2] * brdf[6];
        m_IrradianceMatrix[band][2][3] = u_IrradianceMatrix[band][2][3] * brdf[2];
        m_IrradianceMatrix[band][3][0] = u_IrradianceMatrix[band][3][0] * brdf[3];
        m_IrradianceMatrix[band][3][1] = u_IrradianceMatrix[band][3][1] * brdf[1];
        m_IrradianceMatrix[band][3][2] = u_IrradianceMatrix[band][3][2] * brdf[2];
        m_IrradianceMatrix[band][3][3] = c4L00 * brdf[0] - c5L20 * brdf[6];
    }

    vec4 specular = vec4(0.0, 0.0, 0.0, 1.0);
    specular.r = dot(normal, m_IrradianceMatrix[0] * normal);
    specular.g = dot(normal, m_IrradianceMatrix[1] * normal);
    specular.b = dot(normal, m_IrradianceMatrix[2] * normal);

    float irradiance_r = dot(normal, u_IrradianceMatrix[0] * normal);
    float irradiance_g = dot(normal, u_IrradianceMatrix[1] * normal);
    float irradiance_b = dot(normal, u_IrradianceMatrix[2] * normal);
    float mean = (irradiance_r + irradiance_g + irradiance_b) / 3.0;
    vec4 irradiance = vec4(vec3(mean * 0.3), 1.0);

    specular *= vec4(vec3(0.05), 1.0);
    vec4 diffuse = irradiance * texture2D(u_Texture, v_TexCoord);

    gl_FragColor = diffuse + specular;
}
```

Snippet 2: Fragment Shader Part 1

```
void getBRDF(vec3 normal, inout float brdf[9])
{
    vec2 index = sph2index(cart2sph(normal));
    ivec2 ul = ivec2(int(floor(index.x)), int(ceil(index.y)));
    ivec2 ll = ivec2(int(floor(index.x)), int(floor(index.y)));
    ivec2 ur = ivec2(int(ceil(index.x)), int(ceil(index.y)));
    ivec2 lr = ivec2(int(ceil(index.x)), int(floor(index.y)));

    float hratio = index.x - floor(index.x);
    float vratio = index.y - floor(index.y);

    for(int k=0; k<9; k++)
    {
        brdf[k] = mix(
            mix(u_BRDFCoeffs[brdfIndex(ll.x, ll.y, k)], u_BRDFCoeffs[brdfIndex(lr.x, lr.y,
k)], hratio),
            mix(u_BRDFCoeffs[brdfIndex(ul.x, ul.y, k)], u_BRDFCoeffs[brdfIndex(ur.x, ur.y,
k)], hratio), vratio);
    }
}

int brdfIndex(int i1, int i2, int i3)
{
    return 45 * i1 + 9 * i2 + i3;
}


vec2 cart2sph(vec3 cart)
{
    vec2 sph;
    sph.x = acos(cart.z / length(cart));
    sph.y = atan(cart.y / cart.x);
    return sph;
}


vec2 sph2index(vec2 sph)
{
    float pi = 3.14159265358;
    vec2 index;
    index.x = (sph.x * (4.0 / pi)) + 2.0;
    index.y = (sph.y * (4.0 / pi)) + 2.0;
    return index;
}
```

Snippet 3: Fragment Shader Part 2

2.7. Packages

The Android application is separated into 3 packages: UI, Calculation and Utility. UI package includes MVC architecture based on the Android SDK structure. Calculation package includes classes that perform illumination calculations. Utility package includes miscellaneous classes such as resource loaders.

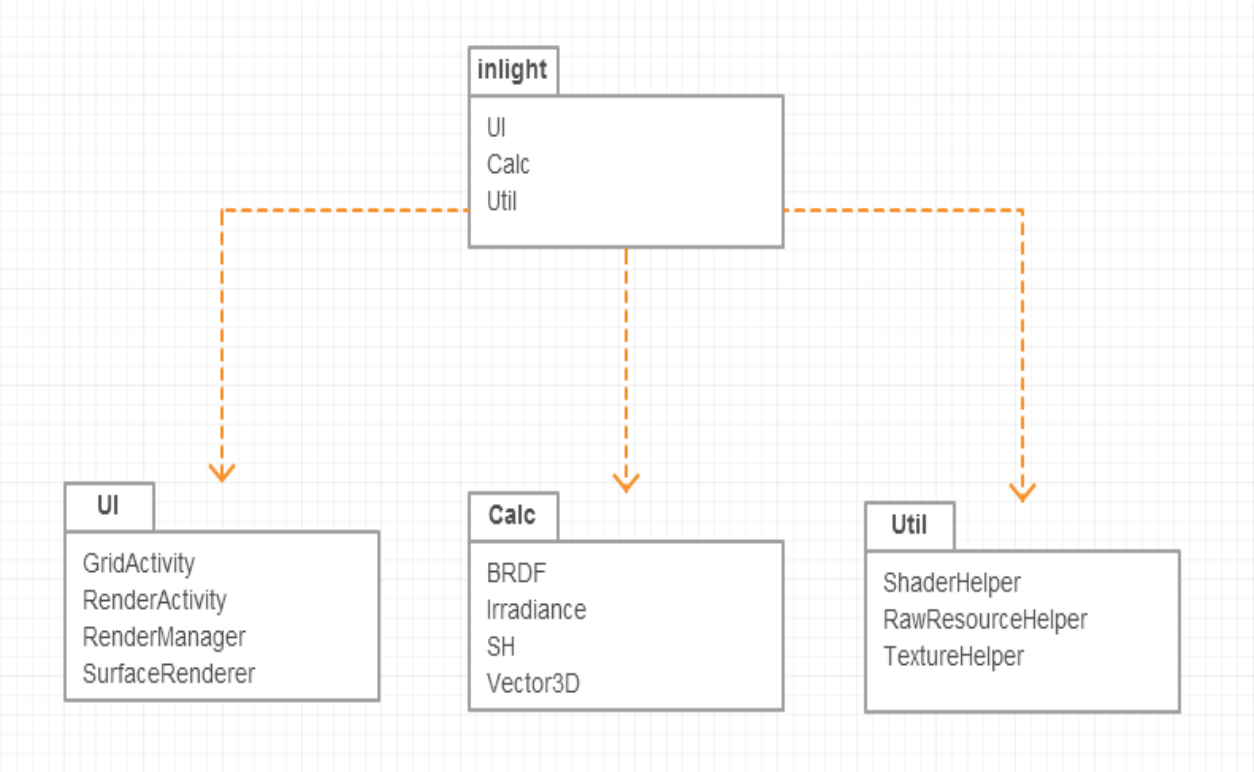

Figure 8: Android Application Packages

3. Impacts

3.1. Global Impacts

As stated before, Inlight is intended for decoration / furniture industry targeted at mobile devices. It may be utilized as a virtual showcase for furniture vendors. Internet catalogs of these vendors usually use only a photo of the furniture or fabric for demonstration purposes. This kind of showcase of fabrics may feel unnatural for most customers. Furniture is regarded as an expression of personality [14]. Therefore a realistic visualization of the product is very important for customers. Photorealism in Inlight helps customers visualize materials in their home or office environments better.

In this sense, our application will boost internet shopping for furniture industry and loosen the physical boundaries between the store and the customer. As a result, people will have a better chance of shopping in a cross-cultural manner. The realistic view of the product we provide may help customers buy items from different regions and cultures more reliably, thus providing them a wide variety of options. This may lead to the globalization of vendor-customer relationship in furniture industry.

3.2. Economic Impacts

Inlight has economic benefits for both vendors, customers and designers. Vendors need big spaces for showcasing their products. They should buy, hire or build this space for themselves which causes great financial burden for them. They should also pay for maintenance and employees to take care of theirs stores. A mobile application is much more suitable in this manner. It will make maintaining and showcasing inventory cost cheaper. It is a common case for customers to experience cognitive dissonance, a feeling of self-doubt, due to bad selection of furniture [14]. One of the reasons is that the lighting in store environment and lighting in customer's own environment may be very different. For example, a kitchen table is probably used at early morning and at evening. Lighting in this two time of day may differ a lot depending on the position of kitchen. However, vendors does not take this variable to account, which may cause for furniture to look different in customer's environment than they look in the store

environment. Customers may feel that they wasted money on a product they do not like. Inlight solves this problem by taking customer's lighting environment into account.

3.3. Environmental Impacts

Raw material for most furniture and fabrics comes from the environment we live in. Great amounts of the material is wasted for showcasing purposes in stores or during interior design. Inlight may be a potential solution for this problem since material showcase is virtual. It will also decrease the amount of synthetic waste since modern furniture may contain good amount of synthetic material.

3.4. Social Impacts

Customers tends to evaluate many alternate options when buying furniture and designing their homes and offices [14]. They spend significant amount of time and money in the process, and it may be tiresome. Inlight will make this process easier and more reliable. As a result, we believe Inlight will change the customer behavior when buying furniture.

4. Contemporary Issues Related to Project inlight

Recent technological advancements and customer needs introduced Augmented Reality into today's world. Augmented Reality, as its name suggests, aims to "augment" the real world surrounding us humans in various ways. The general direction most of the Augmented Reality projects have taken is towards entertainment sector. However, they are not limited to entertainment only. As stated earlier, Augment Reality is a very new and developing concept and possible uses of this concept are still being discovered and improved.

One of the key issues in Augmented Reality is making the features and objects augmented into the perceived world look realistic and believable. This issue is directly related to their visual appearance in the augmented realm and how they interact with the real world around. Augmented objects should appear as though they were actually physically present in the scene in order to blend in with the real world around. An augmented object, in essence, is only a 3D model being projected to a scene. What this 3D model is supposed to look like is determined by a set of factors like lighting, attenuation, occlusion, and so on.

Our project focuses on calculating lighting effects for creating accurate and realistic looking objects in augmented reality. This task of creating photorealistic renders in augmented reality with current technology and devices is considered as one of the biggest challenges. Project inlight tackles this challenge by putting on the table a very radical idea, using the built in camera of the device to simulate lighting effects. The results have been satisfactory and we believe to have created a basis which could be applied to many other contexts as an example.

The calculation of realistic lighting effects brings us to another point, real time computation. As stated earlier, the objects in augmented reality are supposed to be seamless in integration. For this reason, the entire process of creating, computation and placement of an object needs to be done in real time. Otherwise it defeats the entire purpose behind the augmented reality. This situation leads to a trade-off between quality and performance. Creation of realistic looking objects in Augmented Reality calls for more calculations and factors to cut in, which means more computations for the device to handle. Our project has been able to successfully address this trade-off on devices we tested and could produce renders in real time.

5. Software User Manual

5.1. Installation

The application can be installed by obtaining the .apk file and transferring it to the device. Once the .apk file is on the device, the Android OS will detect is an installation file in the file system. Upon running the installer, the customer is asked to grant permissions for the application, namely, Camera, Internet and Storage permissions. When the installer finishes, the application icon is placed in applications menu ready to be run.

5.2. Preparation

The application relies on a wide angle camera to capture a greater field of view for accurately obtaining lighting information. Therefore the front facing camera of the device needs to be fitted with a fisheye lens prior use. The lens needs to be fixed properly to obtain desirable results.

5.3. Execution

The opening screen shows a list of available fabrics to be rendered. The list shows each fabric's thumbnail to allow the user select what they exactly want. The list can be scrolled up and down to load more fabric samples.

Once a fabric has been chosen, the application displays the chosen fabric in full screen. The fabric is rendered using the lighting information obtained from the front facing camera in real time. The user may relocate and rotate the device to observe how the fabric reacts to the change.

Using the "back" button transfers the user to the previous screen where the list of available fabrics is present. From there on, the user can choose another fabric to render.

5.4. Termination

Whenever the user wants to quit the program, they can chose to use default Android controls to do so. That is, using the “back” or “home” buttons on the screen to return to home screen. In this case, the program is only minimized and when the user starts the application again, the program resumes from where it was left. In order to stop the program entirely, the user may choose to user Android task manager to end its execution.

6. References

[1] Palomo C. Real-time High Dynamic Range Image-based Lighting. 2008.

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

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

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

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

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

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

[8] Hoffman, Naty. "Physics and Math of Shading". SIGGRAPH 2014.

[9] "Writing a Cook-Torrance Surface Shader", Pixar Rendermen Documentation. Web. Retrieved Feb 2015.

[10] Heitz, Eric. "Understanding the Masking-Shadowing Function". SIGGRAPH 2014.

[11] Green, Robin. "Spherical Harmonic Lighting: The Gritty Details". GDC 2003.

[12] W.H. Press, G.R. Farrar, Recursive Stratified Sampling for Multidimensional Monte Carlo Integration, Computers in Physics, v4 (1990).

[13] McReynolds, T., Blythe, D., Advanced Rendering in Programming with OpenGL, ACM SIGGRAPH 1997.

[14] Ponder, N., Consumer Attitudes and Buying Behavior for Home Furniture, Mississippi, Franklin Furniture Institute, 2013.