Course: "Computer Graphics," ECS 175, Fall Quarter 2017
Instructor: Bernd Hamann

Project 3: **"THE PHONG LIGHTING MODEL, GOURAUD SHADING, HALF-TONING, AND THE PAINTER'S ALGORITHM"**

Date due: Monday, November 13, 2017

The third project deals with the **Phong lighting model**, **Gouraud shading** for rasterizing shaded triangles, **half-toning** for simulating different brightnesses on a binary display device, and the **painter's algorithm** for solving the hidden surface (*i.e.*, the hidden triangle) problem. Three orthographic projections (onto the $xy-$, $xz-$, and $yz-$plane) are to be generated for a scene in 3D space containing convex polyhedra with triangular faces. The scene should consist of **at least three polyhedra**.

You can utilize the GL graphics library calls whenever appropriate! You can also utilize your own graphics macros you have developed in previous projects.

(i) The Phong lighting model

The formula to be used for computing the intensity at a vertex of a 3D polyhedral object is given as

$$I_{\mathbf{p}} \;=\; k_a I_A + \frac{I_L}{||\mathbf{f}-\mathbf{p}|| + K}\Big(k_d(\vec{\mathbf{l}}\cdot\vec{\mathbf{n}}) + k_s(\vec{\mathbf{r}}\cdot\vec{\mathbf{v}})^n\Big), \tag{1}$$

where $k_a, k_d, k_s \in [0,1]$ are the ambient, diffuse, and specular reflection coefficients, K is a non-negative constant (approximating the "average distance" between the scene and the light source), $n \in \{1,2,3,...\}$ is the Phong constant (affecting the "size" of high lights), I_A is the ambient light intensity, I_L is the light source intensity, $\vec{\mathbf{l}}$ is the light vector $\frac{\mathbf{x}-\mathbf{p}}{||\mathbf{x}-\mathbf{p}||}$ ($\mathbf{p}$ being the vertex position and $\mathbf{x}$ being the light source position), $\vec{\mathbf{n}}$ is the (unit) outward normal vector at $\mathbf{p}$, $\vec{\mathbf{r}}$ is the normalized reflection vector, and $\vec{\mathbf{v}}$ is the viewing vector $\frac{\mathbf{f}-\mathbf{p}}{||\mathbf{f}-\mathbf{p}||}$ ($\mathbf{f}$ being the from point). Having computed the intensities for all vertices in a 3D scene, do not forget to normalize all intensities ($I_{\mathbf{p}} \in [0,1]$). The user must be able to **specify** and **change** any of the parameters in equation (1). You can either store the **normal vectors** at each vertex in the data file describing the polyhedral scene, or, alternatively, you can compute them by averaging normal vectors of triangles sharing a common vertex (as discussed in class).

Regarding the computation of the light vector, the viewing vector, and the reflection vector, the user must be able to specify light source position $\mathbf{x}$ and from point $\mathbf{f}$. Consider the fact that the light vector and the viewing vector are in general different for each vertex in an arbitrary 3D scene. You are supposed to render a 3D scene containing simple 3D convex polyhedra with planar faces that can easily be decomposed into triangles (*e.g.*, tetrahedra, cubes, octahedra, etc.).

(ii) Gouraud shading

You need to project each triangle in the scene onto the $xy-$, $xz-$, and $yz-$ plane. When rasterizing a single triangle pixel-by-pixel, its interior color should vary linearly according to the intensities at its three vertices. This is accomplished by using the **Gouraud shading** algorithm discussed in class. Each triangle is rasterized by first performing **linear intensity interpolation along the three edges** and then performing **linear intensity interpolation on the scanline** that is being rasterized.

(iii) Half-toning for binary display devices

Assume that you have a display device for which each pixel can either be "on" or "off." Since you have to simulate 10 intensity levels, you need to use "**virtual**" **pixels** which actually occupy a 3×3 pixel square on the binary display device. You can define your own "virtual" pixel having intensity i by defining a 3×3 pixel "array" for which i physical pixels are "on." Using M $\times$ N "virtual" pixels requires at least 3M $\times$ 3N physical pixels. Rasterizing triangles should use "virtual" pixels which are then mapped to the proper screen location. Perform linear interpolation using real numbers and round (or truncate) the resulting intensity for a particular "virtual" pixel to an integer – having ten (10) intensity levels available (0, 1, 2, ..., 9).

(iv) Hidden surface removal using the painter's algorithm

Assume that your entire 3D polyhedral scene is defined entirely in the first octant of a 3D coordinate system (*i.e.*, all $x-$, $y-$, and $z-$ coordinates of all vertices are positive). When projecting the scene onto the $xy-$plane, the **triangles must be sorted with respect to their depth** in $z-$direction, when projecting the scene onto the $xz-$plane, the triangles must be sorted with respect to their depth in $y-$direction, and when projecting the scene onto the $yz-$plane, the triangles must be sorted with respect to their depth in $x-$direction. According to the painter's algorithm, the triangle with greatest distance from the projection plane must be rasterized at first, and the one with smallest distance at last. Remember: The depth of a triangle is defined as the minimal depth of its vertices. This requires the implementation of a sorting algorithm. Since it is assumed that the number of triangles in your scene is relatively small, a *bubble sort* is sufficient.

Besides having to hand in a program listing, please prepare a "manual sheet" explaining how to use your program.

The overall grade (on a scale from 0 to 100) will depend on i) **completeness** (40%), ii) **correctness** (40%), iii) **interface quality** (15%), and iv) the **manual sheet** (5%). No project will be accepted when it is more than seven (7) days late; for each day, one (1) point will be deduced.

DO NOT REMOVE YOUR PROGRAM! YOU WILL BE ABLE TO USE IT IN THE NEXT ASSIGNMENT(S).

H A V E F U N ! ! !