

Mean Curvature Using Surface Shaders

By Jacob Thompson

Class of 2020

A thesis (or essay) submitted to the
faculty of Ripon College
in partial fulfillment of the requirements for the
Degree of Bachelor of Arts
in Mathematics

1 Abstract

In this paper, we present a portable tool for displaying the curvature of arbitrary meshes in real time using built-in tools within a recent build of the Unity3D engine (2019.2.8f1). When the developed shader is applied to a game object, the game object is procedurally colored to generate a heat map of the mean curvature at every point along its surface. The shader largely behaves as expected, although the transformation to and from screen-space and use of screen-space differentiation induces a minor loss of information during the texturing process. This results in minor but noticeable visual artifacts produced by our tool and no immediately obvious remedy.

2 Introduction

2.1 Curvature

Signed Curvature

Signed curvature at a point \vec{p} is the rate of change in the direction of the tangent line of a surface with respect to arc length at that location [6]. We consider the two-dimensional case in Figure 1 in which a unit-speed curve $\alpha(s)$ is parameterized by its arc length s ,

$$\kappa(\vec{p}) = \frac{d\phi}{ds}$$

where ϕ is the angle between the depicted tangential line and the x-axis.

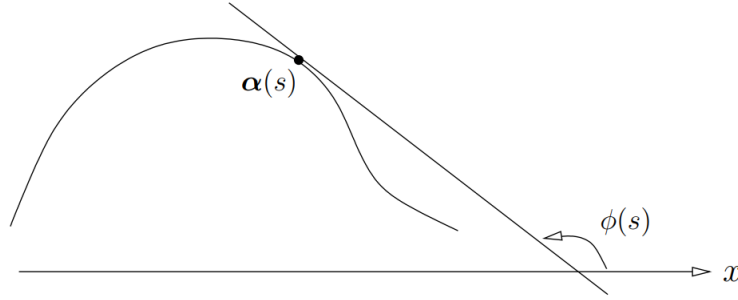


Figure 1: Anatomy of signed curvature. Adapted from [6]

Mean Curvature

Perhaps one of the more intuitive definitions of mean curvature at a point \vec{p} is as the average of signed curvatures κ at that location [5]. When 1 is brought into three dimensions, we find that the tangent line can be drawn from any direction within in a circle around the point in interest. As such, we can write

$$H(\vec{p}) = \frac{1}{2\pi} \int_0^{2\pi} \kappa(\vec{p}, \theta) d\theta.$$

Although this definition is not employed in the final shader, this description provides key insight used to evaluate the behavior of our final product. Under another derivation,

$$2H(\vec{p}) = \text{Tr}(S(\vec{p})),$$

where $\text{Tr}(\mathbf{A})$ denotes the matrix trace of \mathbf{A} , or the sum the elements along its main diagonal,

$$\text{Tr}(\mathbf{A}) = \sum_{i=1}^n a_{i,i}$$

and S is the shape operator, defined by the negative directional derivative:

$$S(\vec{p}) = -\nabla_{\vec{p}} \hat{n},$$

where \hat{n} is used to denote the unit vector normal to the surface at \vec{p} [7]. In \mathbb{R}^3 , this reduces to

$$H(\vec{p}) = -\frac{1}{2} \nabla \cdot \hat{n}.$$

In most standard mesh files, the direction normal to the surface of a model is provided at each vertex for the mesh.

2.2 UV Mapping

Within most modern rendering engines, texturing as achieved by+ applying a geometric transformation in which the x, y, z *object-space* coordinates of a model's polygons are mapped onto a two-dimensional set of coordinates in *texture-space* in which each valid u, v coordinate corresponds to a location on the given texture or material file [4]. This process is shown in Figure 2. The set of u, v values each polygon maps to determines which subsection of the given texture is rendered onto those surfaces in addition to any sort of texture distortions or repetitions made in the mapping process.

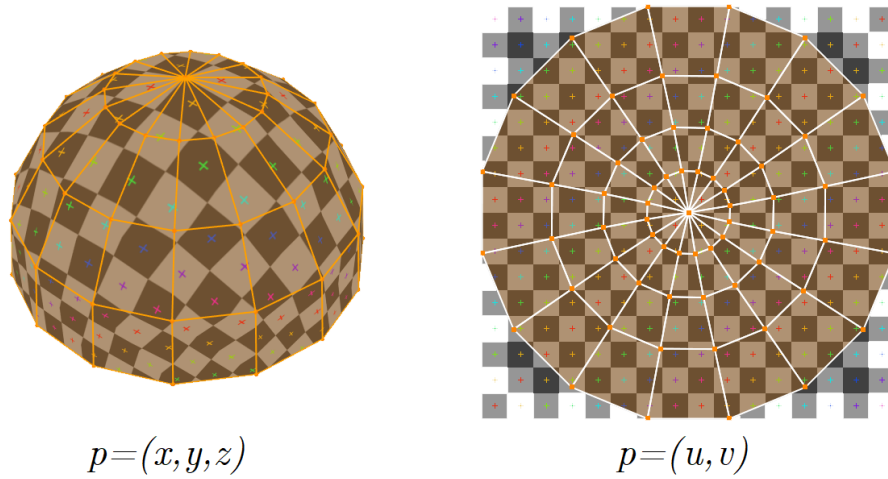


Figure 2: UV mapping. Adapted from [4]

2.3 Surface Shaders

Within the context of Unity, a surface shader describes the HLSL (High-Level Shader Language) *surface function* of an object, which takes the 3D model data of a mesh, modifies the texturing process, and then passes the result to the HLSL *SurfaceOutput* function [2, 8]. This function in turn describes several surface properties including the object's albedo color, bump map data, and specular. Unity's built-in shader graph tool provides an additional layer of abstraction to the process of writing for the rendering pipeline, enabling users to employ a node-based flowchart system to describe the linear algebra taking place underneath the hood of a given surface shader. The shaders in this study were constructed using this tool. Examples of these graphs can be seen in Figures 4 and 3, while additional information can be found in the official documentation.

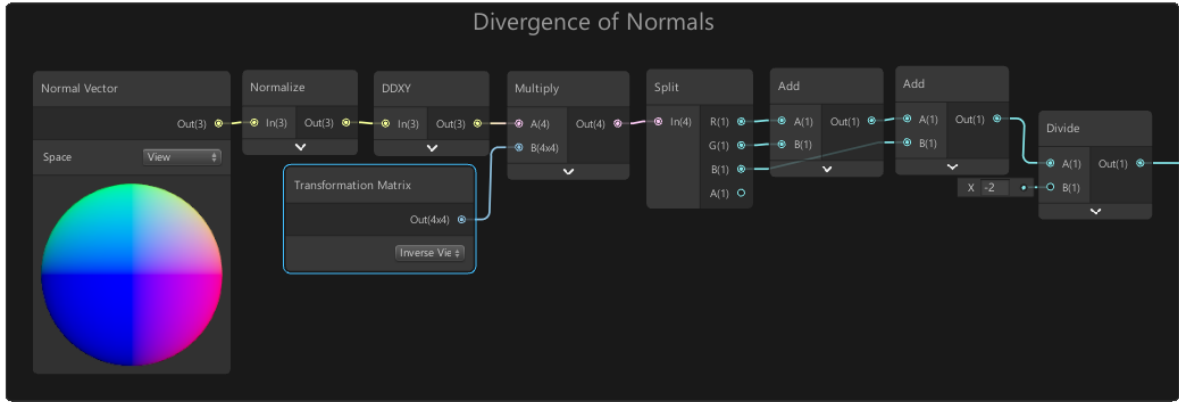


Figure 3: The procedure for calculating mean curvature

3 Methods

3.1 Computation of Curvature

As preciously mentioned, we can assume that necessary normal vectors are supplied by the mesh data of the game object upon which our shader is applied. Incidentally, the normal vectors of a mesh are not necessarily of unit length and must be normalized before differentiation occurs.

The DDXY node used in Figure 3 calculates partial derivative of each supplied normal along each axis, but in order to do so, the passed values must be expressed in terms of the screen-space pixel coordinates of the camera rendering the shaded game object [1]. This is because the DDXY node calls the HLSL GPU functions `ddx()` and `ddy()`. These functions operate by accessing the change in value of each pixel’s set of neighbors in the respective screen x and screen y directions in order to calculate partial derivatives of the passed values [3]. A noteworthy trait is that the gradient vectors produced by these functions are based on linear approximations, and, as such second-order derivatives such as `ddx(ddx())` will yield zero vectors along every screen-space pixel.

The collection of gradients are transformed to object-space to preemptively get the data into the coordinate system needed for the albedo map of our shader. We take the sum of each gradients vector’s components in order to produce the divergence of the unit normals, which are then re-scaled to produce the mean curvature of at each point of the shaded model.

3.2 Color Assignment

The portion of our our shader devoted to applying a color to each point on the passed model is shown in Figure 4. Once a mean curvature x was calculated for each point, a sigmoid curve of the form:

$$S(x) = \frac{1}{1 + b^{-x}}$$

for some base b was applied in a similar manner to a neural net activation function, re-scaling the calculated curvatures to adjusted values in the range of $(0, 1)$. Linear color interpolation (performed by the lerp nodes of Figure 4) was used to assign a color to each corresponding point on the model, with high adjusted values marked in white and low values in black. The chosen base b influenced how aggressively small differences in curvature were shaded differently. Functionality was added to highlight regions of a selected mean curvature using a third, user-defined color.

4 Results

In-engine renders produced by applying our shader to standard computer graphics test models are shown in Figures. 5 and 6. We observe several features that match are consistent with the properties of mean curvature. Through circular

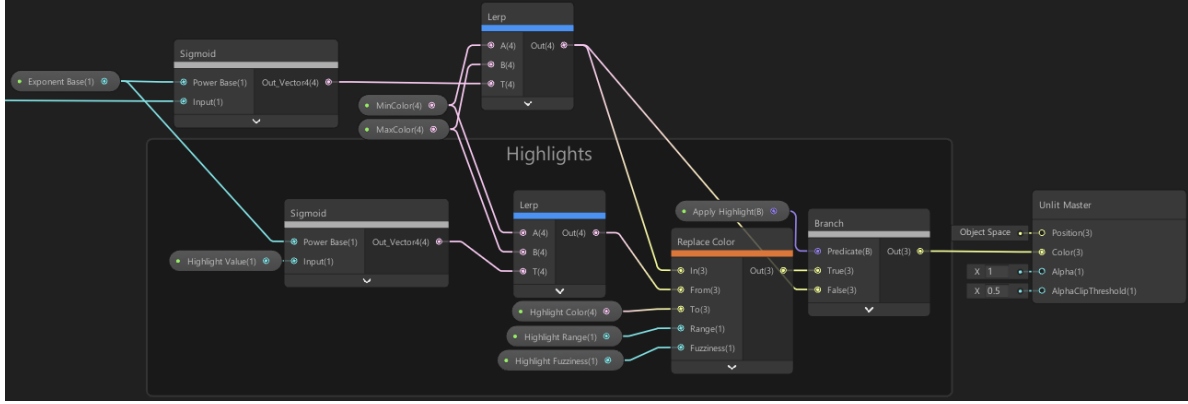


Figure 4: The section of our shader responsible for color selection

groove on the elbow of figure 5 and the inner thigh of the leftmost rabbit in figure 6, we observe that divots and valleys in our models' topography are colored as areas of low mean curvature. With the correct view configuration, we were able to observe a mean curvature of zero in flat planes and a curvature of one along the surface of spheres. This again matches our expected behavior. However, the specific view angles and locations required are topics of concern discussed in the limitation sections.

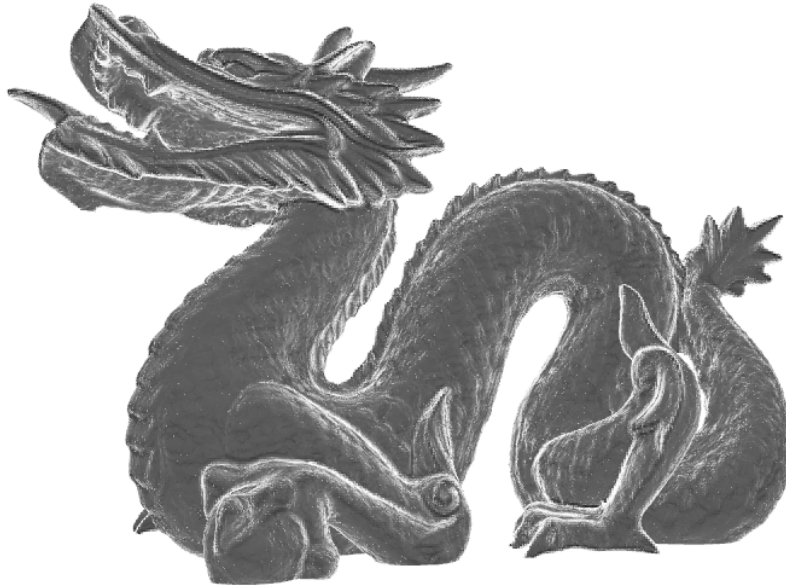


Figure 5: A shaded instance of the Stanford Dragon, $b = 3.5$

Our shaded objects did not receive lighting data. Nonetheless, we consistently found that the coloration from our shader effectively mimicked the presence and interactions due to ambient lighting. This is most clearly demonstrated in 6, where the choice of b in the right-most render resulted in a RGBA value of $(0.5, 0.5, 0.5, 0.1)$ for every shown pixel.

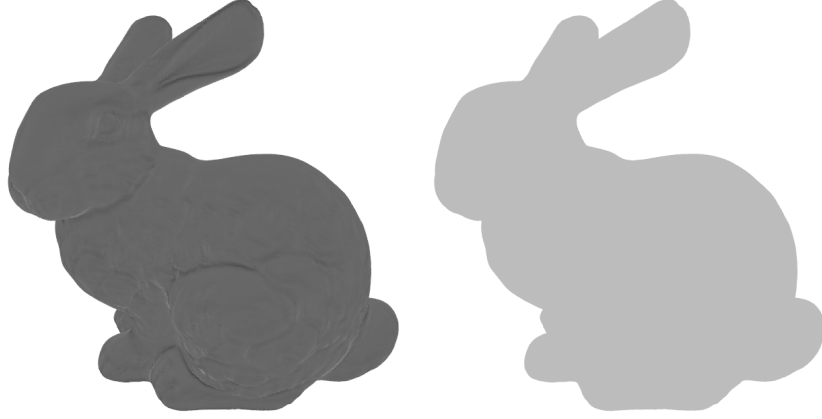


Figure 6: The Stanford Bunny; Loss of natural gradation due to depth and lighting data, ($b = 3.0$, $b = 1.0$)

5 Limitations

As shown in Figure 7, the use of screen-space coordinates results in a non-trivial distortion of the shaders output maps. Although it is entirely unclear how this affects the final render, this alone makes the shader unusable in applications where exact value of mean curvature must be known.

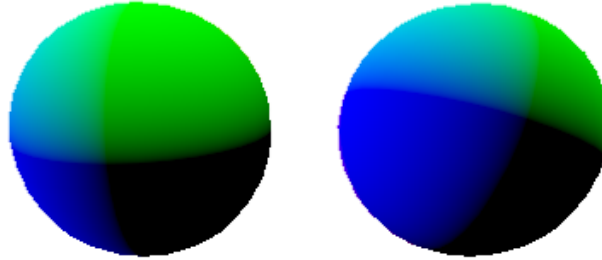


Figure 7: Incorrect texture mapping due to the transformation to and from screen-space. No effects are applied to the leftmost sphere; the right was subject to the transformation and its inverse. The same view angle and direction was used in each case

Additionally, especially triangle-dense portions of a model in which multiple vertices lied inside or near a single pixel resulted in the granular artifacts shown in Figure 8 due to the overlapped shingling of the dragon’s scales. This, of course, is can easily corrected through an increase in render resolution or a more careful selection of models to shade.

Lastly, we noted that the position of the camera relative to the shaded model substantially influenced its coloration.

While examining spheres and planes, we noted that curvature steadily decreased as the camera approached the shaded objects. The expected mean curvatures of one and zero for shaded spheres alluded to in Section 4 occurred when the camera was placed at such a distance that each object appeared less than five pixels wide. Varying camera distances similarly appear to have cause a global decrease in values between Figures 5 and 6. Specifically, we note how there are very few points on the $b = 3.0$ render that are lighter than the grey of the rabbit’s $b = 1.0$ counterpart. This is in stark comparison to the dragon in Figure 5, in which large portions of the render are a lighter grey than the $b = 1.0$ bunny. Moreover, the choice of b should not influence the number of points with positive or negative curvature, but instead *how* positive or negative those values appear to the color selection system.

A possible workaround may lie in introducing a “virtual camera” upon which screen-space calculations are pre-

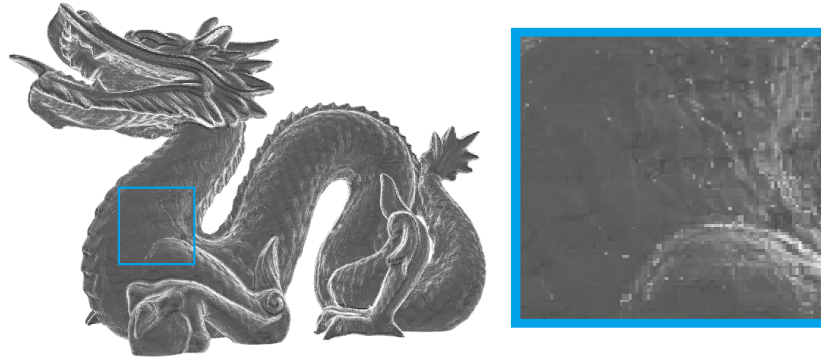


Figure 8: Granular artifacts due to overlapping polygons. Although they significantly reduced their size, these artifacts persist after MSAA and other anti-aliasing techniques were applied.

formed calculated divergence and corresponding color data could then be applied to each point on the shaded object, whereupon the main camera would capture the final render as normal. By manipulating the position of the virtual camera relative to the shaded object, we may be to produce a uniform behavior across all camera depths. Such a rewrite of our shader lies outside the current scope of our paper, however.

6 Future Work

The use of different reference colors during the interpolation process may be worthy of exploration. Due to limited rendering hardware, we were unable to test the behavior of our shader in lit environments, nor were we able to experiment with the application of our tool semi-transparent and/or reflective objects. A rewrite of our shader in pure HLSL may facilitate the implementation of the virtual screen-space system proposed in section 5.

7 References

- [1] Ddxy node. *Unity Technologies*. <https://docs.unity3d.com/Packages/com.unity.shadergraph@5.3/manual/DDXY-Node.html>.
- [2] Writing surface shaders. *Unity Technologies*. Version: 2019.3 <https://docs.unity3d.com/Manual/SL-SurfaceShaders.html>.
- [3] Kevin Bjorke. Understanding gpu derivatives. In *GPU Gems*, chapter 24.2. Nvidia Corporation. <https://developer.nvidia.com/gpugems/gpugems/part-iv-image-processing/chapter-24-high-quality-filtering>.
- [4] Blender Foundation. Uv editor: Introduction. In *Blender 2.82 Manuel*. <https://docs.blender.org/manual/en/latest/editors/uv/introduction.html>.
- [5] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(632):632–658, 2005. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.3008&rep=rep1&type=pdf>.
- [6] Yan-Bin Jia. Curvature. In *Com S 477/577 Notes*. Iowa State University Department of Computer Science, October 2019. <http://web.cs.iastate.edu/~cs577/handouts/curvature.pdf>.
- [7] Eric W Weisstein. Shape operator. *Wolfram Math World*. <https://mathworld.wolfram.com/ShapeOperator.html>.
- [8] Alan Zucconi. Surface shaders in unity3d. June 2015. <https://www.alanzucconi.com/2015/06/17/surface-shaders-in-unity3d/>.