Introduction

Significant work is being expended on photorealistic rendering techniques, techniques that produce rendered images that are indistinguishable from a photograph of an equivalent scene. Advances in areas like global illumination, modelling and rendering of natural phenomena (e.g., fire, water, smoke), and image-based rendered have made significant strides towards photorealistic results.

A separate area of computer graphics studies nonphotorealistic rendering, methods for producing stylized or artistic images. Examples include pen-and-ink sketches, painterly images, and cel shaded models.

Many research-based techniques are now being applied in real-world application environments. In particular, cel shading algorithms are now popular in computer games. Use of these techniques produces a cartoon or anime-like feel to the objects and characters.
Shading Polygons

The basic cel shading algorithm incorporates ambient and diffuse lighting in a manner similar to the OpenGL pipeline. Recall that ambient lighting is the "background" lighting that falls uniformly on every surface due to multisurface scattering. Diffuse lighting occurs when a light source shines on a surface. The diffuse intensity is inversely proportion to the angle between the surface normal and the direction to the light source.

Like OpenGL, the basic cel shading algorithm wants to compute the ambient and diffuse intensities at each vertex, then convert these into a vertex colour that is interpolated across the polygon's face during scanline rendering. Unlike OpenGL, however, cel shading does not use a continuous range of colours or intensities. Animations are characterized by surfaces with constant colour and discrete variations in luminance. Colour differences at a surface or luminance boundary normally have a large, sharp variation.

Rather than trying to implement these variations directly (which would probably require replacing OpenGL's scanline rendering routines), a simpler method is as follows:

1. Request to render a surface with a constant colour (i.e., attach this constant colour to each vertex).
2. Compute ambient + diffuse light intensity at each vertex.
3. Convert intensity into a $u$-value for lookup into a 1D luminance texture containing discrete luminance variations; the texture is used to modulate the brightness of the vertex colours.
4. Render the polygons using the OpenGL pipeline; this will interpolate the $u$ at each pixel, producing sharp variations in luminance wherever $u$ crosses a luminance boundaries in the luminance texture.
Luminance Texture

Unlike a normal lit, shaded object, a cel shaded object uses only a small number of possible luminances for a given surface. Although these values can be computed directly, an easier method is to use a one-dimensional "luminance texture" to darken a surface based on its reflected intensity.

In our simple example, we will use diffuse intensity at each vertex to control texture lookup. Given normalized surface normal N and direction to the light source L, the cosine of the angle \( \Theta \) between these two vectors is:

\[
c = \mathbf{N} \cdot \mathbf{L} = (N_x \cdot L_x) + (N_y \cdot L_y) + (N_z \cdot L_z)
\]

For front-facing polygons, \( 0 \leq c \leq 1 \). We could use arccos to compute \( \Theta \), but there's no need for this. Instead, \( c \) can be used directly as the \( u \) value for texture lookup; when \( N \) and \( L \) are identical, \( c = u = 1 \) and we have a maximum-brightness modulation of the underlying vertex colour. As \( c = u \) tends to 0, the texture makes the underlying vertex colour darker and darker.

Note: The reason this works properly is because OpenGL interpolates between the different \( u \) assigned to each vertex (i.e., it computes \( u \) for each pixel, does a texture lookup, and darkens the pixel). This preserves the discontinuities in the luminance texture.
Silhouette Edges

A second feature of cel shaded objects is a silhouette edge. Boundaries between the polygons that make up an object and the background (i.e., the projected boundary of the object) are highlighted to emphasize the object's silhouette.

In technical terms, a silhouette edge occurs along any edge between a front-facing polygon and a back-facing polygon.

Rather than searching for silhouette edges explicitly, we can use OpenGL's culling routines to highlight the edges automatically. This is done as follows:

1. Render the scene as normal, with filled polygons and back-facing polygons culled.
2. Switch to cull front-facing polygons.
3. Switch to rendering polygons as wireframes rather than filled solids.
4. Increase line width to a suitable pixel size to draw "thick" lines.
5. Re-render the scene. All silhouette edges will now be displayed. Any edges that are not silhouettes will not be shown, because the Z-buffer contains values for the front-facing polygons rendered in step #1 (and since non-silhouette edges of back-facing polygons are always behind front-facing polygons, they are not displayed).
Improvements

This description of cel shading describes only the simplest possible implementation. A number of additional steps can be added to improve the overall appearance of the rendered objects.

Per-Pixel Shading.

Instead of computing vertex $u$-values, then interpolating those values across the entire polygon, $u$ can be computed on a per-pixel basis. This means that light intensity is computed at each pixel, then used to apply a texture to darken the underlying colour as needed.

Per-pixel shading also allows more sophisticated lighting effects to be properly rendered, in particular, specular highlights. Note that no changes need to be made to the original algorithm; adding specular lighting calculations at each pixel simply increases the light intensity at pixels within the specular highlight, and therefore reduces the amount of darkening during texture mapping.

Highlighting Creases.

Silhouette edges are not the only places where lines are drawn in a cartoon or animation. Edges between two polygons with a significant difference in their normal are sometimes drawn to highlight "creases" in the model.