Project Report

Geometry-based shading for shape depiction enhancement

An Implemetation

Real-Time Graphics Programming Project

Author:
Francesco Brischetto

Student Number:
958022

Submitted to:
Dr. Davide Gadia

5th March, 2022

Table of Contents

Executive Summary

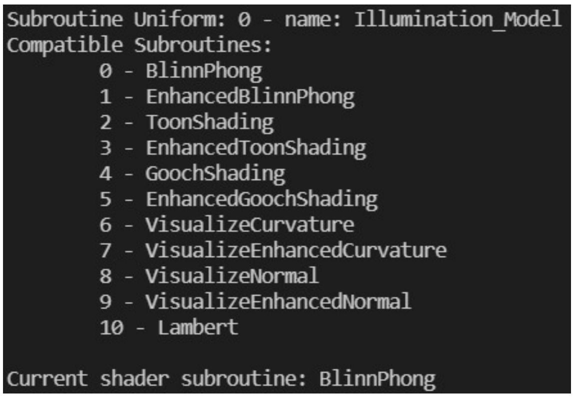
This project report describes the realization and implementation decisions of the shading technique proposed in [1].
The paper's goal is to describe an approach for **enhancing shape depiction** of 3D objects on **Non-Photorealistic Rendering (NPR) shading models** using *local geometry*.
The approach described provides **time-efficiency**, to make it usable in *interactive application*, without any constraints on the choice of material or illumination model.
The proposed method is inspired by **Normal Enhancement** and **Radiance Scaling** Shading approach and it tries to combine the advantages of both techniques while, also, relaxing their constraints.
The overall goal of my project is to implement one of the newest techniques in NPR as well as coding some of the most famouse ones.

1 Introduction

The advent of Computer Graphic technologies has **not replaced artists**. Many non-photorealistic rendering techniques have focused on depicting shape through shading that *mimic hand-drawn illustration* but NPR models still **lack in expressive power**.
The proposed technique should *correlate* the enhancement functionalities to **surface feature variations**. To achieve this result, the paper [1] starts from analyzing two existing techniques that either perturbates the surface normal as in **Normal enhancement** or alterates reflected radiance based on local surface information as in **Radiance scaling**.
However, both of those methods have some limitations:
Normal enhancement operators are *restricted* on specific types of material and illumination models, and they are **not able to enhance** some important geometry features such as **concavities** and **convexities**. In contrary, Radiance scaling overcomes those limitations but (expecially in NPR shading) tends to mask subtle shading variations and hence **reduce effectiveness** of the overall technique.
The proposed techinque's goal is to *reformulate* NPR shading models with respect to **geometry surface features** combining the advantages of Normal enhancement and Radiance scaling while also **relaxing their constraints**.
The enhancement is achieved in two ways. First, by *modifying the surface normal* using a simple high-frequency enhancement operation. Second, by correlating reflected lighting intensity to **surface curvature** using a new scaling function.
In this way, we can achieve enhacement in diffierent extent, allowing users to produce more desirable enhacement results.

1.1 Project Overview

In this project I implemented all the shading models used in [1] :
Blinn-Phong, **Cartoon** and **Gooch** as well as their **enhanced version**, following the reformulation of the *reflectance radiance equation* described in paragraphs 6.1, 6.2 and 6.3 of [1].
Contextually, I realized also all the needed **helping operators**, such as *normal smoothing*, *sharpening* and *curvature* calculation. Lastly, I realized other *subroutines* to better visualize the differences between normal and enhanced normal, as well as curvature and enhanced curvature (that uses enhanced normal).
A list of all the usable subroutine in the project can be seen in **Figure 1**.



```
Subroutine Uniform: 0 - name: Illumination_Model
Compatible Subroutines:
        0 - BlinnPhong
        1 - EnhancedBlinnPhong
        2 - ToonShading
        3 - EnhancedToonShading
        4 - GoochShading
        5 - EnhancedGoochShading
        6 - VisualizeCurvature
        7 - VisualizeEnhancedCurvature
        8 - VisualizeNormal
        9 - VisualizeEnhancedNormal
        10 - Lambert

Current shader subroutine: BlinnPhong
```

Figure 1: All subroutines available in the project

2 Geometry-based shading Approach

The key of this approach is *incorporating surface curvature* information into shape depiction enhancement technique. This is done with multiple steps:

1. **3D Shape Descriptor:** 3D Object surface shape is analyzed using the shape descriptor. **Curvature computation** extracts salient surface features. Tts contribution can be controlled using some parameters.

2. **Geometry-based Shading:** the surface normals are *smoothed* or *sharpened* to attenuate or exaggerate the surface depiction; Then, the **reflected lightning intensity** is scaled using also surface curvature previously calculated.

3. **Rendering:** This shading technique is applied to various **non-photorealistic rendering styles**, reformulating the reflected radiance equation in a way that takes the enhanced shading into account.

In **Figure 2** the rendering pipeline is graphically illustrated.

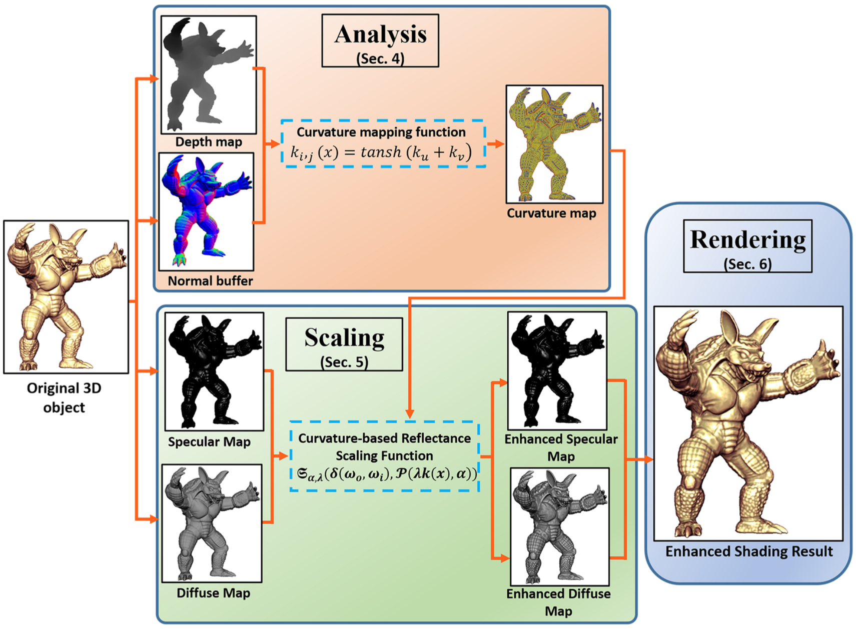


Figure 2: The rendering pipeline of the paper's approach. It combines estimating the surface curvature, scaling the reflected lighting and rendering the NPR result

3 Design and Implementation

In this chapter I will cover all the implementation choices that I made during the realization of the project. Some of the operations described in the paper [1] where not clearly specified so I was forced to take some implementation decisions and use the paper as a guideline. Moreover, some operations where too general and customizable so I tried to simplify it to make the project simpler.
Code-wise, My project started from the code version **lecture03a** that we saw during lectures.

3.1 Normal Enhancement Operators

In the paper [1] are described two types of normal variations: *smoothing* and *sharpening*. Those two kind of enhancement operators works on the high-frequency components of the surface normal. Smoothing reduces those components, while Sharpening increase them.

3.1.1 Normal Smoothing Operator Implementation

To calculate smoothed surface normal I started from this idea[1]. The implementation of normal smoothing operator was done in model_v1 file, while *processing the Assimp mesh* in order to obtain an *OpenGL mesh*. I created a **new position in the vertex attribute**, that stores smoothed normal.
In this way, data can be loaded in vertex shader by simply add the lines in **Code 1**.
The smoothing normal computation is done for each vertex. I **initialized** it to vector zero. Then, I **computed face normal** for each face of the mesh and I *added* it to each vertex of the face. Finally, for each vertex, I **normalized the result** of this sum.
By doing so, I implicitely consider the *kernel* of convolution as the *smallest one*, considering as **neighbourhood** only the faces that share the same vertex. I also implicitely defined the σ parameter, found in Equation 5 of 4.2.1 chapter of [1] that *weights* this operation as *1*. This greatly **simplifies** computation and code while maintaning the same *intent* as the original paper. The added lines of code are showed in **Code 2**.

```
layout (location = 2) in vec3 sm_normal;
```

Code 1: Smoothed normal loaded in vertex shader

```
for(GLuint i = 0; i < mesh->mNumVertices; i++)
{
  Vertex vertex;
  ...
  vertex.Sm_Normal = glm::vec3();
  ...
}
// For each face, I calculate the face normal and I add to the vertices' normal used by the face. Finally, I normalize the normal to obtain the smoothed surface normal.
for ( int i = 0; i < indices.size(); i += 3 )
{
  // I Compute the face normal using triangleNormal method, that computes normal starting from triangle points
  glm::vec3 faceNormal = glm::triangleNormal(
  vertices[indices[i]].Position,
  vertices[indices[i+1]].Position,
  vertices[indices[i+2]].Position);



```

[1] https://www.reddit.com/r/opengl/comments/6976lc/smoothing_function_for_normals/

```
    // I add face normal to each of the 3 vertex normal of the face
    for ( int j = 0; j < 3; j++ )
    {
      vertices[indices[i+j]].Sm_Normal += faceNormal;
    }
  }
  for (auto &v : vertices)
  {
    // Normalizing the vectors accumulating face normals to obtain the smoothed surface normal
    // NOTE: Sigma parameter, found in Equation 5 of 4.2.1 chapter of the reference paper ( to control the quantity of convolution kernel used ) is implicitely 1.
    v.Sm_Normal = glm::normalize(v.Sm_Normal);
  }
```

Code 2: Smoothed normal computations added in processMesh method

3.1.2 Normal Sharpening Operator Implementation

Normal Sharpening operator is implemented in **fragment shader** and uses the smoothed normal previously described and passed to the *vertex shader*.
In vertex shader, I apply *normal Matrix transformation* before passing them to the fragment shader as showed in **Code 3** .
In fragment shader, I calculate the **mask** by subtracting the original normal vector with its smoothed version. Then, I add the mask multiplied by a **scaling factor** λ to the normal vector, with a typical process called **unsharp masking** as described in Equation 6 of 4.2.2 chapter of [1]. Finally, I normalize the result.
The **Code 4** is added to the fragment shader to implement this process.

```
vSMNormal = normalize( normalMatrix * sm_normal );
```

Code 3: Transformation applied to smoothed normal in vertex shader

```
// Computing the mask for Unsharp Masking
vec3 mask = vNormal - vSMNormal;
// calculating enhanced Normal using the Unsharp Masking technique. This is defined, in the reference paper, in equation 6 of chapter 4.2.2
vec3 eNormal = vNormal + lambda * mask;
// normalization of the per-fragment enhanced normal
vec3 N_I = normalize(eNormal);
```

Code 4: Calculation of normal sharpening operator in fragment shader

A Comparison between normal vector and enhanced normal vector (through sharpening) is showed in **Figure 3** that shows two subroutines created in the project.

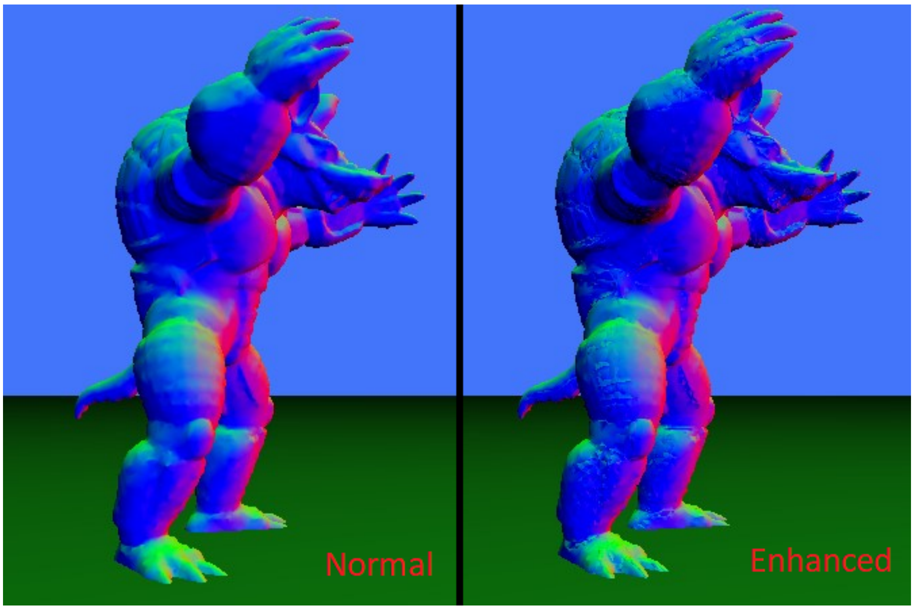


Figure 3: Comparison between normal and enhanced normal vector (through sharpening)

3.2 Curvature Analysis

Curvature analysis is another key point of the technique described in [1].
However, curvature calculation was not so clear and straightforward in the paper because they introduced the **second fundamental tensor** and the computation of **Hessian of depth field** by differentiating the gradient with a *Sobel filter*.
The approach implemented in my project, instead, is simpler. It starts from here[2] and considers **screen-space normals of neighbouring** fragments as well as **depth of the current fragment** to compute a curvature value.

3.2.1 Screen Space Surface Curvature Implementation

Surface curvature is computed in screen space, using OpenGL partial derivatives functions TODO.

3.3 Non-Photorealistic Rendering shading styles

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fringilla fringilla dui id vehicula. Vestibulum aliquet, dui non viverra cursus, odio felis laoreet justo, ultricies blandit massa elit id lacus. Donec est lorem, aliquet ac gravida ac, egestas in elit. Sed sed ultrices tellus. Sed placerat nibh vel leo elementum, at aliquet turpis auctor. Etiam volutpat est risus, sit amet vehicula justo molestie sit amet. Donec fermentum libero in risus lobortis pellentesque. Sed dapibus, leo in congue dapibus, risus risus congue dolor, ac convallis lectus magna placerat ligula. Nam at turpis nisi. In scelerisque, libero vel laoreet tristique, est nunc maximus ipsum, non laoreet orci lorem ut dui. Cras vel est at ex congue congue sit amet id lectus. Nullam in tortor elementum, tempus eros vel, sagittis diam. Curabitur ultrices est nec est rhoncus malesuada. Mauris rhoncus nulla in ligula congue rhoncus. Suspendisse aliquet pellentesque lacus, nec varius neque sagittis et.

[2] https://madebyevan.com/shaders/curvature/

3.3.1 Enhanced Blinn-Phong Shading implementation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fringilla fringilla dui id vehicula. Vestibulum aliquet, dui non viverra cursus, odio felis laoreet justo, ultricies blandit massa elit id lacus. Donec est lorem, aliquet ac gravida ac, egestas in elit. Sed sed ultrices tellus. Sed placerat nibh vel leo elementum, at aliquet turpis auctor. Etiam volutpat est risus, sit amet vehicula justo molestie sit amet. Donec fermentum libero in risus lobortis pellentesque. Sed dapibus, leo in congue dapibus, risus risus congue dolor, ac convallis lectus magna placerat ligula. Nam at turpis nisi. In scelerisque, libero vel laoreet tristique, est nunc maximus ipsum, non laoreet orci lorem ut dui. Cras vel est at ex congue congue sit amet id lectus. Nullam in tortor elementum, tempus eros vel, sagittis diam. Curabitur ultrices est nec est rhoncus malesuada. Mauris rhoncus nulla in ligula congue rhoncus. Suspendisse aliquet pellentesque lacus, nec varius neque sagittis et.

3.3.2 Enhanced Cartoon/Cel Shading implementation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fringilla fringilla dui id vehicula. Vestibulum aliquet, dui non viverra cursus, odio felis laoreet justo, ultricies blandit massa elit id lacus. Donec est lorem, aliquet ac gravida ac, egestas in elit. Sed sed ultrices tellus. Sed placerat nibh vel leo elementum, at aliquet turpis auctor. Etiam volutpat est risus, sit amet vehicula justo molestie sit amet. Donec fermentum libero in risus lobortis pellentesque. Sed dapibus, leo in congue dapibus, risus risus congue dolor, ac convallis lectus magna placerat ligula. Nam at turpis nisi. In scelerisque, libero vel laoreet tristique, est nunc maximus ipsum, non laoreet orci lorem ut dui. Cras vel est at ex congue congue sit amet id lectus. Nullam in tortor elementum, tempus eros vel, sagittis diam. Curabitur ultrices est nec est rhoncus malesuada. Mauris rhoncus nulla in ligula congue rhoncus. Suspendisse aliquet pellentesque lacus, nec varius neque sagittis et.

3.3.3 Enhanced Gooch Shading implementation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fringilla fringilla dui id vehicula. Vestibulum aliquet, dui non viverra cursus, odio felis laoreet justo, ultricies blandit massa elit id lacus. Donec est lorem, aliquet ac gravida ac, egestas in elit. Sed sed ultrices tellus. Sed placerat nibh vel leo elementum, at aliquet turpis auctor. Etiam volutpat est risus, sit amet vehicula justo molestie sit amet. Donec fermentum libero in risus lobortis pellentesque. Sed dapibus, leo in congue dapibus, risus risus congue dolor, ac convallis lectus magna placerat ligula. Nam at turpis nisi. In scelerisque, libero vel laoreet tristique, est nunc maximus ipsum, non laoreet orci lorem ut dui. Cras vel est at ex congue congue sit amet id lectus. Nullam in tortor elementum, tempus eros vel, sagittis diam. Curabitur ultrices est nec est rhoncus malesuada. Mauris rhoncus nulla in ligula congue rhoncus. Suspendisse aliquet pellentesque lacus, nec varius neque sagittis et.

4 Conclusions

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam fringilla fringilla dui id vehicula. Vestibulum aliquet, dui non viverra cursus, odio felis laoreet justo, ultricies blandit massa elit id lacus. Donec est lorem, aliquet ac gravida ac, egestas in elit. Sed sed ultrices tellus. Sed placerat nibh vel leo elementum, at aliquet turpis auctor. Etiam volutpat est risus, sit amet vehicula justo molestie sit amet. Donec fermentum libero in risus lobortis pellentesque. Sed dapibus, leo in congue dapibus, risus risus congue dolor, ac convallis lectus magna placerat ligula. Nam at turpis nisi. In scelerisque, libero vel laoreet tristique, est nunc maximus ipsum, non laoreet orci lorem ut dui. Cras vel est at ex congue

congue sit amet id lectus. Nullam in tortor elementum, tempus eros vel, sagittis diam. Curabitur ultrices est nec est rhoncus malesuada. Mauris rhoncus nulla in ligula congue rhoncus. Suspendisse aliquet pellentesque lacus, nec varius neque sagittis et. [**?**]

References

[1] Riyad Al-Rousan, Mohd Shahrizal Sunar, Hoshang Kolivand (2017) *Geometry-based shading for shape depiction enhancement*