

# Cel-shading and sketch rendering \*

Matthew Belcher  
Digital Computer Laboratory  
University of Illinois at Urbana-Champaign  
matt@mattbelcher.com



Figure 1: Cel-shaded model



Figure 2: Cel shading detail

## 1. INTRODUCTION

This report describes my implementation of a cel-shading and sketch-rendering technique found in "Stylized Rendering Techniques for Scalable Real-time 3D Animation" by Lake, et al.

I adapted the technique for cel and sketch rendering described in this paper to work with 3D models stored in the MD3 format. This format is commonly used in real-time 3D games such as id Software's Quake 3. This file format consists of a triangle mesh and associated textures.

## 2. CEL-SHADING

The cel-shading algorithm is straight-forward. When the model is loaded, we compute and store the vertex normals of the triangle mesh. Then, we average the color of the textures so that each texture is one solid color. We create a 1D texture based on this color, but scaled by the shading factor. This creates a texture with darker tones of the same color past some threshold.

At rendering time, we compute the dot product of the normalized vertex normal and the lighting direction, and use this value as the texture coordinate for the vertex. This creates a hard boundary between the lighter and darker color



Figure 3: Sketch texture

tones based on the angle at which the light is impacting a surface. Surfaces angled away from the light receive a darker tone.

## 3. SKETCH RENDERING

The technique for sketch rendering is similar to that of cel-shading. Again, we use the dot product of the vertex normals with the lighting direction to calculate texture coordinates. However, in this case, we are not using the resultant value as an index into a single texture, but rather as a determinant as to what texture to apply to this vertex. For example, given four examples of sketch textures, we might separate the range of possible dot product values into four "buckets." If the dot product falls within the range for a bucket, then we apply the associated texture to that surface.

Of course, if the values for the vertices of a given triangle fall into different buckets, we must find a way to subdivide

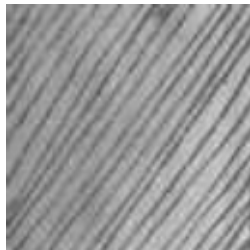


Figure 4: Sketch texture



Figure 5: Sketch rendering detail

the triangle and apply the distinct textures to the computed sub-triangles. We can find the correct subtriangles by linearly interpolating the computed shading value along each edge of the triangle to find the point at which the surface will transition from one texture to another. We choose this point as a vertex for our new subtriangles.

#### 4. SILHOUETTE RENDERING

To render the silhouettes, we simply render all polygons which do not satisfy the OpenGL depth test as lines with 3 point thickness. Once we render the front-facing polygons, part of these lines will show at the edges of the figure. This method uses more computational resources than other edge detection methods (since we are rendering even culled faces), but is far easier to implement. Since neither cel-shading or sketch-rendering is computationally intensive, this has little effect on the performance.



Figure 6: Cel-shaded model with silhouette outlining

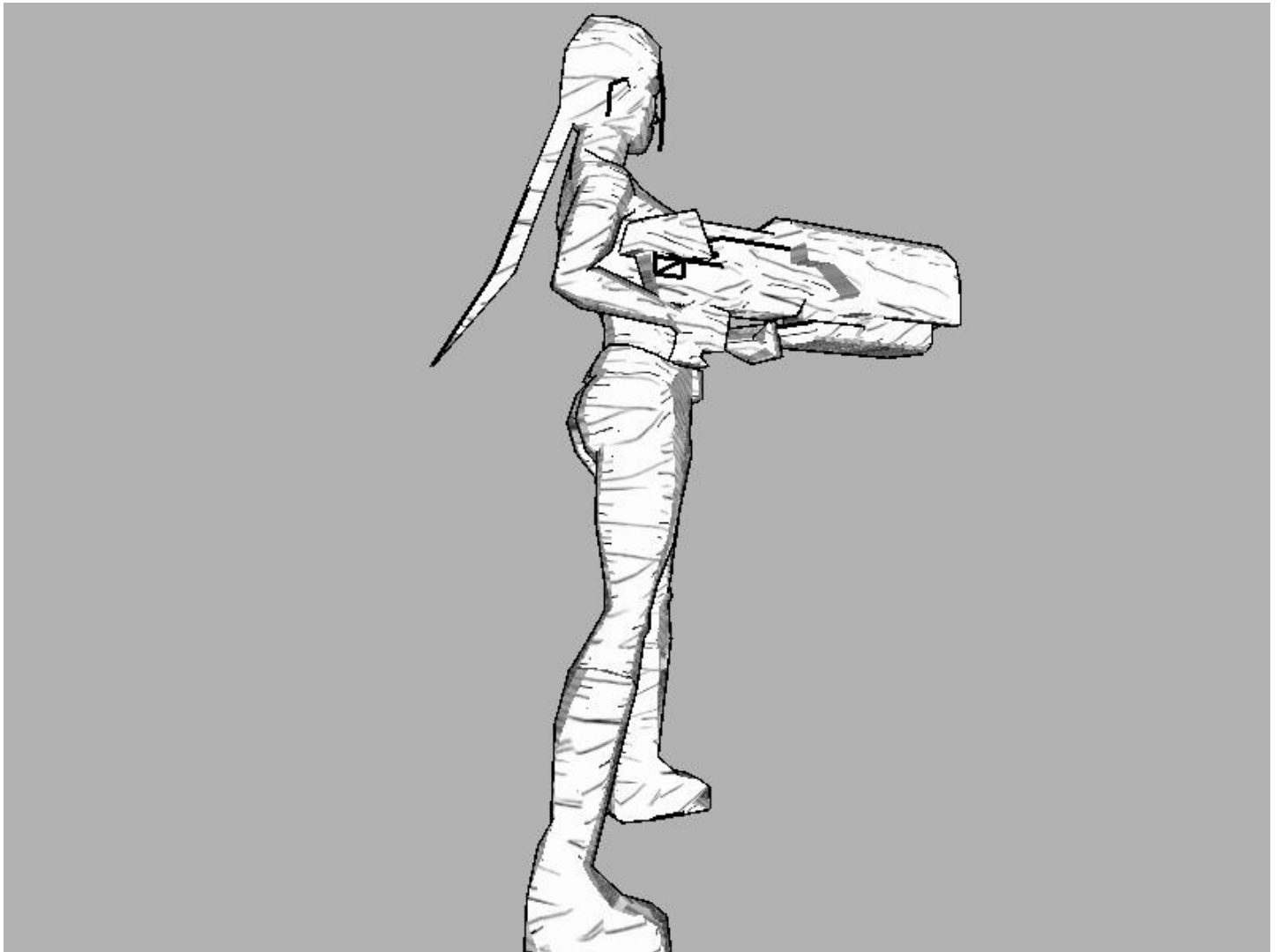


Figure 7: Sketch rendered model with silhouette outlining