3

III

Mobile Toon Shading

Felipe Lira, Felipe Chaves, Flávio Villalva,
Jesus Sosa, Kléverson Paixão
and Teófilo Dutra

3.1 Introduction

Photorealistic rendering is a common goal in the computer graphics industry. This rendering approach intends to produce high quality images through the realistic simulation of light interaction with the environment, and often places a high demand on computational resources. Sometimes, artistic direction or lack of computational resources will motivate developers to use other rendering techniques which are not based on physical illumination models or are not intended to be realistic. Non-photorealistic rendering (NPR) techniques have been widely adopted for giving a painting, drawing, or cartoonish artistic style to 3D rendering.

In this chapter, we describe a unique artistic style through a mobile-ready toon-shading technique. Toon shading is a recurrent NPR technique used in video games for giving a cartoonish aspect to the rendered scenes (Figure 3.1). In order to achieve this, it is necessary to make the 3D scene appear flat, often

Figure 3.1. NPR examples. Left: *The Legend of Zelda: The Wind Waker HD* [Nintendo 2013]; Right: *Guilty Gear Xrd* [Arc System Works 2014].

via nonrealistic illumination models with fewer shading colors and by drawing object outlines. Visual results can be enhanced further by using reprographic techniques such as halftone, which simulate continuous tone imagery through the use of varying sized or spaced geometries (usually dots) in a grid-like pattern.

The idea for the proposed shader arose from the desire to simulate a pop-art style using 3D in real time for games. Initially we tried to make the outline look like a newspaper clipping through the addition of a texture, with overlay blending to simulate the effect. We added particles to complement the outline and give a greater dynamic to the silhouette. For that technique to work nicely, we wanted each animation frame to resemble a paper cut; and then, it was necessary to synchronize each frame with the particle system. However, for animations at usual frame rates (up to 60 fps), the effect turned into a distraction and it was not visually pleasing. We tried to use non-interpolated animations in order to have a stepped animation that would enhance the poses and convey the idea of clipping paper to the screen. Nonetheless, the engine we use forces interpolation between frames by standard and did not allow us to achieve the desired effect. Since we were not able to synchronize the effect with the animation, we abandoned this idea.

Another graphical effect we tried was the use of offset values (exposed in the material), for each color channel of the diffuse texture, in order to simulate a kind of chromatic aberration. The technique works well if used carefully, but very large offsets make UV islands become visible. In the end, these experiments did not end up in the final shader.

In this chapter we describe the individual techniques that comprise our unique artistic style, including a flat shading (Section 3.3), soft light blending (Section 3.4), halftone-based shadows (Section 3.5) and threshold-based inverted hull outline (Section 3.6). We present our implementation of these techniques in Section 3.7 and end by drawing final conclusions in Section 3.8.

3.2 Technique Overview

To achieve a cartoonish artistic style, we used a combination of several different techniques. Toon shading is important in order for the 3D world to appear flat. The transition between light and dark areas has to be hard in order to mimic the shading found in cartoons (Figure 3.2 (left)). Another artistic feature, also observed in cartoons, is the presence of hard outlines (Figure 3.2 (left)). Finally, reproduction of printing patterns (also known as halftone patterns), observed in comics (Figure 3.2 (right)), were the final touch in our artistic style composition.

Our technique starts by pre-processing the model and storing both its normals per face (for flat shading) and normals per vertex (for outline generation). Then, at each iteration of the rendering loop, two passes are used to assemble the desired look. In the first pass, the threshold-based inverted hull outline is generated. Then, in the second pass, the softlight blending equation is used to blend the

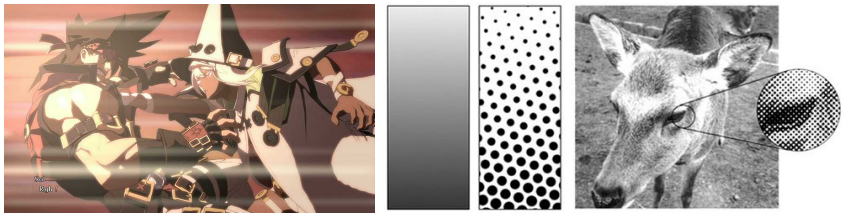

Figure 3.2. Left: rough transition between light and dark areas and hard outlines are features observed in cartoons [Arc System Works 2014]; Right: halftone pattern observed in comics [Scientific American 2012].

albedo with the halftone applied to the flat shading. An overview of the technique is shown in Figure 3.3, and the composition is demonstrated in Figure 3.4.

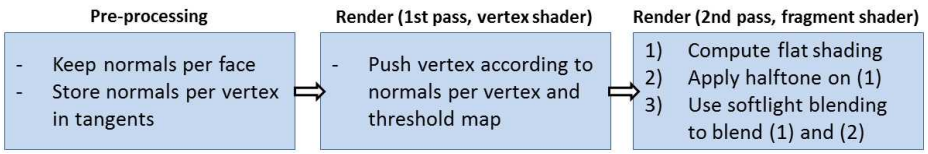

Figure 3.3. Technique overview.

3.3 Flat Shading

The flat shading is achieved through the use of a light-ramp texture in order to define the interaction between light and the surface of the model. The dot product of the surface normal and the light direction is used to sample the ramp texture. The ramp texture provides a hard transition, in our case in three steps, that will be useful for fading our halftone patterns later. Figure 3.4 (b) shows the flat shading obtained using the ramp in Figure 3.5.

3.4 Soft Light Blending

We then use the soft light equation for blending two layers of images. The effect consists of lightning or darkening a base layer according to a blend layer. If a pixel color component in the blend layer is lighter than 50% gray, the correspondent pixel in the base layer will be lighted; otherwise, the pixel is darkened. Mathematically, we have

$$\text{Softlight}(b,a) = \begin{cases} 2ab - a^2(1-2b) & \text{if } b \leq 0.5 \\ 2a(1-b) + \sqrt{a}(2b-1) & \text{if } b > 0.5 \end{cases}$$

(a) Albedo (b) Ramp

(c) Softlight((b), (a)) (d) Halftone((b))

(e) Threshold-based outline (f) Softlight((d), (a)) with outline

Figure 3.4. Technique composition.

Figure 3.5. Ramp texture.

where b is a color component that belongs to the blend layer and a belongs to the base layer. In Figure 3.4 (c), the soft light effect is applied using Figure 3.4 (a) as base layer and Figure 3.4 (b) as blend layer.

3.5 Halftone-based Shadows

We then apply a halftone pattern over the flat shading. In the halftone technique, images are printed using dots with different colors and sizes, and they can be spaced non-uniformly in order to simulate continuously filled areas via an

```
half HalfTone(half RepeatRate , half DotSize , half2 UV)
{
  half size = 1.0 / RepeatRate;
  half2 cellSize = half2(size, size);
  half2 cellCenter = cellSize * 0.5;

  // abs() to avoid negative values after rotation.
  half2 uvlocal = fmod(abs(UV), cellSize);
  half dist = length(uvlocal - cellCenter);
  half radius = cellCenter.x * DotSize;

  // Anti-Aliasing based on differentials
  half threshold = length(ddx(dist) - ddy(dist));

  return smoothstep(dist - threshold, dist + threshold, radius);
}
```

Listing 3.1. Halftone function.

optical illusion. To reproduce the halftone, we define a function based on three parameters: the repeat rate of the dots, the dot size, and a UV position. Our representation is screen-space–based, i.e., the UV passed to the halftone function is the normalized screen position. After the normalization, we are able to rotate the halftone pattern by rotating the UV. The function computes the diameter of the dot and its center position; then, we compute the distance of the UV with respect to the center, and, finally, we perform a smooth step considering that distance and the radius of the dot pattern. The function is shown in Listing 3.1.

The dot size is determined according to the dot product between the normal and light directions, and the result is scaled by a user-defined value. We also define a cutoff threshold for the pattern and a fade distance based on the camera distance. Figure 3.4 (d) shows the halftone pattern applied over the flat shading.

3.6 Threshold-based Inverted Hull Outline

Finally, we need to draw the model's outline. Usually, stencil buffers are used for drawing outlines [Learn OpenGL 2016]. However, since we wanted a mobile-friendly technique, we opted for using the inverted hull outline technique (also know as shell method [Akenine-Möller et al. 2008]) which has a lower computational cost.

The inverted hull outline technique uses an extra pass to render the outline. The outline generation consists of: 1) scaling up the model by pushing its vertices in the direction of their normals; 2) rendering the scaled model with a unique color, without lighting, while culling front faces and showing back faces (first pass); and, 3) rendering the original model (second pass).

This technique works fine if the normals at vertices are the average of the adjacent surface normals. The problem here is that our flat shading needs to use surface normals. The solution we found was pre-processing the model in order

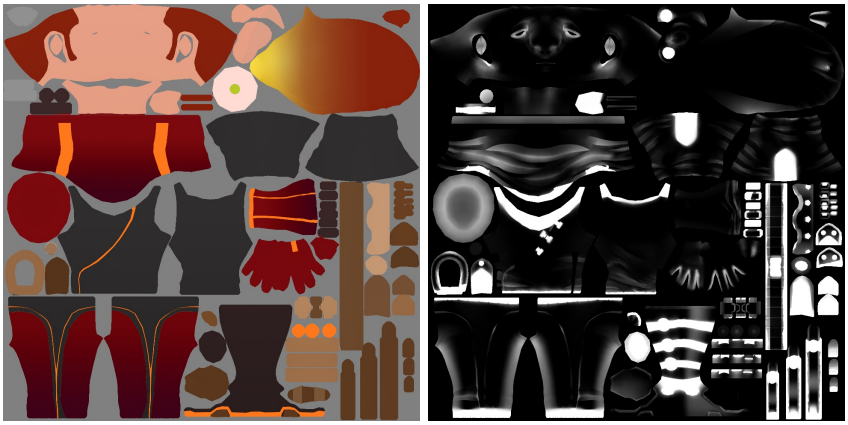

Figure 3.6. Diffuse map (left) and threshold map (right).

to store the averaged normals at the tangents (since we are not using them). We also use a threshold map (Figure 3.6 (right)) to control the scale applied to each vertex. Figure 3.4 (e) shows the outline generated for our example model and Figure 3.4 (f) shows our final composition.

3.7 Implementation

The two passes used to compose the result can be seen in Listing 3.2. An emission map is used in the end of the second pass to highlight some parts of the diffuse map.

```

Pass {
  Name "Outline"
  Cull Front
  ZWrite Off
  ...
  CGPROGRAM
  ...

  VertexOutput vert(VertexInput v) {
    VertexOutput o = (VertexOutput)0;
    half thicknessOffset = tex2Dlod(_OutlineThicknessMap,
        half4(v.texcoord.xy, 0.0, 0.0)).r;
    half outlineWidth = clamp(thicknessOffset * _OutlineScale,
        _OutlineMinOffset, _OutlineMaxOffset);
    o.pos = mul(UNITY_MATRIX_MVP, half4(v.vertex.xyz +
        (v.normal * outlineWidth), 1.0));
    return o;
  }
  ... // frag shader is simple pass through.
  ENDCG
}

```

```
Pass {
  Name "Halftone"
  ...
  CGPROGRAM
  ... // vert shader is simple pass through.

  fixed4 frag(VertexOutput i) : COLOR {
    half2 screenUV = RotateUV(NormalizeScreenSpace(i.screenPos),
        _HtPatternAngle);
    half camDist = length(i.posWorld - _WorldSpaceCameraPos);
    half htAlphaFade = smoothstep(_MaxFade, _MinFade, camDist);

    // Apply Lambert Light term using a lightRamp
    half3 diffuseMap = tex2dLinear(_DiffuseMap,
        TRANSFORM_TEX(i.uv0, _DiffuseMap)).rgb;

    half NdL = saturate(dot(normalize(i.normalDir),
        normalize(_WorldSpaceLightPos0.xyz)));
    half lbtn = tex2dLinear(_LightRamp, half2(NdL, 0)).r * 0.5;
    half htCutoff = step(_HtCutoff, (1.0 - NdL) * _HtScale);
    half ht = HalfTone(_RepeatRate, (1.0 - NdL) * _HtScale,
        screenUV) * htCutoff * htAlphaFade;
    half3 htColor = lerp(half3(lbtn, lbtn, lbtn), _HtColor, ht);

    half4 emissionMap = tex2dLinear(_EmissionMap,
        TRANSFORM_TEX(i.uv0, _EmissionMap));
    half3 result = Softlight(htColor, diffuseMap) + diffuseMap *
        emissionMap.rgb * _EmissionIntensity;

    return gammaConvert(fixed4(result, 1));
  }
  ENDCG
}
```

Listing 3.2. Two passes used for composing the proposed artistic style.

3.8 Final Considerations

We presented a cartoonish shading composed of different NPR techniques. With this shading, we are able to reproduce the art style seen in comics or cartoons. Moreover, it is mobile-friendly and simple to implement. Despite the positive results achieved, there is still room for improvement. The halftone pattern, for example, is screen-space–based, and that can lead to unpleasant visual artifacts depending on the camera motion or the motion of characters in the scene. This is especially noticeable in VR where the user has more control over the camera. The outline can also present artifacts, such as spikes or holes, depending on the scale applied and the model shape (for example, concavities can lead to self-intersection).

Bibliography

Akenine-Möller, T., Haines, E., and Hoffman, N. 2008. Non-photorealistic rendering. In *Real-Time Rendering 3rd Edition*. A K Peters, Ltd., Natick, MA, USA, 1045.

Arc System Works, 2014. Guilty Gear Xrd. Available at: http://guiltygear.us/ggxrds/media.

Learn OpenGL, 2016. Object outlining. Available at: http://www.learnopengl.com/#!Advanced-OpenGL/Stencil-testing.

Nintendo, 2013. The Legend of Zelda: The Wind Waker HD. Available at: http://www.nintendo.com/games/detail/the-legend-of-zelda-the-wind-waker-hd-wii-u.

Scientific American, 2012. Dots, spots, and pixels: what's in a name? Available at: http://blogs.scientificamerican.com/symbiartic/dots-spots-and-pixels-whats-in-a-name/.