



# SDK White Paper

## Fresnel Reflection

WP-01401-001-v01  
July 2004

**NV**SDK

---

## About Fresnel Reflections

When light strikes a material boundary, Fresnel reflection describes how much light reflects at that boundary versus how much refracts and transmits. Fresnel reflection occurs commonly in nature and is thus important for realistic real-time graphics. This paper describes how to implement Fresnel reflection efficiently on DirectX 8.1 hardware such as NVIDIA's GeForce 3. It provides the source for vertex and pixel shaders implementing various approximations, shows the resulting screenshots, and briefly discusses how to choose the most appropriate approximation.

Matthias Wloka  
[mwloka@nvidia.com](mailto:mwloka@nvidia.com)

NVIDIA Corporation  
2701 San Tomas Expressway  
Santa Clara, CA 95050

# Fresnel Reflection

---

## Introduction

When light strikes a material boundary, for example, crossing from air into glass, only some of the light transmits into the new material; some light reflects at the boundary. The name for this effect is Fresnel reflection. Fresnel reflection is most visible when viewing semi-transparent material such as water, window-glass, skin, or car-paint. Fresnel reflection also occurs when viewing opaque materials such as metal or paper.

The commonness of Fresnel reflection makes it important for real-time rendering. This paper explores how to render Fresnel reflection on DirectX 8.1 graphics hardware, GeForce 3 and later.

The discussion of Fresnel reflection here is restricted to single-material boundaries, specifically, single boundaries of less dense to more dense materials. Thus, correct rendering of window panes requires more work, since light hitting a window typically traverses two boundaries: air-glass upon entering and glass-air on exiting. This paper covers the correct rendering of a lake, for example, since a lake presents only a single, less-to-more dense boundary (air-water), as long as the viewer is above water. If the viewer is underwater, the material boundary is more-to-less dense (water-air) and beyond the scope of this paper. The reader may want to consult a textbook on Optics for ideas on how to render these cases; see Eugene Hecht, "Optics," Addison-Wesley, 1987, pp 94-104.

## Fresnel's Formula

Figure 1 depicts the scenario that this paper addresses—material **i** has an index of refraction  $n_i$  which is less than  $n_t$ , the index of refraction for material **t**. The angle  $\theta$  ranges from zero, when the ray of light is normal to the surface, to  $\theta = \pi/2$ , when the ray of light is incident to the surface.

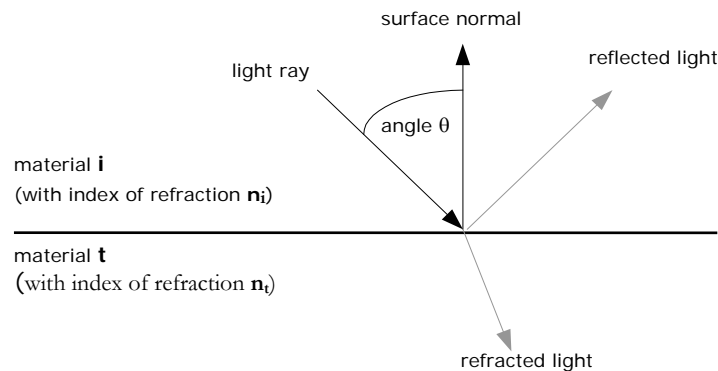


Figure 1. A ray of light traveling through material **i** and striking denser material **t**.

Fresnel's formula describes how much light reflects at the material boundary. The amount of reflection depends on the angle of incidence  $\theta$ , the polarization of the light, the ratio of indices of refraction  $n_t/n_i$ , and since the index of refraction depends on wavelength, the light's wavelength. The formula is:

$$\text{Equation 1. } \mathbf{R}(\theta) = \frac{1}{2} (\mathbf{R}_{\perp}(\theta) + \mathbf{R}_{\parallel}(\theta))$$

$\mathbf{R}_{\perp}$  and  $\mathbf{R}_{\parallel}$  describe the reflectance for light polarized perpendicular to the plane of incidence and parallel to it, respectively:

$$\text{Equation 2. } \theta_t = \arcsin(n_i/n_t \sin(\theta))$$

$$\text{Equation 3. } \mathbf{R}_{\perp}(\theta) = \sin^2(\theta - \theta_t) / \sin^2(\theta + \theta_t)$$

$$\text{Equation 4. } \mathbf{R}_{\parallel}(\theta) = \tan^2(\theta - \theta_t) / \tan^2(\theta + \theta_t)$$

Since  $\mathbf{R}_{\perp}$  and  $\mathbf{R}_{\parallel}$  are undefined for  $\theta = 0$ , the following limit applies:

$$\text{Equation 5. } \mathbf{R}_{\perp}(0) = \mathbf{R}(0) = (n_i - n_t)^2 / (n_i + n_t)^2$$

Figure 2 plots Fresnel's formula given by Equation 1 as well as its two components  $R_{\perp}$  and  $R_{\parallel}$ , given by Equations 3 and 4 respectively. Figure 3 shows a family of curves for various indices of refraction.

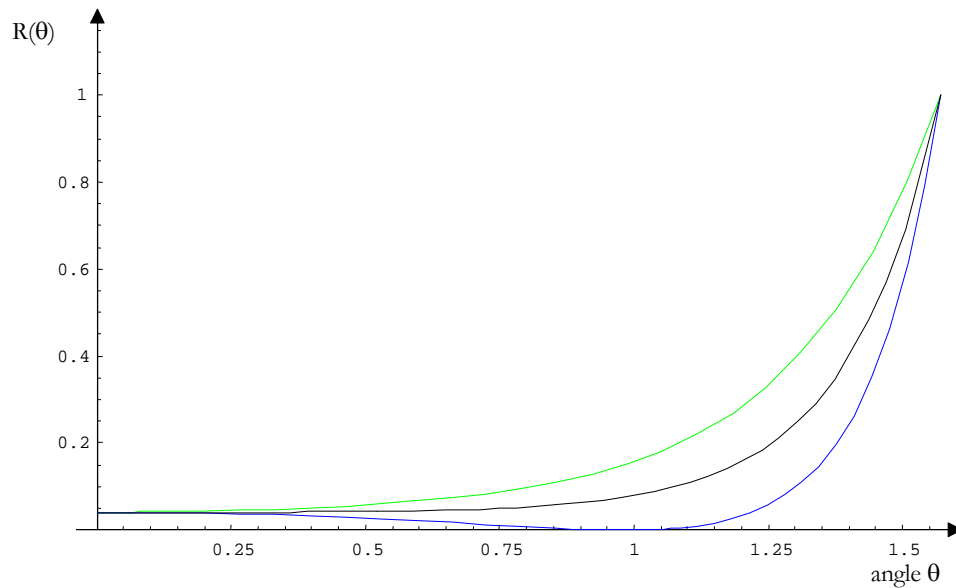


Figure 2. The graphs for  $R$  (black, center),  $R_{\perp}$  (green, top), and  $R_{\parallel}$  (blue, bottom).

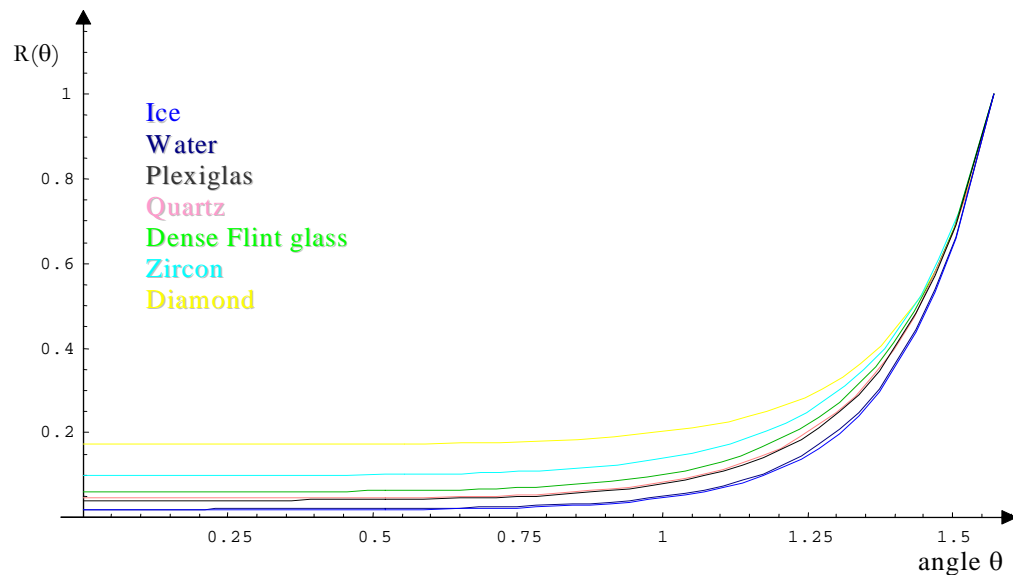


Figure 3. Fresnel's formula for a variety of indices of refraction.

Substituting Equations 2 through 4 into Equation 1 yields the following:

Equation 6.

$$R(\theta) = \frac{1}{2} \left( \frac{\sin^2(\theta - \theta_t)}{\sin^2(\theta + \theta_t)} \right) \left( 1 + \frac{\cos^2(\theta + \theta_t)}{\cos^2(\theta - \theta_t)} \right)$$

Define **c** and **g** as shown in Equations 6 and 7.

$$\text{Equation 7. } \mathbf{c} = \cos(\theta) n_i/n_t$$

$$\text{Equation 8. } \mathbf{g} = \sqrt{1 + \mathbf{c}^2 - (n_i/n_t)^2}$$

Substituting **c** and **g** into Equation 6 simplifies the formula to:

Equation 9.

$$R(\theta) = \frac{1}{2} \left( \frac{\mathbf{g} - \mathbf{c}}{\mathbf{g} + \mathbf{c}} \right)^2 \left( 1 + \left[ \frac{\mathbf{c}(\mathbf{g} + \mathbf{c}) - (n_i/n_t)^2}{\mathbf{c}(\mathbf{g} - \mathbf{c}) + (n_i/n_t)^2} \right]^2 \right)$$

---

## Rendering with Fresnel's Formula

For DirectX 8.1 hardware, computing Fresnel's formula is possible on either a per-vertex or a per-pixel basis. The choice largely depends on whether the surface normals of a model are specified per-vertex or per-pixel. If specified per-pixel, then Fresnel's formula needs to be computed per-pixel. If specified per-vertex, computing Fresnel's formula per-vertex is sufficient. Computing the formula per-pixel is only insignificantly more accurate than using the interpolated per-vertex quantities (assuming reasonable tessellation). The following pages have screenshots and more details.

Choosing to compute Fresnel's formula per-pixel, even though normals are specified per-vertex, may make sense, if an application is heavily vertex-processing bound. In that case, offloading per-vertex work to a per-pixel level rebalances the rendering pipeline and increases overall rendering throughput.

The following approximations greatly improve rendering efficiency:

- ❑ Assume all light is non-polarized.
- ❑ Assume all light is of the same wavelength.

Neither one of these assumptions is generally true. For example, skylight is strongly polarized, and reflected light is generally multi-colored. These are good approximations nonetheless, since their effect on Fresnel's formula is small. This paper assumes these approximations.

## Per-Vertex

While encoding Equation 9 as a DirectX 8.1 vertex-shader is possible, it is also highly inefficient due to the required number of instructions. Approximating Equation 9 yields:

$$\text{Equation 10. } \mathbf{R}(\theta) \approx \mathbf{R}_a(\theta) = \mathbf{R}(0) + (1-\mathbf{R}(0)) (1-\cos(\theta))^5$$

This approximation yields good results; and it introduces less error than the above non-polarized light approximation. Figure 4 compares Equation 9 to its approximation—Equation 10, as well as to a more simplistic approximation to  $(1-\cos(\theta))$ .

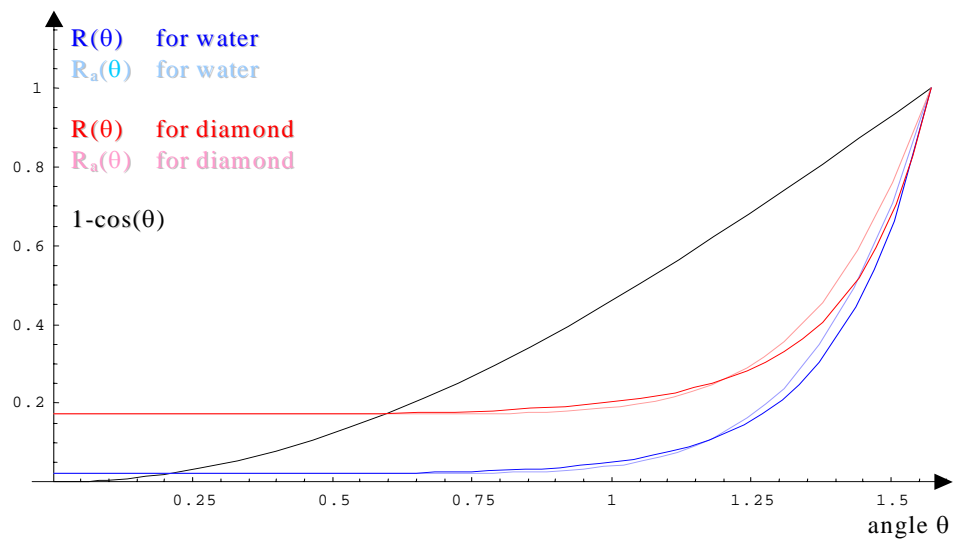


Figure 4. Fresnel reflection and its approximations for indices of refraction of water and diamond.

Figure 5 lists a HLSL/Cg code fragment that implements Equation 10. Supplying  $R(0)$  to the shader instead of the refraction-index ratio, avoids computing the constant  $R(0)$  on the fly and therefore shortens and optimizes the shader. Figure 6 shows the VS 1.0 assembly for the same vertex-shader fragment. Figure 7 shows a result of using this vertex shader approximation.

```
float fresnel(float3 light, float3 normal, float R0)
{
    // Note: compute R0 on the CPU and provide as a
    // constant; it is more efficient than computing R0 in
    // the vertex shader. R0 is:
    // float const R0 = pow(1.0-refractionIndexRatio, 2.0)
    //                / pow(1.0+refractionIndexRatio, 2.0);
```

```

// light and normal are assumed to be normalized
return R0 + (1.0-R0) * pow(1.0-dot(light, normal), 5.0);
}

```

Figure 5. HLSL/Cg shader fragment implementing Fresnel's formula per-vertex.

```

; c[0] contains [R(0), 1-R(0), 0, 1]
; r0  contains normalized surface normal
; r1  contains normalized direction from light
vs.1.0
dp3 r0.w,  r0,    r1
add r0.w,  c[0].w, -r0.w
mul r1.w,  r0.w,  r0.w    // squared
mul r1.w,  r1.w,  r1.w    // quartic
mul r1.w,  r1.w,  r0.w    // quintic
mad oD0.a, r1.w,  c[0].y, c[0].x

```

Figure 6. Vertex shader assembly implementing Fresnel's formula per-vertex.



Figure 7. Screenshot of per-vertex Fresnel reflection.

## Per-Pixel

The per-vertex polynomial approximation used in Equation 10 also applies to computing Fresnel's formula per-pixel. Figure 8 shows a HLSL/Cg pixel shader that implements this per-pixel approximation. Figure 9 lists the corresponding ps.1.1 assembly code and Figure 12 shows a screenshot of the effect.

Using a texture to encode Fresnel's formula is another way to compute it per-pixel. The pixel shader computes  $\mathbf{N} \cdot \mathbf{L} = \cos(\theta)$  and passes the result as a texture coordinate for a dependent read on a texture encoding  $\mathbf{R}(\arccos(x))$ . Figure 10 shows a HLSL/Cg pixel shader implementing this technique. Figure 11 lists corresponding ps.1.3 assembly code and Figure 13 shows a resulting screenshot.



This last approach is the most accurate. Only texture resolution introduces potential computation errors. Visually comparing the two pixel-shader solutions shows little difference. Therefore, choosing between these two solutions depends more on a particular shader having texture stages or register combiners available for implementing Fresnel's formula.

```
float fresnel(float3 light, float3 normal, float R0)
{
    float const cosAngle = 1-saturate(dot3(light, normal));

    float result = cosAngle * cosAngle;
    result       = result * result;
    result       = result * cosAngle;
    result       = saturate(mad(result, 1-saturate(R0), R0));

    return result;
}
```

Figure 8. HLSL/Cg shader fragment implementing Fresnel's formula per-pixel through register-combiner math.

```
ps.1.1
def c0, 1.0, 0.0, 0.0, R(0)

tex t0          // normal map
texm3x3pad t1, t0 // reflect eye-vector around
texm3x3pad t2, t0 // normal in t0 and use result
texm3x3vspec t3, t0 // to look up reflection color

// dot eye-vector with per-pixel normal from t0
dp3_sat r1.rgba, v0_bx2, t0

// run Fresnel approx. on it: R0 + (1-R0) (1-cos(q))^5
mul r0.a, 1-r1.a, 1-r1.a // squared
mul r0.a, r0.a, r0.a // quartic
mul r0.a, r0.a, 1-r1.a // quintic
mad r0.a, r0.a, 1-c0.a, c0.a // r0.a is Fresnel factor
lrp r0, r0.a, t3, v1 // blend based on Fresnel
```

Figure 9. Pixel shader assembly implementing Fresnel's formula per-pixel through register-combiner math.

```
fragout main( vpconn IN,
              uniform samplerCUBE EnvironmentMap,
              uniform sampler2D NormalMap,
              uniform sampler2D FresnelFunc)
{
    fragout OUT;

    float3 environColor = texCUBE(EnvironmentMap).xyz;
    float4 normal = 2*(tex2D(NormalMap)-0.5);
    float fresnelValue = saturate(tex2D_dp3x2(FresnelFunc,
                                              IN.TexCoord2,
                                              normal).w);
```

```

float3 litColor      = IN.Color.xyz + IN.Specular.xyz;

OUT.col.xyz = lerp(litColor, environColor, fresnelValue);
return OUT;
}

```

Figure 10. HLSL/Cg shader fragment implementing Fresnel's formula per-pixel through a texture lookup.

```

ps.1.3

tex    t0      // model color-map
tex    t1      // reflection color
tex    t2      // normal map
texdp3 t3, t2 // Fresnel look-up function R(arccos(x))

// t3.a is Fresnel: blend between no and full reflection
mad t0,      t0,  v0, v1
lrp r0.rgb, t3.a, t1, t0

```

Figure 11. Pixel shader assembly implementing Fresnel's formula per-pixel through texture lookup.



Figure 12. Screenshot of Fresnel reflection using per-pixel register-combiner math.



Figure 13. Screenshot of Fresnel reflection using per-pixel texture lookups.

## Results

Figures 14 and 15 show screenshots of the same scene without Fresnel reflection and with the simplistic  $1-\cos(\theta)$  approximation, respectively. The effect “Fresnel Reflection” generated these screenshots. It is available for download from: <http://developer.nvidia.com/view.asp?PAGE=nvSDK>.

This effect implementation is invaluable for evaluating visual differences between the various approximations described here. In particular, it should clarify that adding Fresnel-based reflection adds important visual detail. Furthermore, the simplistic  $1-\cos(\theta)$  approximation is visually unacceptable. Better approximations are available at little to no added performance expense. Finally, the per-vertex and per-pixel approximations are visually distinguishable only through A/B-style comparisons.



Figure 14. Screenshot of the test scene without Fresnel reflection.



Figure 15. Screenshot approximating Fresnel reflection through  $1-\cos(\theta)$ .

# Appendix: Indices of Refraction

Table 1. Selected indices of refraction.

Material	Index of Refraction
Vacuum	1.0 (min. index of refraction)
Air	1.000293
Ice	1.31
Water	1.333333
Ethyl Alcohol	1.36
Fluorite	1.43
Poppy Seed Oil	1.469
Olive Oil	1.47
Linseed Oil	1.478
Plexiglas	1.51
Immersion Oil	1.515
Crown Glass	1.52
Quartz	1.54
Salt	1.54
Light Flint Glass	1.58
Dense Flint Glass	1.66
Tourmaline	1.62
Garnet	1.73-1.89
Zircon	1.923
Cubic Zirconia	2.14-2.20
Diamond	2.417
Rutile	2.907
Gallium Phosphide	3.5 (max. index of refraction)



## Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

## Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

© 2004 NVIDIA Corporation. All rights reserved



**NVIDIA.**

NVIDIA Corporation  
2701 San Tomas Expressway  
Santa Clara, CA 95050  
[www.nvidia.com](http://www.nvidia.com)