COSC422 Assignment 2 Non-Photorealistic Rendering Report

Student ID: 97245310
Student Name: Yuezhang Zhu

Overview

The dolphin can be rendered in two-tone or pencil shading by toggling the space key, as shown in Figure 1 and Figure 2.

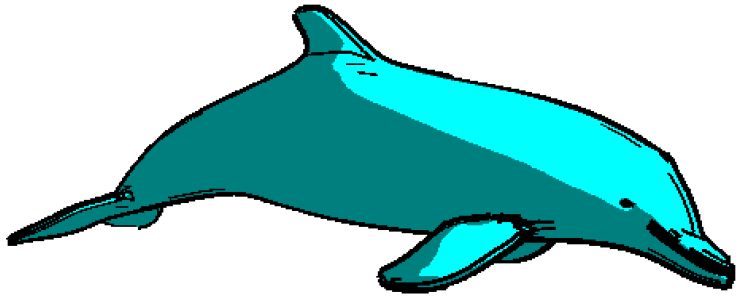

Figure 1. Two-tone shading

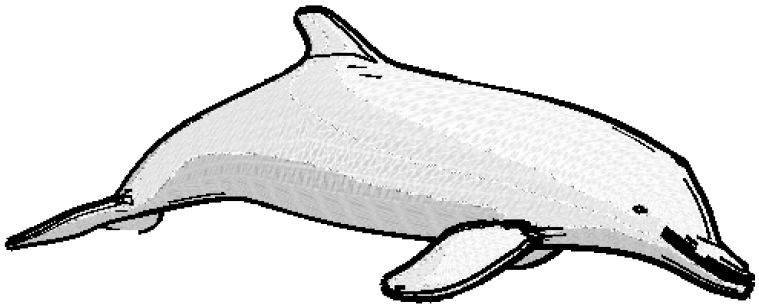

Figure 2. Pencil shading

Extra Features

1. A mipmap set of 12 textures is used to improve the rendering of pencil stroke textures. 4 different sizes (64x64, 32x32, 16x16, 8x8) are used for each of the 3 pencil stroke types shown in Figure 3. The pencil textures are drawn and scaled using GIMP. Please refer to the directory “Textures” to see the individual texture sizes. For code implementation, please see function loadTextures() in MeshViewer.cpp, and loadTGA_mipmap(string filename, int level) in loadTGA.h.

Figure 3. Pencil textures used for rendering

2. Multi-texturing of pencil shading is implemented based on the diffTerm from lighting calculations. The multi-texturing can be toggled by key ‘t’. Figure 4 and 5 below shows the

dolphin with and without multi-texturing. The code is implemented in the fragment shader as shown in Figure 6, where texSample 0,1 and 2 correspond to the 3 levels of pencil stroke density in increasing order.

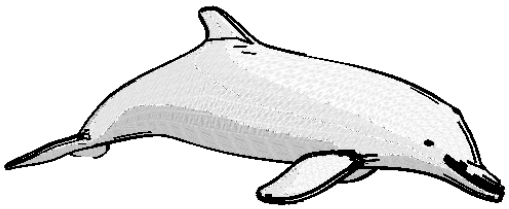

Figure 4. With multi-texturing

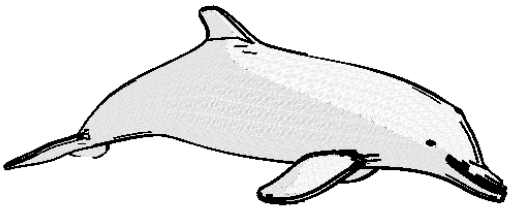

Figure 5. Without multi-texturing

```
vec4 getPencilColor()
{
    vec4 penColor;
    if (diffTerm < 0.0){
            penColor = texture(texSample[2], texCoords);
    } else if (diffTerm > 0.6) {
             penColor = texture(texSample[0], texCoords);
    } else {
            if (multiTexture){
                if (diffTerm > 0.45){
                    penColor = mix(texture(texSample[1], texCoords), texture(texSample[0], texCoords), (diffTerm - 0.45) / 0.45);
                } else if (diffTerm < 0.25){
                    penColor = mix(texture(texSample[2], texCoords), texture(texSample[1], texCoords), (diffTerm - 0.25)/ 0.25);
                } else {
                    penColor = texture(texSample[1], texCoords);
                }
            } else {
                penColor = texture(texSample[1], texCoords);
            }
    }
    return penColor;
}
```

Figure 6. Code for multi-texturing

3. Silhouette edges and crease edges are implemented with adjustable thickness. Press key ‘q’ (increase) and ‘a’ (decrease) to change silhouette thickness, and ‘w’ (increase) and ‘s’ (decrease) to change crease thickness. Figure 7 - 10 shows silhouette and crease edges in different thicknesses. Please refer to the geometry shader for code implementation.

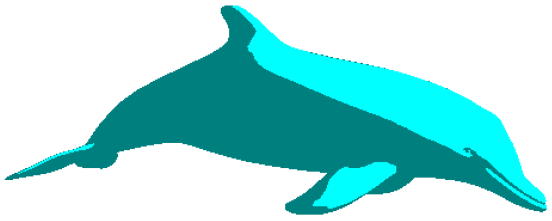

Figure 7. No silhouette or crease edge

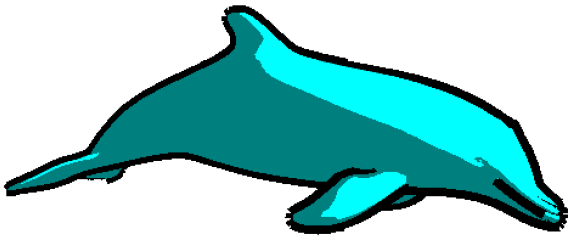

Figure 8. Increased silhouette edge

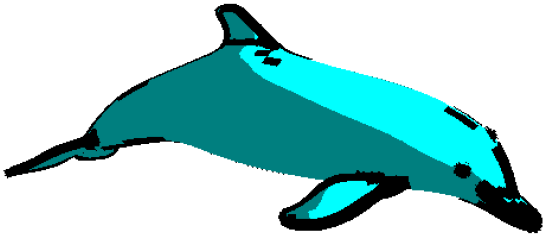

Figure 9. Increased crease edge

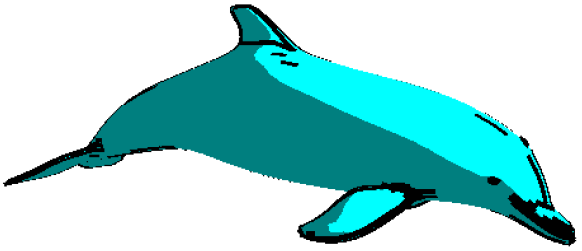

Figure 10. Decreased crease edge

4. The edge gaps can be reduced or increased by pressing key ‘m’ and ‘n’ respectively. The gaps are reduced by extending the 2 edge points in opposite directions in each triangle of a triangle strip (by adding multiples of the unit vector between these 2 points to each point). Figure 11 shows a comparison between increased and decreased edge gaps. Please refer to the geometry shader for code implementation.

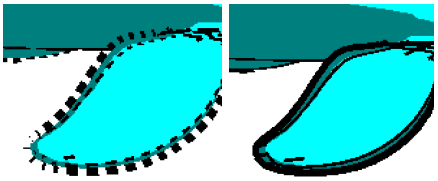

Figure 11. Increased edge gaps (left) and decreased edge gaps (right)

5. Mesh simplification can be toggled by pressing key ‘2’. Please allow 10 seconds for the application to re-render. Figure 12 and 13 shows a comparison between the original and simplified mesh. The code implementation is shown in Figure 14.

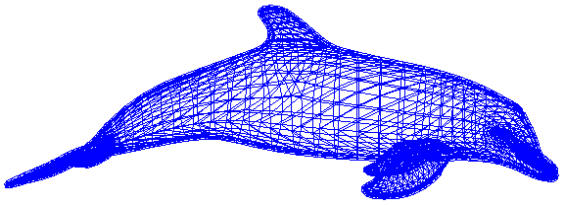

Figure 12. Original mesh

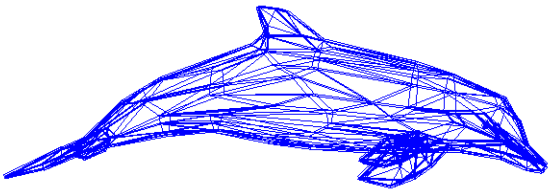

Figure 13. Simplified mesh

```
void simplifyMesh()
{
    // Decimation Module Handle type
    typedef OpenMesh::Decimater::DecimaterT< MyMesh > MyDecimater;
    typedef OpenMesh::Decimater::ModQuadricT< MyMesh >::Handle HModQuadric;
    MyDecimater decimater(mesh); // a decimater object, connected to a mesh
    HModQuadric hModQuadric; // use a quadric module
    decimater.add(hModQuadric); // register module
    decimater.initialize();
    decimater.decimate_to(nverts); // nverts = 200
    mesh.garbage_collection();
}
```

Figure 14. Code for mesh simplification

Keyboard functions

Key	Function
←	rotate left
→	rotate right
↑	rotate up
↓	rotate down
Page Up	zoom in

Page Down	zoom out
1	switch between polygon mode and wireframe mode, default polygon
2	switch between normal mesh and simplified mesh, allow 10s for the switching to happen, default normal mesh
space	switch between two-tone rendering and pencil shading, default two-tone
t	enable/disable multi-texturing of pencil shading, default enabled
h	enable/disable silhouette edge, default enabled
c	enable/disable crease edge, default enabled
q	increase silhouette edge thickness silhouette (‘h’) must have been enabled
a	decrease silhouette edge thickness silhouette (‘h’) must have been enabled
w	increase crease edge thickness crease (‘c’) must have been enabled
s	decrease crease edge thickness crease (‘c’) must have been enabled
m	reduce edge gap either silhouette (‘h’) or crease (‘c’) must have been enabled
n	increase edge gap either silhouette (‘h’) or crease (‘c’) must have been enabled
i	increase crease edge threshold crease (‘c’) must have been enabled
d	decrease crease edge threshold crease (‘c’) must have been enabled
ESC	exit program

Reference

1. COSC422 lecture and tutorial materials by Professor Ramakrishnan Mukundan
2. “3D mesh processing and character animation: with examples using OpenGL, OpenMesh and Assimp” by Professor Ramakrishnan Mukundan