Realtime Marble Rendering

Zahra Ghavasieh
University of Calgary, Canada
zahra.ghavasieh@ucalgary.ca

ABSTRACT

Marble is a beautiful stone with a unique texture that cannot be represented by simple texture mapping and Blinn Phong Shading. This paper introduces a rendering method to capture this material and its characteristics in a real-time setting. During the course of two months, a combination of three techniques — namely, procedurally generated textures, subsurface scattering and reflection approximations — were implemented to produce the best and most realistic visualization of the marble stone.

KEYWORDS

Marble Rendering, Perlin Noise, Texture Generation, Subsurface scattering, Fresnel Schlick Approximation

1 INTRODUCTION

In real-time rendering, textures are commonly imported from images and mapped onto objects in a scene. Finally, a simple shading algorithm such as Phong[1] is applied to create volume. However, this is a very rough and generalized method which does not fully capture the characteristics of various materials, one being the marble stone. Marble is a beautiful stone used as a luxurious material for buildings and furniture. Furthermore, the texture is unique and easily recognizable. Because marble objects are directly cut out from the stone, the texture is continuous and does not contain any clippings which can be caused by using image mapping. Furthermore, for the texture to appear seamless, each object must contain its own customized texture image and mapping function. Furthermore, some marble stones contain a degree of subsurface scattering which is not captured by a basic shading algorithm.

In this project, three methods were explored and combined to create a better and more realistic marble rendering. First, a texture is procedurally generated by adding turbulence to a smoothed noise such as Perlin Noise[2]. Then, an approximation of Subsurface Scattering is applied to the object via depth map calculations[3]. Finally, to depict the shine of marble, the Schlick Fresnel Approximation is used as a reflectance algorithm.

This project[4] is an attempt to combine these three techniques to create a realistic rendering of marble in real-time. Furthermore, the project contains a series of customizable parameters that can be used to adjust and experiment for the ideal render.

2 RELATED WORK

The three main techniques used for this project include Perlin Noise Texture Generation, Subsurface Scattering, and Schlick Reflectance. Each of these methods are discussed and researched separately. This project references various of these resources and combines them to create the results discussed in this paper.

Other approaches to marble rendering either focus entirely on the texture generation aspect, such as Paul's Marble Demo[5], or are not applied in real-time[6]. It is worth noting that most efforts are concentrated on making generalized rendering algorithms for multiple materials[7] rather than for a specific one such as marble.

3 METHOD

This project was implemented using OpenGL[8], a popular graphics API, and makes use of the Dear ImGUI[9] Library to allow for easy parameter adjustments and experimentations. As seen in Figure 1, the general pipeline consists of first setting up the window and shaders, and generating the marble texture while loading in an object mesh. Once the program enters the render loop, it renders the object in a deferred fashion using three passes:

1. The Depth Pass simply creates a depth map originating from the light source as opposed to the camera.

2. The Geometry Pass calculates the vertex positions, normals, depths and colours and saves them to a gbuffer. During this step, the marble texture is coloured and mapped onto the object.

3. Finally, the Lighting Pass receives all the buffers generated in the past two steps and calculates the Subsurface Scattering and Schlick Reflectance.

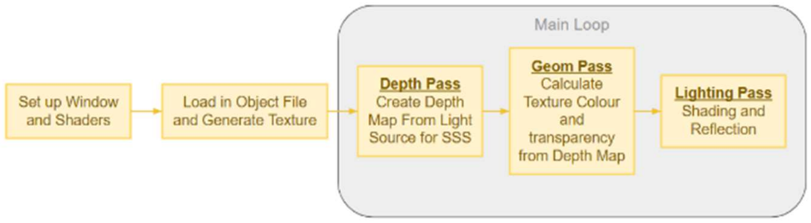
Main Loop
Set up Window and Shaders
Load in Object File and Generate Texture
Depth Pass
Create Depth Map From Light Source for SSS
Geom Pass
Calculate Texture Colour and transparency from Depth Map
Lighting Pass
Shading and Reflection

Figure 1: Program Pipeline

3.1 Texture Generation

Before discussing the process of texture generation, one must configure the problem of texture mapping. The texture generated in this project is a two-dimensional image, while the targeted object is three-dimensional. There are various approaches to this issue. In this project, the s and t components of the texture were simply mapped onto the x and z axes of the object, as this was the simplest method of demonstrating the generated marble texture. As seen in figure 2, this approach works best of objects elongated on the x and z axes as it stretches the texture along the y-axis. Furthermore, the texture is mirrored wherever it is mapped beyond its bounds. As this was not a major issue for the purposes of marble rendering, it was not addressed. However, for a more seamless mapping, one would need to consider the bounds of the objects as well as the generated image and map them accordingly.

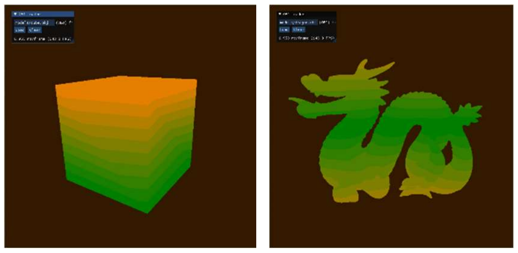

Figure 2: Texture Mapping by using the s and t values as red and green. Notice the stretch in the y-axis for the cube, and the mirroring for the dragon object.

The marble texture generation in this project is built upon Perlin Noise to create a more natural pattern. As explained in Lode's Texture Generation Tutorial[11], random noise is generated by using a function that randomly picks a float between 0 and 1 (black and white). By iterating this process through the s and t directions, we can produce a white noise image (Figure 3A). However, to create a natural looking texture, we must interpolate between each pixel to make a **smooth noise** such as the Perlin Noise. This method makes the image look almost blurry, as neighbouring pixels are now much closer in value. Next, different resolutions of this smooth noise are generated and combined with maximum values closer to 0 (darker) to create **turbulence**. This turbulence will have an appearance similar to smoke or fog as seen in Figure 3B. Finally, by combining this turbulence to a wave function such as sine or cosine (Figure 3C), a texture resembling marble patterns is formed (Figure 3D).

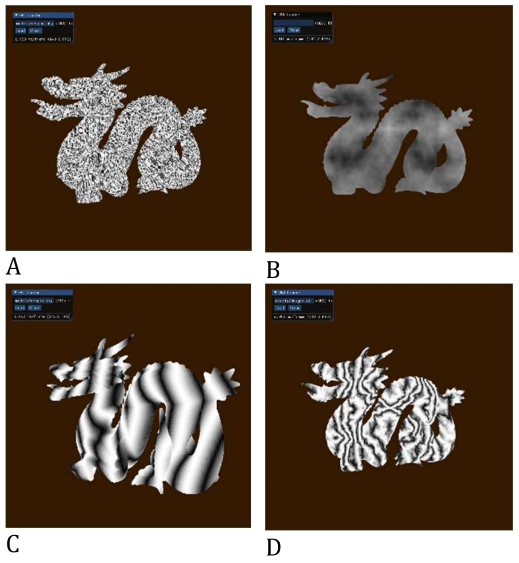

Figure 3: Process of Marble Texture Generation via Perlin Noise

By changing parameters such as the period of the wave or the turbulence power, as well as applying more realistic colours, we can get reasonable looking marble textures.

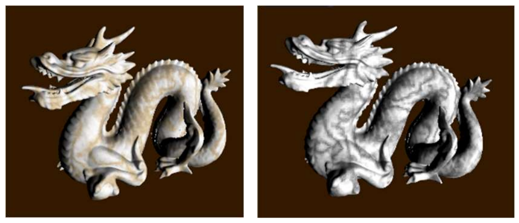

Figure 4: Coloured Marble Textures with Phong Shading

3.2 Subsurface Scattering

Subsurface Scattering is the entrance of a light ray at a point on the object and its exit at another point on that object. This phenomenon creates depth and substance to an object. Although marble is not as absorbent as a material like jade, it does scatter light to a certain degree. This project uses an approximation method described by Nvidia[12]. This process involves calculating the distance from the light source to the object and subtracting it from the distance of the view point to that object. By calculating each of these depth maps, we can grasp the thickness of the object and the degree of which light penetrates the object.

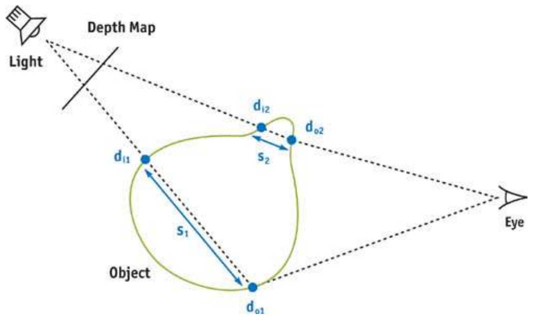

Figure 5: Calculating the Distance Light Has Travelled Through an Object Using a Depth Map[13]

This allows us then to dictate different colours for different depths and create the illusion of subsurface scattering. In this project, the depth of the object, s_i, is used as follows to apply the results shown in Figure 6.

$$sss_{valu} \quad = e^{s_i \cdot \sigma_t}$$

$$FragColor = vec4(sss_{value} \cdot diffuse, 1.0)$$

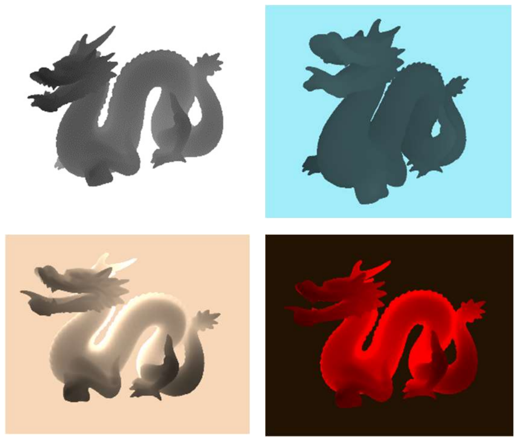

Figure 6: Subsurface Scattering Approximation. The top two images are the depth maps using different projection matrices. The bottom two images are the results of applying the above equation to different diffuse colours.

3.3 Fresnel Schlick Reflectance

The final technique used to make the marble rendering complete is a reflectance algorithm. As the Fresnel Reflectance is too computationally intensive for a real-time setting, the Schlick Approximation was applied instead. The algorithms described in LearnOpenGL[14] and Wikibooks[15] were used as reference to implement the following.

$$H = \frac{V + L}{|V + L|}$$

$$F_0 = vec3(0.04)$$

$$F_0 = F_0(1 - metalness) + diffuse \cdot metalness$$

$$schlick = F_0 + (1 - F_0) \cdot \left(1 - (H \cdot N)\right)^5$$

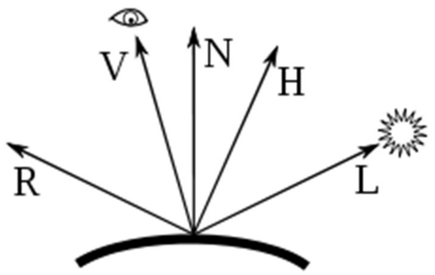

Figure 7: In addition to most of the vectors used by the Phong reflection model, we require the normalized halfway vector H, which is the direction exactly between the direction to the viewer V and the direction to the light source L. [16]

After calculating the Schlick Reflectance, its opacity is customizable via the *k_schlick* parameter on the ImGUI panel.

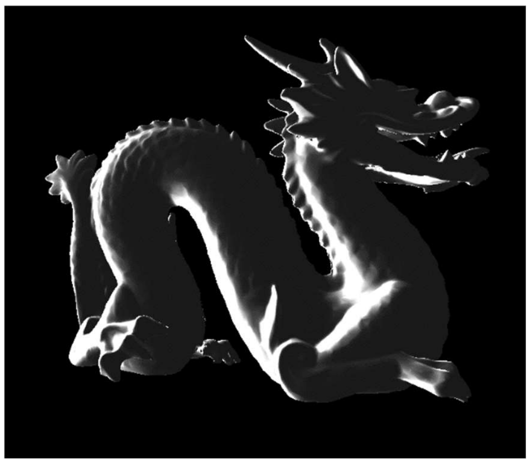

Figure 8: Fresnel Schlick Reflectance applied to a dark grey dragon

4 RESULTS

After calculating the three different techniques described above, they are then combined by various customizable degrees. First, the marble texture is generated before entering the render loop. It is then mapped onto the object in the Geometry pass and saved to the gbuffer as the diffuse colour. Next, in the Lighting pass, subsurface scattering is calculated and applied to the marble texture (diffuse). And finally, the Fresnel Schlick Reflectance is computed by using the results of the Subsurface scattering as its input colour. This order can be reversed via the ImGui panel for different results.

Comparing the results from Figure 10 to the basic shading of the same model shown in Figure 9, it is clear that there is a significance change in the likeness of the model to a marble object.

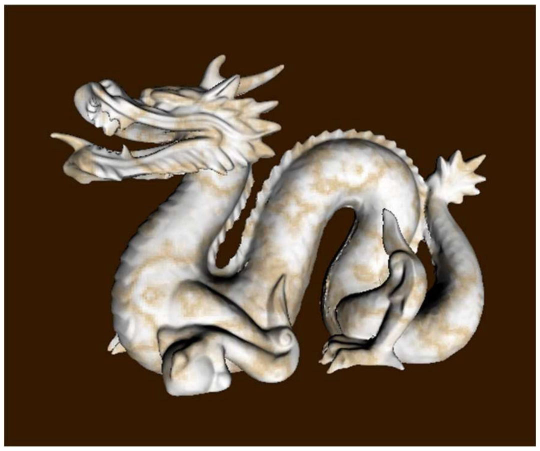

Figure 9: Generated Marble texture with Oren-Nayar Lighting Model

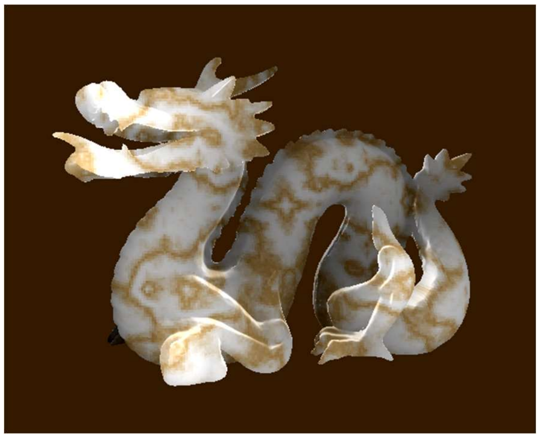

Figure 10: Marble Rendering combining the three techniques of texture generation, subsurface scattering, and Schlick approximation

However, this project has room for improvement. Parameters still need to be finetuned. Furthermore, the Schlick Fresnel Reflectance is perfect for highlighting the silhouette of the object but falls apart when moving the light source behind the object to showcase the subsurface scattering. A different reflectance model may be needed to better represent the property of the marble stone.

The executable can be found on the GitHub page[10].

REFERENCES

[1] *Basic lighting*. LearnOpenGL. https://learnopengl.com/Lighting/Basic-Lighting

[2] Moore, H. (n.d.). *Procedural Noise/Perlin Noise.* http://physbam.stanford.edu/cs448x/old/Procedural_Noise(2f)Perlin_Noise.html

[3] Green, S. (2004). *Chapter 16. real-time approximations to subsurface scattering*. NVIDIA Developer. https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-16-real-time-approximations-subsurface-scattering

[4] Ghavasieh, Z. (2022). *Marble Rendering*. marble-renderable. https://judgyknowitall.github.io/marble-renderable/

[5] http://www.paulsprojects.net/opengl/marble/marble.html

[6] https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model

[7] Rusinkiewicz S. (2014) Reflectance Models. In: Ikeuchi K. (eds) Computer Vision. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-31439-6_537

[8] OpenGL. https://learnopengl.com/Introduction

[9] Imgui. https://github.com/ocornut/imgui

[10] Marble Renderable GitHub. https://github.com/judgyknowitall/marble-renderable

[11] https://lodev.org/cgtutor/randomnoise.html

[12] https://developer.nvidia.com/gpugems/gpugems/part-iii-materials/chapter-16-real-time-approximations-subsurface-scattering

[13] https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/elementLinks/fig16-03.jpg

[14] https://learnopengl.com/PBR/Theory

[15] https://en.wikibooks.org/wiki/GLSL_Programming/Unity/Specular_Highlights_at_Silhouettes

[16] https://commons.wikimedia.org/wiki/File:Blinn_Vectors.svg