Layered Car Paint Shader

Chris Oat
ATI Research

Natalya Tatarchuk
ATI Research

John Isidoro
ATI Research

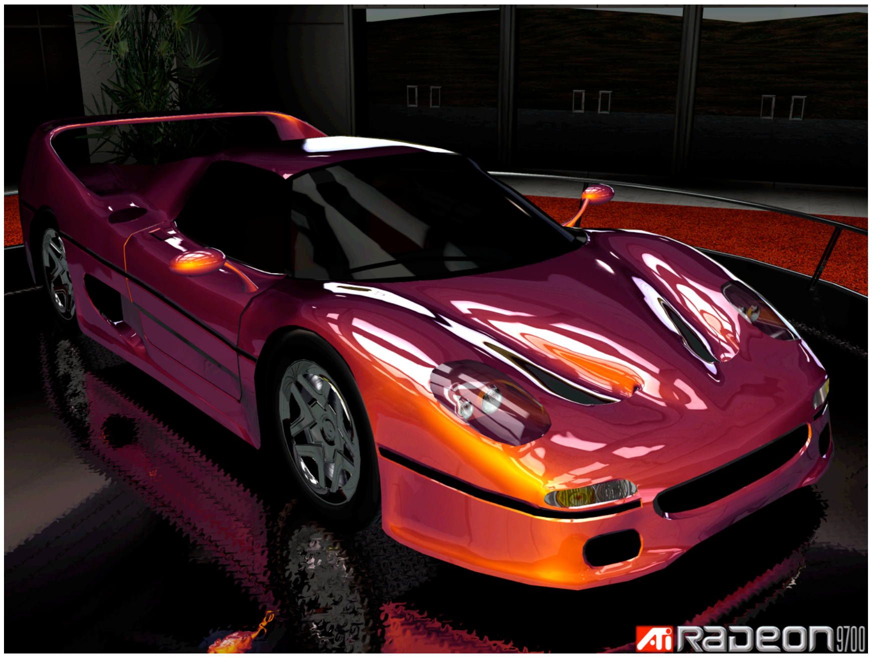

Figure 1: Two-tone, suspended micro-flake car paint rendered in real-time using an HLSL Pixel Shader in DirectX9.0.

The application of paint to a car's body can be a complicated process. Expensive auto body paint is usually applied in layered stages and often includes dye layers, clear coat layers, and metallic flakes suspended in enamel. The result of these successive paint layers is a surface that exhibits complex light interactions, giving the car a smooth, glossy and sparkly finish. The car model shown here uses a relatively low number of polygons but employs a high precision normal map generated by an appearance preserving simplification algorithm (visit http://www.ati.com/developer/ for more information on the ATI Normal Mapper Tool). Due to the pixel shader operations performed across the smoothly changing surfaces (such as the hood of the car), a 16-bit per channel normal map is necessary.

Normal Map Decompression

The first step in this pixel shader is normal decompression. Since the normals are stored in surface local coordinates (*a.k.a.* tangent space), we can assume that the z component of the normals will be positive. Thus, we can store x and y in two channels of a 16-16 texture map and derive z in the pixel shader from $+\text{sqrt}(1 - x^2 - y^2)$. This gives us much higher precision than a traditional 8-8-8-8 normal map (even 10 or 11 bits per channel is not enough for this particular shader) for the same memory footprint.

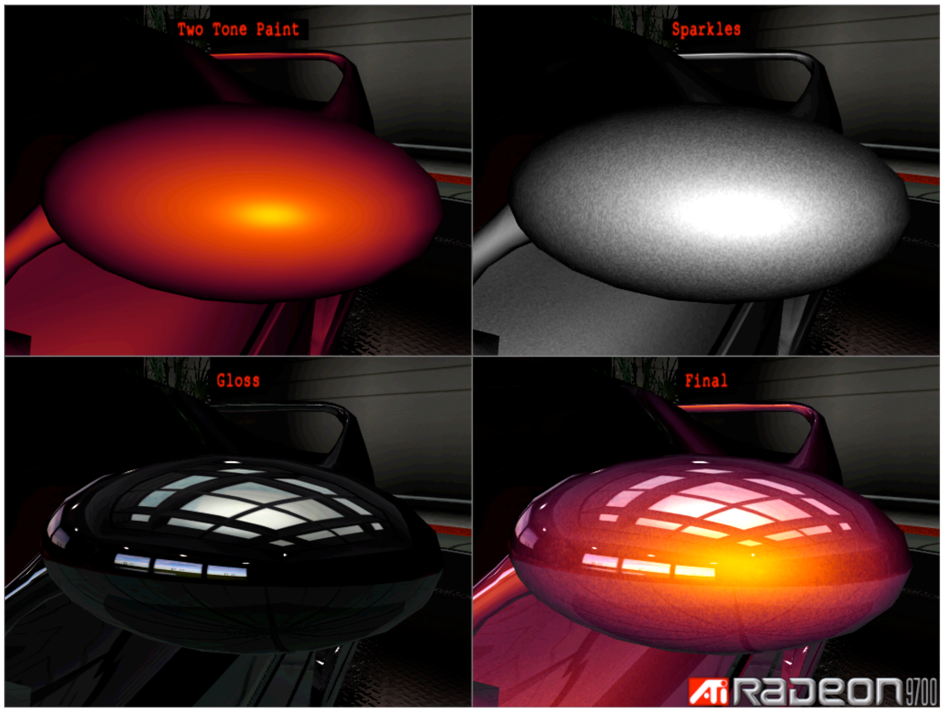

Figure 2: Two-tone, micro-flake, clear coat and final lighting on side rear-view mirror.

Base Color

The normal decompression described above is performed on a surface normal map which is generated from an appearance preserving simplification process (N) and a high frequency normalized vector noise map (N_n) which is repeated across the surface. These two normals are used to compute two perturbed normals which are used to simulate the two-toned nature of the paint as well as the microflake suspended in an inner coat of the paint.

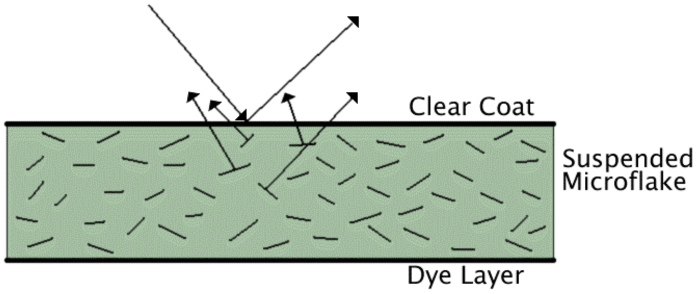

Figure 3: Metallic micro-flakes, suspended in clear enamel, are applied over a base paint coat (dye layer) and results in subsurface light scattering.

These normals, N_s and N_{ss} are computed as follows:

$$N_s = \frac{aN_n + bN}{|aN_n + bN|} \text{ where } a < b$$

$$N_{ss} = \frac{cN_n + dN}{|cN_n + dN|} \text{ where } c = d$$

The coefficients *a*, *b*, *c* and *d* above are constant input parameters to the pixel shader which determine the distributions of the perturbed normals. The magnitude of these perturbed normals determines the width of the region in which the micro-flake is readily visible. The two normals are dotted with the view vector and used as parameters in the following polynomial, which determines the color of the base coat and strength of the micro-flake term:

$$c_0(N_s \bullet V) + c1(N_s \bullet V)^2 + c2(N_s \bullet V)^4 + c3(N_{ss} \bullet V)^{16}$$

The first three terms of this polynomial perform the blend between the two tones of the paint. The fourth term adds an extra layer of sparkle for the micro-flake's contribution. Constants *c*0, *c*1, and *c*2 correspond to the base paint colors while *c*3 corresponds to the micro-flake color.

Clear Coat Paint Layer

The final step in rendering the painted areas of the car is the inclusion of the clear coat through the addition of an environment map as shown below.

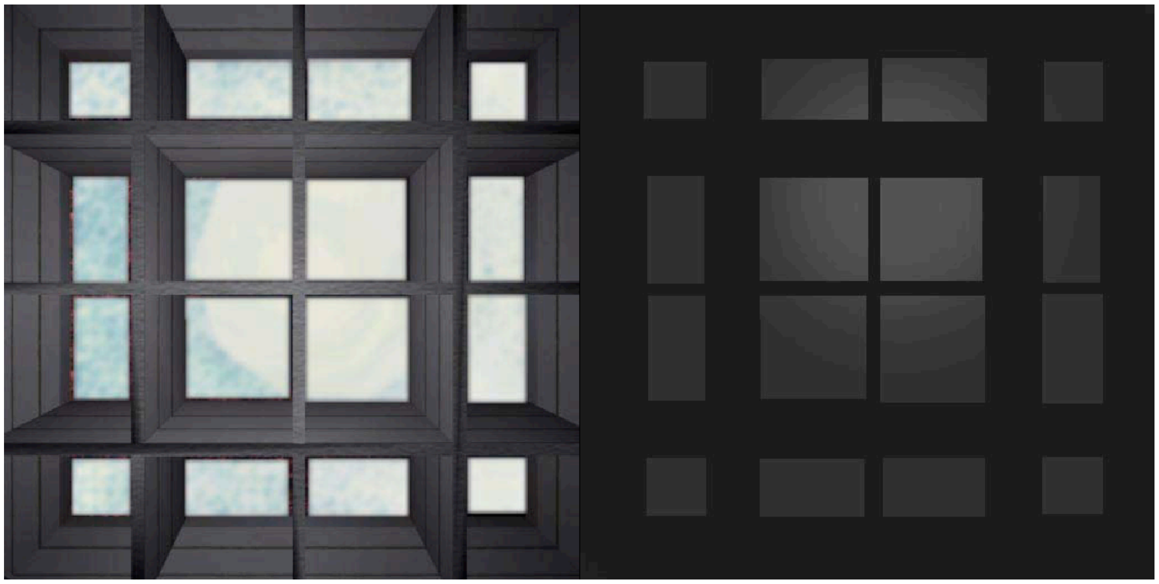

Figure 4: Top face of HDR cubic environment map (left) RGB Channels (right) Alpha Channel.

One interesting aspect of the clear coat term is the decision to store the environment map in an RGBScale form to simulate high dynamic range in a low memory footprint. The alpha channel of the texture, shown on the right in figure 4, represents 1/16th of the true range of the data while the RGB, shown on the left, represents the normalized color. In the pixel shader, the alpha channel and RGB channels are multiplied together and multiplied by eight to reconstruct a cheap form of HDR reflectance. This is multiplied by a subtle Fresnel term before being added to the lighting terms described above.

The full HLSL pixel shader for the car paint and trim is shown below.

```
struct PsInput
{
   float2 Tex        : TEXCOORD0;
   float3 Tangent    : TEXCOORD1;
   float3 Binormal   : TEXCOORD2;
   float3 Normal     : TEXCOORD3;
   float3 View       : TEXCOORD4;
   float3 SparkleTex : TEXCOORD5;
};

float4 main(PsInput i) : COLOR
{
   // fetch from the incoming normal map:
   float3 vNormal = tex2D( normalMap, i.Tex );

   // Scale and bias fetched normal to move into [-1.0, 1.0] range:
   vNormal = 2.0f * vNormal - 1.0f;

   // Micro-flakes normal map is a high frequency normalized
   // vector noise map which is repeated across the surface.
   // Fetching the value from it for each pixel allows us to
   // compute perturbed normal for the surface to simulate
   // appearance of micro-flakes suspended in the coat of paint:
   float3 vFlakesNormal = tex2D(microflakeNMap, i.SparkleTex);

   // Don't forget to bias and scale to shift color into [-1.0, 1.0] range:
   vFlakesNormal = 2 * vFlakesNormal - 1.0;

   // This shader simulates two layers of micro-flakes suspended in
   // the coat of paint. To compute the surface normal for the first layer,
   // the following formula is used:
   //    Np1 = ( a * Np + b * N ) /  || a * Np + b * N || where a << b
   //
   float3 vNp1 =
          microflakePerturbationA * vFlakesNormal + normalPerturbation * vNormal ;

   // To compute the surface normal for the second layer of micro-flakes, which
   // is shifted with respect to the first layer of micro-flakes, we use this formula:
   //    Np2 = ( c * Np + d * N ) / || c * Np + d * N || where c == d
   float3 vNp2 = microflakePerturbation * ( vFlakesNormal + vNormal ) ;

   // The view vector (which is currently in world space) needs to be normalized.
   // This vector is normalized in the pixel shader to ensure higher precision of
   // the resulting view vector. For this highly detailed visual effect normalizing
   // the view vector in the vertex shader and simply interpolating it is insufficient
   // and produces artifacts.
   float3 vView =  normalize( View );

   // Transform the surface normal into world space (in order to compute reflection
   // vector to perform environment map look-up):
   float3x3 mTangentToWorld = transpose( float3x3( Tangent, Binormal, Normal ) );
   float3   vNormalWorld    = normalize( mul( mTangentToWorld, vNormal ));

   // Compute reflection vector resulted from the clear coat of paint on the metallic
   // surface:
   float  fNdotV      = saturate(dot( vNormalWorld, vView));
   float3 vReflection = 2 * vNormalWorld * fNdotV - vView;

   // Here we just use a constant gloss value to bias reading from the environment
   // map, however, in the real demo we use a gloss map which specifies which
   // regions will have reflection slightly blurred.
   float fEnvBias = glossLevel;

   // Sample environment map using this reflection vector:
   float4 envMap = texCUBEbias( showroomMap, float4( vReflection, fEnvBias ) );

   // Premultiply by alpha:
   envMap.rgb = envMap.rgb * envMap.a;

   // Brighten the environment map sampling result:
   envMap.rgb *= brightnessFactor;

   // Compute modified Fresnel term for reflections from the first layer of
   // microflakes. First transform perturbed surface normal for that layer into
   // world space and then compute dot product of that normal with the view vector:
   float3 vNp1World = normalize( mul( mTangentToWorld, vNp1) );
   float  fFresnel1 = saturate( dot( vNp1World, vView ));

   // Compute modified Fresnel term for reflections from the second layer of
   // microflakes. Again, transform perturbed surface normal for that layer into
   // world space and then compute dot product of that normal with the view vector:
   float3 vNp2World = normalize( mul( mTangentToWorld, vNp2 ));
   float  fFresnel2 = saturate( dot( vNp2World, vView ));

   // Combine all layers of paint as well as two layers of microflakes
   float  fFresnel1Sq = fFresnel1 * fFresnel1;

   float4 paintColor = fFresnel1   * paintColor0 +
                       fFresnel1Sq * paintColorMid +
                       fFresnel1Sq * fFresnel1Sq * paintColor2 +
                       pow( fFresnel2, 16 ) * flakeLayerColor;

   // Combine result of environment map reflection with the paint color:
   float  fEnvContribution = 1.0 - 0.5 * fNdotV;

   float4 finalColor;
   finalColor.a = 1.0;
   finalColor.rgb = envMap * fEnvContribution + paintColor;
   return finalColor;
}
```

Conclusion

This shader was developed using empirically gathered phenomenological illumination characteristics rather than actual physical material attributes. Many different car paint swatches were observed under various lighting conditions, this shader strives to reproduce the observed characteristics of those swatches. This chapter demonstrates a compelling simulation for the illumination of car paint using a real-time pixel shader.