CSC 2521 Final Project Report

Hanieh Bastani

December, 2007

NPR Renderer: Overview

I implemented a 3D NPR renderer which supports contours, suggestive contours, and *toon* shading. For this implementation, I used the *trimesh2* library written by Szymon Rusinkiewicz. The mesh models used are in PLY format from the Suggestive Contours gallery page. I also used the *rtsc* software (*http://www.cs.princeton.edu/gfx/proj/sugcon/index.html#software*) as a learning tool and reference during the research phase of writing this implementation, and also adopted their method of calculating the directional derivatives (identified as $DwKr$ in their implementation) of vertices for suggestive contours.

I used the convention of normals pointing outward from the surface, to be consistent with the literature.

Visibility issues are resolved by rejecting triangles that are entirely on the back face of the mesh. If the dot product of the view vector with the vertex normals yield a negative value, for all vertices across a triangle, the triangle lies on the back face of the mesh and is ignored.

Contours

Determining contours requires finding the zeros of the dot product between n and v, n being the surface normal at point p, v being the vector from the view position to the point p. Working with polygons in my implementation, $n.v$ is calculated exhaustively for every vertex. As it is often the case, the zeros of $n.v$ may occur between two vertices, which means one vertex has a positive $n.v$ value, while another has a negative $n.v$ value. Thus, for every triangle of the mesh, I perform a check for a sign change among the triangle vertices. If it is determined that $n.v$ at one vertex has a different sign than the other two, a simple linear interpolation determines the point on the triangle edges at which $n.v = 0$. For example,

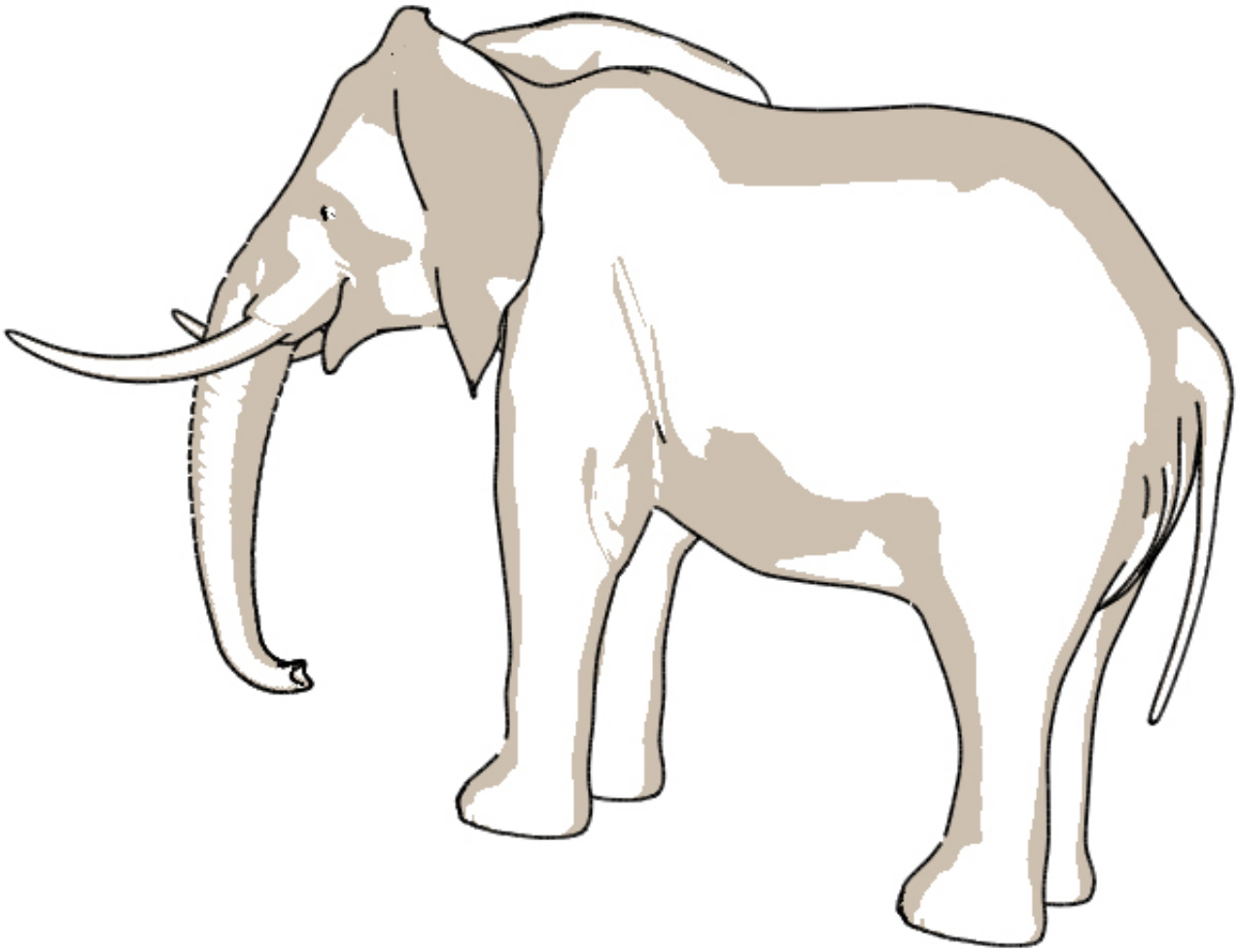

Figure 1: Elephant Contour with 'Toon' Shading

if $n.v$ at $vertex_0$ has a different sign than $n.v$ at $vertex_1$ and $vertex_2$, a linear interpolant finds the points p_1 and P_2 on the edges $< vertex_0, vertex_1 >$ and $< vertex_0, vertex_2 >$, respectively. A line with a specified thickness is then drawn between p_1 and p_2. Figures 1 to 4 are the resulting contour renderings, shown with different styles of shading, covered in section .

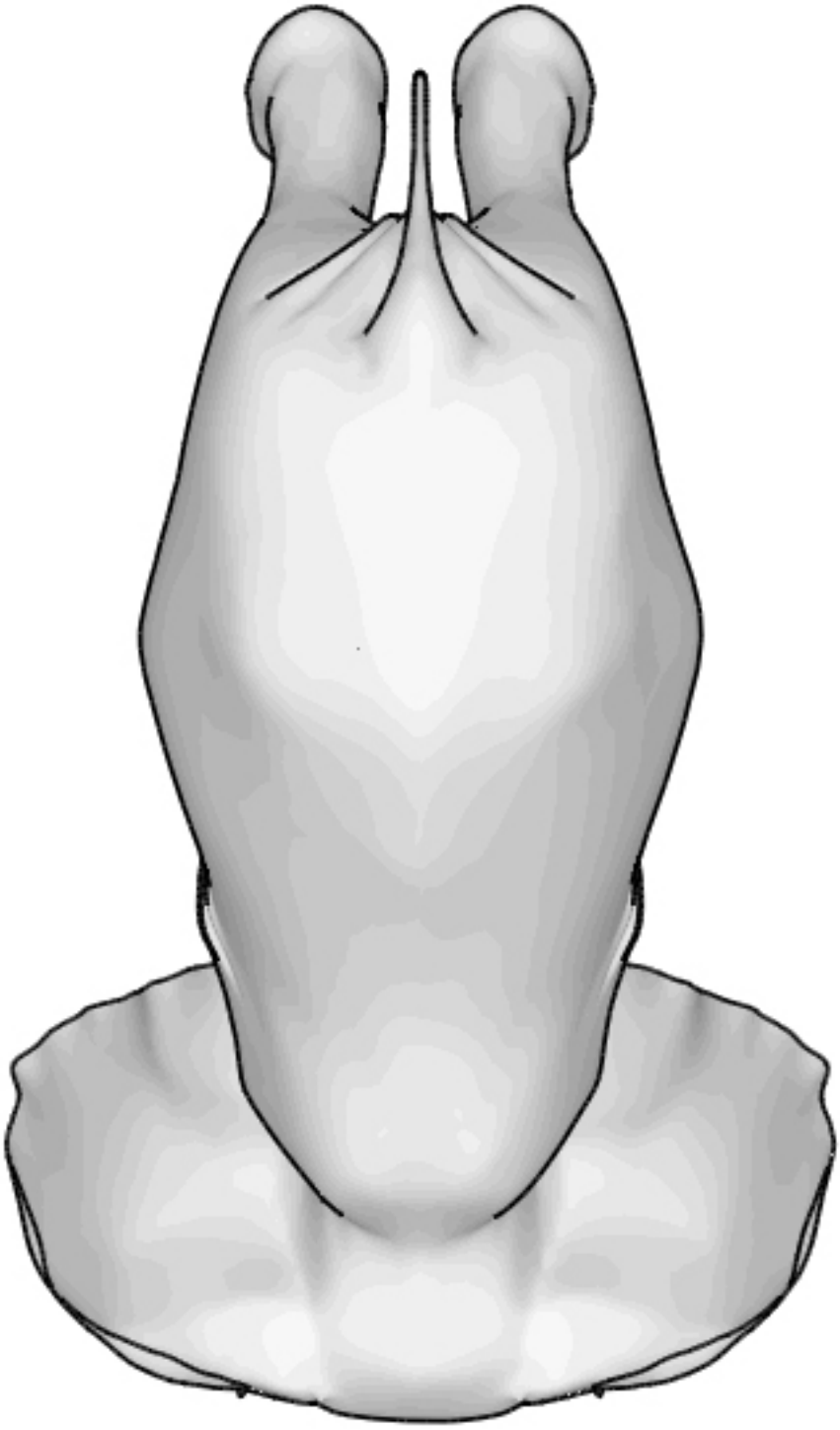

Figure 2: Elephant Contour with ‘Smooth’ Shading

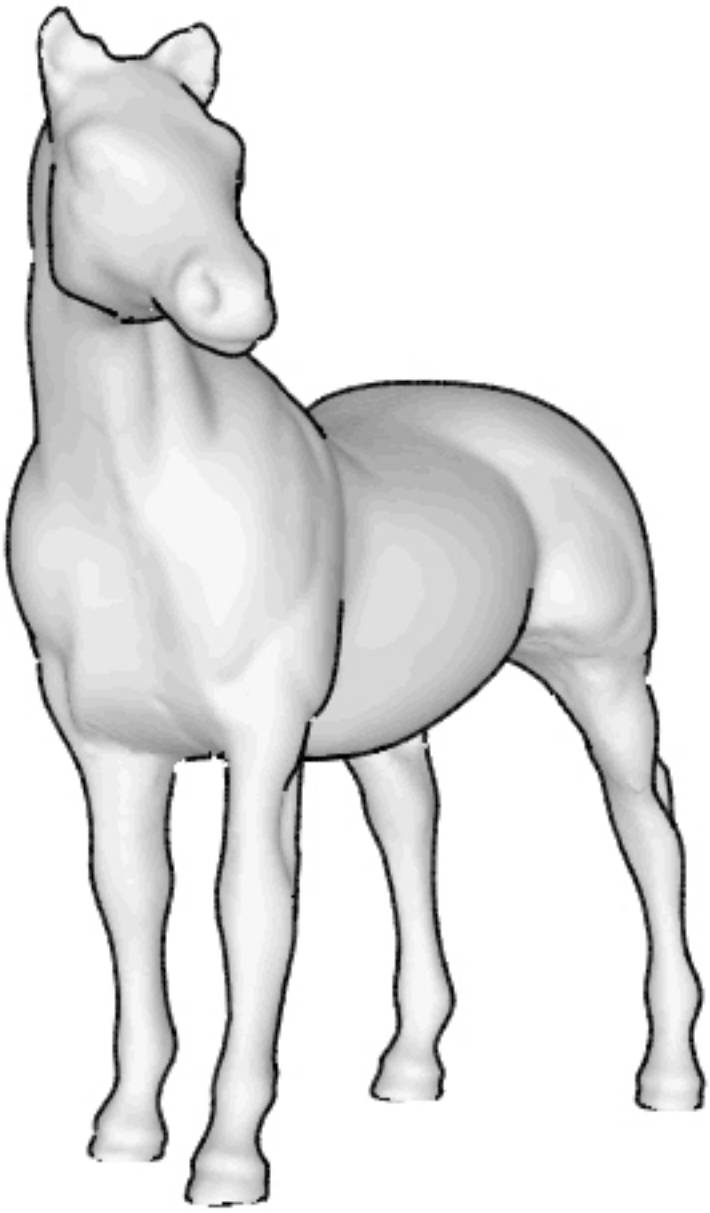

Figure 3: Horse Contour with ‘Smooth’ Shading (Front)

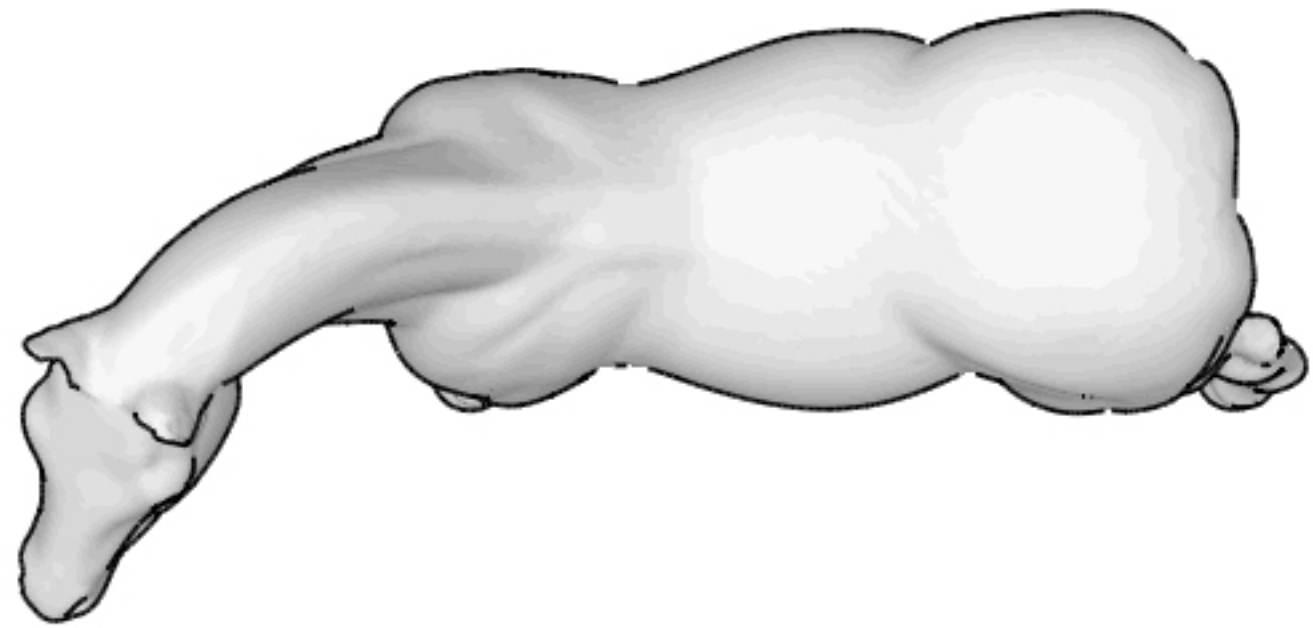

Figure 4: Horse Contour with 'Smooth' Shading (Top)

Suggestive Contours

Calculating Radial Curvature

Suggestive contours require the calculation of the radial curvature κ_r at every vertex of the polygonal mesh, and are drawn where $\kappa_r = 0$ subject to the constraint that the directional derivative in the direction of the view vector is positive. In my implementation, κ_r for every vertex i is calculated as follows:

$$k_r[i] = meshObj->curv1[i] * cos2Theta + meshObj->curv2[i] * sin2Theta \quad (1)$$

where $curv*[i]$ indicates the principal curvatures at vertex i, provided by the *trimesh2* library. $Theta$ is the angle between the view vector and the principal directions corresponding to the principal curvatures, provided through the *pdir** member of the mesh object in the *trimesh2* library. $cos2Theta$ and $sin2Theta$, representing $cos^2(\theta)$ and $sin^2(\theta)$ are calculated

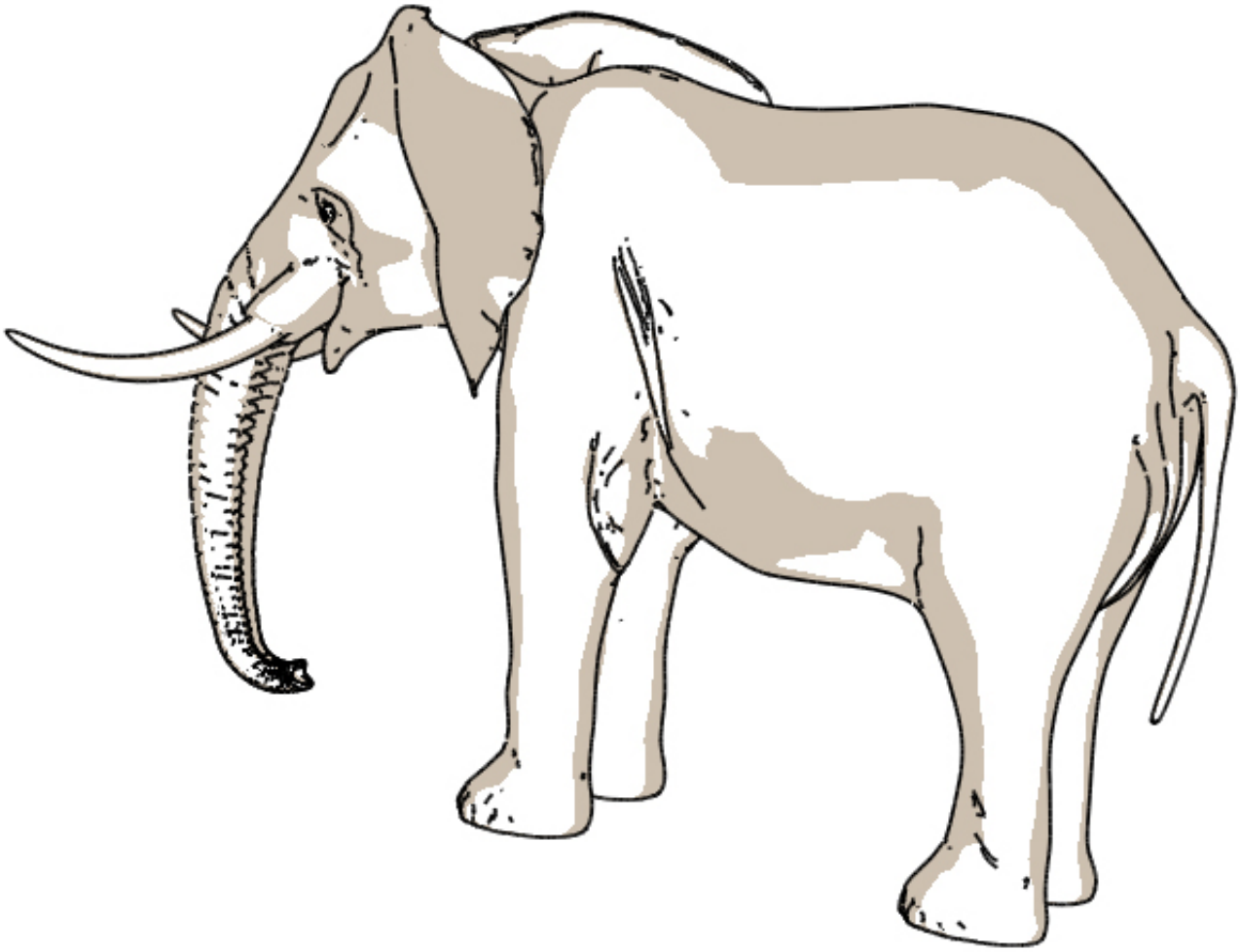

Figure 5: Suggestive Contours on Elephant with 'Toon' Shading

with respect to the dot and cross products between the view vector and principal directions at the vertices. Figures 5 to 8 are the resulting suggestive contours renderings.

Determining Zero Crossings

To get the appearance of smooth contours, I check once again for triangles in which one vertex has a radial curvature with a different sign than the other two. Upon encountering such a triangle, I check the value of the directional derivatives D_k of the vertices of the triangle. The directional derivatives must be positive, which would mean that the eye is viewing the

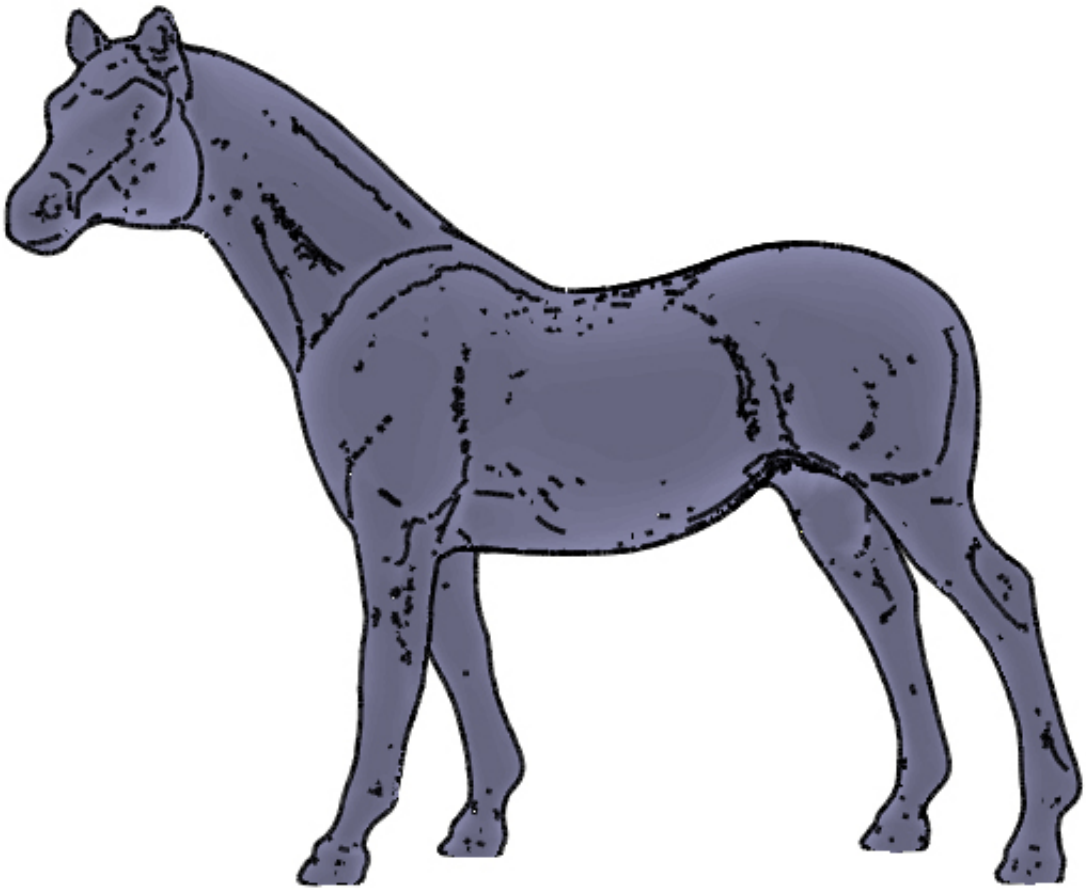

Figure 6: Suggestive Contours on Horse with ‘Smooth’ Shading

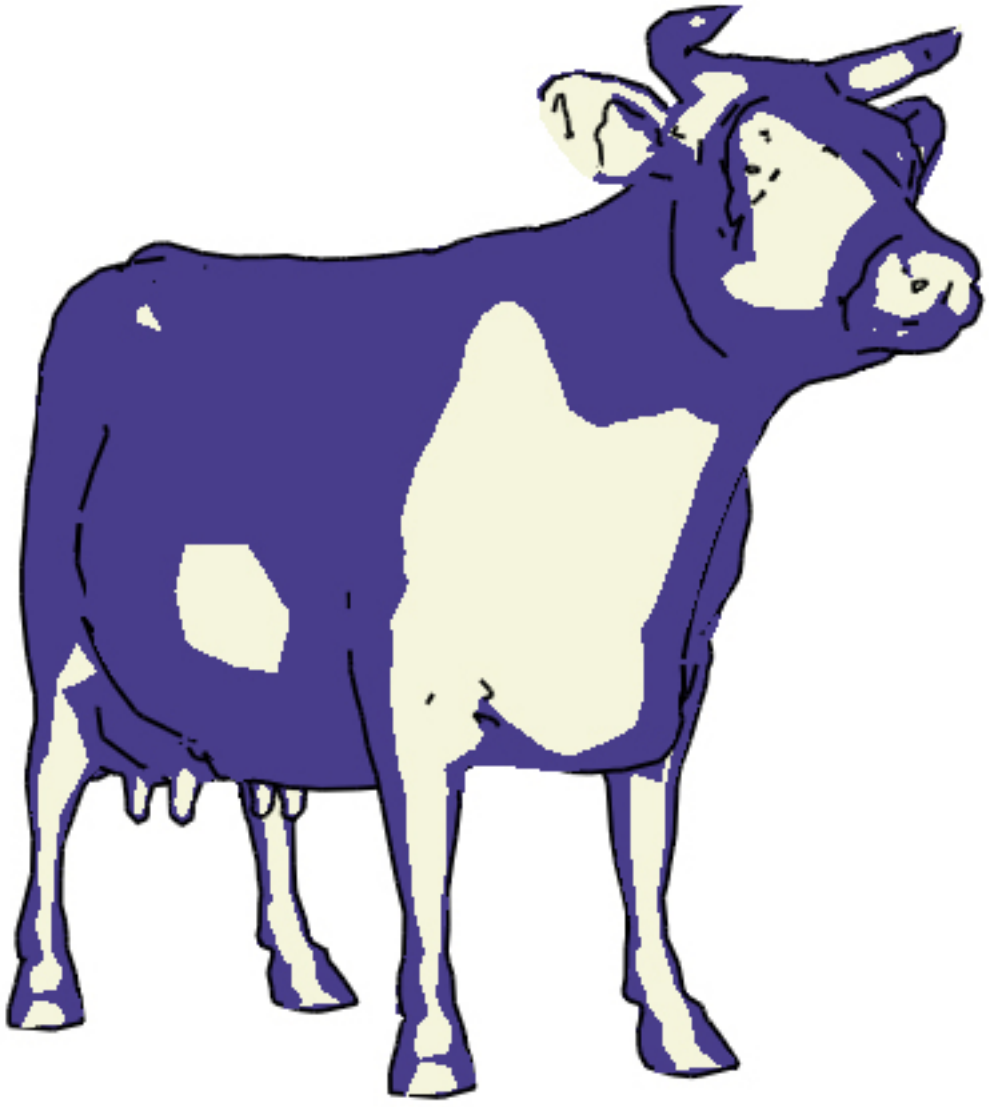

Figure 7: Suggestive Contours on Cow with ‘Toon’ Shading

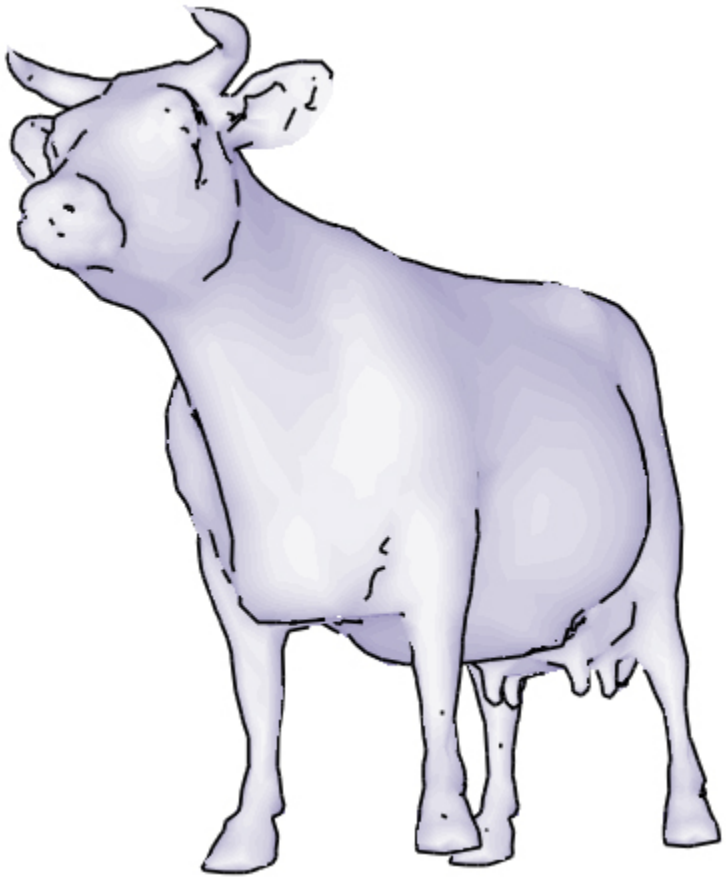

Figure 8: Suggestive Contours on Cow with ‘Smooth’ Shading

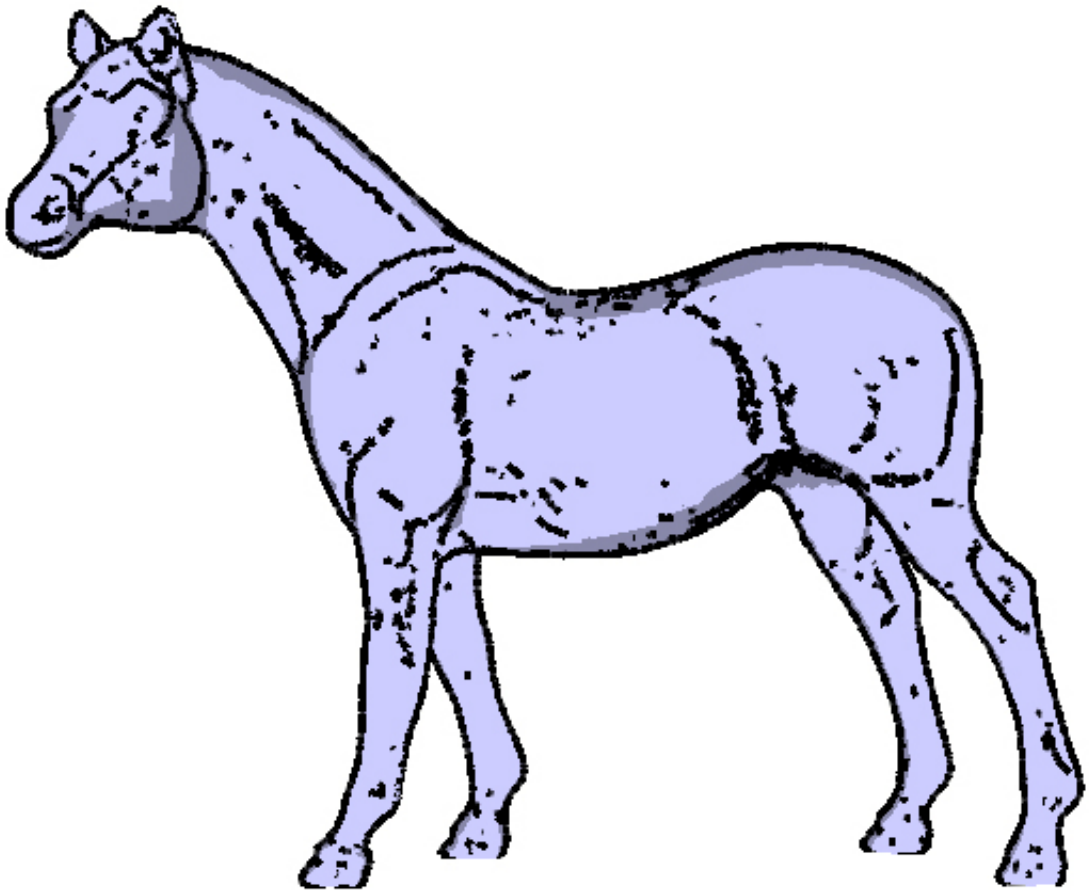

Figure 9: Suggestive Contours with No Line Stylization

point from its convex side. If the D_k values are negative, the exhaustive search for a potential suggestive contour continues. However, if they are positive, the two points at which $k_r = 0$, lying on the edges leading to the vertex with a singular k_r sign are approximated using linear interpolation, and are used to specify endpoints of a (suggestive) line.

Styling Elements

Figure 9 is an example of equal line density/thickness throughout the line drawings. Normally artists avoid such harsh lines, and tend to vary line intensities and/or thickness. The *Siggraph* lecture notes suggest varying line thickness as a function of the square root of the radial

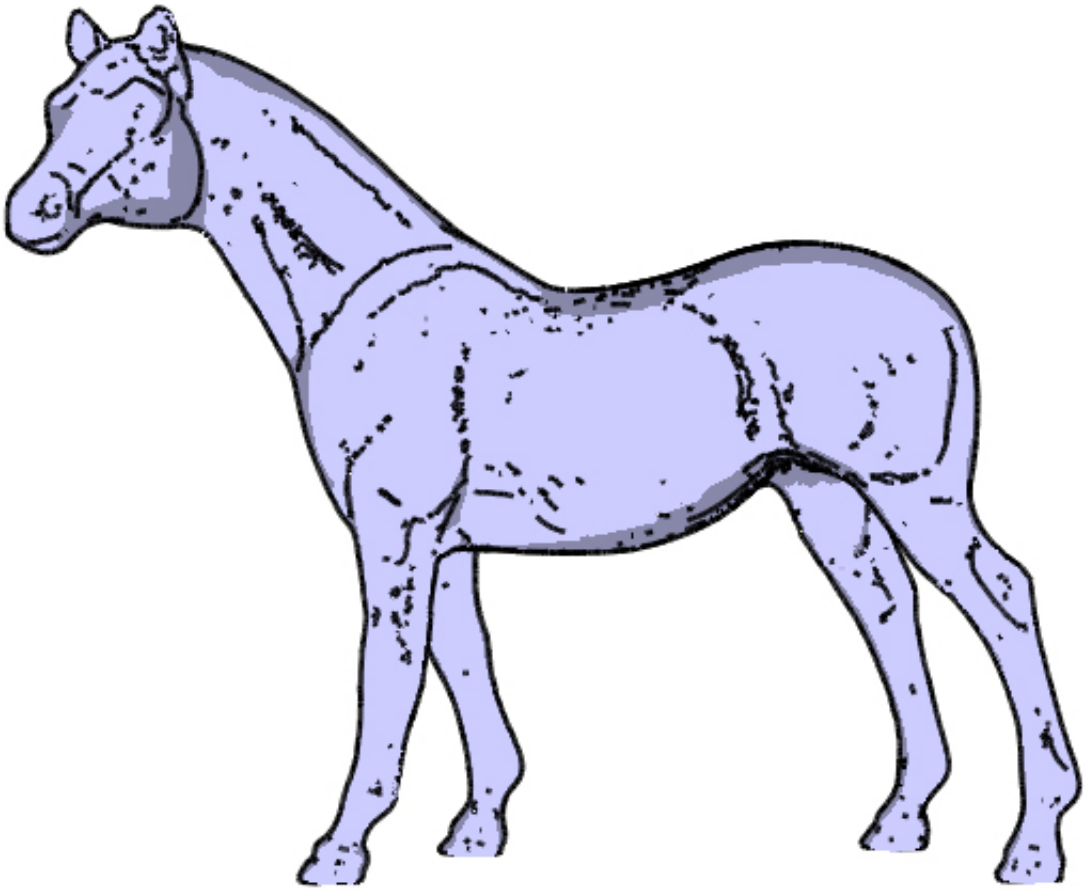

Figure 10: Suggestive Contours with "Alpha" Line Stylization

curvature. After some experimentation, I used the following expression to specify the alpha parameter of the line:

$$1.0 - \sqrt{(}p_{Kr} * \alpha) \tag{2}$$

where α is some large number (10000 in most of my renderings), and p_{Kr} is the interpolated radial curvature value of the zero crossing point. Figure 10 was rendered with alpha values based on κ_r, and shows aesthetic improvement in terms of line smoothness and edge softness over the previous rendering.

Toon Shading

I implemented two types of shadings, a smooth and a hard *toon* shading. A texture image is constructed for each type. The texture coordinates are assigned based on the dot product between the normal n at point p, and the vector from point p to the light position, call it l. For the hard shading, the texture image is constructed from multiple distinct colors, with no transitional colors. Hence, within a certain range of $n.l$ the same color is used. As the values of $n.l$ pass a specified threshold, the color changes abruptly. In the case of the smooth shading, the texture map varies smoothly by interpolating the color values of multiple specified colors. The user can specify color values, as well as the threshold parameter controlling the regions of the distinct parameters. Figures 11 to 13 show renderings with different shading parameters.

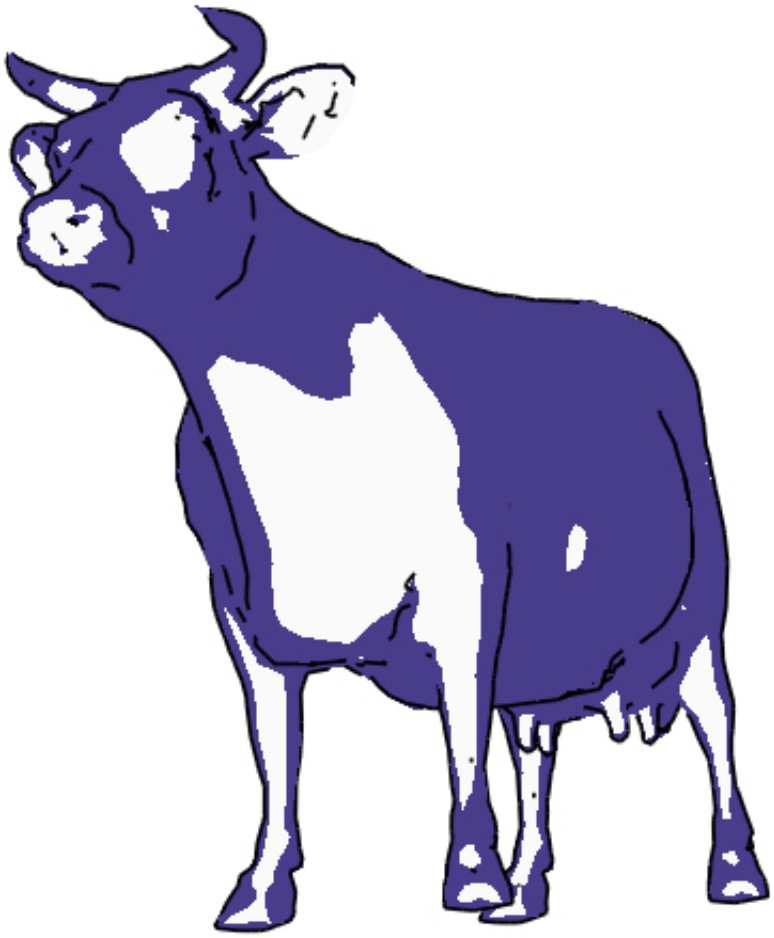

Figure 11: Cow Toon Shaded, Two Colors

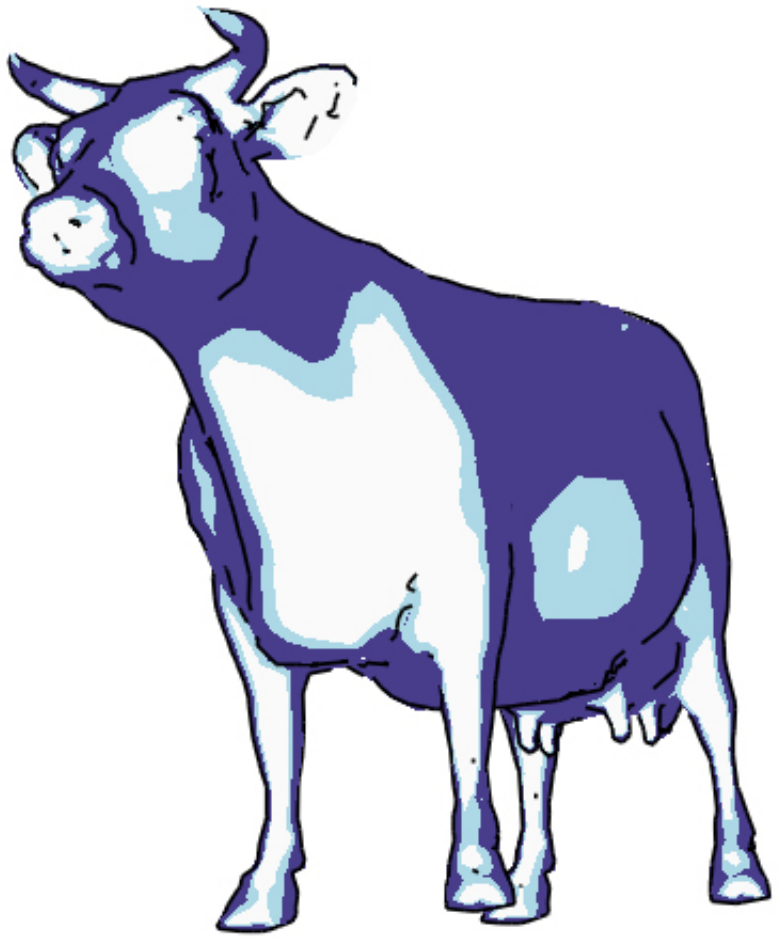

Figure 12: Cow Toon Shaded, Three Colors, Threshold Between Regions (1)

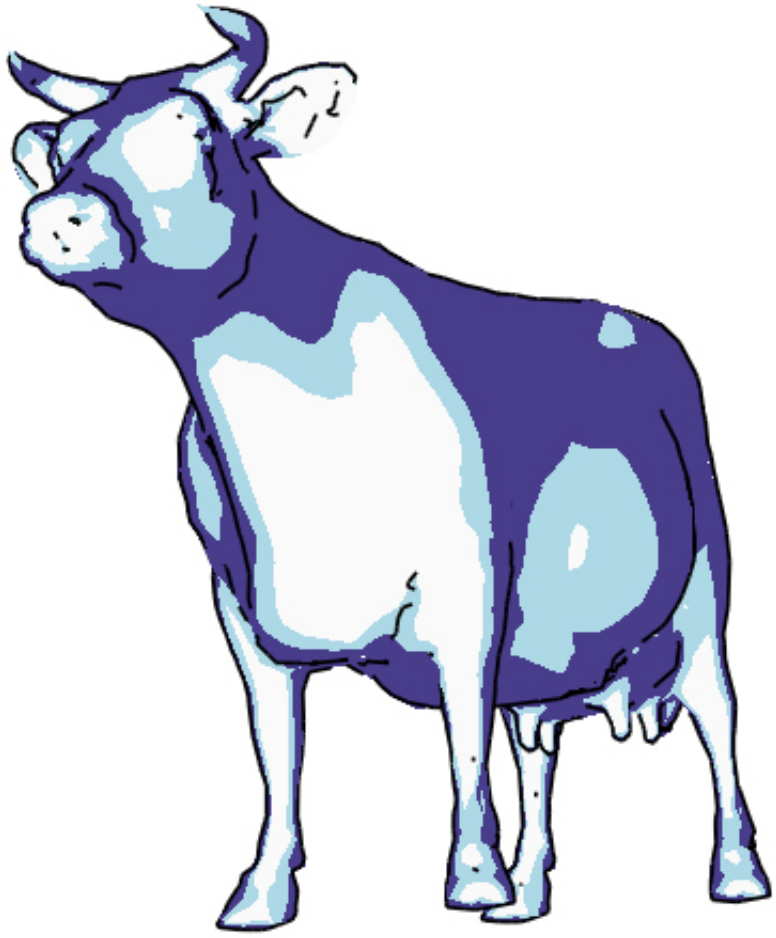

Figure 13: Cow Toon Shaded, Three Colors, Threshold Between Regions (2)