Rendering Metals and Worn or Weathered Metallic Objects

Veselin Mihaylov

Kongens Lyngby 2013
IMM–M.Sc.–2013-89

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet
Building 303 B
DK – 2800 Kgs. Lyngby
Phone +45 45253351, Fax +45 45882673
reception@compute.dtu.dk
www.compute.dtu.dk

Summary

This thesis focuses on computer generated visualization of metal material objects. There are two basic rendering approaches that have been taken into account in the scope of this project: ray traced rendering approach using NVidia's MentalRay render engine, and a real-time rendering approach using Unity3D game engine. In both cases the goal of the thesis was to find and implement an effective solution to simulate a metal material surface. In ray traced rendering as well as real-time rendering, a physically-based light reflectance model has been used to simulate as accurately as possible the interaction between the light coming from the light source and the object's surface. That interaction is crucial to the appearance of the object.

Once the metal material simulation has been achieved, the thesis focuses on simulation of weathering and aging effects on metallic surfaces. Techniques for simulating physical damage on surfaces such as scratches, cavities, and grooves are discussed for both ray tracing and real-time rendering approach. The project focuses on the simulation of some non-physical damages on metal surfaces that are due to environmental and weathering conditions. Simulation of metal corrosion resulting in iron oxide (red rust) has been covered. The rusty surface has been simulated with a physically-based light reflectance model designed to simulate rough surfaces. The thesis covers the development and implementation of a system for simulating the life span of a patina formation and development process. The system does not take any physically accurate input into consideration when simulating patina formations for specific time period (i.e. 1 month, 5 years) but instead uses user-defined colors and procedurally generated noise textures to simulate the spread and color of the metallic patina.

Preface

This thesis was prepared as a partial fulfillment of the requirements for acquiring a Master of Science degree in the field of Digital Media Engineering, Department of Informatics and Mathematical Modeling at Technical University of Denmark.

Acknowledgements

I would like to thank my family and friends for the moral and financial support throughout this project. I would also like to thank my supervisor Jeppe Revall Frisvad for his involvement and mentoring throughout the whole project.

Contents

Chapter 1

Introduction

The ever expanding demand for creating closer to photorealistic computer generated visualizations leads to the need for more accurate simulations of real-world materials. It is rather a rare case scenario to have a perfectly clean material in the real world. For the purpose of increasing the level of realism, objects' materials have to account for surface imperfections, weathering and aging conditions, collection of dust and dirt on their surfaces etc.

In reality, the reason one is able to perceive any object or surface is due to the presence of some sort of a light source. Without light nothing would be visible to an observer. Having that observation, the simulation of a surface material of a certain type relies mainly on the light-surface interaction and the properties that define a certain material type. With the continuously increasing computer power and technological advancements, it becomes possible to compute with higher precision the interaction between light and virtual materials using physically-based environmental conditions and material properties.

1.1 Challenges and Goals

This thesis is focused on the simulation of metal materials and weathered or aged metallic surfaces. All of the research and development of the techniques needed for getting the desired results was driven by two main goals – simulation of metal surfaces and simulation of worn or weathered surfaces.

There are two distinctive approaches taken in this thesis for accomplishing the goals. Each approach is targeting the same goals only it uses different software and hardware components to accomplish this. The first approach is using a rendering method called ray tracing which uses the CPU (Central Processing Unit) of the computer for its computational needs. That method produces very detailed, close to photorealistic, simulations of 3-dimensional computer generated scenes. The method is tracing the path of light rays through pixels on the image plane producing a single image (frame), like a snapshot of a 3D scene. The disadvantage is that the ray tracing method is also very time consuming which makes it incompatible for real-time simulations. A real-time 3D computer simulation is the generation of multiple still images (frames) consequently that when put together gives a real-time interactive 3D environment. The second approach is using a real-time rendering method that uses the GPU (Graphical Processing Unit) of the video card to perform the computations needed. The GPU has been designed and optimized to process graphical data fairly fast compared to the CPU, which makes it an adequate choice in the simulation of 3-dimensional scenes in real-time.

Both approaches face different challenges throughout their development. Some of the challenges are related to the light reflectance model for the metal part and for the weathered or aged part (i.e. rust, patina); some others are related to the surface reflections. Since the two approaches differ from each other, even though they have the same goals in common, the challenges introduced by the development process for each approach are specific to it.

In a summary, the goals introduced by this project are as follows:

- Simulation of metal surfaces (clean)
  - Ray traced rendering (CPU approach)
  - Real-time rendering (GPU approach)
- Simulation of aged or weathered metallic surfaces
  - Ray traced rendering (CPU approach)
  - Real-time rendering (GPU approach)

1.2 Potential Application

The models described in this thesis could be used for the visualization of still images, animations, or real-time interactive applications such as computer games. The simulation of metal materials for achieving realistically looking metallic objects could find potentially a great application in architectural visualizations and many other fields.

The system for simulating weathering and aging effects on an object's surface introduced in this project could be used in creating realistic metal corrosion, surface damage, and metallic patina. That system allows for procedural creation of the effects mentioned above. That way a lot of work that has been done by artists in the past to recreate a rusty surface for instance, could be replaced by the procedural methods for generating the same effect. The system introduced gives the opportunity to animate the process by increasing or decreasing some of the values that drive the procedural process. That feature allows for creating an animation of a rusting or patina formation process for instance which could find potential application in movies, games, or even architecture. Another application of the weathering and aging system related to computer games could be used as a method of having game scene objects to change in appearance over time or under certain circumstances or events.

1.3 Thesis Flow

The thesis starts with a chapter giving information about what a bidirectional reflectance distribution function is and why is it important for the development of this project. That chapter is important for the reader to understand the core light-matter interaction techniques used later in the thesis. Chapter 3 focuses on the creation of metal surfaces and weathering or aging effects using the first approach mentioned above – ray traced rendering. The chapter describes the methodology behind the scenes and shows the results gathered from the experimental phase. The ray traced rendering engine used to perform all the experiments for this thesis is MentalRay by NVidia under the Autodesk's 3DS Max environment. The next chapter focuses on the research and implementation of metal materials in a real-time rendering environment. The chapter focuses on the development of small programs that run on the GPU (Graphical Processing Unit) called graphical shaders. Through the use of those shaders, the lighting models, surface reflections, surface imperfections etc. has been developed for use in a real-time environment such a computer game. For the implementation of that part of the thesis Unity3D game engine has been used as a development environment. Finally, the thesis

ends with a comparison of the results from the two approaches and a summary of the project. A discussion of how much difference there is in the results produced by the two methods rendering the same object has taken place in this chapter.

Chapter 2

Reflectance Models

To simulate a real-world material in a computer generated environment, a reflectance model needs to be defined to compute the interaction of the incoming light with a point on a surface. Reflectance models could be divided into two main groups: empirical and theoretical models. The first group, empirical models, provides reflectance models that are computationally efficient in a rendering process but are lacking physical accuracy such as conservation of energy[1] for example. Those models are limited and not precise enough to be used in the visualization of photorealistic three-dimensional scenes. Even though the empirical models are lacking precision, they are still popular nowadays in applications that do not require rendering of photorealistic objects. The other group of reflectance models, theoretical models, provides quantitative values that could be filled with measured physical data. The theoretical models use much more computational

[1] On a contact between light and a material, the incoming light gets reflected, transmitted, or absorbed. The conservation of energy is equal to the addition of the amount of reflected, transmitted, and absorbed light.

power which makes them heavy to render but they provide much better results in terms of quality and photorealism [IMPBR] [ASBRCG].

This thesis focuses on the simulation of realistic metallic surfaces which should use a theoretical reflectance model to account for a physically-based render. A complete representation of a surface behavior when interacting with light takes into account phenomena such as scattering, polarization, phosphorescence, and fluorescence. Each of those could be represented by variables and plugged into a reflectance function. Those variables are as such: [ASBRCG]

- Incoming and outgoing light angle
- Incoming and outgoing wavelength
- Incoming and outgoing polarization
- Incoming and outgoing position (sometimes differs due to subsurface scattering)
- Time delay between the incoming and outgoing light rays

Having to include all those variables in the calculations of a reflectance model in computer graphics would be very computationally inefficient. Eliminating most of those variables (subsurface scattering, fluorescence, phosphorescence, and polarization) leaves the reflectance function with only the incident and reflected light angles. The forming function is called bidirectional reflectance distribution function (BRDF) and is a function of four variables as follows: [ASBRCG] [IMPBR]

$$\rho(\theta_i, \phi_i, \theta_r, \phi_r) = \frac{dL_r(\theta_r, \phi_r)}{dE_i(\theta_i, \phi_i)} \tag{2.1}$$

where dL_r is the reflected radiance and dE_i is the irradiance.

When the incoming light interacts with a material, the light gets reflected, transmitted, or absorbed. The BRDF is a function that describes how much of the incoming light gets reflected off the surface being illuminated, similarly a function called BTDF (Bidirectional Transmittance Distribution Function) describes how much light gets transmitted through the surface.

Additionally, on interaction of incoming light and a surface, different light wavelengths (colors) could be reflected, absorbed, or transmitted to various degrees depending on the material's physical properties. That phenomenon is giving the color to an object [AIBL].

BRDF functions could be classified into two classes: isotropic and anisotropic. The isotropic BRDFs represent light reflectance that remains unchanged when the surface is

rotated around the surface normal[2]. Smooth materials such as smooth plastics have that type of BRDF. The anisotropic BRDFs on the other hand change with respect to the rotation of the surface. As an example of a material that exhibits an anisotropic BRDF is a brushed metal or satin. Most real-world materials are anisotropic to some degree but the effect on some materials is very subtle which is why it could be neglected in computer graphics simulations and using isotropic BRDFs for the sake of optimization.

There is no generic BRDF function that could serve all types of material simulations but instead there are many BRDFs that are specific in the simulation of a certain type of material. Once we have a defined bidirectional reflectance distribution function (BRDF) we can use it to compute the surface reflectance at a point viewed by an observer for every light source that illuminates that point. All the results are then summed up into a single reflectance amount [AIBL] (*See Figure 01*).

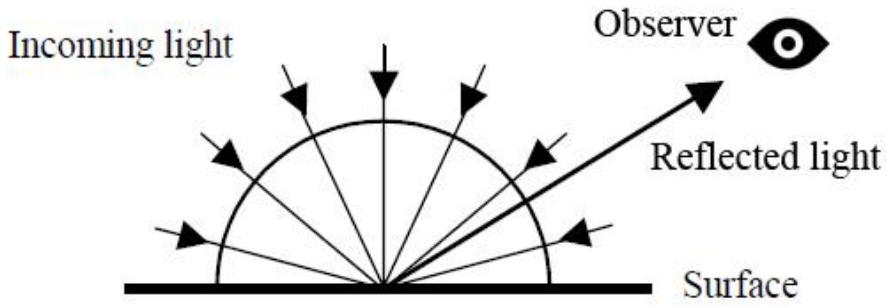

Figure 01: Incoming light illuminating a surface point from all possible directions. All of them contribute to the final amount of reflected light towards the observer.

[2] A surface normal in three-dimensional geometry is a vector that is perpendicular to the tangent plane of a surface.

Chapter 3

Offline Rendering of Metal Materials and Weathered or Worn Metallic Surfaces

Reproducing the appearance of a surface as closely and accurately as possible in computer graphics has always been a challenge in a number of ways. Surfaces in the real-world are made of complex materials that react to light in different ways such that they get different appearances. In computer graphics we often try to use the physical material properties to simulate as closely as possible the visual appearance of a certain type of material such as metal, fabric, plastic, etc. For the sake of optimization in rendering an approximation of the interaction between light and matter is often used. To render a realistically looking image, a reflectance model of how objects interact with light would be needed.

3.1 Simulation of Plain Metal Surfaces

Real-world surfaces exhibit a broad variety of reflectance phenomena. Some surfaces are very reflective and some are not as much. In general, reflection from surfaces can be separated in four main categories: perfect diffuse, perfect specular, glossy, and retro reflective [RMPLT] (*See Figure 02*). In the real world, a surface reflection does not fall under a single category but instead it is some sort of a mixture of the all four.

3.1.1 Previous Work

There has been an evolution of methods to simulate reflections in computer graphics throughout the years. The first attempt that increased significantly the level of realism in computer renderings was developed using simple Lambertian reflectance model. That method turns the surfaces into perfectly diffuse reflectors which mean that the reflected light is scattered equally in every direction. That way a surface always appears the same no matter from what direction the viewer is looking at it. Even though it is not physically possible to have a perfect diffuser in nature, the Lambertian reflectance model could still be used to achieve matte look to a surface [ICGGL]. In 1975, Phong created an empirical model that increased the level of realism and richness significantly [ICGP]. The main contributions of the Phong reflectance model were the interpolation of surface normals across triangles and the use of a cosine lobe to get highlights around the direction of a perfect light source reflection. That reflectance model allowed surfaces to appear glossy-like. The appearance was controlled by ambient, diffuse, and specular parameters that could be tweaked to achieve different results. The ambient parameter represents a uniform constant color that approximates light coming from the environment. The ambient parameter in the Phong equation is an added constant that brightens up the entire object of interest. The specular and diffuse parameters are taking into account the light coming from specific light sources such as point lights and directional lights. Just like the Lambertian model, the diffuse component represents light that is distributed equally in every direction. The specular parameter represents highlights, and it is concentrated around the mirror direction. The ambient and diffuse parameters are meant to affect the color of the surface and the specular parameter is affecting the color of the light source [ICGP]. The Phong reflectance model is computed by the following equation:

$$I = I_a + I_d + I_s = L_a K_a + L K_d (N \cdot L) + L K_s (R \cdot V)^{\alpha} \quad (3.1)$$

where $R = 2N(N \cdot L) - L$

where *K* is reflection coefficient, *L* is the light source, *N* is the surface normal, *V* is the viewing direction, *R* is the reflected direction, and *α* is the shininess controlling the size of the highlight lobe.

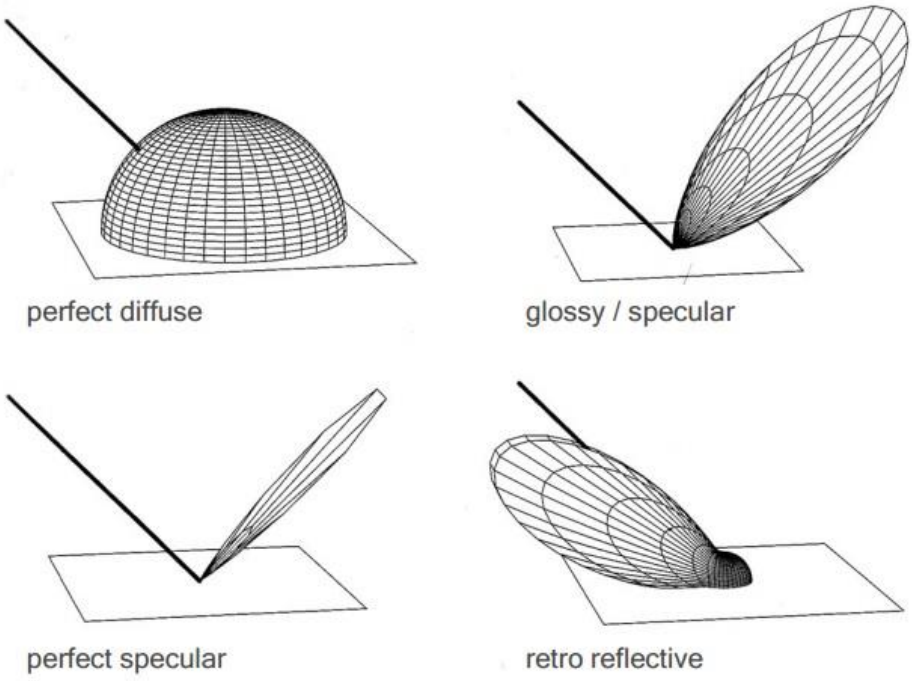

Figure 02: A perfect diffuse surface scatters light equally in every direction in a hemi-sphere (such as matte paint and chalkboard). A perfect specular surface scatters light in a single reflected direction, which shows sharp reflections (such as mirror and glass). A glossy surface scatters light in multiple reflected directions, which shows blurry reflections (such as plastic). A retro reflective surface scatters light back along the incident direction [RMPLT].

Several years after Phong created his model, Jim Blinn developed a solution based on the original reflectance model but with simplified calculations. For the sake of optimization a half vector has been calculated instead of using the true reflection vector [MLRCSP]. Later on, Ward developed a reflectance model that was much more physically accurate than the Phong model. Another major difference in Ward's model was that it could be expressed in isotropic and anisotropic forms [MMAR]. That reflectance model allowed for metal-like surfaces to be simulated. Another reflectance model that is based on Torrance and Sparrow work on developing BRDFs for metal surfaces [TORFRS] is the Cook-Torrance reflectance model. The model shows a surface as a set of many microfacets [ARMCG]. Those microfacets reflect light in many different directions (*See Figure 04.c*). Some of the reflected light could be blocked by

nearby microfacets when going towards the viewer, and some of the incoming light could be blocked in the same fashion. That phenomenon is known as masking (*See Figure 03.a*) and shadowing (*See Figure 03.b*), and is affecting the amount of diffuse and specular reflection that gets back to the viewer [MMRRS] [IMPBR] [ARMCG]. The Cook-Torrance model incorporates a Fresnel term to control the surface reflection amount at different angles of incidence.

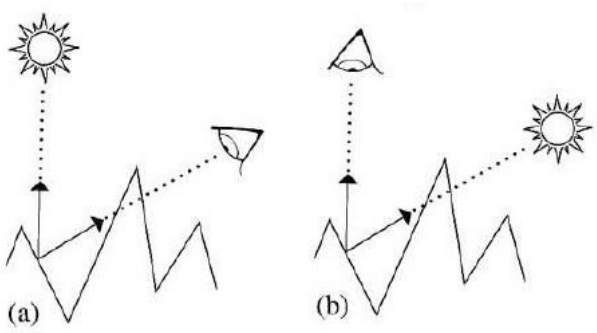

Figure 03: (a) Masking - some of the reflected light could be blocked from nearby microfacets that stay in the way towards the observer. (b) Shadowing – some of the microfacets could block (occlude) the light towards near to them microfacets.

3.1.2 BSSRDF and BRDF

For convenience most of the materials could be separated into two main groups: conductors and dielectrics. Dielectric materials are usually translucent and to some extend exhibit absorption and subsurface scattering. That means light rays enter the material, parts of them are scattered around and parts are absorbed, and some of the scattered light exits the material from a different place at a different angle [APMSLT] (*See Figure 04.b*). As the light travels through the body of the material, some wavelengths of the light rays are absorbed more strongly than others which results in the color of the object. In order for a material to appear realistic and have a natural look, the light scattering which is done by a light transport algorithm is of very big importance. Depending on the material that needs to be reproduced, different techniques could be suitable. Some might require more computations than others but some materials are in fact more complex to reproduce visually than others. For instance, complex multi-layered materials such as cloth, paper, meat, skin, and candle wax, etc. would require a light transport algorithm that is heavier on computations. A bidirectional surface scattering reflection distribution function (BSSRDF) would be required to render such translucent materials. Such function is considered computation heavy because the

incoming light enters the surface, scatters around internally, and exits the surface from a different place at a different angle [APMSLT].

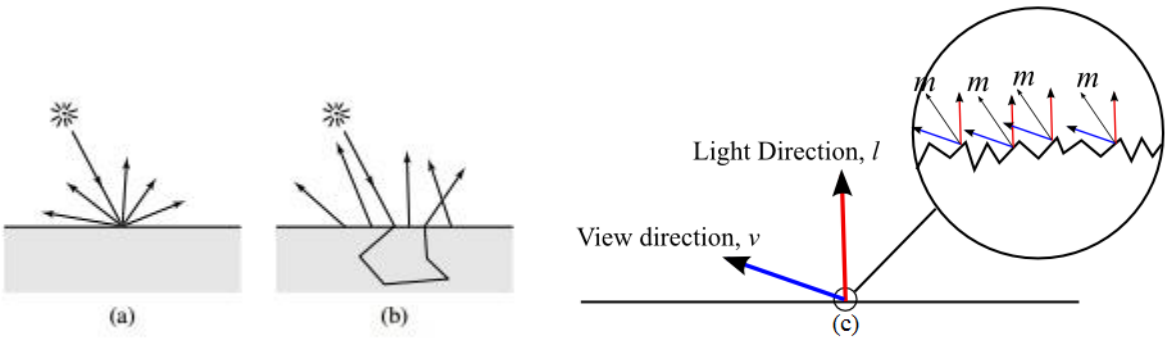

Figure 04: (a) A bidirectional reflectance distribution function (BRDF) – the light enters and exits the material from the same place. (b) A bidirectional surface scattering reflectance distribution function (BSSRDF) – the light enters the material, scatters around internally, and exits from another place [APMSLT]. (c) A Cook-Torrance reflectance model showing a surface as a set of many microfacets, each with its own surface normal represented by the letter "m" on the image.

Translucent materials definitely have the need for more accurate subsurface scattering computations to appear more realistic in computer graphics. Unlike dielectrics, conductors such as metals are hardly translucent and exhibit very little subsurface scattering if any at all. Light is interacting with metal materials just like with dielectric materials: some of the light is reflected and some is transmitted or possibly scattered. However, the absorption in metals is much higher than in dielectrics. That way, some of the light still gets reflected but the transmitted light gets absorbed almost immediately after its interaction with the material. The energy from the absorbed light turns into heat. Reproducing such a material in computer graphics would not require using a computationally heavy function such as the BSSRDF. Instead a simplified version can be used called a bidirectional reflectance distribution function (BRDF). That function is approximating the effect of the BSSRDF but instead of exiting the material at a different place, the light is assumed to enter and leave the material at the same position [APMSLT] (*See Figure 04.a*). The bidirectional surface scattering reflectance distribution function takes into account eight geometric variables, 4 for point of incidence and emergence on the surface, and another 4 for the incoming and outgoing light directions. The BRDF computes only four out of those eight for homogeneous materials, which simplifies the calculations and shortens rendering time. Among other materials, metals have extremely distinctive appearance. They have a very high level of reflectivity. When seeing a metal material one could almost always recognize its metallic nature

[APMSLT]. The specular component is of great importance because it provides highlights on a surface which gives a strong visual hint for the location of the rendered object with respect to the light sources, and the general shape of the object. The color of the specular reflection of non-metallic surfaces is usually the same as the color of illumination. However, for metals that statement is not necessarily true. Some metals change the color of their specular reflection (*See Figure 05*).

Figure 05: An image showing objects made of copper (left) and gold (right). In colored metals such as bronze, copper, and gold the specular reflection changes its color based on the material properties. In this particular example the copper pot has a yellow-orange colored specular highlight and the golden amphora has brown-yellowish color of highlights.

3.1.3 Fresnel Term and Surface Reflectance

Depending on the roughness of the object's surface, the direction of the outgoing light would behave differently for specular reflection and for transmission. Given a perfectly smooth surface, for specular reflection, the angle the incoming light and the surface normal make is the same as the angle the outgoing light makes with the surface normal [RMPLT] (*See Figure 06.a*):

$$\theta_i = \theta_o \tag{3.2}$$

The outgoing direction of the light, for transmission, is calculated using Snell's law, which relates the angle θ_i between the surface normal and the incident light ray to the angle θ_t between the surface normal and the transmitted direction. Snell's law is based on an index of refraction (*See Table 01*), which is a ratio of how much slower light speed is in a particular medium compared to vacuum. Snell's law is taking into account the index of refraction of the medium the incident light ray is currently in, and the index of refraction of the medium that is entering [RMPLT] (*See Figure 06.b*). The letter η denotes the index of refraction, the Snell's law equation is

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t \tag{3.3}$$

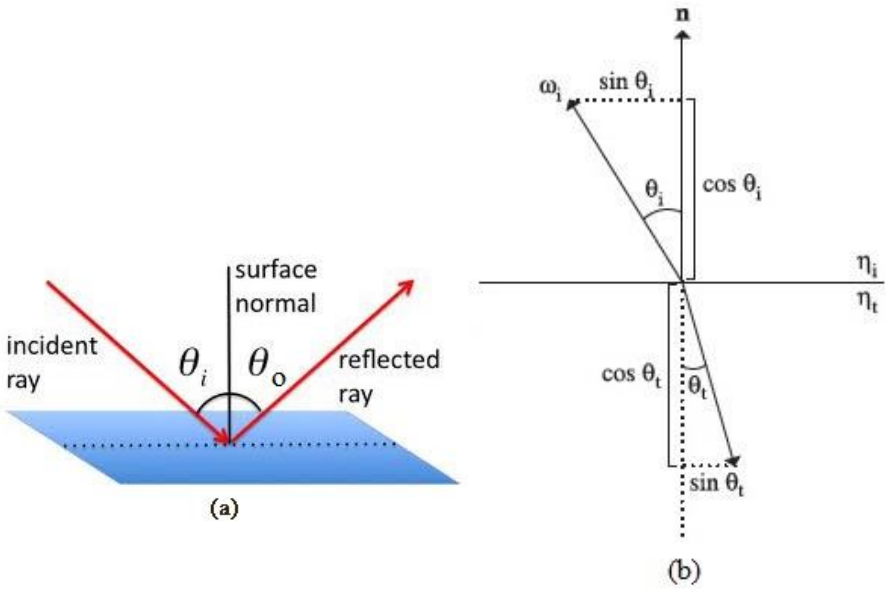

Figure 06: (a) For a perfectly smooth surface, the incident and the surface normal make an angle equal to the reflected ray and the surface normal for a specular reflection. (b) Refraction based on Snell's law using the indices of refraction of the two mediums the light passes through.

Medium	Index of refraction η
Vacuum	1.0
Air at sea level	1.00029
Ice	1.31
Water (20°C)	1.333
Fused quartz	1.46
Glass	1.5–1.6
Sapphire	1.77
Diamond	2.42

Table 01: The index of refraction is the ratio of the speed of light in a vacuum environment to the speed of light in a particular medium. This table shows indices of refraction for various mediums.

Computing the specular reflection and transmission directions is only half of the work that needs to be done. It is necessary to calculate the amount of light that gets reflected or transmitted in the directions discussed above. Simple, non-physical, render engines do not waste resources to compute the amount of reflected or transmitted light. Instead they use predetermined values for those parameters, which are uniformly spread over the entire surface. Of course, that is not the case with physically based renderers, where the reflection and refraction are directionally dependent. For dielectric media and for conductors, the Fresnel equations are the same but the refractive indices differ from each other. Those equations have two forms for each case, depending on the polarization of the incident light. For the sake of optimization and simplification, we would assume that light is unpolarized. That means it is oriented randomly with respect to the light wave. As a next step, the perpendicular and parallel polarization terms needs to be computed. Computing those terms for conductors requires an extra variable *k*, which is the imaginary part of the complex index of refraction. The Fresnel equation is as follows [PPFT]:

$$Fr = \frac{r\perp + r\parallel}{2} \tag{3.4}$$

where the perpendicular ($r\perp$) and parallel ($r\parallel$) polarization terms are given by [RMPLT]:

$$r\perp = \left|\frac{n_1\cos\theta_i - n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2}}{n_1\cos\theta_i + n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2}}\right|^2 \quad r\parallel = \left|\frac{n_1\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2} - n_2\cos\theta_i}{n_1\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2} + n_2\cos\theta_i}\right|^2 \qquad (3.5)$$

A commonly used approximation of the parallel and perpendicular polarization terms used to compute the Fresnel reflectance for conductors is:

$$r_\perp^2 = \frac{(\eta^2+k^2) - 2\eta\cos\theta_i + \cos\theta_i^2}{(\eta^2+k^2) + 2\eta\cos\theta_i + \cos\theta_i^2} \quad r_\parallel^2 = \frac{(\eta^2+k^2)\cos\theta_i^2 - 2\eta\cos\theta_i + 1}{(\eta^2+k^2)\cos\theta_i^2 + 2\eta\cos\theta_i + 1} \qquad (3.6)$$

Conductors such as metals have a fairly high level of reflectivity. Unlike dielectrics, which have low reflectance throughout most of the angular range, and a very high, almost mirror-like, near grazing angles. Smooth metal surfaces have reflectance relatively independent of angle. Of course some do not reflect as much light as others do but generally speaking metals have reflectance of over 60% for the whole angular range while dielectrics have 20% or less for most of the range [AFGBM] (*See Figure 07*).

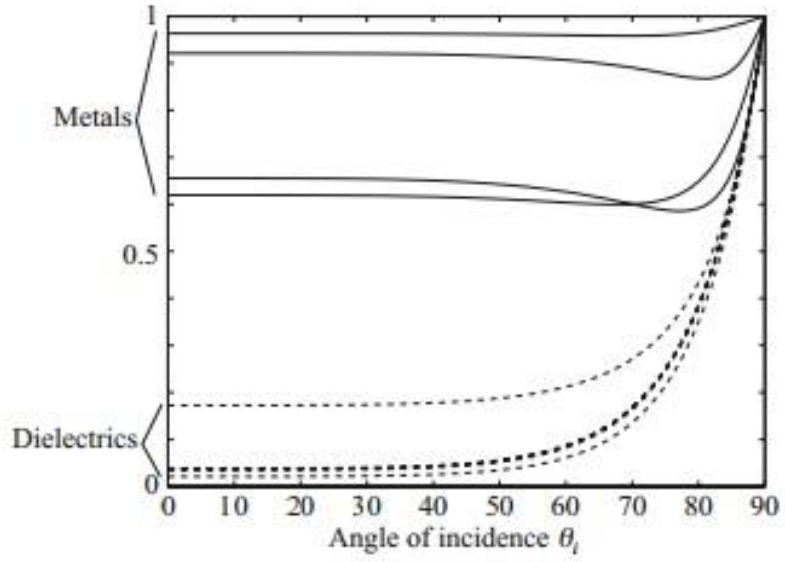

Figure 07: Fresnel reflectance graph showing the difference between dielectrics and metals. For dielectric materials surfaces have relatively low reflectance level of 20% or less for most of the angular range, while metals have pretty high level of reflectance of 60% and above [AFGBM].

The metal material shown in *Figure 08* is a computer graphics simulation of a smooth polished steel material. For the creation of that material a physically based shader has been set up. The three dimensional scene consists of a plane and a sphere, for a ground and an object to demonstrate the metal material respectively. An HRD environment map has been used to give the metallic object an environment to reflect. There is a directional light in the scene coming from the left. The scene has been set up and rendered in Autodesk's 3DS Max environment using MentalRay render engine. For the purpose of this demonstration a multi-layer type of material shader has been used as a starting point (*See Figure 09*). Since metals are highly reflective, a reflection component needs to be added by adding ray traced reflections and an HDR environment map so the object would have something to reflect. As discussed above a Fresnel term needs to be applied, which is done by adding a mask function to the reflection section in the shader properties. Eventually, the specular component has been added to bring the metal shader completeness.

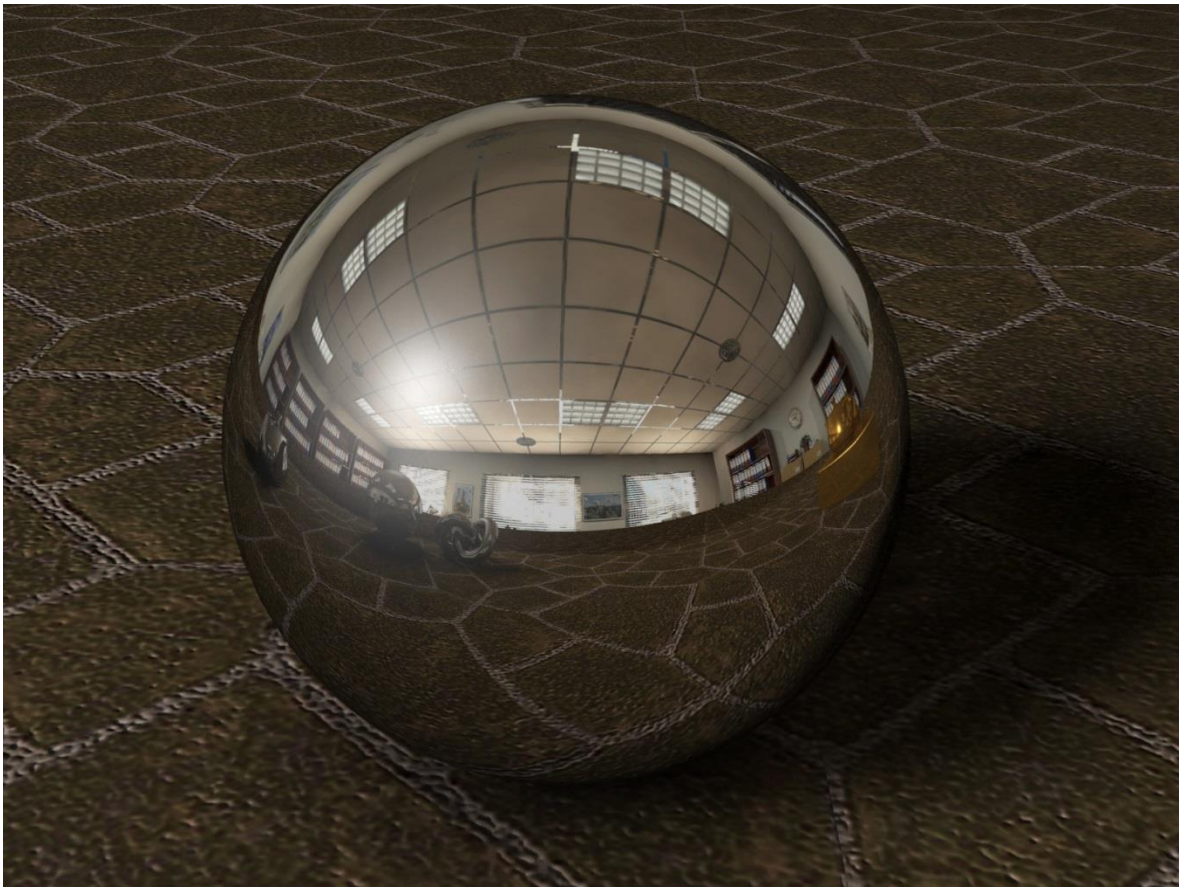

Figure 08: Rendered image of a smooth metallic surface. This is a polished surface which results in a regular shape isotropic specular reflection.

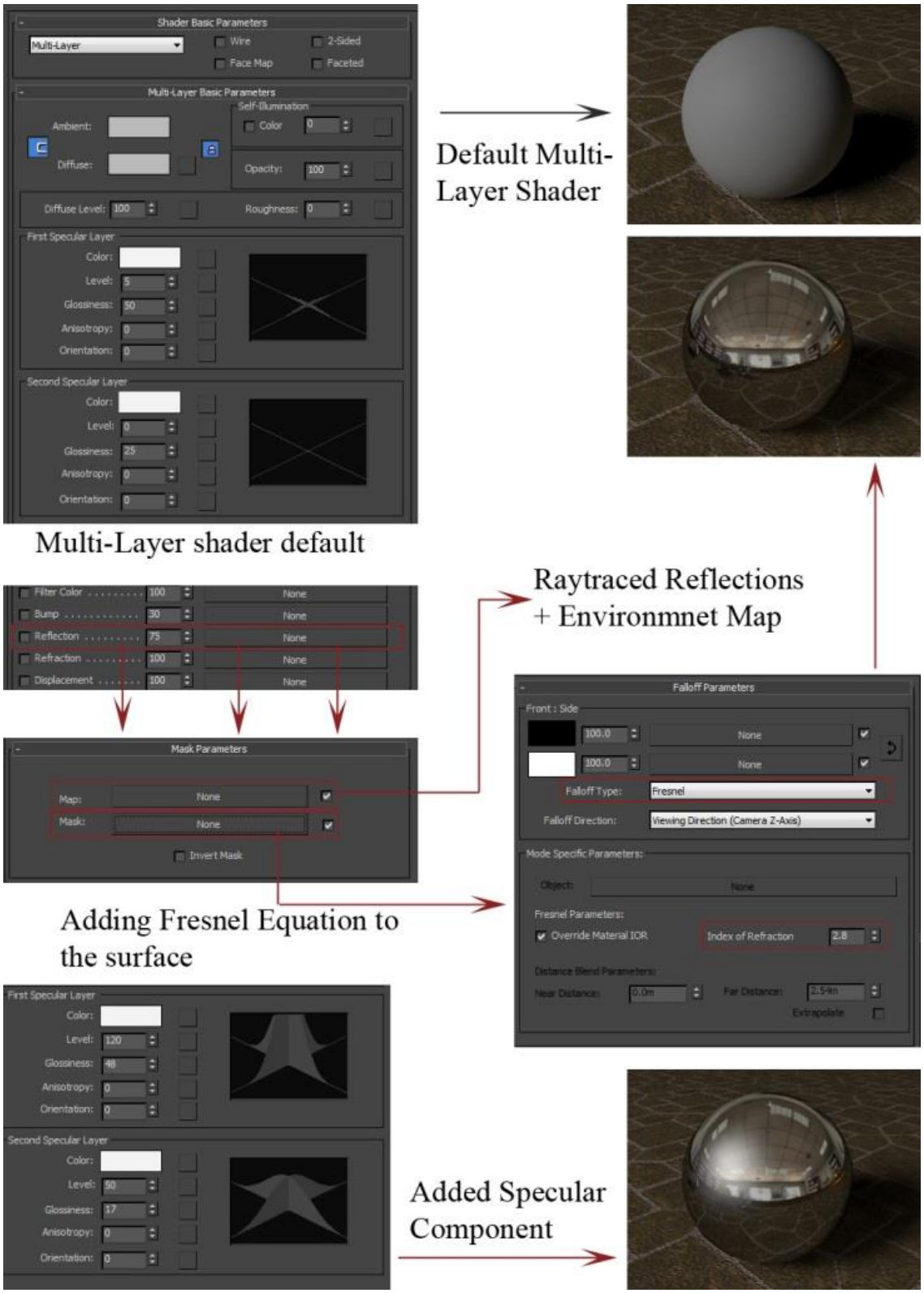

Figure 09: Setting up a metal material shader in 3DS Max rendered with MentalRay

3.1.4 Isotropic and Anisotropic Surfaces

Depending on a surface's roughness or smoothness, the characteristics of the reflected light differ. For most of the surfaces simulated in computer graphics, the light reflection is isotropic. That is, the roughness of the surface is scattered uniformly, instead of being spread out to follow a particular direction like brushed steel and aluminum. Isotropic reflection of light is independent of the angle of the surface in relation to the observer. In contrast, anisotropic surfaces have little dents and microgrooves that are mostly oriented in a given direction. As a result, anisotropic surface reflection changes depending on the viewing angle and the surface orientation [MMAR] [PRRSC]. Brushed metals possess long microgrooves which stretch anisotropic reflection across the entire surface. Such phenomenon could be seen on various brushed metal objects (*See Figure 10.a*). The reflection could even be in a circular pattern depending on the orientation of the microgrooves and dents, such an example could be seen on the bottom part of a cooking pot.

Figure 10: (a) A real life brushed metal object showing long anisotropic reflections across the entire surface. (b) A compact disk showing light diffraction effect on an anisotropic surface.

Whenever light interacts with diffractive surface, the wavelengths of light would be split out, which would cause white light to reveal its color spectrum as seen in soap bubbles, compact disks, and oil stains. When the light reflection is of isotropic type, it is usually shown as highlight rims of different color. With anisotropic surface, as the reflection is of irregular type, the diffracted light covers longer streaks resulting in an effect that could be seen on the surface of a compact disk (*See Figure 10.b*).

The computer graphics simulation shown in *Figure 11* demonstrates a physically based brushed steel material. For the purpose of this simulation, the shader used to create the brushed metal is based on the previously developed BRDF model for rendering metal surfaces. The material shader is of anisotropic type since the main goal is to simulate a brushed metal surface. An HDR environment map has also been applied to the reflectance of the material to simulate an environment around the rendered object. Also a bump map has been applied to the surface to add visible scratches. This technique does not add up to the anistropic reflectance effect, it is just for the sake of achieving a certain scratchy look to the brushed metal material. The bump map has been procedurally created from a Perlin noise function that generates a random black and white dotted two dimensional texture. After that those dots have been stretched along the x-axis to create a scratchy look to the steel surface (*See Figure 11*). According to the information on the Autodesk's official web site, the material shaders built into the software that we used as a base to create the metal material tend to be physically accurate. Unfortunately there is no clear information regarding which reflectance model those shaders are using for their computations.

3.2 Simulation of Mechanical Damage on Metal Surfaces

In reality, often times object's surface exhibit some sort of damage or abrasion over time. Depending on the material itself, damages could vary and have different effect over the surface. A damaged surface could result in change of object's geometry and appearance depending on the level of damage and the material properties. For instance, a cloth material could get its fabric strings stretched or even ripped on contact with pointy or sharp object. Some plastic surfaces could easily get scratched, broken, or even molten under certain conditions. One of the most common types of damage that metal surfaces exhibit are scratches. In real world materials, scratches are divided into two major types: individually visible scratches, also called isolated scratches, and microscratches [SSMMR] (*See Figure 12*). As the name suggests, microscratches are tiny little scratches which are individually invisible to the naked eye. Materials such as brushed metal results in a microscratched surface that could produce anisotropic reflection as explained in the previous section. Microscratches are distributed uniformly along the entire surface. Isolated scratches could be seen by naked eye on a surface. They usually appear as small grooves following a path along the object's surface. Depending on the shape of the object causing them, the pressure applied to that object, and the material properties of the scratched surface, the grooves of the scratches would have a certain depth.

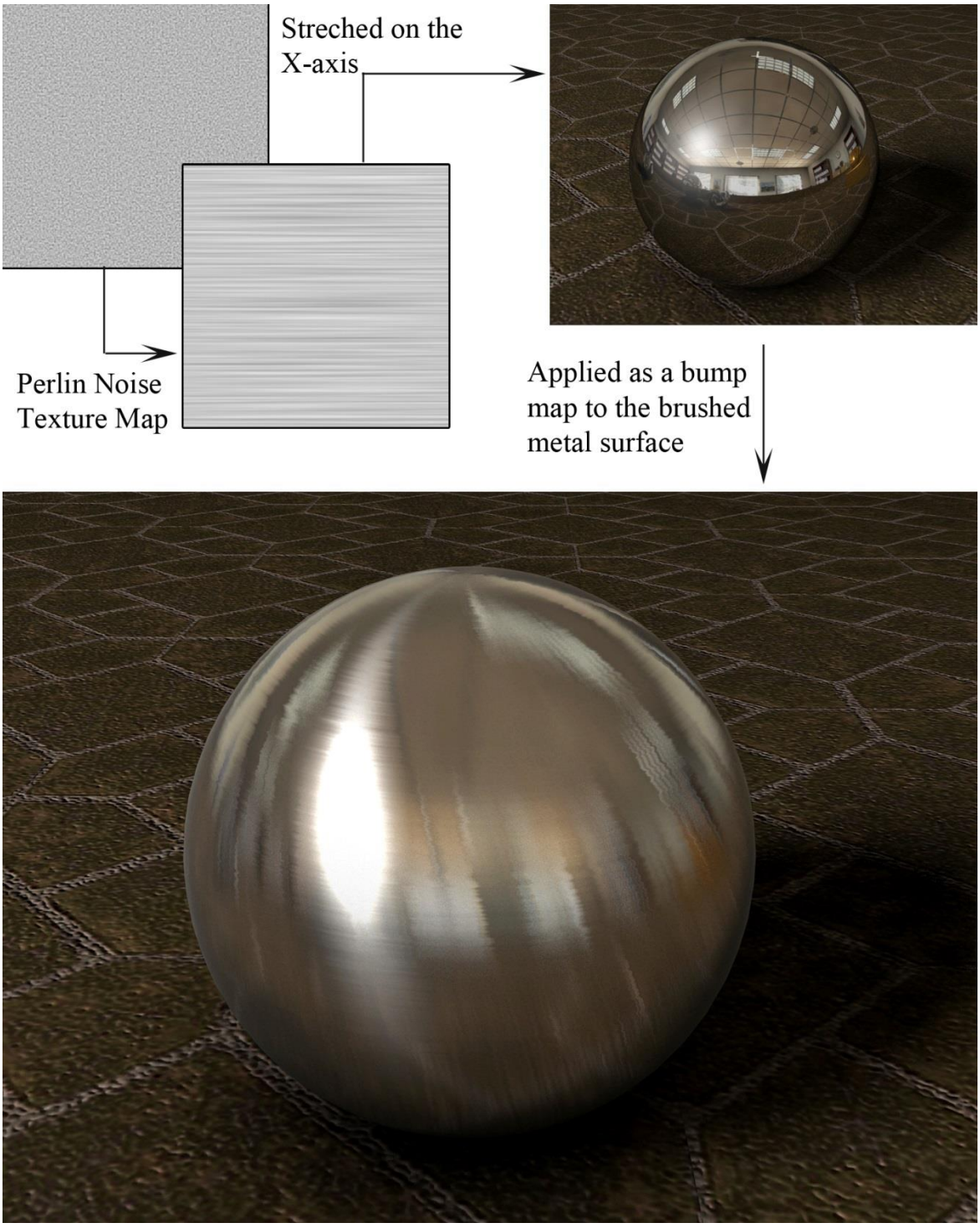

Figure 11: Rendered image of a brushed metallic surface using MentalRay render engine. For the purpose of this simulation the BRDF model used has been set up to create anistropic reflections and highlights. To enhance the visualization a procedurally generated bump map has been applied to the surface, as well as HDR environment map.

By increasing the depth of a groove, those imperfections would appear more visible to the observer. The more damage in the form of scratches there is on a surface, the less reflective it would be [APBMRRS].

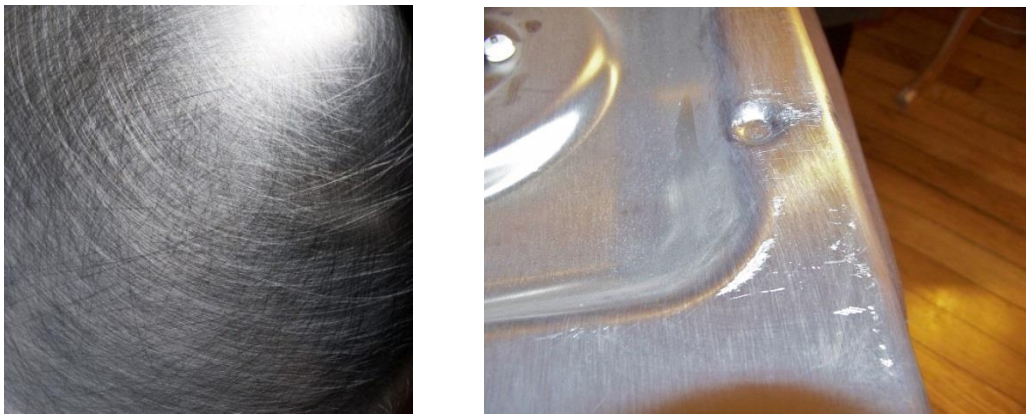

Figure 12: Real world metal materials showing isolated scratches on their surfaces. On the left there is an interior bottom of a steel cooking pot. Scratches got there from scrubbing and cleaning the cooking pot with a metal brush sponge. On the right side there is a picture of a kitchen sink.

3.2.1 Scratches on Metal Surfaces

Most of the objects we interact with on daily basis have scratches of a certain amount or possess some sort of other damage. Very rarely we can find a material that has a perfectly smooth surface. Human eyes are used to notice those types of detail, even on subconscious level. For that reason if object appears to be too perfect to an observer, it would look fake or unrealistic. Adding those types of defects to an object's surface in a computer generated image plays an important role in achieving a higher level of realism. As stated earlier, in reality scratches are imperfections of an object surface which result in change of geometry of that particular surface. In computer graphics, modeling all of those details could result in a tremendously big number of surface faces (polygons) which would be a serious performance hit when rendering the image. Not only in rendering time, but modeling such a massive amount of detail would increase modeling time as well as making such an object unusable to any kind of application in the field of computer graphics.

Some of the first to include isolated scratches into rendered images were W. Becket and N. Badler during the early 90s at the University of Pennsylvania [IRIS]. They have developed a number of ways to simulate different damages and surface imperfections such as rust, stains, mold, and scratches. Their model was using a texture to place scratches onto the surface represented by straight lines of random direction and length. The reflection of those lines was simulated by assigning random specular intensities for each scratch. However, that model completely ignored the anisotropic behavior of the surface. Later on, Lalonde and Buchanan improved the existing model by adding the specular highlight of all the scratches traversing through a texel and then during rendering add that to the reflection of the surface [AOMIIS].

Instead of modeling each scratch's geometry on a particular surface, another much more efficient method in terms of optimization could be used. Since scratches usually do not appear by themselves on a surface, but are caused by an interaction with another object, they are not randomly located on the surface. To represent scratch's exact position onto an object, a two-dimensional texture could be used to serve as a map showing where the scratches are located and the path they go along. That way, the damaged surface could have a different bidirectional reflectance distribution function (BRDF) from the area that remains undamaged [APBMRRS]. For instance, a surface that is made of smooth polished steel material would have a BRDF and Fresnel reflectance parameters of a certain type on the polished part and different settings for the part that is scratched, which is supposed to appear much rougher compared to the polished part (*See Figure 13.a*).

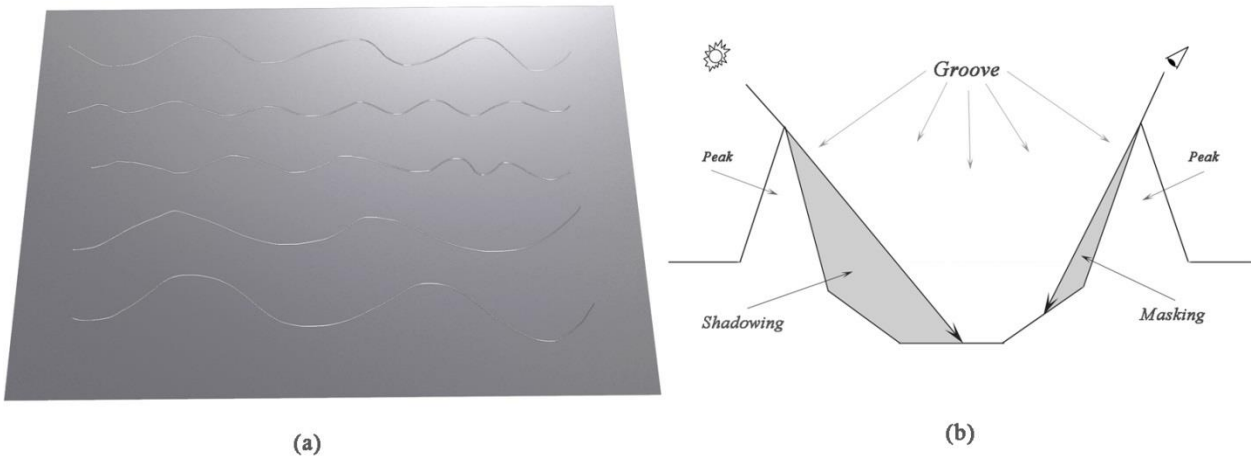

Figure 13: (a) Rendering of a scratched aluminum surface where the scratched area of the surface has a different BRDF compared to the undamaged area. (b) A close look to the geometry of a scratch showing the profile of a groove and the two peaks

surrounding it. Parts of the groove could be occluded due to the light source position or masked due to the observer's position.

On a real world metal material a scratch's geometry would normally consist of a groove and two peaks surrounding it. Just as with anisotropic type of surfaces, isolated scratches could also experience shadowing and masking depending on the viewing direction and the light source direction. For that reason they do not appear the same along their entire path. A lit area of a scratch that reflects light towards the observer without any obstacles on the way would appear shiny, and alternatively an unlit area would appear darker (*See Figure 13.b*). Researchers from University of Girona in Spain and University of Limoges in France have developed a method to recreate scratches on a surface in a computer generated environment [APBMRRS]. The method takes into account all kinds of factors such as the scratch tool shape, the pressure applied on the surface while scratching, the penetration forces, the material properties of the object, etc. All of that collected data contributes to the process of computing the BRDF of the scratched surface. That research has been done to provide a way to simulate physically-based modeling of scratches for the sake of having a virtual environment for testing materials resistance to scratches [APBMRRS].

In a computer graphics simulation instead of modeling those grooves that scratches leave on object's surface, which would be somewhat inefficient and time consuming in terms of modeling and rendering, they can be simulated through the use of a normal map. A normal map is a two dimensional texture that contains information about surface normals, where the X, Y, and Z coordinates are stored as red, green, and blue (RGB) values on the texture. That technique is used to simulate details, such as scratches on a metal surface, with no need to model them and add all of that extra geometry on an object.

Figure 14 shows a model of a Newell teapot, which is a standard primitive in a number of computer graphics software applications, named after its original creator Martin Newell in 1975 at the University of Utah. The rendered scene consists of a textured plane which serves as a floor and an environment map that surrounds the object to simulate a real scene atmosphere. The teapot itself is made of a brushed steel material. The material shader for it has been made in a similar manner as the metallic shader for brushed anisotropic surfaces described in the previous section, but the effect is more subtle, resulting in clearer reflections. As explained above, a normal mapping technique has been used to simulate scratches, imperfections, and details on the surface of the teapot while keeping the geometry in a low, manageable state. A black and white two dimensional texture map has been used to specify the exact position of the scratches

Figure 14: A rendered image of the Newell teapot, also known as the Utah teapot, demonstrating a brushed metal material with some scratches along its surface. The image is the end result of a technique explained in figure 15.

on the surface. The texture map serves as a mask where the black color of the texture map represents the main or also called base material and the white color represents the scratched material. In this particular example the scratched material is similar to the base material, only modified to appear much rougher with decreased reflectivity and much blurrier reflections. Using that method to simulate the appearance of scratches and other sorts of damage details on an object's surface is only an approximation of the effect taking place in a real world environment (*See Figure 15*).

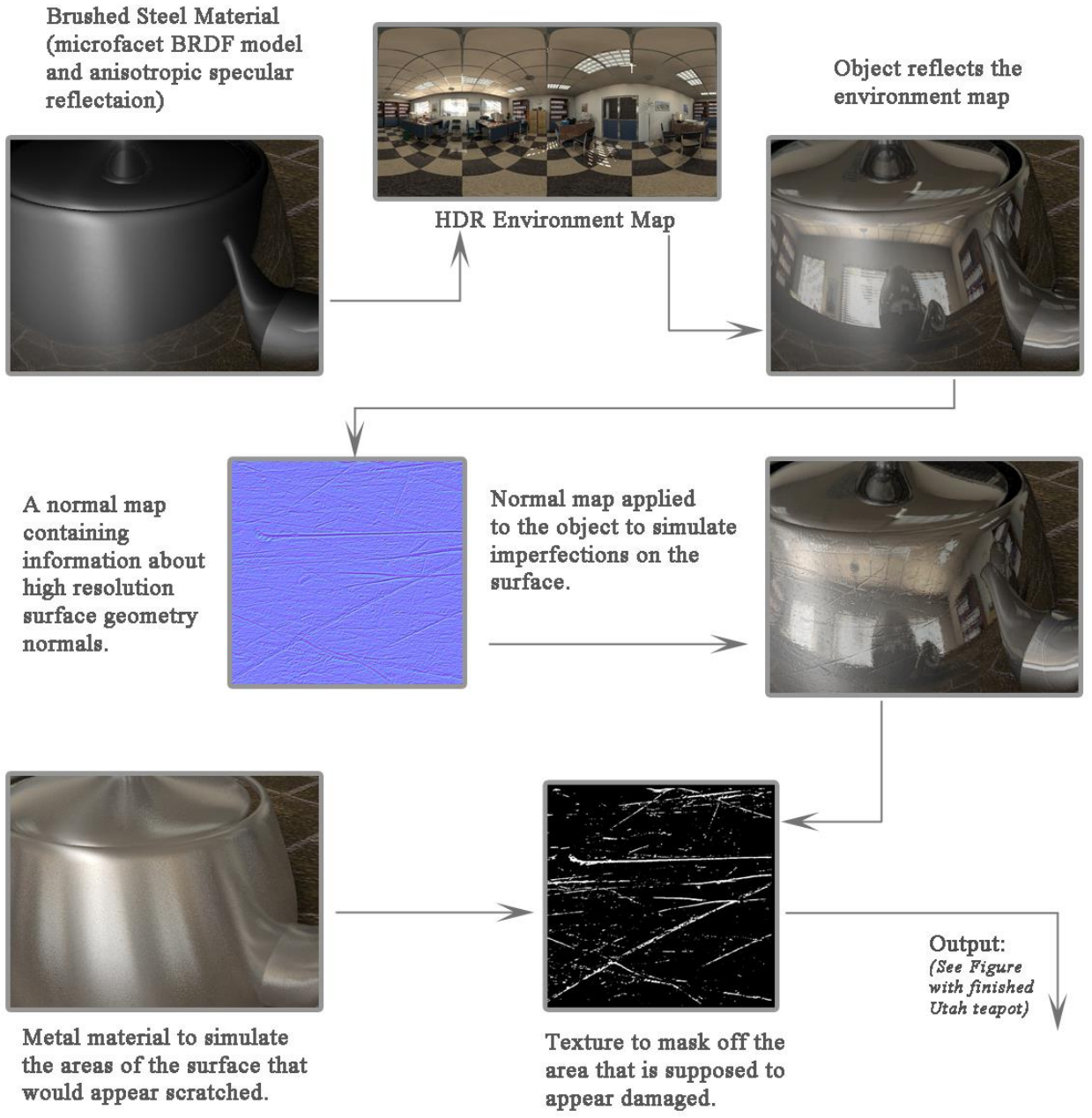

Figure 15: This model illustrates the workflow of the creation of the Newell teapot shown in figure 14. The undamaged part of the object has been created using a physically-based BRDF model and an HDR environment map to enhance its mettalic feel. The plain metal part of the surface or the so called base material is the same as the metal material described and developed in the previous section. All the surface imperfections have been simulated through the use of a normal map. The damaged area of the surface has been simply masked off and swaped with a different BRDF model as shown.

Considering how scratches would behave in a real world situation, the level of realism could be enhanced. On a smooth surface a dent, a scratch, or any type of surface imperfection would collect dirt over time. That would decrease reflectivity and make it appear more contrasty compared to the smoother part of the surface. In the image below, the methods for rendering metal materials and scratches described above have been used to visualise a low polygon model of a knight's armor. Besides the methods explained before, an additional layer has been placed as an overlay on top of the scrached area of the surface. That overlay is simply a texture representing a height map computed by taking into account the actual depth of all the imperfections. The deeper a scratch is, the more likely it would collect more dirt. That suggests the higher the number on the height map (0 to 1), the darker and less reflective a scratch would appear. The height map is represented as a texture also called cavity map which consist of grey values between the pure white and black range (*See Figure 16*).

Figure 16: A rendered image of knight's armor showing a physically-based metal material with damage (dents and scratches) applied to it. The image shows the use of an extra layer adding dirt into the scratches to enhance realism.

3.3 Simulation of Weathered Metal Surfaces

Real-world materials are constantly changing their material properties and appearance over time. Those changes are due to the dynamic environment surrounding the materials and to the constantly changing weather conditions. A complex combination of the weather conditions, such as the temperature and the air humidity level, the cleanness of a material surface (layers of dirt, dust, and grease serve as a protective coating against some aging conditions), and the material surface properties themselves are all playing an important role in the formation of aging and weathering effects such as metallic patinas, dust accumulation, erosion, mold and mildew, corrosion on metals, and many others. A surface exposed to certain weather conditions would experience a certain level of chemical decomposition and of physical changes. The properties of a material are experiencing degradation over time, a process known as aging of the material.

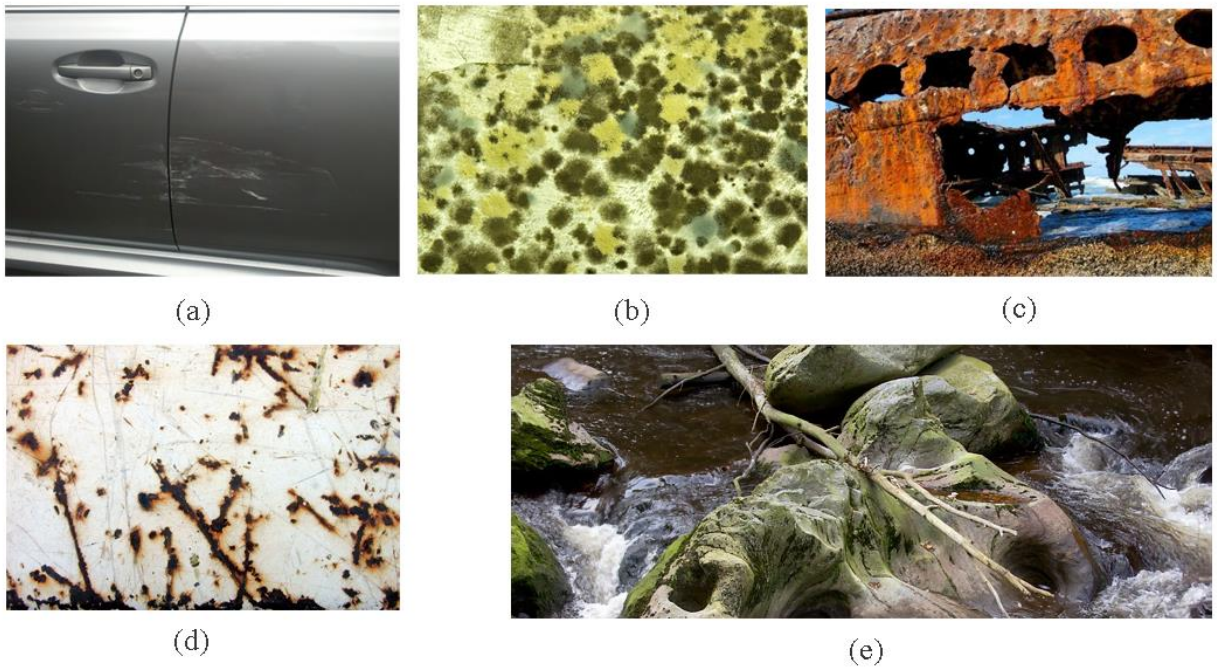

Figure 17: (a) A mechanical class of aging showing a vehicle door with scratches onto its surface. (b) A biological class of aging – a wall covered with mold and mildew. (c) A chemical class of aging – a destructive corrosion on a ship's body. (d) A combination of mechanical and chemical classes – scratched sheet of metal developing rust in the scratches. (e) A combination of biological and chemical classes – erosion of rocks having lichens and moss growing on the surface.

Aging phenomena could be classified as a process that deteriorates objects over time. That process usually takes multiple internal and external factors, such as usage mechanical damages, atmospheric conditions, inclusion of impurities into metals that lead to formation of rust, and organic material growth. Those internal and external factors are further falling into three main classes: chemical, mechanical, and biological. Under each of those classes fall aging factors, related to the specific category. For instance, the mechanical class would contain external factors that lead to geometric changes of the object's surface. Those changes are due to matter that is removed from the object's surface. That can occur at a variety of geometric scales. It could be in the form of scratches or other types of surface damage. Under the biological class, external factors such as mold and mildew growth onto a surface could be witnessed, and finally under the chemical class both internal and external factors could deteriorate the condition of the material. Some effects of aging could also introduce structural damage of the material, such as destructive corrosion that would destroy completely a metal surface over time. A real-life surface might not be affected by just a single class of the ones mentioned above but of any combination of those classes and even having factors from all three simultaneously. Moreover, simultaneous processes could influence each other (*See Figure 17*) [SAWPCG].

In a computer generated environment, a rendered image of a scene could increase its level of realism significantly by adding as much details as possible. Most of the objects in reality have some sort of surface imperfections or have experienced aging and weathering in some way. It is nearly impossible for a material surface to appear "brand new" in a real environment. For that reason, in a computer generated environment, simulating details such as aging and weathering defects is of great importance. However, the formation of those defects has many factors that take part in the process which makes it difficult to simulate in computer graphics. As mentioned above there are external and internal factors of three major classes that could be present simultaneously on a single surface. There are a lot of unique combinations that could lead to many different appearances of a certain weathering effect. For instance, in a formation of rust due to corrosion on a metal surface, the rust layers might have many variations of color patterns depending on the weathering conditions the material has been exposed to (atmospheric humidity level, wind, direct sunshine, moisture, etc.) and the material properties of the surface (the type of metal, impurities, alloys). On certain conditions rust color might be dark brown to black, and on others it might be moderate to light brown.

Simulating weathering and aging of material surfaces in computer graphics has been an area of research in the past and also active today. It is of significant importance such simulations to be as close to reality as possible in the area of architecture, for instance. That way architects and engineers could recreate a virtual environment that matches the real one where a building is supposed to be built and simulate how the materials

would react to that environment for a certain amount of time. It would be important not only for the visual appearance of the materials, but for their change of structural properties too [AAWS].

Scientists have done extensive research in the past in the field of computer graphics about simulating different aging and weathering effects. In 1982, James Blinn has done a study about simulating light reflectance behavior in clouds and dusty surfaces [LRFSCDS]. His work has originally started as an extension of simpler reflectance models developed some years prior to his work. The main focus was to address light passing through and being reflected by cloud structures. His work was extended to simulate surfaces completely covered by dust, providing a reflectance model taking into consideration light passing through small particles of dust. Julie Dorsey and some more researchers from the Massachusetts Institute of Technology has published a research paper on how aging and weathering affects rocks and stones [MRWS]. The model developed was simulating the flow of moisture and the recrystallization and transport of minerals within the stone volume. To simulate closer to reality appearance of some of the stones, a subsurface scattering reflectance model was used to simulate materials like marble and sandstone. Another research done by Dorsey in the field of weathering is simulating metallic patinas [MRMP]. The method developed for the simulation uses light reflectance and transmission model based on the Kubelka-Munk model. For the simulation of patinas, forming on the surface of copper and bronze materials, the researchers have used layered structure. Beside patinas, some research has been done in the area of destructive corrosion of metal surfaces by Stephane Merillou [CSR]. His work dealt with the simulation of corrosion over metallic surfaces resulting in rust. The formation and spread of rust was studied, as well as rendering and shading of rusty surfaces.

The focus of this thesis is on rendering metallic surfaces including changes in the material properties caused by atmospheric and environmental conditions, as well as changes caused by the time factor, known as weathering and aging respectively. In the next section some of the most common weathering effects on metals have been researched and simulated in computer generated environment.

3.3.1 Simulating Corrosion on Metal Surfaces (Rust)

Most of the metal surfaces would experience some sort of material properties change over time due to their exposition to certain environment conditions. However, on some metal materials known as precious metals, the material properties change on a very

slow rate because chemically those metals are less reactive than the rest of the metal materials. As an example of precious metals are gold, silver, and platinum. Unlike precious metals, most of the metals are getting gradually destroyed over time due to their chemical reaction with the environment surrounding them. That process is also known as corrosion. On an atomic level, a metal surface combined with oxygen forms a compound called oxide which combined with electrolyte, such as water or moisture, leads to an electrochemical reaction called oxidation. The most common case of oxidation on metals is iron oxide or more commonly known as rust. The typical rust color is in the red-brownish or yellowish shades. Although, iron has another oxide called black oxide or black rust, that forms when the surface is heated to a certain temperature. That type of rust that black oxide makes as the name suggests is black in color (*See Figure 18*).

Figure 18: (a) An example of the most common type of rust – red rust. (b) Black oxide also known as black rust. Often used as a protective coating against red rust.

Iron oxide forms on iron surfaces or alloys that contain iron in their structure. An example of an alloy containing iron is steel, which is a substance made primarily of iron with the mixture of carbon and other elements. The formation and development of rust over a metallic surface would depend on a number of internal and external factors. For instance, atmospheric and environmental conditions, such as the level of moisture a metal surface is exposed to (high air humidity, excessive rain, near water or under water environment) and chemical as well as physical structure of the object (amount of iron contained, thickness of the object or surface, presence or lack of protective coating) would be essential in the development speed of the rusting process. In environments near water containing sodium chloride such as seas and oceans where metal surfaces are with direct contact with the water or exposed to seawater mists, the rusting process speeds up significantly. Sodium chloride, commonly known as salt, when added to the

electrochemical process of oxidation of iron surfaces or metal alloys containing iron, increases the degradation rate of the material. The oxidation of other metals such as aluminum differs in the structure of the oxide that the electrochemical reaction produces. Instead of rapid degradation of the surface, the aluminum oxide makes a protective coating along the surface. Some other metals such as bronze and copper result in protective coating called patina [CUEIC] [CSR].

In the process of corrosion a metal surface gradually loses its properties. The formation of rust on a surface weakens its strength and other properties such as its electrical conductivity. The thinner a surface is the shorter it would take the corrosion process to completely dissolve the surface. The rusting appears in the form of nested layers. To protect an iron surface from rusting, a thin layer of a protective coating made out of resistant to corrosion metal could be used. For instance a thin layer of gold around an iron piece of silverware would protect it from corrosion and rust. Any method that could stop water and air to interact with an iron surface would prevent it from rusting or at least slow down the process [CUEIC].

Figure 19: A rendered image of a steel mechanical wrench demonstrating formation of iron oxide onto its surface.

Figure 19 shows a rendered image of a mechanical wrench made out of steel material with some rust spots along its surface. This wrench has been modeled in Autodesk 3DS Max software package and rendered with NVIDIA MentalRay render engine. The base material of the object is made of steel, which shader setup has been covered in detail in the previous section. In short, the brushed steel material has been simulated using a physically based anisotropic surface reflectance model. The damages such as scratches and dents along the surface have been simulated with slightly different version of the steel reflectance model which results in decreased reflectivity and blurrier reflections. A two-dimensional texture has been used as a map to describe scratches exact position onto the object's surface.

The type of corrosion effect desired onto the surface is iron oxide or more commonly known as red rust. It develops forming layers, where each layer has different age and color. Depending on many conditions rust colors may vary quite a bit which makes it a difficult task to simulate precisely. Layers also vary in age, where the older a layer is the deeper it goes into the surface. Given enough time and proper conditions, a layer of rust would go deeper and deeper into the surface until it completely dissolves it. To simulate the steepness of the rust forming layers a bump map technique was used. Unlike normal map, which contains information about surface normals in three-dimensional space storing the X, Y, and Z into the R (red), G (green), and B (blue) values of the texture, the bump map takes only two inputs. Those inputs represented by black and white values on a texture map serve as a guide to where to push or pull onto the surface. The effect is a bumpy surface under different lighting conditions. A randomly generated height map based on a procedural noise function has been created to simulate the randomly distributed layers of rust. The function is of type Perlin noise [MN], named after its inventor Ken Perlin, a professor in the department of computer science at New York University. A technique called fractal noise has been used in the generation of the height map. A fractal noise consists of multiple iterations of the Perlin noise function performed with different set of parameters. All of those iterations are then combined into a single noise map with richer level of detail. The height map generated is a greyscale texture that serves as an input to the bump function. After the bump map has been applied to the surface, it increases the level of realism.

The next step is choosing a reflectance model to simulate how light reacts with the surface. In the real world, rust appears rough and has a low level of specularity. To simulate light interaction with the surface more accurately, a reflectance model designed for rough surfaces should be chosen. Michael Oren and Shree Nayar from the department of computer science at Columbia University have developed a reflectance model designed to address rough surfaces such as concrete, sand, etc. [GLRM]. For many materials the diffuse component is often assumed to be Lambertian but for rough surfaces it would provide inadequate approximation. A surface that is a perfect diffuser

(obeys Lambert's Law) appears equally lit from all viewing angles. That would be sufficient if a surface is perfectly smooth. A rough surface could be represented as a set of differently sloped facets each with individual Lambertian reflectance. That way a surface is no longer view independent but instead it changes appearance depending on the viewing direction. The Oren-Nayar reflectance model takes into account masking and shadowing techniques explained earlier to create a more adequate simulation and more realistic end results.

Another challenge in the simulation process is the color of the rust itself. As stated earlier an iron oxide results in red-brownish color rust. Each layer of the rust has a different color because its age is different. In fact, the older a layer is the darker it appears. For instance, the oldest layer that has almost completely destroyed the iron surface has a dark-brown to almost black color. A newly formed layer results in more light-brown or red color. To simulate such a variety of colors a technique called a "gradient color ramp" has been used. Two colors have been pre-selected and a gradient ramp between them has been generated. The ramp stretches from 0 to 1 where 0 represents the dark shade color (i.e. dark-brown) and 1 represents the light shade color (i.e. red). All the values in between the two colors are mixed shades from the two. The color ramp is then matched with a greyscale ramp where each shade of grey corresponds to a specific shade from the color ramp. The previously generated height map is used as an input of the gradient color ramp (*See Figure 21*). To increase realism even further an extra Perlin noise function is performed on each layer of rust. For instance, if a relatively large layer of dark red color rust is just single colored it would look unrealistic like it has been painted. Running a Perlin noise function on that layer would bring some variety in the color and enhance realism (*See Figure 20*).

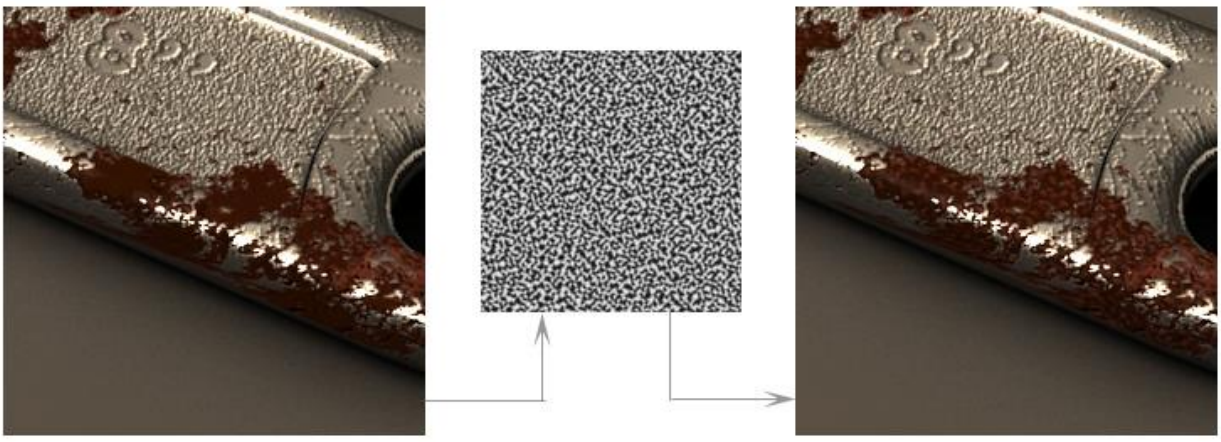

Figure 20: An extra Perlin noise function applied to each layer of rust to bring variety to the layer color. That way a dark-brown layer for example would have some small light brown and red spots.

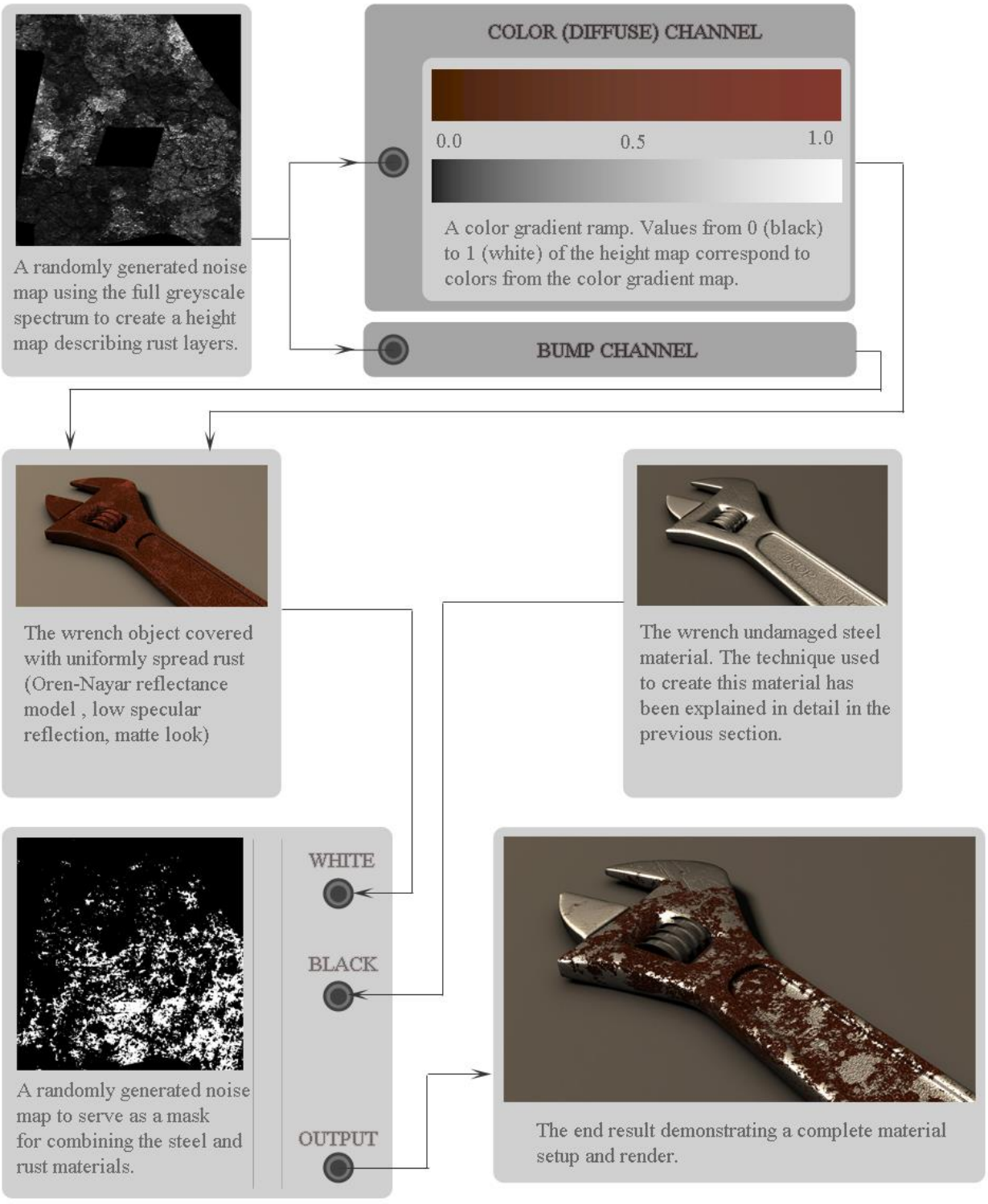

Figure 21: A workflow of the rust creation process. The technique allows the creation of rust spots along an object surface or an object entirely covered with rust.

Eventually, rust spots over the steel material need to be created. If the desired effect for the object is to be completely rusted then the rust shader developed above could be applied to the entire object. In this simulation the desired effect is to have a steel mechanical wrench with some rust spots along its surface. Using the fractal noise technique explained earlier, a black and white texture has been created to serve as a mask. The purpose of that mask is to combine the steel and the rust materials into one single material. All the white spots on the texture would get input from the rust shader, and all the black ones would get input from the steel shader respectively. A schematic view of the rust creation process could be seen on figure 21.

The method described above does not take into account internal and external factors that cause rust in reality. Instead the formation is randomly generated onto the object's surface. This research could be taken further and procedurally generate rust spots on locations that are more adequate for forming rust than others. For instance, places on an object that have direct contact with water or damages such as scratches would remove the protective coating of the surface and it would make it more vulnerable to corrosion. Even though rust formation is usually hard to predict, in certain situations it could be procedurally created.

3.3.2 Simulating the Formation and Development of Metallic Patinas

Throughout the process of oxidation among some metals, layers of coating form on an object's surface called metallic patina. That phenomenon naturally forms on the surface of copper metals or alloys that contain copper such as brass and bronze. Copper is a soft metal material used primarily in the development of electrical wires, industrial machinery parts, plumbing, and roofing. Unlike iron which oxidation process leads to the formation of rust, copper is resistant to rust. Even rust free, copper and its alloys still go through a process of oxidation. When exposed to the atmosphere, a copper surface quickly forms a thin layer of tarnish that is dark brown in color. That tarnish is caused by the formation of copper sulfide and copper oxide onto the surface. Gradually the color of the forming patina changes toward red-brownish tones. Once the base layer has formed, the new layers developing on top grow much slower. In the matter of years the process would develop to a point where the surface would be in the well-known green-bluish color tones (*See Figure 22*). Chemically, the green colored layer of patina consists of two copper sulfates: antlerite and brochantite [ECP]. The development speed of patina would highly depend on the environment the copper object is exposed

to. For instance, a patina would develop much faster near water than inland. All kinds of environmental factors play a role in the speed of development, such as annual air temperature, wind, humidity level, rain and more. Besides the natural formation of metallic patina on a copper surface, in some cases it is developed artificially. The metallic patina could serve as a protective coating on metals vulnerable to rust. It is also used as an artistic technique in many sculptures. For example after the restoration of the Statue of liberty in New York City in 1986, it has been artificially covered with bluish patina to look like its original appearance.

Figure 22: (a) A newly built copper roof on a building. (b) Dark-brown tarnish layer gradually turns into red-brownish color. (c) and (d) Green patina development aging 25 years or more.

In the field of computer graphics researchers have been trying to simulate metallic patinas. Julie Dorsey and Pat Hanrahan have developed a physically based method for modeling and rendering of such patinas [MRMP]. Their work simulates a copper surface as a set of layers. Different types of operators (erode, polish, and coat) are applied to the surface layers to simulate the formation of patina under different environmental conditions. The geometry of the object is also taken into account in the development of patinas. Using the Kubelka-Munk reflectance model, they have developed a technique for simulating the reflectance and transmission of light through the layered structure. In their model each layer inherits its input values from its predecessor allowing for a physically based realistic appearance of the surface. The Kubelka-Munk reflectance model was originally developed to simulate light transmission and reflectance of paint film. The model was developed by Kubelka and Munk in 1931. Later on their model was quickly adapted by the papermaking industry and it has been used in the prediction and measurement of color, brightness, and opacity of paper sheets for decades [KMTDOP].

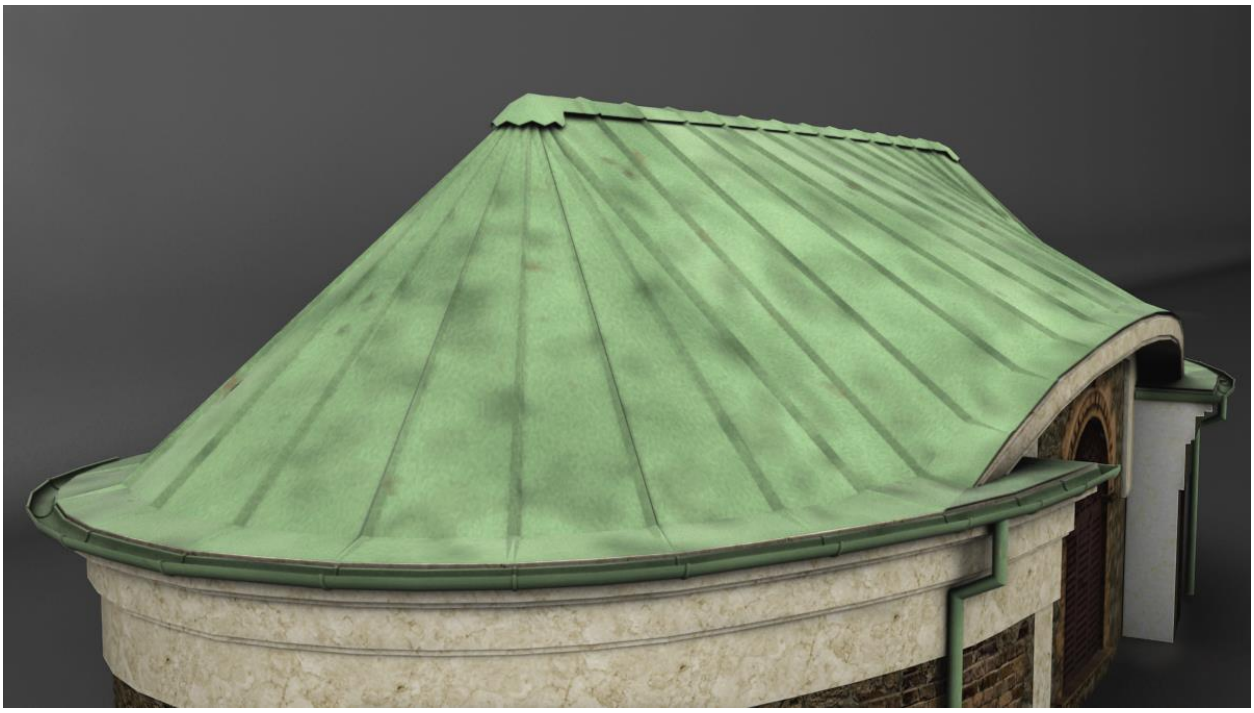

Figure 23: A rendered image of a copper roof patina formation aging 25-30 years or more. The oxidation process has covered the whole roof of the building resulting in a green color patina.

Figure 23 shows a rendered image of a building with metallic patina formed on its roof. The roof itself is made of copper material which is vulnerable to formation of patina. The rendered scene is very simple, consisting of a model of the building over a plane backdrop with grey diffuse material applied to it, one light source, and an HDR environment map that surrounds the whole scene. The purpose of this image is to demonstrate a system for simulating the development of metallic patina over a copper surface. The modeling is done in Autodesk 3DS Max and the image is rendered using MentalRay.

The copper material has been created in the same manner as the previously presented metal materials using the technique discussed earlier. A MentalRay Arch (architectural) & Design shader has been used as a base to build the copper material. The Arch & Design is a physically based type of shader that has some default material presets to be used as a starting point to achieve different architectural materials. For the simulation of the copper roof a copper type of preset has been chosen and used with the default settings. The material uses a reflectance model allowing for anisotropic type of reflectance. The copper material also changes the color of its specular component which some metals do in reality. The actual color of the illumination in the scene is pure

white but the specular highlight color of the copper roof is in the orange-brown spectrum.

As stated earlier the patina formation and development speed depends on some internal and external factors. After careful examination of the development of patina for certain amounts of time, it is clear that the color of the patina changes throughout the process (*See Figure 22*). When the formation has just begun the colors are in the dark brown shades. Gradually, colors change towards green and blue tones when exposed to oxidation process for a longer amount of time. To simulate the change of colors over time, a method similar to the rust simulation discussed earlier has been used. For demonstration purposes, a set of five different colors has been chosen, each one representing different age of the process. Since the patina formation and development is a complex process that could have many different scenarios, the colors and their age representation are only approximations of the real process. All of those colors have been put on a single line and interpolated between each other creating a color gradient ramp (*See Figure 24*). The color gradient ramp requires an input of greyscale values between 0 and 1 where certain color is matched to certain greyscale value. All of the age colors are arranged in a linear fashion where each 20% of the gradient ramp is occupied by one of the pre-selected colors and its light and dark shades. To create variety of colors and forms, a greyscale noise map has been procedurally generated and used as an input. The noise function is of type Perlin noise. To enhance more detail on the generated map, a technique called fractal noise has been performed by adding several more iterations to the base Perlin noise function. The noise map generated uses the whole greyscale spectrum which when matched to the values from the color gradient ramp would result in a messy colorful map and an incorrect overall appearance of the object. As a solution to that problem when generating the noise map, instead of using the full greyscale spectrum (black to white), a limited one could be used (i.e. black to dark grey). For instance, if the black and white gradient (0 is black and 1 is white) is split into five parts, the values from 0.8 to 1.0 (light grey to white) would be used as up and down limits in the generation of the noise map. That way, five different greyscale spectrums are used to create five separate noise maps, each corresponding to its appropriate colors from the gradient color ramp.

The metallic patina results in a relatively rough and diffuse surface. Its behavior is similar to iron oxide or red rust in terms of interaction with light. The technique for simulating light interaction with rusty surfaces discussed in the previous section could be applied to the simulation of metallic patinas as well. The patina surface uses Oren-Nayar reflectance model designed for simulation of rough surfaces at Columbia University by Michael Oren and Shree Nayar in 1994 [GLRM]. As mentioned earlier, the gradient color ramp generates a two-dimensional image based on the input from the greyscale noise map. That two-dimensional image is then mapped onto the

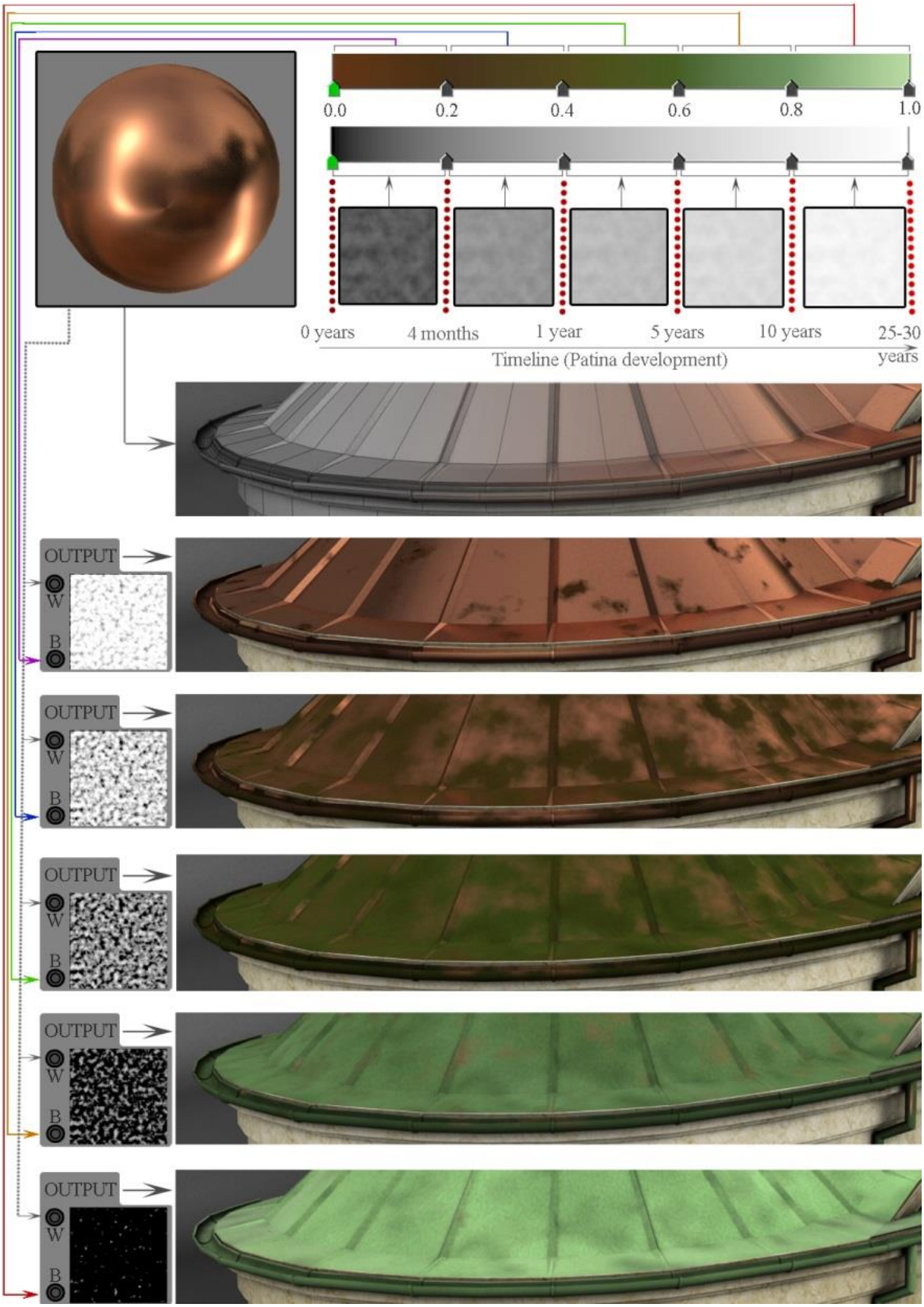
0.0
0.2
0.4
0.6
0.8
1.0
0 years
4 months
1 year
5 years
10 years
25-30 years
Timeline (Patina development)
OUTPUT
W
B
OUTPUT
W
B
OUTPUT
W
B
OUTPUT
W
B
OUTPUT
W
B

Figure 24: A schematic view of the patina formation and development simulation process. The image describes the technique of picking the appropriate patina color for a certain age on the timeline and mixing the two materials using masking technique. The result is: different color area coverage patina corresponding to an age on the timeline.

object's surface. The greyscale noise map has been used as an input to the bump channel as well. The bump functionality has been discussed in greater detail in the previous section.

Observation of patina development in reality proves that the process is of layered structure. When the forming process have just started or in early stages it results in a fewer amount of patina spots along the surface. As the time goes by new layers form, covering larger area until the whole surface is covered. To simulate the behavior of the patina development process, a masking technique has been used. The mask is a procedurally generated two-dimensional texture that serves as a map of how two materials should be blend. The different shades of grey represent different layers. The mask has been generated with a Perlin noise function. It starts with predominantly light shades of grey and gradually gets darker and covers larger area (*See Figure 24*). By controlling the size and the noise threshold within 3DS Max, allows for creating a noise map that could serve as a mask to blend the copper and patina materials. For blending two materials, the masking technique requires two inputs (each one of the materials) one goes in the black color of the mask and one in the white color respectively. All of the grey colors generated by the mask would mix the two materials. For instance, if the copper material goes in the white slot and the patina material goes in the black slot, an area of the mask that is dark grey would appear as a mix between the two but closer to the patina color. Using that technique allows for animation of the patina development process by dynamically changing the parameters of the noise functions over time.

Figure 25: Bronze coins with engraving on the surface demonstrating the formation of metallic patina in the low point areas (cracks, creases, and cavities).

The formation and development of metallic patina on a copper roof would slowly cover up the entire surface over time. That is due to the fact that the roof remains untouched by external factors such as rubbing, scratching, or any kind of interaction with another object. In a situation where an object is interacted with on a regular basis, the formation of patina on the areas of interaction slows down significantly. For instance, a bronze coin with engravings on its surface would develop patina in the areas that get touched the least (cavities, creases, cracks and scratches) much faster than the areas that are exposed to everyday contact with another object (*See Figure 25*).

Figure 26 shows a rendered image of a bronze battle shield. The image demonstrates the formation of patina in unexposed to direct contact areas of the object surface. The simulation of the patina itself has been done using the same technique used to simulate the copper roof. However, a slightly different approach has been taken when blending the patina and the bronze materials. The mask used for mixing up the two materials has not been randomly generated using a Perlin noise function, but instead pre-computed with a technique called ambient occlusion. The ambient occlusion, originally developed by Zhukov and Landis in the late 1990s, is a crude approximation of a full global illumination technique. The basic idea is to compute the occluded amount of a point on a surface by shooting random rays from that point in a hemisphere and check if those rays hit another object or not [LTPLT]. The end effect of the ambient occlusion technique is to generate deep type of shadows in the areas that are hard for light to reach. Since the ambient occlusion generates a two-dimensional greyscale texture describing the location of cavities, cracks, and not directly exposed to light spots, it could be used as a mask for blending the bronze and patina materials on the battle shield. All of the spots that appear to be in shadow would have patina forming on them, and all of the spots that are not occluded are assumed to be exposed to interaction with other objects. That way the high points of the surface would not have patina developed over them.

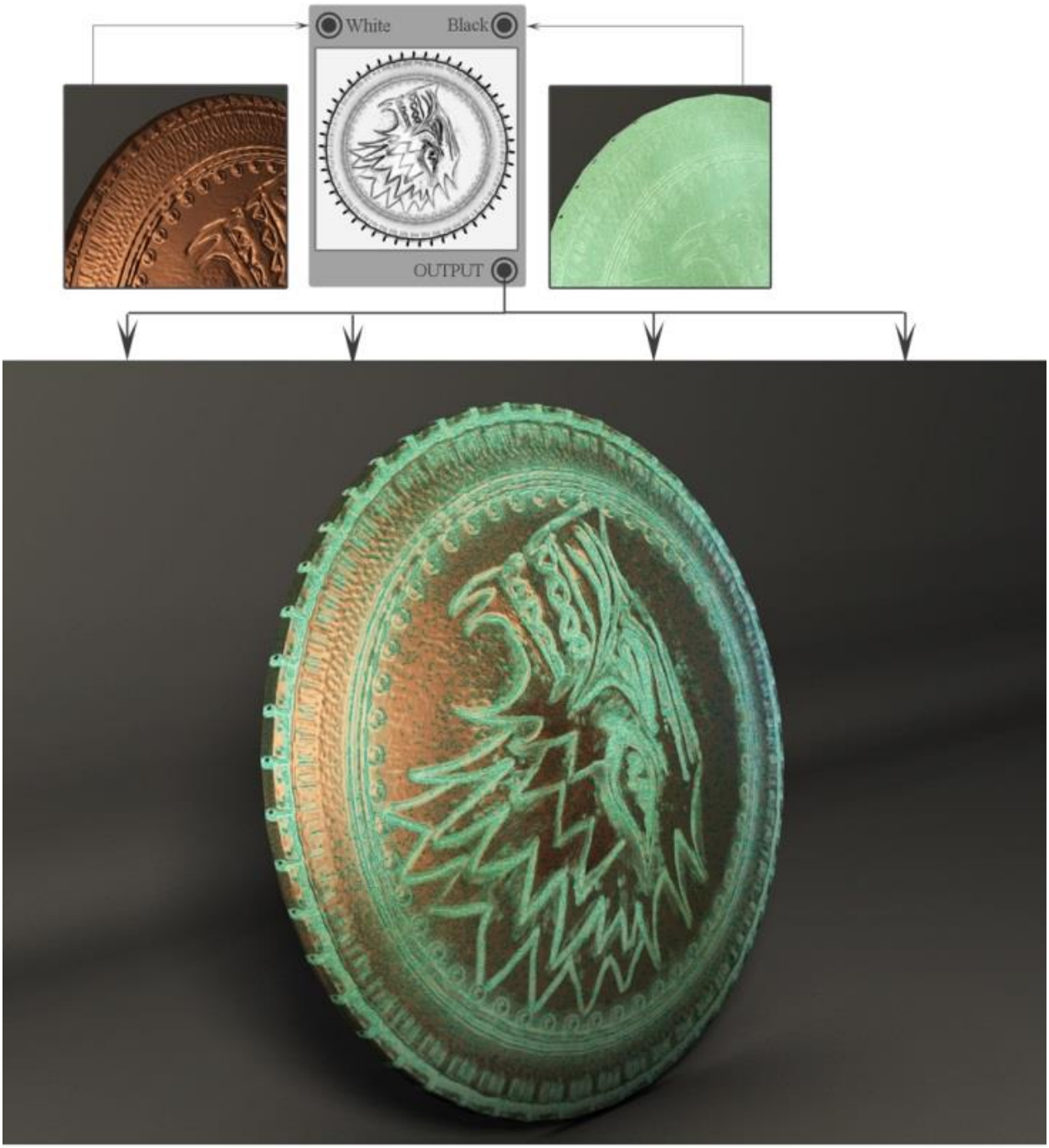

Figure 26: Workflow of a patina formation on low points of a surface as well as cracks, cavities, and creases. The technique for creating the bronze and patina materials required for the simulation of this battle shield have been explained earlier. Instead of using a randomly generated noise map as a blending mask, a pre-computed global illumination technique has been used called ambient occlusion to generate a two-dimensional blending mask.

Chapter 4

Real-Time Rendering of Metal Materials and Weathered Metallic Surfaces

Rendering a three-dimensional scene with a ray tracing render engine such as MentalRay could result in a high detail even photo-realistic image. Rendering with such an engine has many uses in computer generate animation movies, special effects, architectural visualizations, game cinematics and many more. The method has advantages and disadvantages. It can produce highly detailed renders of surface materials and special effects but it is relatively slow in production speed. Ray tracing engines use primarily the CPU (central processing unit) for their computations which could take from seconds to days to render a single image depending on the complexity of the calculations needed. What it loses in speed though it gains in quality having better approximated lighting models for more accurate physically based light simulation.

Taking the production speed of a ray tracing engine into consideration makes it inadequate solution when it comes to real-time rendering. Having a 3D interactive virtual world environment such as a computer game or an architectural walkthrough, would require a render engine that is capable of producing 30 to 100 images per second. Tracing light rays around a 3D scene is just not fast enough for such a demand. In the production of 3D interactive software such as computer games, the rendering process is primarily done on the GPU (graphics processing unit). The reason for that is the GPU is specifically designed and optimized for processing 3D graphics making it many times faster than the CPU. Only through the use of the GPU a high render frame rate could be accomplished. As a disadvantage the real-time rendering lacks precision and it results in an overall lower image quality level. Many ray tracing and global illumination techniques that significantly increase realism cannot be used with real-time rendering. Often times those techniques are simulated through highly approximated methods (i.e. ambient occlusion) or even through hand painted textures. All the per-vertex and per-pixel computation instructions are fed to the graphics processing unit through small programs called shaders. All the material properties and light reflection models are implemented in those shaders. The previous chapter introduced a method for simulating metal materials and aging or weathering effects on their surface through the use of a ray tracing render engine. This chapter focuses on using the same techniques for developing GPU shaders that could simulate metal surfaces in real-time. For the accomplishment of that goal a game engine would be needed as well as a shader programing language. For the needs of this project the chosen game engine would be Unity3D with a shader programing language that integrates very well with Unity3D called “Cg”. This chapter is split into three sections each focusing on different aging or weathering effect. First, a shader program has been built to simulate metal material and then modified to simulate scratches, rust, and metallic patina formations on the material surface.

4.1 Metal Shader

The metal shader developed to simulate metallic material surface consist of three main components: ambient, diffuse, and specular. Each one of the components contributes to the final appearance of the material. All of the components are computed separately per-pixel and then added together to get the pixel’s final color (*See Figure 27*). The shader model uses two separate passes in order to allow for rendering the contribution from multiple light sources. The end results from the rendering passes is then additively blended resulting in a single pixel color on the screen.

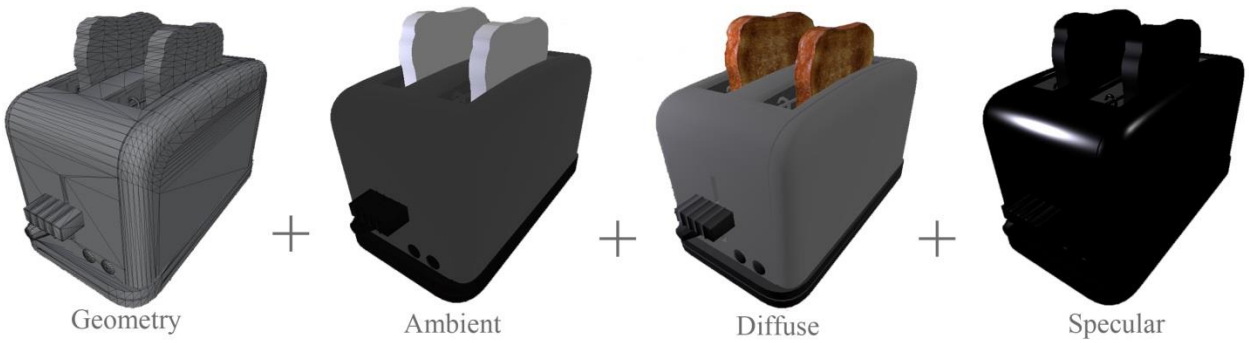

Figure 27: The ambient, diffuse, and specular components are computed separately and added together in order to achieve a metallic material applied to the 3D mesh geometry.

4.1.1 Ambient Component

The ambient term is meant to provide uniform lighting to all parts of an object's surface equally. In reality the light rays coming from a light source bounce off different surfaces and contribute to the whole scene lighting indirectly. That way, parts of an object that are in shadow do not appear completely black. The ambient term brightens up an entire object simulating indirect illumination of some extend. The way it is calculated is by multiplying the constants of ambient light uniform intensity and ambient diffuse coefficient of reflection [ICGGL].

$$Ambient\ component = \mathrm{I_a} * \mathrm{K_a} \tag{4.1}$$

where I_a is the ambient light intensity and K_a is the ambient diffuse coefficient, which has three values for colored surfaces: red, green, and blue.

4.1.2 Diffuse Component

The diffuse component for the metal shader being developed contributes primarily to the overall color of the object. The diffuse term is based on the Lambert's cosine law which states that the light intensity a surface gets is directly proportional to the angle θ between the surfaces' normal and the light direction. The closer to perpendicular to a

surface point the light source is - the more light intensity that particular surface point would receive (*See Figure 28.a*) [GLRM].

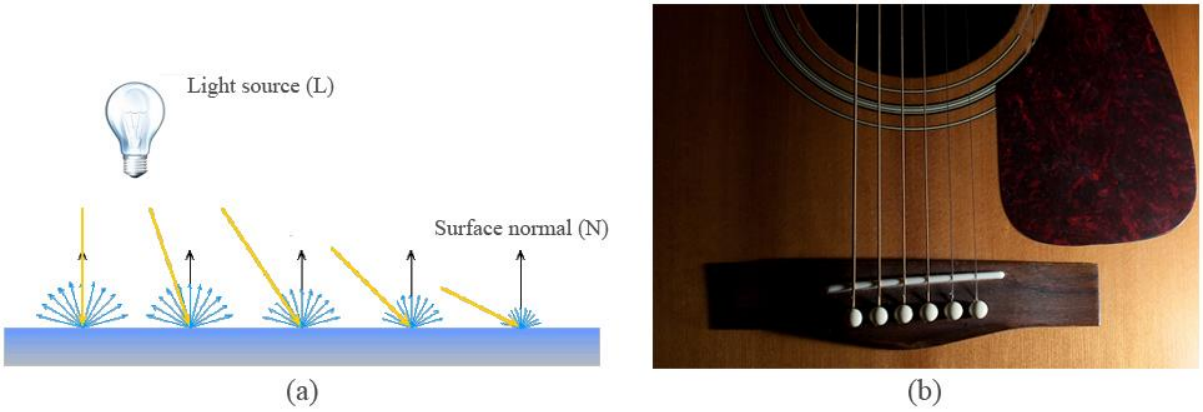

Figure 28: (a) Lambert's cosine law stating that light intensity on a perfectly diffuse surface is directly proportional to the cosine of the angle between the light source and the surface normal. (b) A photograph of a acoustic guitar showing how the light intensity decreases proportionally going away from the center of the light source.

The diffuse term takes into account the distance from the light source to the point being illuminated. The further away from the light source the less light intensity a surface would receive (*See Figure 28.b*). The light attenuation is calculated in a linear fashion:

$1/d$, where *d* is the distance between the light source and the point of interest on a surface.

Taking into account the Lambert's cosine law and the linear light attenuation, what is left to compute the diffuse term is the contribution of the light intensity L_d and the diffuse reflection coefficient K_d. Having all of that the diffuse reflectance term becomes:

$$Diffuse\ component = \frac{1}{d} Kd(N \cdot L)Ld \qquad (4.2)$$

where *d* is the distance between the light source and a point on the surface, K_d is the diffuse reflectance coefficient, L_d is the light source intensity, *N* is the surface normal direction, and *L* is the direction towards the light source [ICGGL].

Since the purpose of the shader being developed is to simulate metal material, the diffuse component calculated thus far would not be completely sufficient. Metal surfaces are usually very reflective. In order to make the diffuse term simulate metallic surface as accurately as possible, a reflection component has to be added to the equation. Ray

tracing render engines allow for ray traced reflections which produce very accurate results. In the case of a real-time rendering approach, the ray traced reflections are not possible to achieve. As a solution to that problem a technique called cube map could be used. A cube map is a pack of six 2D square texture maps, equal in size, each corresponding to one of the sides of a cube (*See Figure 29*) [GSTP].

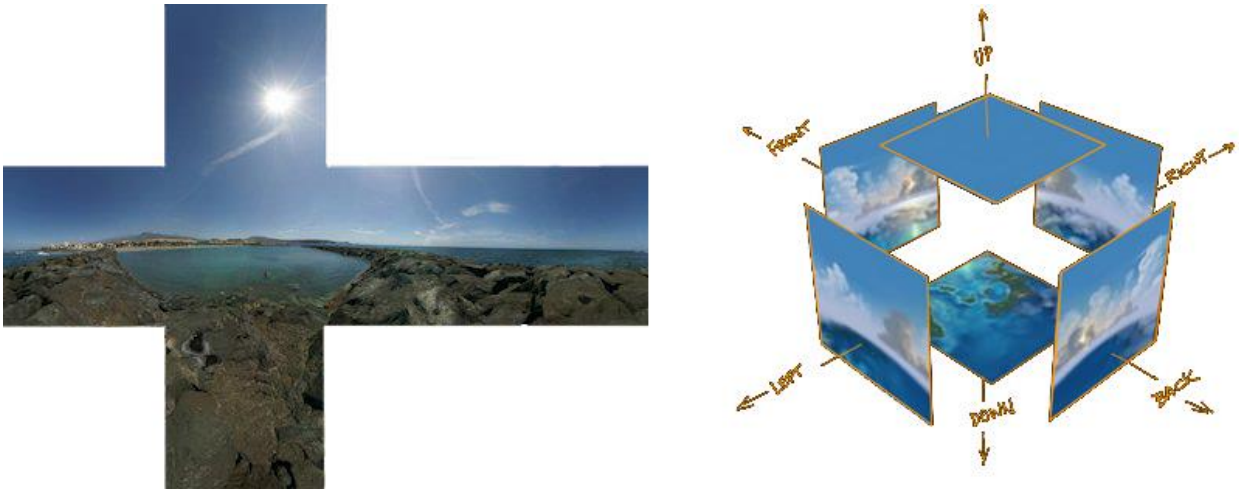

Figure 29: A cube map composed of an HDR environment map split into six equal in size square textures and wrapped around the 3D scene in a cubic form.

To accomplish real-time reflections on an object surface through the cube map technique, six 2D textures need to be fed to the cube each render frame. A virtual camera is placed in the center of the object of interest and it takes six snapshots of each direction in a cubic manner. Each of the snapshots is recorded into a 2D texture map, a technique known as “render to texture”. All of the six texture maps are then placed in a cube map, one texture per side of the cube (*See Figure 30.a*). To have a real-time updating cube map the procedure described above needs to be repeated for every frame, which makes it a relatively “expensive” technique. For the sake of optimizing the process and demonstrating how cube map technique has been used to simulate reflections, a simplified (non-real-time updating) cube map has been used. Instead of updating each of the cube faces every frame, a set of six textures has been used constantly simulating a three dimensional environment around the rendered object (*See Figure 29*).

Once we have the cube map set up, the next step would be to calculate how it reflects onto the object’s surface. The way this is achieved is by shooting a ray from the camera position towards the point of interest on the object’s surface, computing the reflected direction of the ray after it hits the surface and following that direction to the point where

it hits the cube map. A texture lookup is then performed on the cube map to determine what color corresponds to the point hit by the reflected ray. The color gathered from the texture lookup is then used as diffuse input on the object being rendered (*See Figure 30.b*).

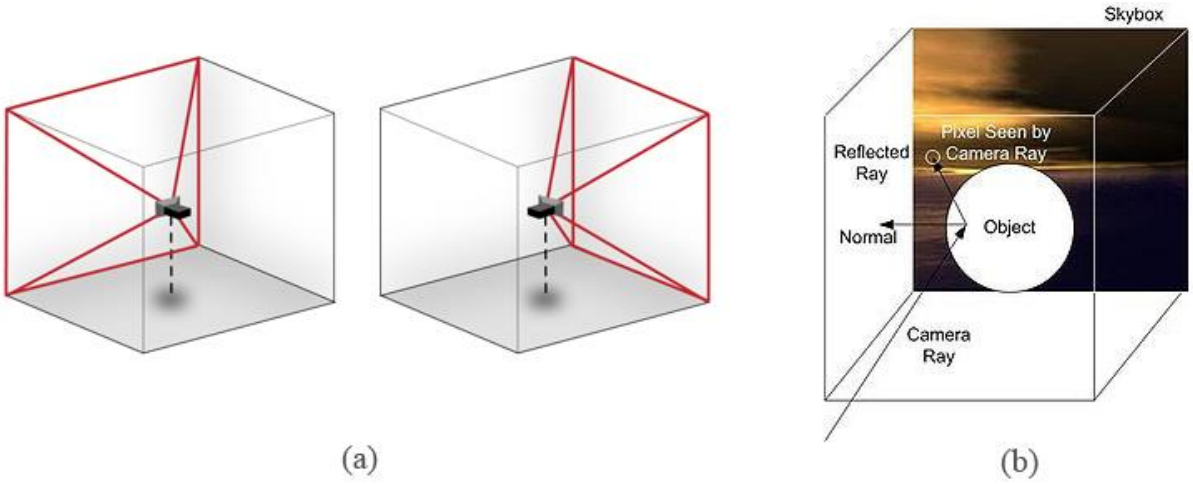

Figure 30: (a) Assuming a virtual camera is positioned in the center of an object needing a dynamic cube map. The virtual camera takes six snapshots of one of the six sides in a cubic manner using a 90 degree field of view. The resulted textures are fed to the cube map every render frame. (b) A reflected direction is computed given the camera direction and the surface normal. A texture lookup in the cube map is then performed to determine what color corresponds to the point hit by the reflection ray.

Having the surface normal direction and the viewing direction, allows calculating the reflected direction. The law of reflection states that the angle of incidence θ_i between the viewing direction and the surface normal is the same as the angle of reflection θ_r between the surface normal and the reflected direction [RRRT]. Knowing that $\theta_i = \theta_r$ the reflected direction is computed as follows:

$$R = 2N(N \cdot V) - V \tag{4.3}$$

where *N* is the surface normal direction and *V* is the viewing direction.

Adding reflections to the diffuse component surely brings the level of realism to a higher ground. In reality, for many materials the surface reflection level is not uniform over the entire surface. Most materials have certain level of reflectivity when viewed head-on and much higher level when viewed edge-on. Metals are usually highly reflective throughout most of the viewing angular range (60% and above) but become almost mirror-like at

grazing angles. That behavior is simulated through the use of Fresnel reflectance [PPFT].

The calculation of the reflectance *Fr* (Fresnel reflectance) depends on the polarization of the incident ray. To compute the Fresnel reflectance two cases of polarization need to be addressed: one for parallel and one for perpendicular polarization [RMPLT]. Having the polarization cases computed, the reflectance *Fr* is:

$$Fr = \frac{r\perp + r\parallel}{2} \tag{4.4}$$

The parallel and perpendicular polarization terms needed for the calculation of the Fresnel reflectance are as follow:

$$r\perp = \left|\frac{n_1\cos\theta_i - n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2}}{n_1\cos\theta_i + n_2\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2}}\right|^2 \quad r\parallel = \left|\frac{n_1\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2} - n_2\cos\theta_i}{n_1\sqrt{1-(\frac{n_1}{n_2}\sin\theta_i)^2} + n_2\cos\theta_i}\right|^2 \tag{4.5}$$

where n_1 and n_2 are refractive indices of the two mediums.

Computing the Fresnel reflectance term in real-time rendering would be somewhat heavy on calculations using the actual equations for the parallel and perpendicular polarization terms. There are a number of approximations for computing those terms for dielectric and conductor materials [RMPLT]. An approximation widely used to calculate the polarization of light for conductor materials such as metals is:

$$r_\perp^2 = \frac{(\eta^2+k^2) - 2\eta\cos\theta_i + \cos\theta_i^2}{(\eta^2+k^2) + 2\eta\cos\theta_i + \cos\theta_i^2} \quad r_\parallel^2 = \frac{(\eta^2+k^2)\cos\theta_i^2 - 2\eta\cos\theta_i + 1}{(\eta^2+k^2)\cos\theta_i^2 + 2\eta\cos\theta_i + 1} \tag{4.6}$$

where *k* is the imaginary part of the complex index of refraction which is related to the absorption coefficient and η is the refractive index of the metal material (*See Figure 31*).

Having the Fresnel reflectance term and the cube map surface reflections computed, the diffuse component for the metal material shader being developed becomes:

$$Diffuse\ component = \frac{1}{d}Kd(N\cdot L)Ld + CM * Fr \tag{4.7}$$

where *d* is the distance between the light source and a point on the surface, K_d is the reflectance coefficient, L_d is the light source intensity, *N* is the surface normal direction, and *L* is the direction towards the light source. *CM* represents the cube map surface reflection and *Fr* is the Fresnel reflectance term.

Figure 31: The plain diffuse component applied to a 3D model of a toaster on the far left. A cube map reflection applied to the model in the middle render and just the Fresnel term contribution on the far right.

4.1.3 Specular Component

The last part that needs to be added to the metal shader equation is the specular component. That component adds highlights to the object's surface based on a light reflection model. There have been many lighting models developed throughout the years to simulate how light interacts with matter on contact. When light touches upon an object's surface that light gets reflected, absorbed, or transmitted through the surface. In order to achieve more realistic simulation of the light-surface interaction a preferably physically based light reflectance model could be used [AIBL].

In the case of simulating the light behavior when interacting with a metal material several light reflection models could be taken into consideration depending on the desired results. The light reflectance model chosen for this simulation of metal materials is based on the Cook-Torrance reflectance model. The model was developed by Robert Cook and Kenneth Torrance in 1982 based on the original work of Torrance and Sparrow on electromagnetic waves on rough surfaces in 1967 [TOSRFRS]. Compared to the Phong reflectance model and the more accurate Blinn-Phong model developed by James Blinn in 1977 as an extension to the original Phong model, the Cook-Torrance reflectance model provides a closer to physical reality light-surface interaction. There

are a number of materials that could be simulated through the Cook-Torrance model but it is primarily targeted to simulate metal and plastic materials. The model treats a surface as a set of microfacets each reflecting the incoming light separately. The microfacet approach allows for simulating smooth surfaces as well as rough surfaces. Simulating a smoother surface would require the microfacets to be oriented in a similar direction, and to get a rougher surface would simply vary the microfacet orientation to a greater level. This approach allows for simulating a broader range of metal materials: from mirror-like chrome surfaces to brushed metal.

The specular contribution from the Cook-Torrance light reflectance model depends on three factors: the Fresnel reflectance term (F), the geometric attenuation (G), and the directional distribution of the microfacets (D) [ALMFS] [ARMCG]. Once those terms have been computed the Cook-Torrance specular reflectance is given by:

$$CTs = \frac{FDG}{\pi(N \cdot L)(N \cdot V)} \tag{4.8}$$

where *F* is the Fresnel term, *D* is the microfacet directional distribution, *G* is the geometric attenuation, *N* is the surface normal, *L* is the light direction, and *V* is the direction towards the camera.

The Fresnel term defines what fraction of the incoming light gets reflected, transmitted, or absorbed. Using the full formulas for calculating the polarization of light in the Fresnel equation (4.4, 4.5) is computationally inefficient in real-time rendering. Instead, an approximation could be used to optimize the process. A widely used approximation of the Fresnel term for specular reflections has been developed by Christophe Schlick in 1994 [IMPBR]. The Schlick's approximation is:

$$Fr = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2 + \left(1 - \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2\right)(1 - (H \cdot V))^5 \tag{4.9}$$

where n_1 and n_2 are refractive indices of different mediums, *V* is the viewing direction vector towards the camera, and *H* is the halfway vector between the viewing direction and the incident light direction vectors.

To compute the Fresnel term needed for the specular contribution of the Cook-Torrance model, the halfway vector *H* needs to be computed. That is done by simply adding the viewing vector *V*, from the surface point to the camera position, and the light vector *L*, from the surface point to the light position (*See Figure 32.a*).

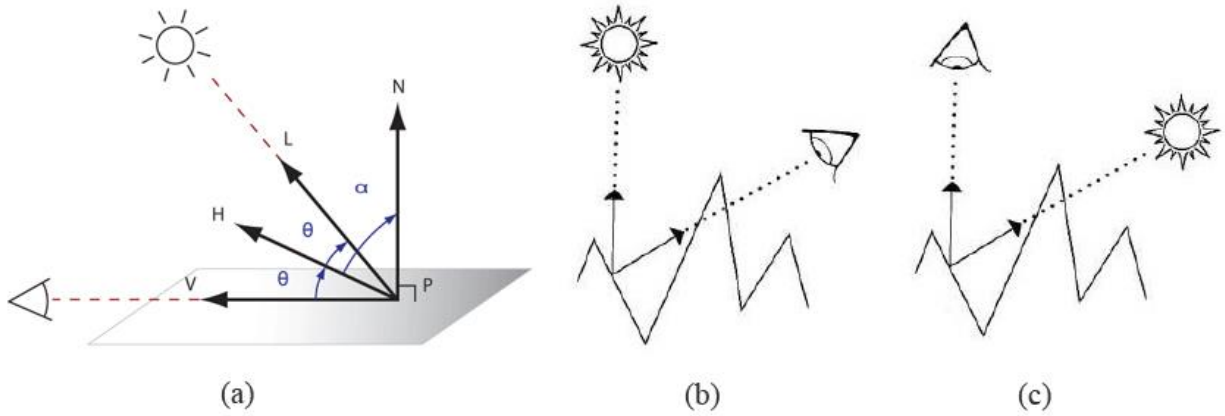

Figure 32: (a) The halfway vector H is in the middle between the viewing vector V and the light direction vector L, (b) masking – in treating a surface as a set of microfacets the reflected light from a facet is blocked by a neighboring microfacet, (c) shadowing – similar effect as masking but instead the a microfacet blocks the incoming light towards a neighboring facet.

The next thing that needs to be computed is the geometric attenuation. Since the Cook-Torrance model is physically based it has to account for shadowing and masking. Shadowing is the effect of having some microfacets blocking the incoming light towards other microfacets. Masking is a similar effect only this time the reflected light or outgoing light is blocked by a microfacet [MMRRS] [IMPBR] [ARMCG] (*See Figure 32.b and 32.c*). Since the microfacets are considerably tiny it might seem irrelevant to waste computational power for shadowing and masking but essentially they have a very high importance on the final look of the surface. Having more of the shadowing and masking effect on a surface would result in a very rough, matte appearance of that particular surface. The geometric attenuation factor that needs to be computed is the amount of light that remains after the masking and shadowing has taken place. The value for that factor is in the range between 0 and 1. There are three cases of how light would react on contact with a surface:

- no interference while reflecting the light
- shadowing – microfacets are blocking some of the incoming light
- masking – microfacets are blocking the reflected light

In the first case where there is no interference in the light reflection, the geometric attenuation factor would be equal to 1, meaning all of the incoming light gets reflected. In the case of shadowing or masking, some of the light would get reflected and some

blocked. The formulas for calculating the masking and shadowing attenuation factors are given by:

$$Gm = \frac{2(N \cdot H)(N \cdot V)}{V \cdot H} \qquad Gs = \frac{2(N \cdot H)(N \cdot L)}{L \cdot H} \tag{4.10}$$

where *Gm* is the geometric attenuation factor for masking and *Gs* for shadowing. *N* is the surface normal, *H* is the halfway vector between the viewing direction and the incoming light, *V* is the viewing direction, and *L* is the direction towards the light source.

Having those three values computed, the geometric attenuation factor *G* is the minimum value of those [ARMCG]:

$$G = \min(1, \frac{2(N \cdot H)(N \cdot V)}{V \cdot H}, \frac{2(N \cdot H)(N \cdot L)}{L \cdot H}) \tag{4.11}$$

The next thing that needs to be computed is the microfacet directional distribution. Depending on how the microfacet normals are oriented the surface might appear smooth or rough. On smoother surfaces all microfacet normals are oriented in a similar manner while on rougher surfaces their orientation distribution varies widely. The computation of the microfacet distribution could be done with several distribution functions. The Gaussian distribution is one option that could produce plausible results but it is not physically correct. The distribution function of choice for this project is the Beckmann distribution which is a physically-based and could deliver better results [MMRRS] [ARMCG]. The Beckmann normal distribution of microfacets is as follows:

$$D = \frac{1}{\pi m^2 (N \cdot H)^4} e^{(\frac{(N \cdot H)^2 - 1}{m^2 (N \cdot H)^2})} \tag{4.12}$$

where *N* is the surface normal, *H* is the halfway vector between the light direction vector and the viewing direction vector, and *m* is a user-defined variable that controls the roughness of a surface (*See Figure 33*).

Figure 33: By controlling the roughness variable (m) a simulation of smoother or rougher surface could be achieved. The figure shows a 3D render of a toaster having three different settings for the variable (m): Left – shiny smooth surface simulation (low value for m), Middle – average value for (m), Right – simulation of a rough surface (high value for m). The values for (m) in this implementation of the Cook-Torrance model are ranging from 0 to 1.

4.1.4 Implementation

For the implementation of the Cook-Torrance reflectance model, a shader language called CG has been used under the Unity3D game engine. There are two passes developed for this model, the first one accounts for the ambient light and the first light source, and the second pass accounts for every additional light source [CGT] (*See Figure 34*).

There are a number of variables defined by the user to control the appearance of the material surface. Tweaking those variables allows for simulating a broader range of materials. The variables are defined as properties in the game engine and could be tweaked at any time by the user. That allows for creating very dynamic materials that could change in appearance over a period of time or triggered by some sort of an event in game runtime. The user-defined variables are implemented as follows:

```
1  // User-defined surface color (white by default) allows for tinting the surface
2  _Color ("Diffuse Color", Color) = (1.0, 1.0, 1.0, 1.0)
3  // User-defined specular color (white by default) allows for tinting highlights
4  _SpecColor ("Specular Color", Color) = (1.0, 1.0, 1.0, 1.0)
5  // Index of refraction of the second material n2, assuming n1 is air with ior = 1.0
6  _IOR ("Index of Refraction", Float) = 1.5
7  // The absorbtion coefficient needed to compute the complex Fresnel term
```

```
8   _K ("Absorbtion Coefficient", Float) = 0.0
9   // The level of roughness of the surface
10  _M ("Roughness: 0 - 1", Range(0.01, 1.0)) = 0.3
11  // Level of surface reflectivity
12  _Refl ("Level of Reflection: 0 - 1", Range(0.0, 1.0)) = 0.5
13  // Environment map slot (Cube Map Reflections)
14  _Cube ("Reflection Map (Environment)", Cube) = "white" {}
15  // Normal map slot
16  _BumpMap ("Normal Map", 2D) = "bump" {}
17  // Specular map slot
18  _SpecMap ("Specular Map", 2D) = "white" {}
```

Figure 34: 3D model of a toaster rendered in real-time (100fps) with Unity3D engine. The metal material described in this section is the one used in this particular render. The shader is based on the Cook-Torrance reflectance model.

The normal map and spec map are not needed for the implementation of the basic shader but are required for the creation of the damaged and weathered surface shaders explained later on in this chapter.

As stated above the metal shader final contribution requires the addition of three components: ambient, diffuse, and specular. All of the following computations are per-pixel and take place in the fragment program. For the ambient component Unity3D provides a built-in variable giving the ambient intensity and color.

```
1 // Ambient Component contribution
2 float3 ambientComponent = float3(UNITY_LIGHTMODEL_AMBIENT);
```

Computing the diffuse component for the metal shader being developed uses the Lambertian reflectance model as a starting point. Additionally the diffuse component requires a light attenuation factor, environment surface reflections through the use of a cube map, and Fresnel reflectance term. The computation of the elements needed to get the diffuse component requires the use of the vectors for the surface normal, viewing direction, and light direction. In computing the light attenuation factor a distinguishment between directional and point/spot lights should be made because light attenuation depends on the position of the light with respect to the object and directional light's position is not been taken into account, only direction matters. Unity3D provides a 4 dimensional vector that holds the position of a light source in XYZ coordinates in world space. The 4th dimension W contains a value of 0 or 1 and is meant to specify the type of light source: directional or point/spot respectively. The code snippet below shows only operations from the fragment program which gets some of its input from the vertex program.

```
1  // Getting the normalized surface normal
2  float3 N = normalize(i.normalDir);
3  // Computing the viewing direction towards the camera
4  float3 V = normalize(_WorldSpaceCameraPos - i.posWorld.xyz);
5
6  float attenuation;              // Declaring a variable for light attenuation
7  float3 L;                       // Declaring a vector for the light direction
8
9  if(_WorldSpaceLightPos0.w == 0.0f)          // 0.0 = Directional light
10 {
11     // No attenuation (directional light position does not matter)
12     attenuation = 1.0f;
13     L = normalize(_WorldSpaceLightPos0.xyz);    // Light direction vector
14 }
15 else           //Point and/or spot light
16 {
```

```
17      // Direction from the vertex being processed to the light source
18      float3 vertexToLightSource = float3(_WorldSpaceLightPos0 - i.posWorld);
19      // Distance from the vertex to the light source (needed to compute attenuation)
20      float distance = length(vertexToLightSource);
21      attenuation = 1.0f / distance;              // Linear light attenuation
22      L = normalize(vertexToLightSource);     // Light direction vector
23  }
```

To calculate the reflection on a surface point the reflected direction needs to be computed. As explained in formula (4.3) the reflected direction is computed by giving the viewing direction and the surface normal direction. The method *reflect()* is a built-in method in the CG API and it returns the computed reflection direction based on formula (4.3). The method *texCUBE()* performs a texture lookup in the cube map based on the reflected direction given. The color gathered from the texture lookup is then multiplied by a user-defined variable between 0 and 1 for controlling the amount of reflectivity.

```
1  // Calculating the reflect ray direction used for the cube map texture lookup
2  // multiply by the level of reflectivity variable to increase/decrease reflections
3  float3 cubeRefl = texCUBE(_Cube, reflect(-V, N)) * _Refl;
```

To account for physically based reflectivity, the amount of reflection should be angle dependent. The more perpendicular to the viewer the surface is the less reflective it should be. That effect is achieved by approximated light polarization terms (4.6) required for the computation of the Fresnel equation (4.4). The parallel and perpendicular polarization terms have been implemented inside a method that returns the Fresnel contribution. The method takes in three parameters: the index of refraction of the metal material, the absorption coefficient, and the cosine of the angle θ.

```
1   // A method for calculating the Fresnel reflectance term
2   float fresnelTerm(float ior, float k, float cosTheta)
3   {
4      // Parallel light polarization
5      float r1 =
6      ( (pow(ior, 2.0f) + pow(k, 2.0f)) * pow(cosTheta, 2.0f) - 2.0f * cosTheta + 1.0f ) /
7      ( (pow(ior, 2.0f) + pow(k, 2.0f)) * pow(cosTheta, 2.0f) + 2.0f * cosTheta + 1.0f );
8
9      // Perpendicular light polarization
10     float r2 =
11     ( (pow(ior, 2.0f) + pow(k, 2.0f)) - 2.0f * ior * cosTheta + pow(cosTheta, 2.0f) ) /
12     ( (pow(ior, 2.0f) + pow(k, 2.0f)) + 2.0f * ior * cosTheta + pow(cosTheta, 2.0f) );
13
14     // Fresnel term equation
15     return 0.5f * (r1 + r2);
```

Having all the necessary parts required for the calculation of the diffuse component, the final diffuse contribution is computed by:

```
float3 diffuseComponent =
cubeRefl + attenuation * float3(_LightColor0) * float3(_Color) *
max(0.0f, dot(N, L)) * fresnelTerm(_IOR, _K, max(0.0f, dot(V, N)));
```

where *_LightColor0* is the color of the light source, *_Color* is the user-defined diffuse color, *_IOR* is the user-defined index of refraction of the metal material, *_K* is the absorption coefficient.

The next component that needs to be implemented is the specular component. As shown in equation (4.8) the specular highlights are based on the Cook-Torrance reflectance model. The contribution of the specular highlights depends on three factors: the Fresnel reflectance term (4.9), the geometric attenuation factor (4.11), and the Beckmann microfacet normal distribution (4.12). For the Fresnel reflectance term the approximation of choice is the Schlick's approximation (4.9). It has been implemented into a method that takes two parameters as an input to the formula calculation. It is similar to the method described earlier, except the Schlick's approximation does not include the absorption coefficient into the equation.

```
// A method for calculating the Fresnel reflectance term (Schlick's approximation)
float fresnelSchlick(float ior, float cosTheta)
{
    float r1 = pow((1.0f - ior) / (1.0f + ior), 2.0f);
    return r1 + (1.0f - r1) * pow(1.0f - cosTheta, 5.0f);
}
```

The specular component only has certain contribution when the surface is facing the light source in a range of degrees, if the light source is behind the surface then the specular component is simply 0. Prior the calculation of the Fresnel reflectance, the geometric attenuation, and the microfacet distribution several dot product operations need to be performed. Those dot products are then used as input in the computation of the Cook-Torrance lighting model.

```
// Computing the specular component contribution
float3 specularComponent;          //Declaring a variable for the specular component

if (dot(N, L) < 0.0f)              //If the light source is on the opposite side
{
    specularComponent = float3(0.0f, 0.0f, 0.0f);       // No specular highlight
}
```

```
8   else     // Light source on the correct side of the surface
9   {
10     // Computing the halfway vector (H)
11     // between the view direction (V) and the light direction (L)
12     float3 H = normalize(V + L);
13
14     // Required dot product operations for the Cook-Torrance lighting model
15     float HdotV = max(0.0f, dot(H, V));
16     float HdotN = max(0.0f, dot(H, N));
17     float VdotN = max(0.0f, dot(V, N));
18     float LdotN = max(0.0f, dot(L, N));
19
20     // Calculating fresnel term approx for metals (F)
21     float F = fresnelTerm(_IOR, _K, max(0.0f, HdotV));
22
23     // Calculating geometric component (G)
24     float G1 = (2.0f * HdotN * VdotN) / HdotV;
25     float G2 = (2.0f * HdotN * LdotN) / HdotV;
26     float G = min(1.0f, min(G1, G2));
27
28    // Calculating microfacet distribution (D)
29     float D1 = 1.0f / (3.1415f * pow(_M, 2.0f) * pow(HdotN, 4.0f));
30     float D2 = exp((pow(HdotN, 2.0f) - 1.0f) / (pow(_M, 2.0f) * pow(HdotN, 2.0f)));
31     float D = D1 * D2;
32
33     // Calculating the final specular component contribution
34     specularComponent =
35     // Adding a small constant to the denominator to prevent division by 0
36     float3(_SpecColor) * ((F * G * D) / 3.1415f * (VdotN * LdotN + 1.0e-7 ));
37  }
```

Once the ambient, diffuse, and specular components are computed, they are simply added together to contribute to the pixel's final color.

```
1  float4 frag(fragmentInput i) : COLOR                    // FRAGMENT PROGRAM
2  {
3     ……..
4     ……..
5     ……..
6     return float4((ambientComponent + diffuseComponent + specularComponent), 1.0f);
7  }
```

4.2 Damaged Metal Surfaces

The metal shader developed thus far has been used as a starting point in the development of a shader that could simulate damaged metallic surface. Figure 35 shows an optimized for real-time rendering, low poly 3D model of a battle axe made of steel and gold materials. The shader applied to the 3D mesh is the Cook-Torrance metal shader described in the previous section.

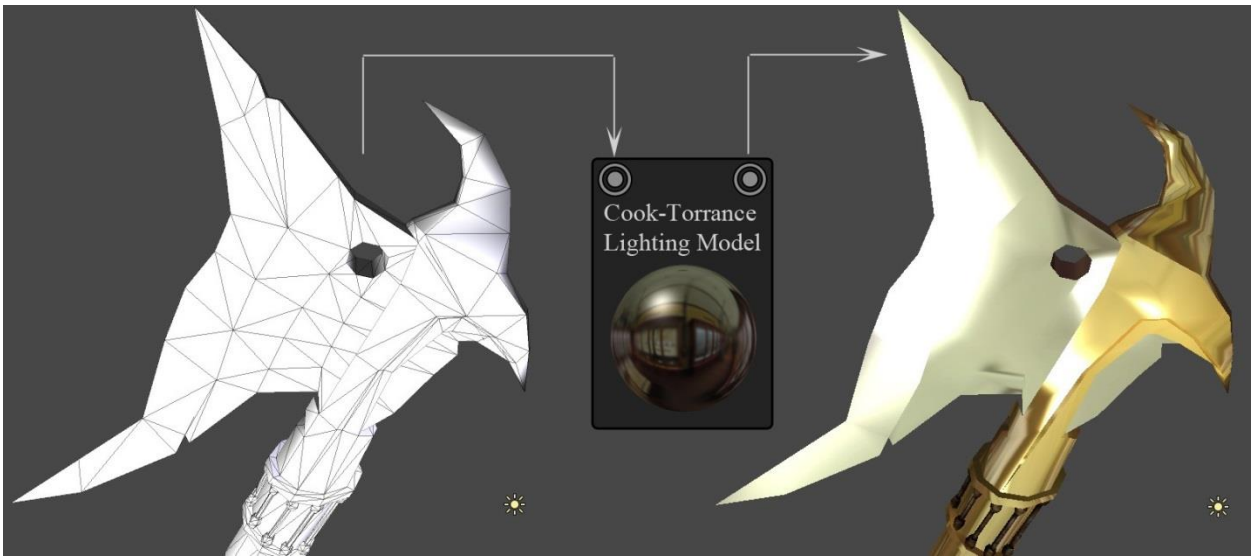

Figure 35: The Cook-Torrance based metal shader applied to a low-poly 3D model of a battle axe rendered in real-time with Unity3D engine.

The damage on the surface would be in the form of scratches, varying in size and depth. The scratches are not procedurally created on the surface but hand drawn in external sculpting software. The technique allows for placing scratches on specific places on the object's surface. Since the object has to have a low amount of surface geometry (polygons) to be manageable in a real-time rendering environment, the surface damage had to be simulated through the use of a normal map. The normal map is created by baking the surface normal information from the high detail 3D sculpt into a two-dimensional texture map. Instead of using the direction of the actual surface normal vectors when the lighting is computed on the low polygon model, the ones from the normal map are used to create an illusion of a bumpy surface. Since the texture map containing the normal information is encoded, the fragment shader needs to decode that normal vectors and use them on the place of the original surface normal vectors [GSTPDT].

Unity3D uses a local space coordinate system to specify the normal vectors. The z-axis of the local space coordinate system is represented by the interpolated normal vector (N), and the x-axis and y-axis are forming a two-dimensional plane tangent to the surface. The x-axis is represented by a tangent vector (T), and the y-axis by a binormal vector (B). The binormal could be calculated by the cross product of the normal (N) and the tangent (T). Since the normal, tangent, and binormal are in local space coordinates they need to be converted into world-space coordinates. The tangent vector (T) is transformed to world-space coordinate system by multiplying it with the model matrix since it specifies a direction between two points on a surface. The normal vector (N) is transformed with the inversed model matrix because it is orthogonal to the surface. The binormal vector (B) is simply gathered from the cross product of the normal and tangent vectors in world-space coordinate system. The transpose from local-space to world-space of those vectors needs to be done in the vertex shader [CGT].

```
1  fragmentInput vert(vertexInput i)      // VERTEX PROGRAM
2  {
3     fragmentInput o;
4  ……….
5  ……….
6     o.normalDir = normalize(float3(mul(float4(i.normal, 0.0f), _World2Object)));
7     o.tangentWorld =
8           normalize(float3(mul(_Object2World, float4(float3(i.tangent), 0.0f))));
9     o.binormalWorld = normalize(cross(o.normalDir, o.tangentWorld));
10    o.uv = i.texcoord;
11
12    return o;
13 }
```

where *_World2Object* and *_Object2World* are Unity3D built-in matrices representing the inversed model matrix and the model matrix respectively.

Once the normal map texture is created and ready to be used, the fragment shader needs to compute the modified normal direction vector by decoding the normal map. The normal map texture color is stored in a two component image: color component and alpha component. The color component contains red (R), green (G), and blue (B) channels, each of which contains the same value. In Unity3D the green (G) channel has to be used together with the alpha component (A) which are represented by numbers between 0 and 1. Using the green (G) and alpha (A) components the normal vector N' (N'_x, N'_y, N'_z) could be computed in local-space coordinate system [CGT].

$$N'_x = 2A - 1 \qquad N'_y = 2G - 1 \qquad N'_z = \sqrt{1 - N_x^2 - N_y^2} \tag{4.13}$$

The implementation of the method described above is as follows:

```
1  // Performing a texture lookup to get the normal map values
2  float4 normalMap = tex2D(_BumpMap, i.uv);
3
4  // Computing of the modified vector N from the normal map
5  float3 localCoords = float3(2.0f * normalMap.ag - float2(1.0f), 0.0f);
6  localCoords.z = sqrt(1.0f - dot(localCoords, localCoords));
```

Having the T, B, and N converted to world-space a matrix needs to be constructed that could transpose the N'_x, N'_y, and N'_z from local-space to world-space [GSTPDT]. In the shader language CG matrices are constructed row-by-row [CGT]. The transpose matrix for mapping N_x, N_y, and N_z to world-space is as follows:

$$M^T_{local \rightarrow world} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \tag{4.14}$$

The implementation of the transpose matrix and the normal vector conversion from local to world-space in CG is:

```
1  // Local to World transpose matrix
2  float3x3 local2WorldTranspose =
3              float3x3(i.tangentWorld, i.binormalWorld, i.normalDir);
4
5  // The normal vector direction converted from local to world space
6  float3 N = normalize(mul(localCoords, local2WorldTranspose));
```

Adding a normal map to the existing metal shader increases the level of realism significantly without adding more geometry to the scene. Through the use of normal maps, damage such as scratches and dents could be simulated on the object's surface (*See Figure 36*).

In real world materials the reflectivity is not usually uniformly spread across a surface. A mirror would produce a perfect reflection spread across the entire surface uniformly but for most materials that is not the case. The reflectivity of a surface usually decreases over time under the influence of multiple factors such as dirt and dust collection, mechanical damage, or chemical changes in the material structure.

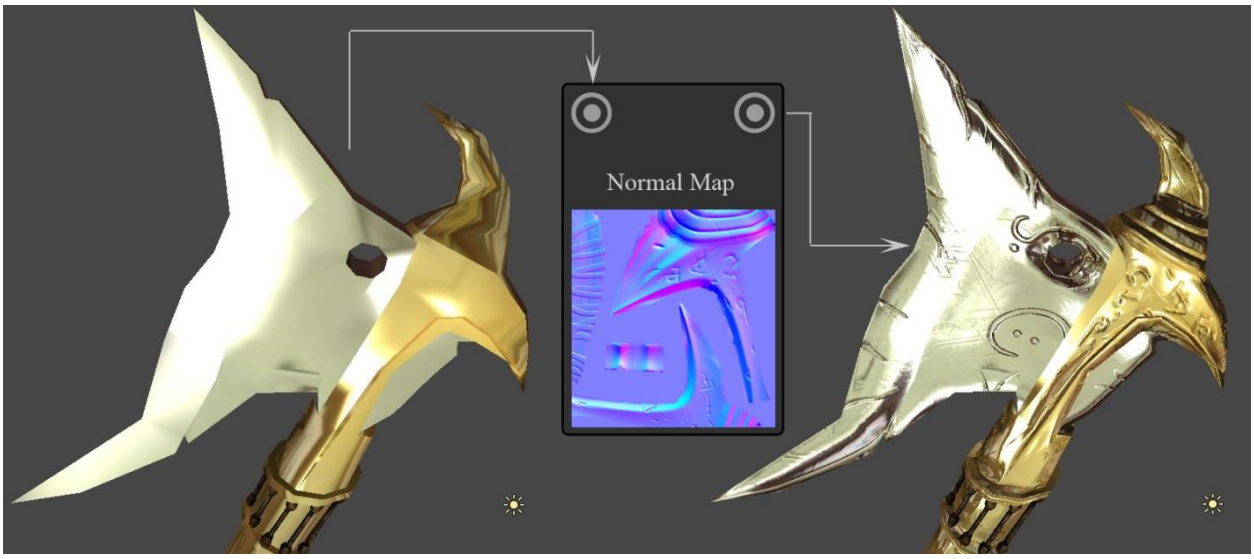

Figure 36: Normal direction vectors gathered from a normal map and used on the place of the original surface normals gives an illusion of a bumpy and irregular surface. The method has been used to add small details to an object without having to increase the object's surface geometry. In this particular case the method has been used to add small ornamental details and damage to the surface in the form of scratches and dents.

In the case of a mechanical damage such as surface scratches or dents on a metallic surface, the reflection level would be decreased inside the scratched cavities. That is due to the removal of material which results in a rougher surface with lowered reflectivity compared to a part of the surface that is undamaged. Additionally, depending on the size of a scratch, the cavity gets gradually filled with dirt which results in less and less reflectivity over a period of time. Eventually, the final appearance of the metallic surface varies in reflectivity level – some parts are more reflective than others.

That effect is simulated in real-time through the use of a specular map. A specular map is a two-dimensional texture of greyscale colors. The different shades of grey represent a value between 0 and 1, where 0 is pitch black and 1 is pure white. That value is then multiplied by the output of the cube map reflection. That way, based on the specular map some parts of the surface would be more or less reflective than others.

```
1  // Performing a texture lookup to get the specular texture values
2  float3 specMap = tex2D(_SpecMap, i.uv);
3
4  // Calculating the reflect ray direction used for the cube map lookup
5  // and the reflectivity level adjustment based on the specular map
6  float3 cubeRefl = texCUBE(_Cube, reflect(-V, N)) * specMap;
```

The texture is usually created by hand with the use of the normal map as guidelines. That way all of the scratches and dents from the normal map would get decreased reflectivity as well as some smaller scratches added only through the specular map (*See Figure 37*).

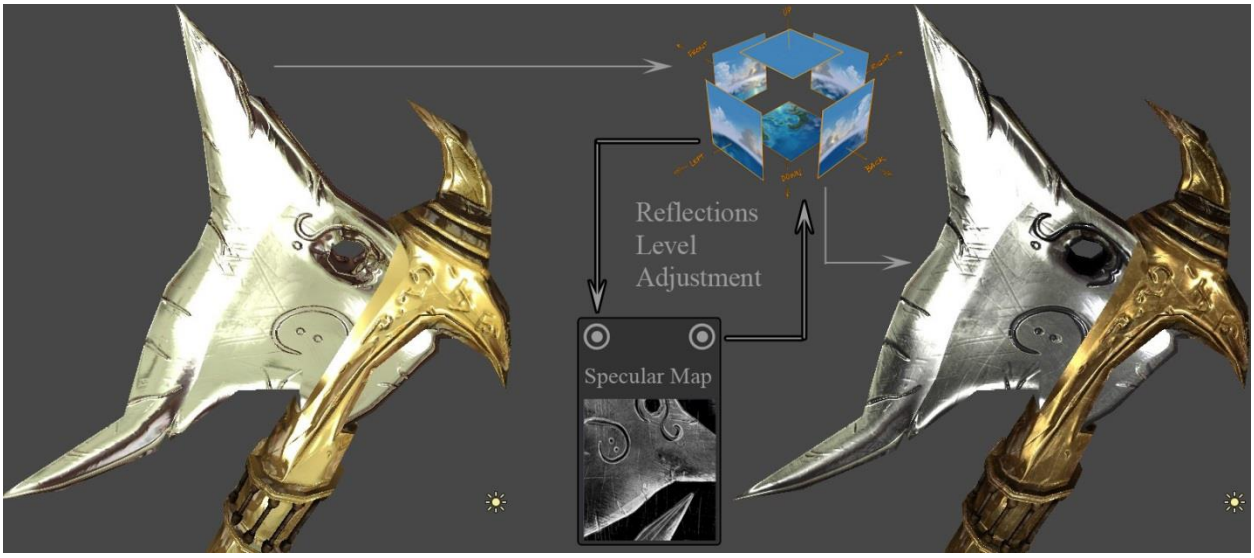

Figure 37: To achieve a higher level of realism the surface reflections should be decreased in the scratched cavities and dents. That effect is achieved by a greyscale 2D texture called a specular map.

Following the same model, the specular highlights also need to have different level of intensity. The dents and scratches should have a lower intensity for specular highlights compared to the areas on the surface that are flat and undamaged. The same method as the reflection level control has been used – a specular map. The values from the specular map are multiplied by the specular component to control the highlights intensity of the surface.

```
1   // Calculating the final specular component contribution
2 specularComponent =
3 float3(_SpecColor) * ((F*G*D) / 3.1415f * (VdotN * LdotN + 1.0e-7 )) * specMap;
```

By simply using a normal map to give the illusion of surface imperfections such as dents and scratches, and decrease specular intensity and reflections in the area of the cavities, the realism in the rendered image increases significantly. Figure 38 shows the completed 3D model of the battle axe rendered in real-time with Unity3D with a damaged metal shader applied to the surface.

Figure 38: A real-time render of a low-poly model of a battle axe. The metal material applied to the mesh uses Cook-Torrance reflectance model with normal and specular maps to simulate a scratched surface.

4.3 Weathered Metallic Surfaces

To simulate a metallic surface with rusty spots over it in real-time rendering, two materials need to be mixed together: a metal and a rust material. For the metal part the shader developed thus far would be used. The metal shader uses the Cook-Torrance light reflectance model for the specular part and an environment cube map with Fresnel reflectance for its diffuse component. The Cook-Torrance model allows for simulating shinier surfaces or rougher surfaces based on a roughness parameter controlled by the user as explained in the previous section.

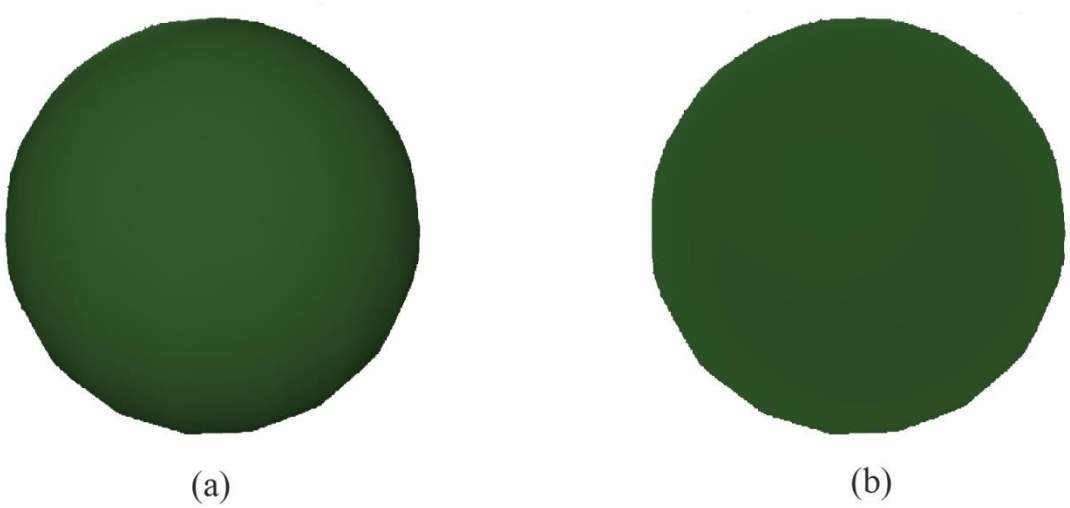

Figure 39: (a) A 3D sphere with Lambertian reflectance model applied to it. The edges of the sphere appear darker than its middle part. (b) The same sphere with Oren-Nayar reflectance model applied to its surface. The model uses inter-reflection technique to simulate sort of indirect illumination resulting in a more evenly lit surface. The model is used primarily to simulate rough surfaces.

The rusty part of the surface cannot be simulated through the use of the metal shader described above. Since the rusty surface is very rough, it has a very low amount of specular highlights. For the sake of optimization in the real-time rendering process, the specular component for the rust material would be completely ignored. The diffuse component could be simulated with the Lambertian model that threats a surface as a perfect diffuser, meaning that the brightness of a surface would appear the same regardless of the viewer's position. The problem with using that model is that the closer to perpendicular a surface normal in relation to the light source, the darker it would

appear. For instance, if we take a sphere and apply the Lambertian model to it, the edges of the sphere would appear darker compared to its middle part (*See Figure 39.a*). To simulate a rough surface more accurately the light interaction (scattering, reflectance) with the tiny imperfections of the rough surface should be taken into account. Michael Oren and Shree Nayar developed a reflectance model based on a micro-facet approach, which was meant for simulation of rough surfaces [GLRM] (*See Figure 39.b*). The model assumes that each microfacet is a Lambertian reflector. Unlike the Cook-Torrance model where the microfacets use distribution functions and account for self-shadowing and self-masking, the Oren-Nayar model focuses on the concept of inter-reflection. If the model does not take inter-reflection into consideration when a micro-facet that does not get direct illumination is viewed by the observer, it would appear totally dark (*See Figure 40.a*). In reality those unlit areas of the surface would receive some light reflected from nearby directly illuminated areas of the surface. In the micro-facet approach taken in the development of the Oren-Nayar reflectance model, unlit facets would receive some amount of reflected light from nearby lit micro-facets (*See Figure 40.b*) [GLRM] [PVGPS].

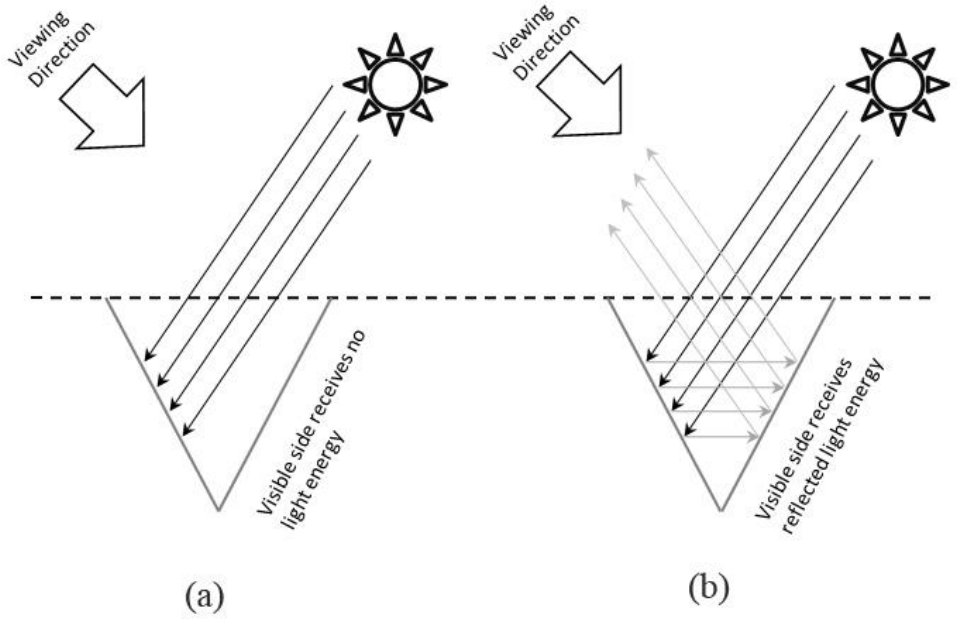

Figure 40: (a) No inter-reflection used in a micro-facet based reflectance model. The unlit facet would appear very dark to the observer. (b) The same reflectance model but taking into consideration inter-reflections. That method allows for not directly lit facets to still receive some amount of reflected light [PVGPS].

The Oren-Nayar model uses a lot of spherical coordinates in its original equations which have to be converted to Cartesian coordinate system in order to be implemented in a shader program running on the GPU. To compute the diffuse component for the Oren-Nayar reflectance model, the following equation would be used [PVGPS]:

$$Diffuse\ Component = Kd(N \cdot L)Ld * (C_1[\sigma] + \chi + \epsilon) \tag{4.15}$$

where K_d is the diffuse reflectance coefficient, L_d is the light source intensity, N is the surface normal direction, and L is the direction towards the light source. The letter σ is a user-defined roughness coefficient of values between 0 and 1. The evaluations for C_1, χ, and ϵ are as follows [PVGPS]:

$$C_1 = 1 - 0.5 * \frac{\sigma^2}{\sigma^2 + 0.33} \tag{4.16}$$

$$\chi = \gamma * C_2[\alpha, \beta, \gamma, \sigma] * \tan\beta \tag{4.17}$$

$$\epsilon = (1 - |\gamma|) * C_3[\alpha, \beta, \sigma] * \tan\left(\frac{\alpha + \beta}{2}\right) \tag{4.18}$$

where α, β, and γ are required components for the computation of C_2 and C_3. The calculation of all those components is as follows:

$$\alpha = \max[\cos^{-1}(V \cdot N), \cos^{-1}(L \cdot N)] \tag{4.19}$$

$$\beta = \min[\cos^{-1}(V \cdot N), \cos^{-1}(L \cdot N)] \tag{4.20}$$

$$\gamma = |V - N * (V \cdot N)| \cdot |L - N * (L \cdot N)| \tag{4.21}$$

where V is the viewing direction or in other words the direction towards the camera, N is the surface normal direction, and L is the direction towards the light source.

$$C_2 = \begin{cases} 0.45 * \frac{\sigma^2}{\sigma^2 + 0.09} * \sin\alpha & if\ \gamma \geq 0 \\ 0.45 * \frac{\sigma^2}{\sigma^2 + 0.09} * \left(\sin\alpha - \left(\frac{2\beta}{\pi}\right)^3\right) & if\ \gamma < 0 \end{cases} \tag{4.22}$$

$$C_3 = \frac{1}{8} * \left(\frac{\sigma^2}{\sigma^2 + 0.09}\right) * \left(\frac{4\alpha\beta}{\pi^2}\right)^2 \quad (4.23)$$

To implement the Oren-Nayar equations into Unity using the CG shading language the surface normal direction (N), the direction towards the camera (V), and the direction towards the light source (L) must be present. Having those direction vectors is the basic requirement for computing the Oren-Nayar reflectance model. The implementation of the model itself is as follows:

```
1   // Calculating the diffuse component using the Oren-Nayar lighting model
2
3   float alpha = max(acos(dot(V, N)), acos(dot(L, N)));
4   float beta = min(acos(dot(V, N)), acos(dot(L, N)));
5   float gamma = dot((V - N * dot(V, N)), (L - N * dot(L, N)));
6
7   float C1 = 1.0f - 0.5f * (pow(_Q, 2.0f) / (pow(_Q, 2.0f) + 0.33f));
8   float C2 = 0.45f * (pow(_Q, 2.0f) / (pow(_Q, 2.0f) + 0.09f));
9
10  if(gamma >= 0.0f)
11  {
12      C2 *= sin(alpha);
13  }
14  else
15  {
16      C2 *= (sin(alpha) - pow((2.0f * beta) / 3.14159f, 3.0f));
17  }
18
19  float C3 = (1.0f / 8.0f);
20  C3 *= pow(_Q, 2.0f) / (pow(_Q, 2.0f) + 0.09f);
21  C3 *= pow((4.0f * alpha * beta) / pow(3.14159f, 2.0f), 2.0f);
22
23  float A1 = gamma * C2 * tan(beta);
24  float B1 = (1.0f - abs(gamma)) * C3 * tan((alpha + beta) / 2.0f);
25
26  float3 diffuseComponent =
27      float3(_Color) * float3(_LightColor0) * max(0.0f, dot(N, L)) * (C1 + A1 + B1);
```

where Pi is approximated to 3.14159, and _Q is the user-defined variable that controls the amount of roughness on the surface.

Computing the diffuse component with the Oren-Nayar lighting model surely simulate a rough look to a surface. The method developed above uses a single color for its diffuse component, pre-defined by the user, for the entire surface. In the case of rust, there

should be a variety of colors used to simulate the effect of a rusty surface. To achieve that in a procedural way, a Perlin noise function has been used to create a noise texture map. That greyscale noise texture is then used as an input to a gradient color ramp, where each shade of gray represents a certain color. The Perlin noise function has been developed on the CPU instead of the GPU for the sake of optimization. Being on the CPU gives the advantage of executing the method only a single time when the game simulation is started. The Perlin noise function itself has been taken directly from the Ken Perlin's implementation of the method. [MN] The method takes some user-controllable variables needed for the generation of the noise map such as amplitude, frequency, the resolution of the map being generated, and octaves, which is the number of iterations for generating a more cloud like noise effect (*See Figure 41*).

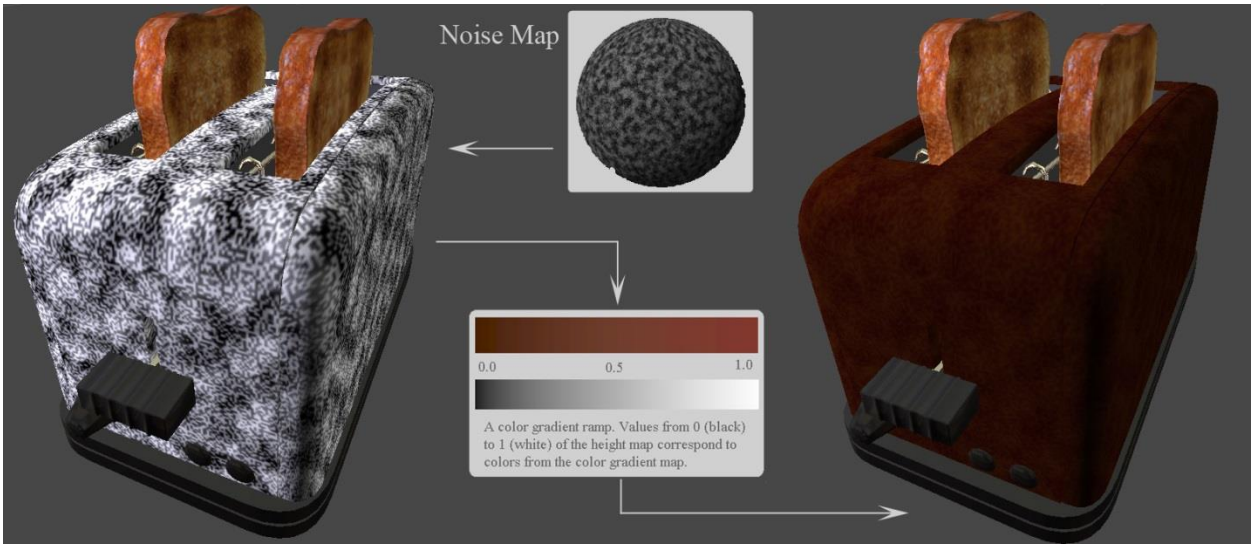

Figure 41: A procedural Perlin noise map applied to a model of a toaster. A method for mapping the greyscale values to a gradient color ramp has been used to create the color for the rust material.

Once the noise texture map is generated, it needs to be used as an input to a gradient color mapping method. That method is taking place in the GPU shader fragment program. The functionality of the gradient color map technique simply has a linear gradient between two pre-defined colors. That gradient is stored in a variable and ranges between 0 and 1, where 0 is the first color and 1 is the second color. Anything in between 0 and 1 is a mixture of the two colors. Taking the value of the greyscale noise map which is also between 0 and 1 and map it to the gradient color ramp would result in a color simulation of a rusty surface (*See Figure 41*). The gradient color ramp method is implemented as follows:

```
// A method for mapping a greyscale noise map to a color ramp

float3 colorRamp(float3 _Color1, float3 _Color2, float3 noiseMap)
{
    // Linear interpolation between color1 and color2
    return (1.0f - noiseMap) * _Color1 + _Color2 * noiseMap;
}
```

where *_Color1* and *_Color2* are pre-defined colors by the user, and *noiseMap* is the procedurally generated Perlin noise texture map.

To create rusty spots on a metallic surface, a masking technique could be used. The idea is to have two materials, a metal and a rust material, and a black and white texture used as a mask. The area of the mask that is white in color renders the metal material and the black color renders the rust material. The texture mask is procedurally generated using another instance of the Perlin noise function as described above. The generated noise map has only black and white colors, values of only zeros and ones, which would serve the masking method very well (*See Figure 42*). The masking method implemented in Unity3D game engine using the CG shading language is:

```
// A masking method (returns either a metal material or a rust material)
float3 maskDiffuse(float3 metalDiffuse, float3 rustDiffuse, float3 mask)
{
    if(mask.x <= 0.5f)
    {
        return metalDiffuse;
    }
    else
    {
        return rustDiffuse;
    }
}
```

where the *metalDiffuse* is the diffuse component of the metal material, *rustDiffuse* is the Oren-Nayar diffuse component of the rust material, and *mask* is the Perlin noise texture used as a blending mask.

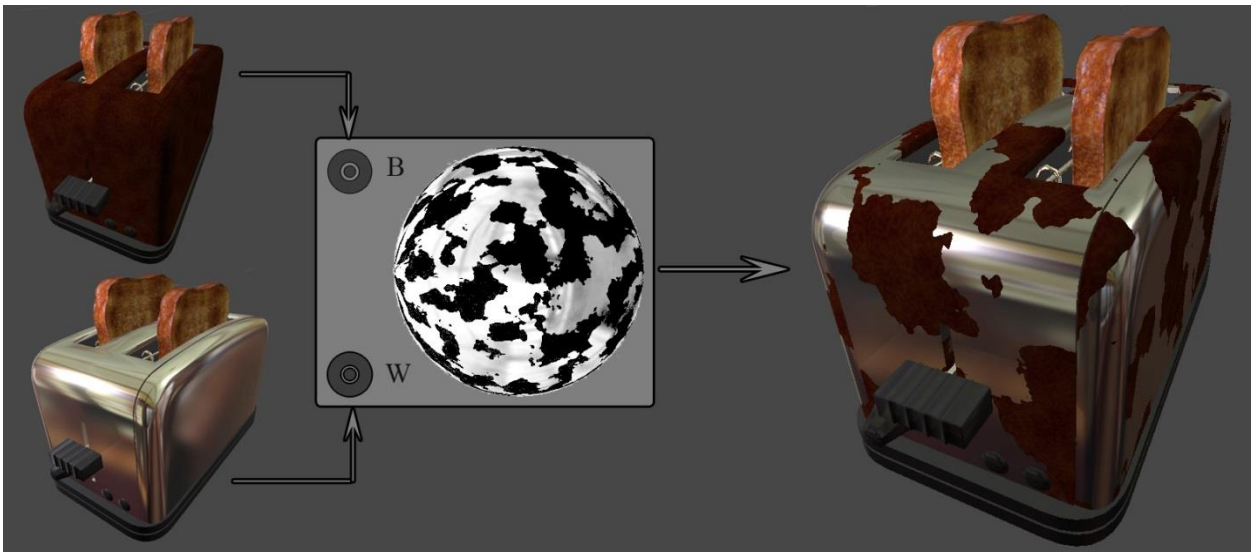

Figure 42: Blending two materials (metal and rust) with a procedurally created noise mask.

To bring the level of detail and realism a little higher a normal map should be used to simulate the bumpiness of the rusty material. The normal map should be generated from the already existing Perlin noise texture maps mentioned above. The procedurally created noise textures used in the diffuse component and for the creation of the material blending mask could be combined together into a single texture (height map). The height map is used to compute the perturbed surface normal and then use it during lighting calculations. That technique produces less accurate results compared to an actual normal map but still satisfactory. Figure 43 shows a 3D model of a toaster rendered in real-time with Unity3D, demonstrating a metal surface with rusty spots over it.

The technique explained above for simulating rusty surfaces in real-time could be applied also to simulate metallic patinas. The patinas would also have to use the Oren-Nayar reflectance model as well as color gradient ramp fed by a Perlin noise texture map.

Figure 43: A real-time render of a model of a toaster. The metal material uses the Cook-Torrance lighting model and the rust material – Oren-Nayar lighting model. The two materials are blended together with a procedurally generated masking texture.

Chapter 5

Conclusion

This thesis focused on simulation of metal materials and weathered or worn metallic surfaces in a computer generated environment. The thesis work addresses two distinctive approaches for accomplishing the main goals. The first one takes on visualization of metal materials using an offline rendering solution. The second method still targets the same goals but using a real-time rendering approach.

The summary of the offline rendering method is described in section 5.1, and a summary of the real-time rendering solution – at section 5.2. Since both methods are targeting the same results, section 5.3 focuses on a comparison between the two approaches, stating advantages and disadvantages from both methods.

5.1 Summary of the Offline Rendering Solution for Visualization of Metal Materials

The first method for computer generated visualization of metal materials and weathered or worn metallic surfaces uses a ray tracing rendering solution. A three-dimensional scene and objects have been created within Autodesk's 3DS Max environment and rendered using NVidia's MentalRay render engine.

The first part of the chapter focuses on the research and development of plain metal surfaces. The accuracy of the simulation of those surfaces is directly dependent on the surface reflectance model. Some previously developed reflectance models related to the topic have been described briefly. The chapter flow then moves towards surface reflections and Fresnel reflectance term. The surface reflections used in this chapter are ray-traced reflections which results in very accurate and detailed reflections. A section of this chapter addresses the difference between the isotropic and anisotropic metal surfaces and how can both be simulated in a 3D environment.

Once the metal material has been simulated, the next part would be the simulation of damage and weathering or aging effects on a metallic object's surface. The thesis focuses on surface damage in the forms of scratches. Those scratches have been simulated through the use of normal maps and modified version of the metal reflectance model. There has been introduced a technique for simulating accumulation of dirt in the scratch's cavities.

The last part of the offline rendering chapter addresses weathering and aging effects such as the formation of iron oxide or also known as red rust on metal surfaces and the formation and development of metallic patinas forming on copper surfaces.

5.2 Summary of the Real-time Rendering Solution for Visualization of Metal Materials

The second method for simulating computer generated metal materials and weathered or worn metallic surfaces addressed in this thesis, focuses on the use of a real-time rendering solution. The real-time render engine or game engine used for the purpose of this demonstration is Unity3D game engine.

The main goals of this chapter are still the same as the offline rendering approach. The different part is the path taken to get there. The real-time rendering approach uses mainly the graphical processor unit (GPU) instead of the central processor unit (CPU) as the ray-tracing method does. The implementation of the metal surface reflectance model and the weathering and aging effects have all been simulated using the same techniques as described in the offline method. The implementation of the above techniques has been done through small programs that run on the GPU called shaders.

5.3 Summary of the Results

Even though the two methods described in this thesis are aiming for the same goals and use almost the same techniques, the results gathered from both methods are quite different from each other. A ray traced rendered image usually has better quality and details compared to an image rendered in real-time. The reason for that is the ray-tracing rendering has the advantage of taking the time necessary to compute with greater precision all the calculations needed (light reflectance models, surface reflections, etc.). Since time is of no concern a single image rendered through ray-traced renderer could take from seconds to days to complete.

In the case of real-time rendering the rendered images (frames) have to be completed in the matter of fractions of a second. From 30 to 100 or more rendered images should be done in a single second. Having that requirement, the computations needed have to be simplified and optimized to maintain that speed. Since all the calculations are simplified as much as possible the results are also not as precise as the ones produced by the ray-traced renderer.

Figure 44 and 45 show rendered images of a 3D model of a rusty mechanical wrench. Using the same view direction and lighting, the 3D model has been rendered in ray-tracing renderer (MentalRay) and in a real-time renderer (Unity3D). The purpose of those rendered images is to compare the results gathered from both methods. The ray-traced results are providing better quality compared to the real-time results but the time taken to produce a single shot is much more than the time taken by the real-time approach. The two methods could be used for different types of applications. The ray-traced method could be used in producing photorealistic single shot renders or for computer animation. The real-time rendering method could be used for 3D interactive applications such as computer games or architectural walkthroughs.

Figure 44: (Top) An image rendered with ray-tracing renderer (MentalRay). (Down) Same object rendered using a real-time rendering engine (Unity3D)

Figure 45: (Top) An image rendered with ray-tracing renderer (MentalRay). (Down) Same object rendered using a real-time rendering engine (Unity3D)

References

[IMPBR] Schlick, Christophe. "An Inexpensive BRDF Model for Physically-based Rendering.” *Computer Graphics Forum* 13.3 (1994): 233-46. Print.

[ASBRCG] Rusinkiewicz, Szymon. "A Survey of BRDF Representation for Computer Graphics." Princeton University, Winter 1997. Web. 02 June 2013. <http://www.cs.princeton.edu/~smr/cs348c-97/surveypaper.html>.

[AIBL] Wynn, Chris. "An Introduction to BRDF-Based Lighting." *NVIDIA Corporation.* Print.

[PRRSC] Kang, Young-Min, Hwan-Gue Cho, and Sung-Soo Kim. "Plausible and Real-time Rendering of Scratched Metal by Deforming MDF of Normal Mapped Anisotropic Surface." *Journal of WSCG* 101-10. Print.

[PVGPS] Engel, Wolfgang. "Lighting." *Programming Vertex, Geometry and Pixel Shaders.* 2nd ed. Delmar, 2008. 251-264. Print.

[AFGBM] Westin, Stephen H., Hongsong Li, and Kenneth E. Torrance. "A Field Guide to BRDF Models." *SIGGRAPH '04* (2004): Print.

[ICGGL] Angel, Edward, and Dave Shreiner. "Lighting and Shading." *Interactive Computer Graphics: A Top-down Approach with Shader-based OpenGL*. 6th ed. Boston: Addison-Wesley, 2012. 257 -302. Print.

[GSTP] Bailey, Michael, and Steve Cunningham. "Surface Textures in the Fragment Shader. "*Graphics Shaders: Theory and Practice*. 2nd ed. Wellesley, MA: K Peters, 2009. 179-212. Print.

[IRIS] Becket, Welton, and Norman I. Badler. "Imperfection for Realistic Image Synthesis." *The Journal of Visualization and Computer Animation* 1.1 (1990): 26-32. Print.

[SSMMR] Merillou, S., J. Dischler, and D. Ghazanfarpour. "Surface Scratches: Measuring, Modeling and Rendering." *The Visual Computer* 17.1 (2001): 30-45. Print.

[AOMIIS] Buchanan, John W., and Paul Lalonde. "An Observational Model for Illuminating Isolated Scratches." *Electronic Arts Research* Print.

[ICGP] Phong, Bui T. "Illumination for Computer Generated Pictures." *Graphics and Image Processing* 18.6 (1975): 311-17. Print.

[ARMCG] Cook, Robert L., and Kenneth E. Torrance. "A Reflectance Model for Computer Graphics." *ACM SIGGRAPH Computer Graphics* 15.3 (1981): 307-16. Print.

[CGT] Fernando, Randima, and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics*. Boston, MA: Addison-Wesley, 2003. Print.

[MLRCSP] Blinn, James F. "Models of Light Reflection for Computer Synthesized Pictures." *ACM SIGGRAPH Computer Graphics* 11.2 (1977): 192-98. Print.

[APBMRRS] Bosch, Carles, Xavier Pueyo, Stephane Merillou, and Djamchid Ghazanfarpour. "A Physically-Based Model for Rendering Realistic Scratches." *Computer Graphics Forum* 23.3 (2004): 361-70. Print.

[RRRT] De Greve, Bram. "Reflections and Refractions in Ray Tracing." *Stanford University*(2006): Print.

[TORFRS] Torrance, K. E., and E. M. Sparrow. "Theory for Off-Specular Reflection From Roughened Surfaces." *Journal of the Optical Society of America* 57.9 (1967): 1105. Print.

[APMSLT] Jensen, Henrik W., Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. "A Practical Model for Subsurface Light Transport." *SIGGRAPH '01* (2001): 511-18. Print.

[MMAR] Ward, Gregory J. "Measuring and Modeling Anisotropic Reflection." *ACM SIGGRAPH Computer Graphics* 26.2 (1992): 265-72. Print.

[SAWPCG] S. Merillou and D. Ghazanfarpour. "A Survey of Aging and Weathering Phenomena in Computer Graphics." *Computers & Graphics* 32.2 (2008): 159-74. Print.

[LRFSCDS] Blinn James F. "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces." *ACM SIGGRAPH Computer Graphics* 16.3 (1982): 21-29. Print.

[GSTPDT] Bailey, Michael, and Steve Cunningham. "Discrete Techniques. "*Graphics Shaders: Theory and Practice*. 2nd ed. Wellesley, MA: K Peters, 2009. 357-424. Print.

[AAWS] C. Mansuy. “Aging architecture: Weathering simulation.” TU Eindhoven DDSS laboratory.

[MRWS] Dorsey Julie, Alan Edelman, Henrik W. Jensen, Justin Legakis, and Hans K. Pedersen. "Modeling and Rendering of Weathered Stone." *ACM SIGGRAPH Computer Graphics* 4 (2006): 225-234. Print.

[MRMP] Dorsey Julie and Pat Hanrahany. "Modeling and Rendering of Metallic Patinas." *ACM SIGGRAPH Computer Graphics* 2 (2006): 387-96. Print.

[CSR] Merillou Stephane, Jean-Michel Dischler, and Djamchid Ghazanfarpour. "Corrosion: Simulating and Rendering." *GRIN'01* (2001): 167-74. Print.

[DDRM] Matusik, Wojciech, Hanspeter Pfister, Matt Brand, and Leonard McMillan. "A Data-driven Reflectance Model." *ACM Transactions on Graphics* 22.3 (2003): 759. Print.

[CAT] Lu, Jianye, Athinodoros S. Georghiades, Andreas Glaser, Hongzhi Wu, Li-Yi Wei, Baining Guo, Julie Dorsey, and Holly Rushmeier. "Context-aware Textures." *ACM Transactions on Graphics* 26.1 (2007): 3-Es. Print.

[TVB] Sun, Bo, Kalyan Sunkavalli, Ravi Ramamoorthi, Peter N. Belhumeur, and Shree K. Nayar. "Time-Varying BRDFs." *IEEE Transactions on Visualization and Computer Graphics* 13.3 (2007): 595-609. Print.

[IBMW] Xuey, Su, Jiaping Wang, Xin Tong, Qionghai Dai, and Baining Guo. "Image-based Material Weathering." *Computer Graphics Forum* 27.2 (2008): 617-26. Print.

[CUEIC] Ullmann, Fritz, Wolfgang Gerhartz, Y. Stephen. Yamamoto, F. Thomas. Campbell, Rudolf Pfefferkorn, James F. Rounsaville, and Fritz Ullmann. "Corrosion." *Ullmann's Encyclopedia of Industrial Chemistry*. Weinheim, Federal Republic of Germany: VCH, 2000. Print.

[MN] Perlin, Ken. "Making Noise." *Noise Machine*. Ken Perlin, 9 Dec. 1999. Web. 19 June 2013. <http://www.noisemachine.com/talk1/>.

[GLRM] Oren, Michael, and Shree K. Nayar. "Generalization of Lambert's Reflectance Model." *ACM SIGGRAPH Computer Graphics* (1994): 239-46. Print.

[LTPLT] Pharr, Matt, and Greg Humphreys. "Light Transport III: Pre-computed Light Transport." *Physically Based Rendering: From Theory to Implementation*. 2nd ed. Amsterdam: Morgan Kaufmann/Elsevier, 2010. 925-88. Print.

[ECP] Zumdahl, Steven S., and Donald J. DeCoste. "Electrochemistry." *Chemical Principles*. 7th ed. Cengage Learning, 2011. 497-501. Print.

[KMTDOP] Džimbeg-Malčić, Vesna, Željka Barbarić-Mikočević, and Katarina Itrić. "Kubelka-Munk theory in describing optical properties of paper. "*Tehnički Vjesnik* [Zagreb, Croatia] 1 Jan. 2011, 18th ed., sec. 1: 117-24. Print.

[PPFT] Brennan, Chris. "Per Pixel Fresnel Term." *ATI 3D Application Research Group*: Print.

[RMPLT] Pharr, Matt, and Greg Humphreys. "Reflection Models." *Physically Based Rendering: From Theory to Implementation*. 2nd ed. Amsterdam: Morgan Kaufmann/Elsevier, 2010. 423-74. Print.

[TOSRFRS] Torrance, K. E., and E. M. Sparrow. "Theory for Off-Specular Reflection From Roughened Surfaces." *Journal of the Optical Society of America* 57.9 (1967): 1105-114. Print.

[ALMFS] Weidlich, Andrea, and Alexander Wilkie. "Arbitrarily Layered Micro-Facet Surfaces." *GRAPHITE* (2007). Print.

[MMRRS] Walter, Bruce, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. "Microfacet Models for Refraction through Rough Surfaces." *Eurographics Symposium on Rendering* (2007). Print.