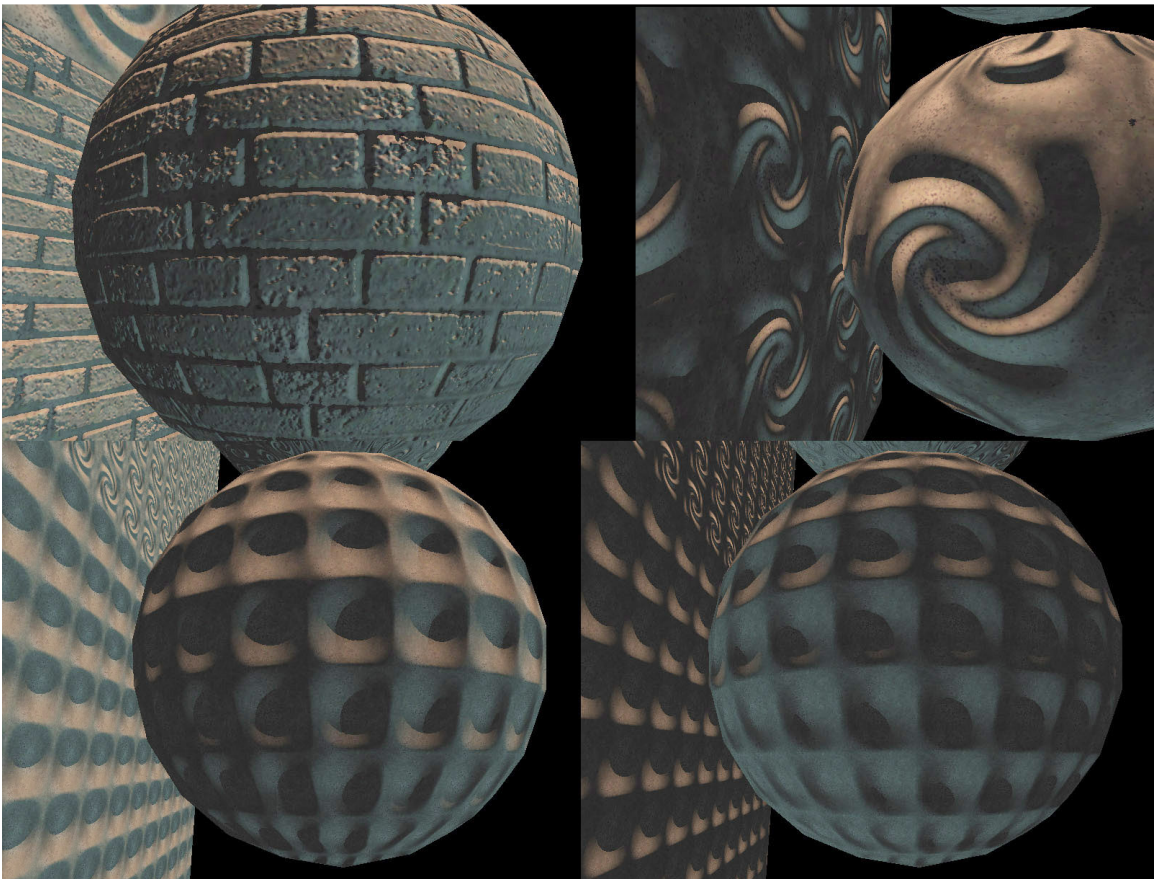


# Dot3 Self Shadow on PS.1.1 and Xbox

By Danny Oros  
frost@frost-art.com

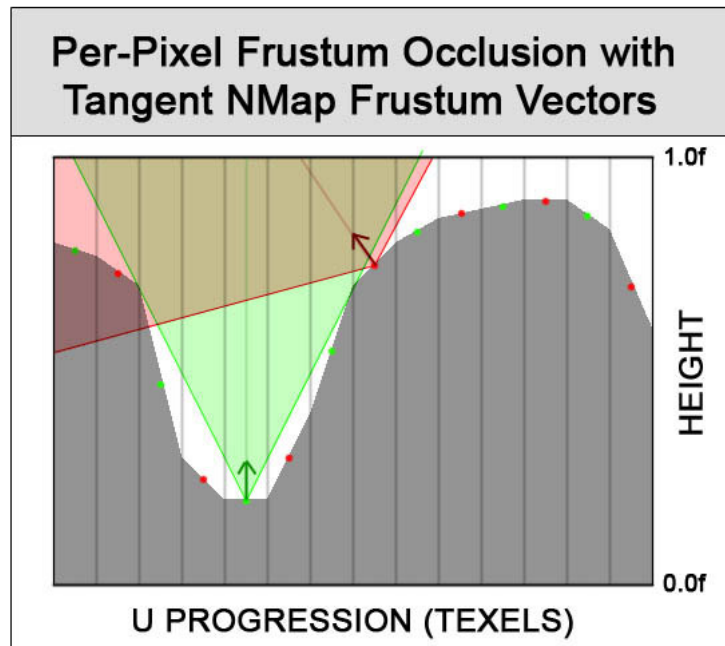
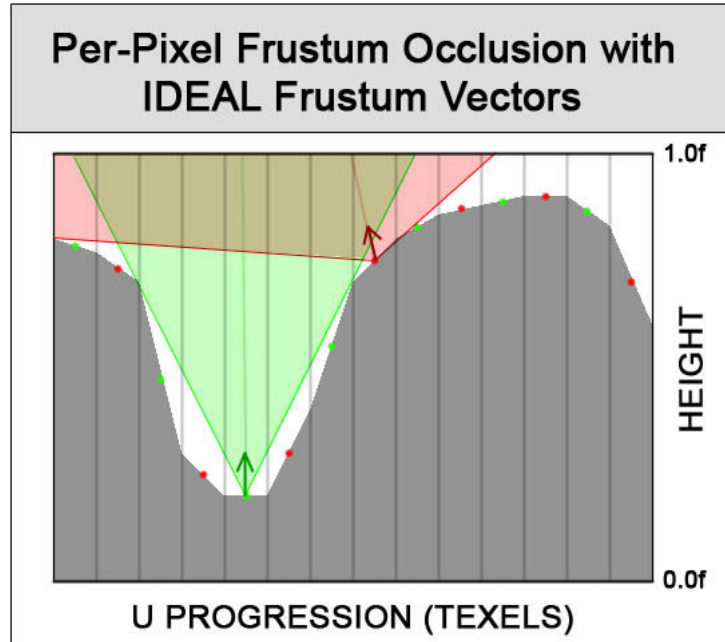
This document describes a simple way to perform per-pixel dot3 self shadowing on very low end programmable pixel hardware such as the Xbox and upwards.



The idea behind my approach is rather simple. The basic idea is that each pixel in the tangent space normal map has an FOV frustum that determines the aperture into which it can receive light. Anything outside that FOV frustum consists of an obstruction occluding the light from reaching the specified pixel.

These frustums are ideally centered along the vector that best fits the center of the opening, with the frustum size expanding to the extents of this opening each of equal distance from that center frustum vector. We will call this the IDEAL frustum vector.

However, in our PS.1.1 example, we will assume the tangent space normal texel as being our frustum vector (for memory and GPU processing considerations), which more or less (mostly less =>) describes the occlusion frustum for our pixel.



Seeing as we are basing ourselves on the tangent space normal texel for our per-texel frustum vector, all we need to do is scale our FOV frustum to fit the aperture size of our opening on this pixel.

This is achieved by having a single scalar define this opening. This is of course another factor in having incorrect results, as our FOV frustum is assumed to be of 1:1 aspect ratio between U and V (tangent and binormal) opening. Ideally, we would have many occlusion and vector samples based on axial occlusion tests, but, this is way too hardcore for our ps.1.1 generation hardware as well as for memory consumption.

All we need to do in our pixel shader is BIAS the per-pixel dp3 result by our FOV frustum aperture scalar (height), and we get a per-pixel self-shadowing result. This will attenuate the per-pixel lighting and allow us to perform a conditional (CND) to cut out the pixels that are in shade (values less than 0.5f).

Here is a sample pixel shader (PS.1.1) to perform this Dot3 Self Shadowing technique:

```
=====
;
; SHADER: DOT3SELSHADOW.PS
;
;
; DESCRIPTION:
;   Very simplified per-pixel lit self-shadowing pixel shader for use
;   on Xbox and other low-end pixel shader versions (ps.1.1).
;
; HISTORY:
;   2004/04/16: Initial implementation, by Danny Oros.
;
; CONSTANTS:
;   c0.rgb = ambient RGB
;   c0.a   = 0.0f
;   c1.rgb = light color RGB
;   c1.a   = n/a
;
; INPUT:
;   t0      = tangent space normal map :: RGB=nmap A=depth/occlusion
;   t1      = diffuse texture :: RGB=diffuse
;   v0.rgb  = tangent space from VS
;   v0.a    = n/a
;   v1.rgb  = n/a
;   v1.a    = n/a
;
; OUTPUT:
;   r0.rgb  = final pixel
;   r0.a    = n/a
;
=====

; SHADER MODEL VERSION
;
ps.1.1

; TEX OPS
;-----
tex t0      // tangent space normal map :: RGB=nmap A=depth/occlusion
tex t1      // diffuse texture :: RGB=diffuse
```

```

; PIXEL OPS
;-----

; PERFORM BASIC PER PIXEL DP3
;-----
; dp3_result = dp3 ( ((tex_nmap - 0.5f) * 2), ((tangent_space - 0.5f) * 2))
;-----

dp3 r1, t0_bx2, v0_bx2

; ESTIMATE PER PIXEL OCCLUSION FACTOR
;-----
; occlusion_factor = dp3_result * tangent_space.blue
; occlusion_factor += tex_nmap.alpha
;-----

mad r0.a, r1, v0.b, t0.a

; PERFORM PER PIXEL SHADOW MASKING
;-----
; if (occlusion_factor >= 0.5)
;   pixel_result = dp3_result
; else
;   pixel_result = 0.0f
;-----

cnd_sat r0, r0.a, r1, c0.a

; MULTIPLY LIGHT RESULT BY LIGHT COLOR
;-----
; pixel_result *= light_color
;-----

mul r0, r0, c1

; ADD AMBIANT
;-----
; pixel_result += ambient
;-----

add r0, c0, r0

; MULTIPLY BY DIFFUSE TEXTURE
;-----
; pixel_result *= tex_diffuse
;-----

mul r0, r0, t1

```

If you have any comments or additional questions on this technique, please let me know.

Best regards,  
Danny