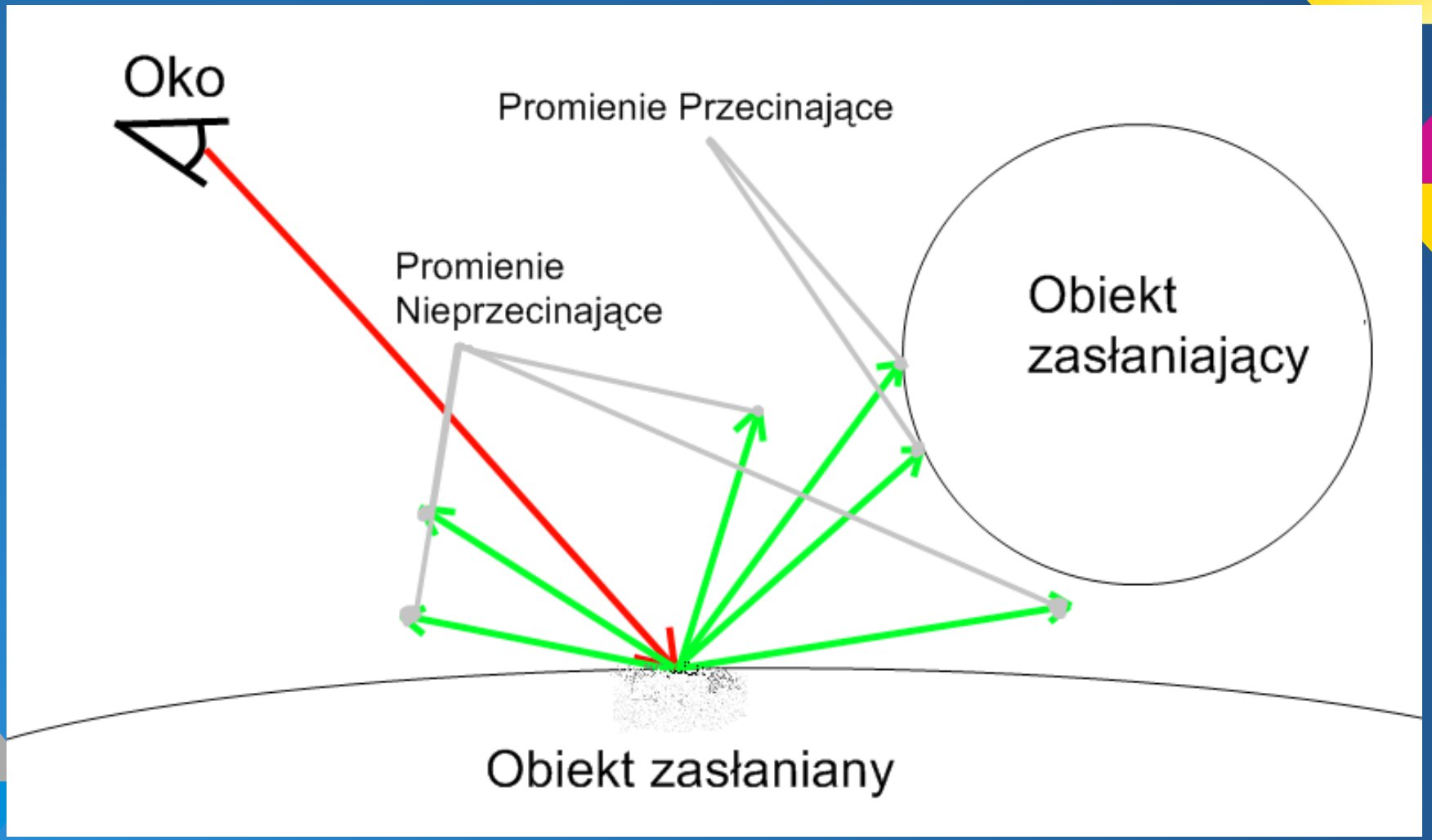
Screen Space Ambient Occlusion

Karol Grzybowski
Piotr Ruchwa

Ambient Occlusion

- metoda cieniowania powierzchni obiektów przestrzennych używana w grafice trójwymiarowej,
- technika ta szacuje stopień zacienienia danej powierzchni przez rozproszone światło otoczenia,
- pozwala na self-shadowing

Ambient Occlusion



Screen Space Ambient Occlusion

- technika renderowania przybliżająca efekt Ambient Occlusion w czasie rzeczywistym,
- efekt *post-process*,
- została opracowana przez Władimira Kajalina *(Crytek)*,
- pierwszy raz użyta w grze *Crysis (Crytek, 2007)*

Screen Space Ambient Occlusion

Chapter 8: Finding Next Gen – CryEngine 2

Figure 15. *Screen-Space Ambient Occlusion in a complete ambient lighting situation (note how occluded areas darken at any distance)*

Screen Space Ambient Occlusion - zalety

- niezależny od stopnia skomplikowania sceny,
- zerowy koszt pamięciowy,
- działa z dynamicznymi scenami,
- działa w ten sam sposób dla każdego piksela na ekranie,
- brak narzutu na CPU,
- w całości na GPU,
  - pixel shader

Screen Space Ambient Occlusion - wady

- lokalny, w wielu przypadkach zależny od widoku,
- zależy od sąsiednich tekseli,
- trudno wygładzić szum bez zakłócania ciągłości głębi,
- "wylewanie" się efektu poza geometrię

Implementacja

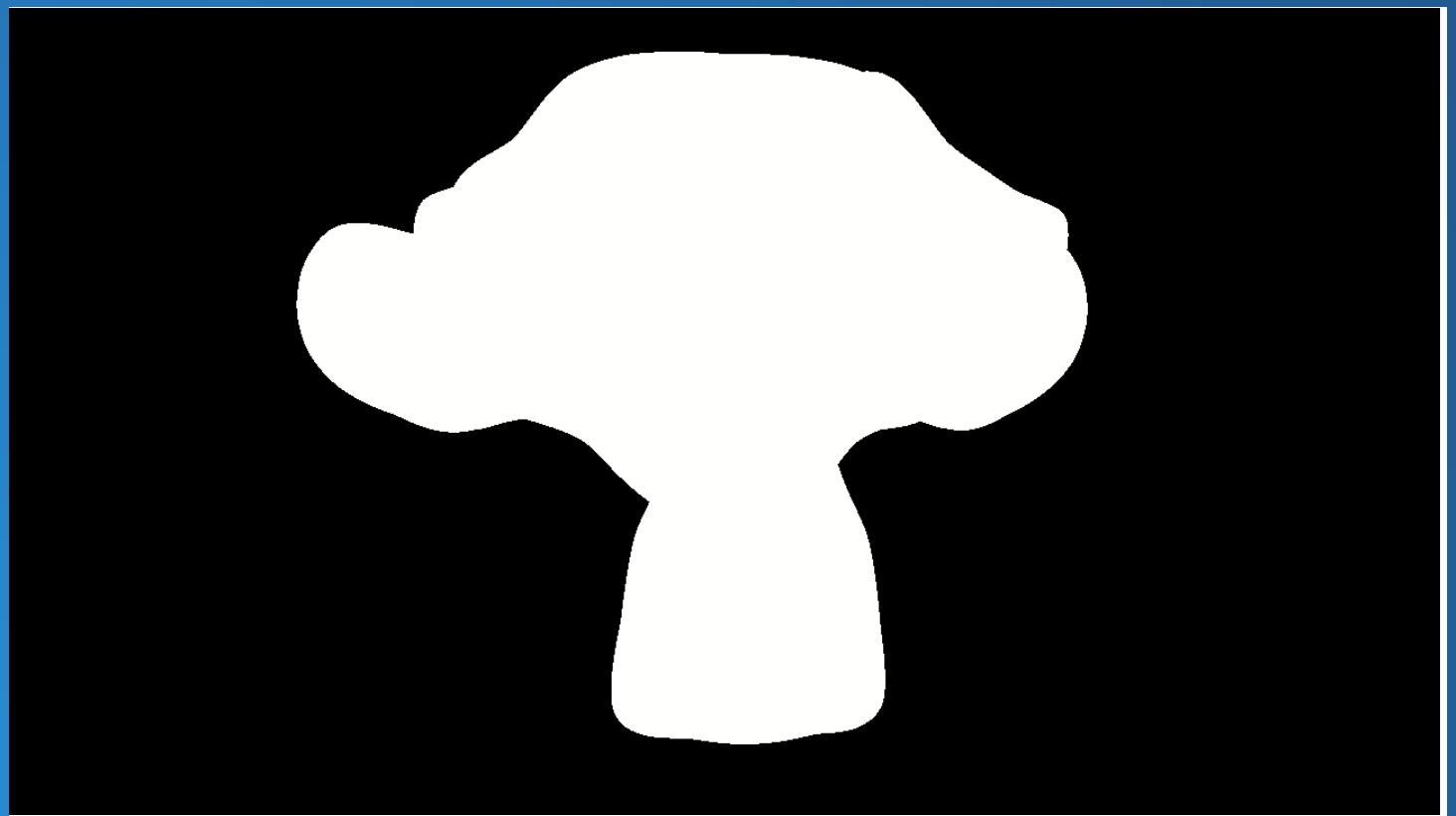
Direct3D 11 / HLSL SM5

Geometry Buffer

- Depth - DXGI_FORMAT_D32_FLOAT

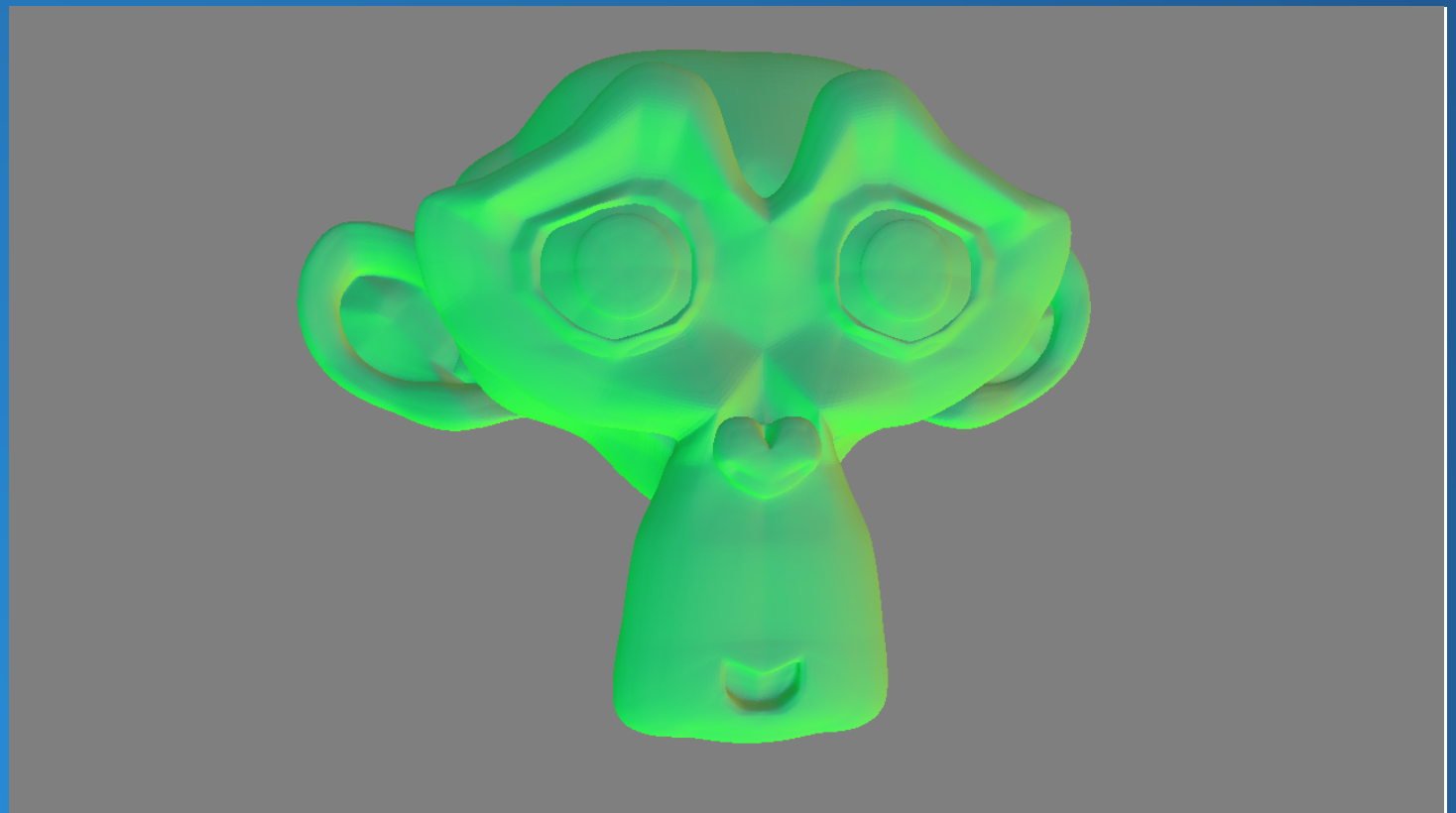
Geometry Buffer

- Depth - DXGI_FORMAT_D32_FLOAT
- Position - DXGI_FORMAT_R32G32B32A32_FLOAT



Geometry Buffer

- Depth - DXGI_FORMAT_D32_FLOAT
- Position - DXGI_FORMAT_R32G32B32A32_FLOAT
- Normal - DXGI_FORMAT_R32G32B32A32_FLOAT



Geometry Buffer

- Depth - DXGI_FORMAT_D32_FLOAT
- Position - DXGI_FORMAT_R32G32B32A32_FLOAT
- Normal - DXGI_FORMAT_R32G32B32A32_FLOAT

Shader SSAO - algorytm

- dla każdego piksela:
  - wyznacz jego pozycję (gbuffer / depth)
  - wyznacz N próbek wokół piksela źródłowego
    - sprawdź, czy go przesłaniają porównując głębokość / pozycję
  - oblicz średnią ze wszystkich

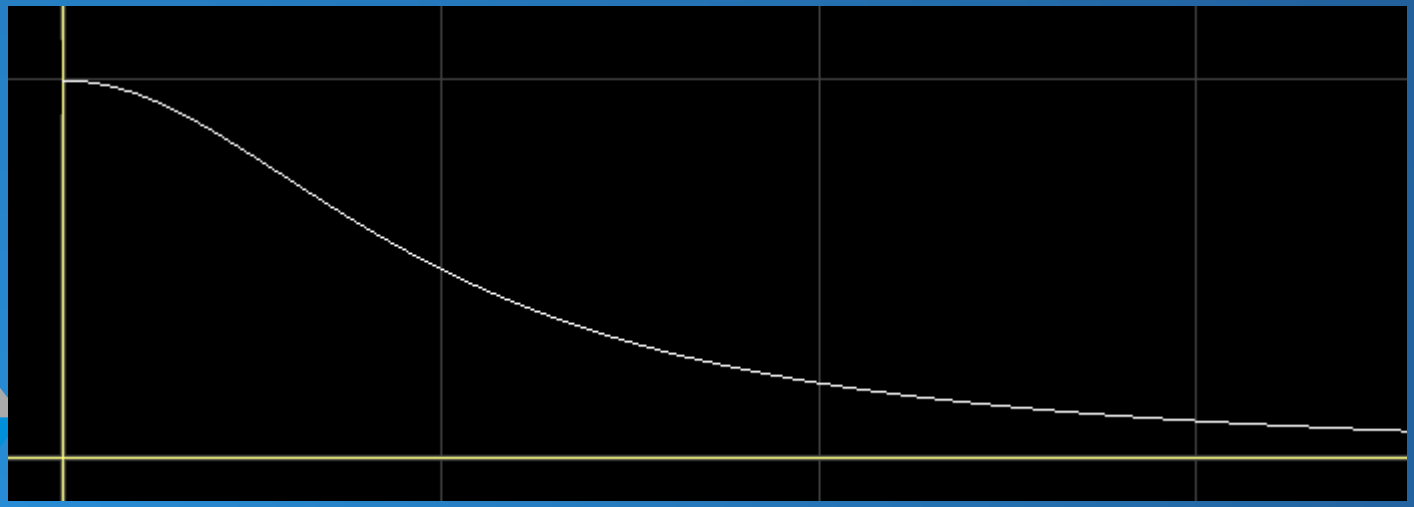
Shader SSAO - wyznaczenie współczynnika AO

```
float gather_occlusion(float3 p, float3 n, float3 occ_p)
{
    float3 v = occ_p - p;
    float d = length(v);
    v /= d;
    d *= g_scale;

    return max(0.0F, dot(n, v) - g_bias) * (1.0F / (1.0F + d)) * g_intensity;
}
```

- g_scale - skala wektora v
- g_bias - przesunięcie głębi
- g_intensity - intensywność

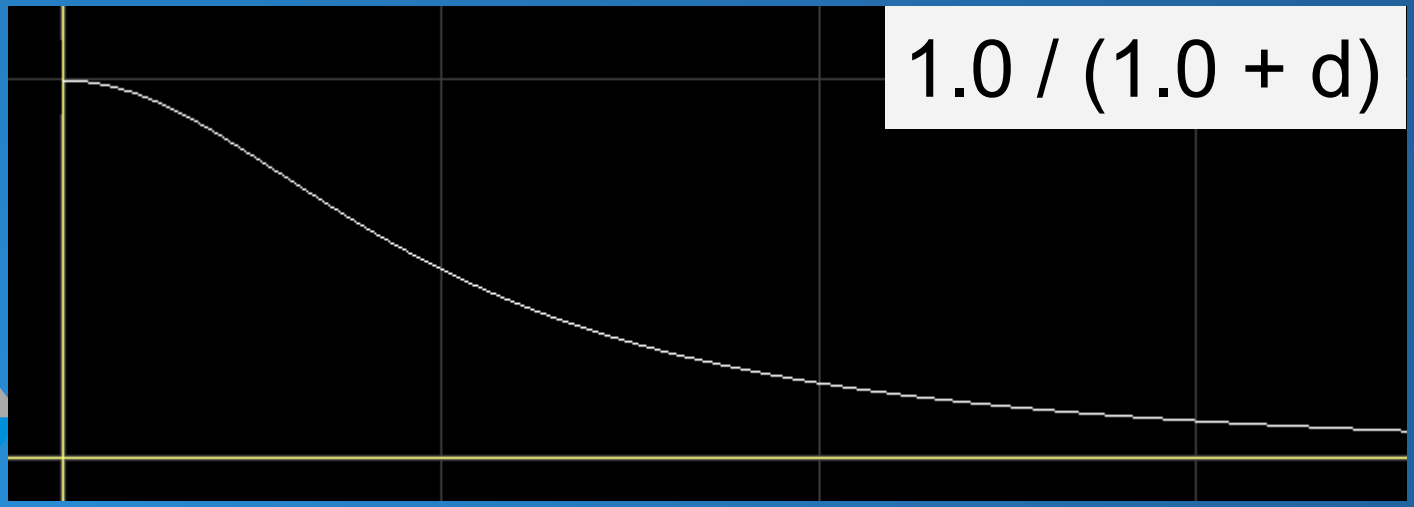
Shader SSAO - wyznaczenie współczynnika AO

```
float gather_occlusion(float3 p, float3 n, float3 occ_p)
{
    float3 v = occ_p - p;
    float d = length(v);
    v /= d;
    d *= g_scale;

    return max(0.0F, dot(n, v) - g_bias) * (1.0F / (1.0F + d)) * g_intensity;
}
```

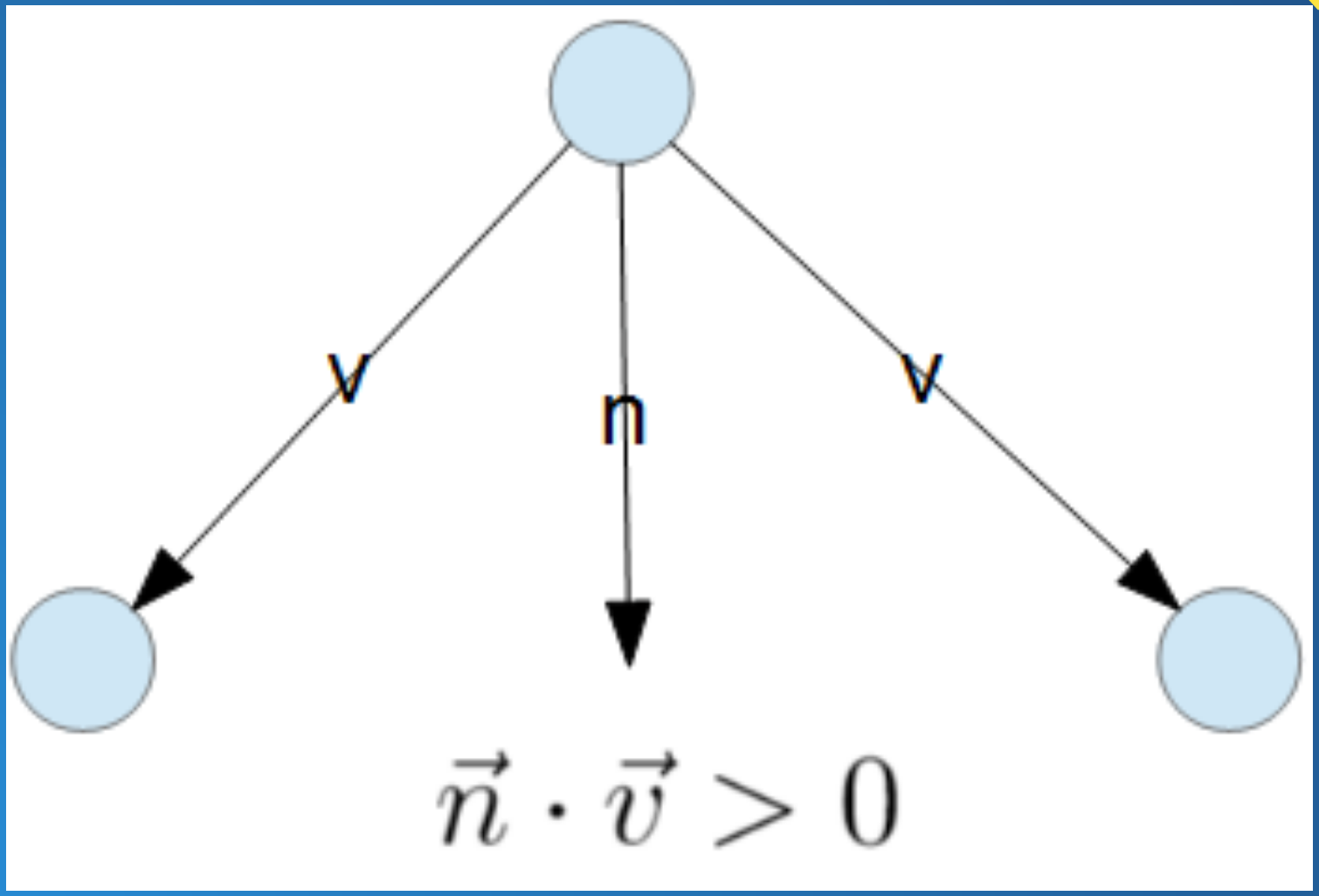


Shader SSAO - wyznaczenie współczynnika AO

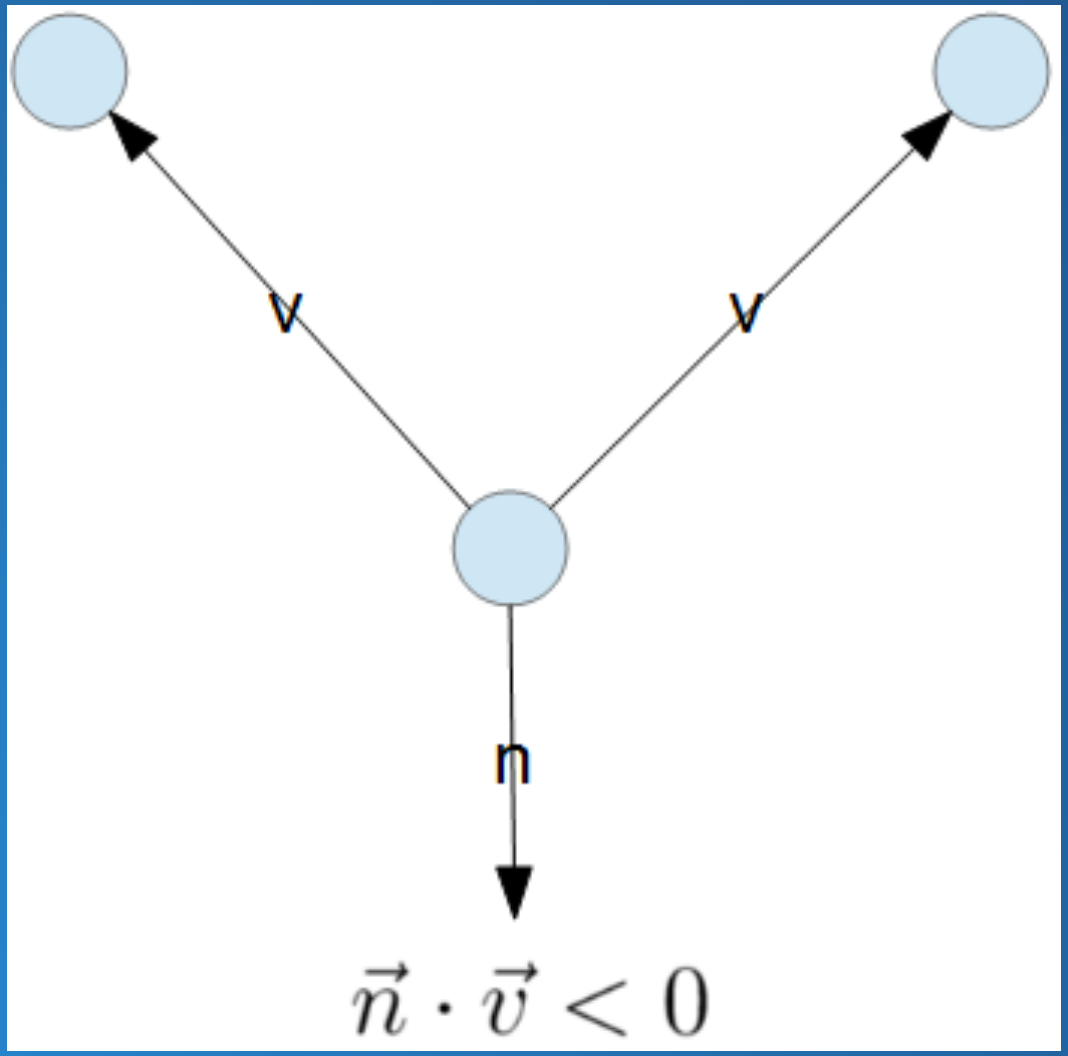
```
float gather_occlusion(float3 p, float3 n, float3 occ_p)
{
    float3 v = occ_p - p;
    float d = length(v);
    v /= d;
    d *= g_scale;

    return max(0.0F, dot(n, v) - g_bias) * (1.0F / (1.0F + d)) * g_intensity;
}
```



Shader SSAO - wyznaczenie współczynnika AO



Shader SSAO - wyznaczenie współczynnika AO



Shader SSAO - wyznaczenie współczynnika AO

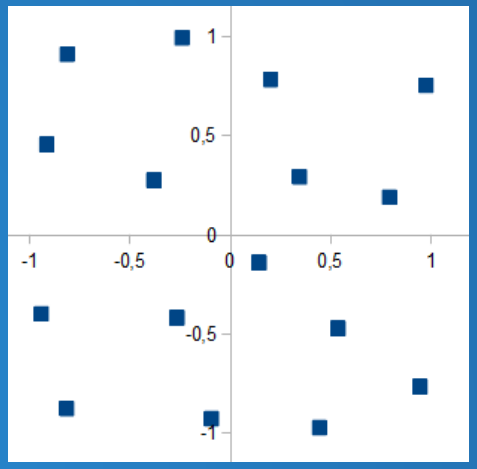
```
float compute_ssao(float2 src_coord) {
    float src_depth = sample_depth(src_coord);

    float3 src_position = sample_position(src_coord);
    float3 src_normal = sample_normal(src_coord);

    float occ_total = 0.0F;
    float samples = 16.0F;

    [loop]
    for (int i = 0; i < samples; ++i) {
        float2 offset = poissonDisk[i];

        float2 occ_coord = src_coord + offset * g_pixel_radius;
        float occ_depth = sample_depth(occ_coord);

        [branch]
        if (occ_depth < src_depth) {
            float3 occ_point = sample_position(occ_coord);
            occ_total += gather_occlusion(src_position, src_normal, occ_point);
        }
    }

    return 1.0F - (occ_total / samples);
}
```

- g_pixel_radius - promień próbkowania

Shader SSAO

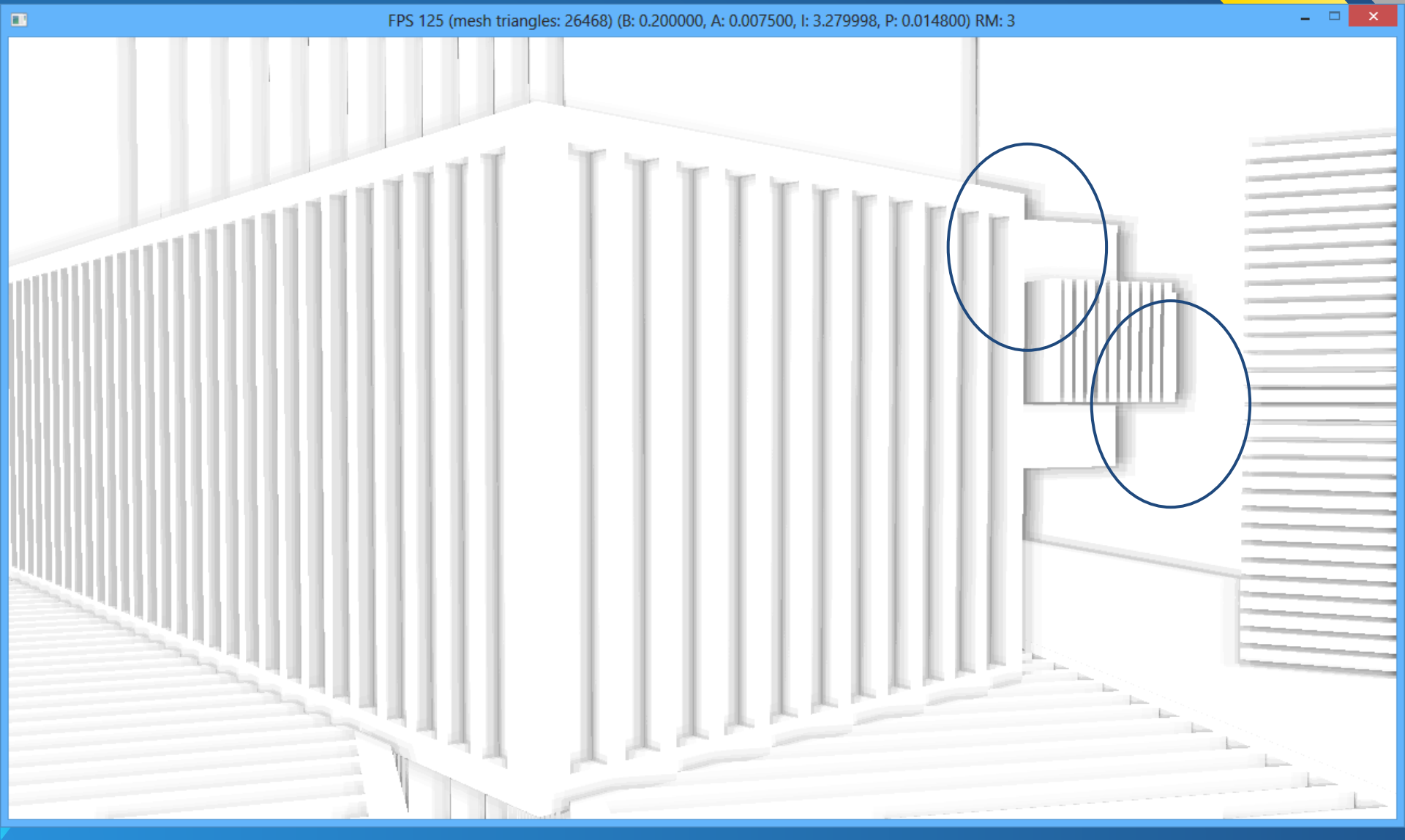
- ~31 instrukcji

```
static float2 poissonDisk[16] = {
    float2(-0.94201624F, -0.39906216F),
    float2( 0.94558609F, -0.76890725F),
    float2(-0.09418410F, -0.92938870F),
    float2( 0.34495938F,  0.29387760F),
    float2(-0.91588581F,  0.45771432F),
    float2(-0.81544232F, -0.87912464F),
    float2(-0.38277543F,  0.27676845F),
    float2( 0.97484398F,  0.75648379F),
    float2( 0.44323325F, -0.97511554F),
    float2( 0.53742981F, -0.47373420F),
    float2(-0.26496911F, -0.41893023F),
    float2( 0.79197514F,  0.19090188F),
    float2(-0.24188840F,  0.99706507F),
    float2(-0.81409955F,  0.91437590F),
    float2( 0.19984126F,  0.78641367F),
    float2( 0.14383161F, -0.14100790F)
};
```




```
// Name                 Index   Mask Register SysValue Format   Used
// -------------------- ----- ------ -------- -------- ------ ------
// SV_POSITION              0   xyzw        0      POS  float
// TEXCOORD                 0   xy          1     NONE  float   xy
//
//
// Output signature:
//
// Name                 Index   Mask Register SysValue Format   Used
// -------------------- ----- ------ -------- -------- ------ ------
// SV_TARGET                0   xyzw        0   TARGET  float   xyzw
//
ps_5_0
dcl_globalFlags refactoringAllowed
dcl_constantbuffer cb2[6], immediateIndexed
dcl_sampler s0, mode_default
dcl_resource_texture2d (float,float,float,float) t0
dcl_resource_texture2d (float,float,float,float) t1
dcl_resource_texture2d (float,float,float,float) t2
dcl_input_ps linear v1.xy
dcl_output o0.xyzw
dcl_temps 4
dcl_indexableTemp x0[5], 4
mov x0[0].xy, l(-0.942016,-0.399062,0,0)
mov x0[1].xy, l(0.945586,-0.768907,0,0)
mov x0[2].xy, l(-0.094184,-0.929389,0,0)
mov x0[3].xy, l(0.344959,0.293878,0,0)
mov x0[4].xy, l(-0.915886,0.457714,0,0)
sample_l_indexable(texture2d)(float,float,float,float) r0.x, v1.xyxx, t0.xyzw, s0, l(0.000000)
sample_l_indexable(texture2d)(float,float,float,float) r0.yzw, v1.xyxx, t2.wxyz, s0, l(0.000000)
sample_l_indexable(texture2d)(float,float,float,float) r1.xyz, v1.xyxx, t1.xyzw, s0, l(0.000000)
mad r1.xyz, r1.xyzx, l(2.000000, 2.000000, 2.000000, 0.000000), l(-1.000000, -1.000000, -1.000000, 0.000000)
mov r2.xy, l(0,0,0,0)
loop
  itof r1.w, r2.y
  ge r1.w, r1.w, l(4.000000)
  breakc_nz r1.w
  mov r2.zw, x0[r2.y + 0].xxxy
  mad r2.zw, r2.zzzw, cb2[4].xxxx, v1.xxxy
  sample_l_indexable(texture2d)(float,float,float,float) r1.w, r2.zwzz, t0.yzwx, s0, l(0.000000)
  lt r1.w, r1.w, r0.x
  if_nz r1.w
    sample_l_indexable(texture2d)(float,float,float,float) r3.xyz, r2.zwzz, t2.xyzw, s0, l(0.000000)
    add r3.xyz, -r0.yzwy, r3.xyzx
    dp3 r1.w, r3.xyzx, r3.xyzx
    sqrt r1.w, r1.w
    div r3.xyz, r3.xyzx, r1.wwww
    dp3 r2.z, r1.xyzx, r3.xyzx
    add r2.z, r2.z, -cb2[4].w
    max r2.z, r2.z, l(0.000000)
    mad r1.w, r1.w, cb2[4].z, l(1.000000)
    div r1.w, l(1.000000, 1.000000, 1.000000, 1.000000), r1.w
    mul r1.w, r1.w, r2.z
    mad r2.x, r1.w, cb2[5].x, r2.x
  endif
  iadd r2.y, r2.y, l(1)
endloop
mad o0.xyz, -r2.xxxx, l(0.250000, 0.250000, 0.250000, 0.000000), l(1.000000, 1.000000, 1.000000, 0.000000)
mov o0.w, l(1.000000)
ret
```

SSAO - render


FPS 125 (mesh triangles: 26468) (B: 0.200000, A: 0.007500, I: 3.279998, P: 0.014800) RM: 3

Shader SSAO

Podsumowanie:

- sample_depth: 1 + 16 wywołań
- sample_position: 1 + 16 wywołań
- sample_normal: 1 wywołanie
- jedna instrukcja IF :(

W sumie 35 wywołań

Można to zrobić lepiej

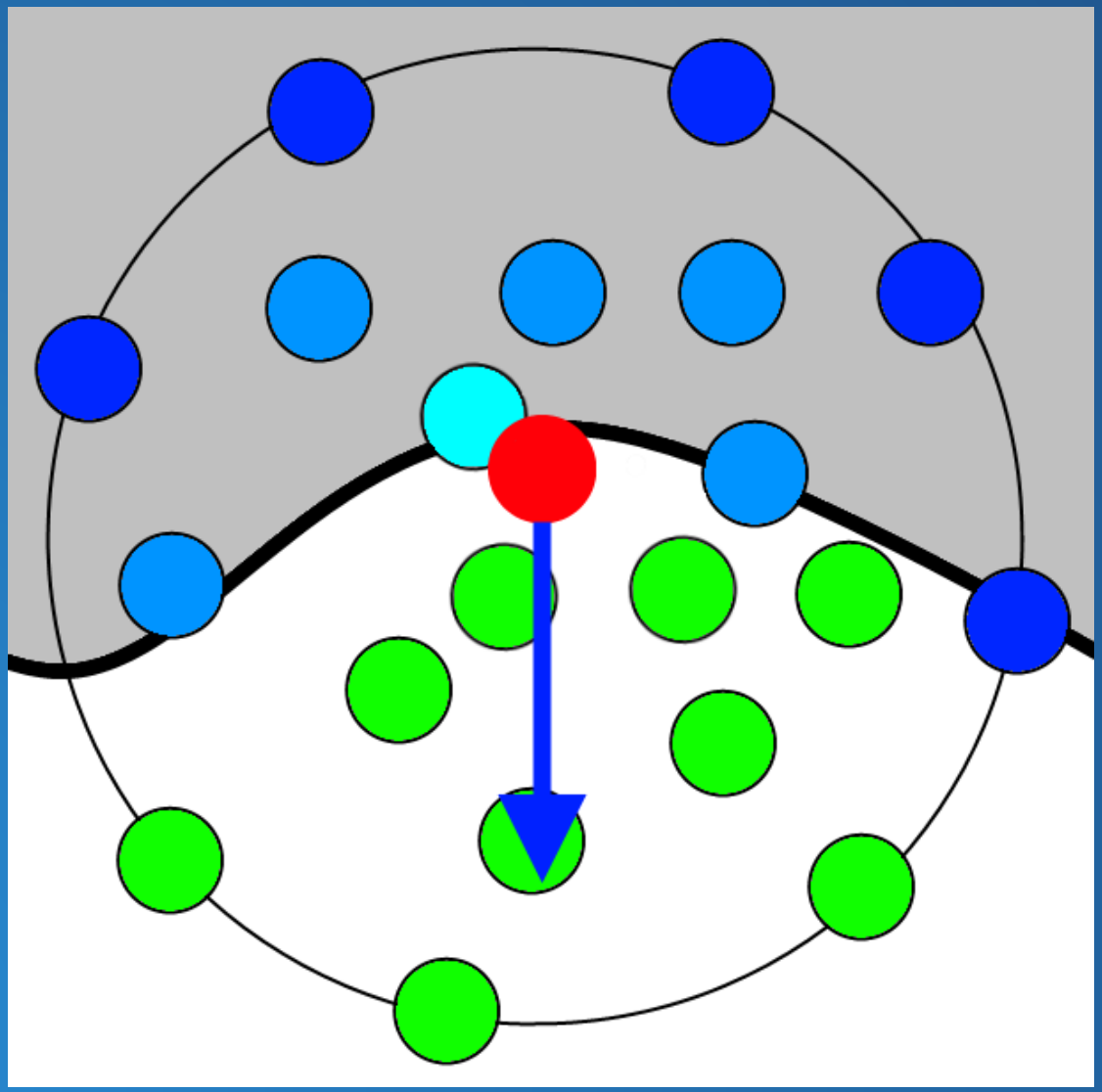
korzystając jedynie z depth buffor-a,

- ale w g-buffer mamy dokładniejsze normalne,
- więc użyjemy ich!

SSAO - podejście drugie

- zamiast liczyć dot-product pomiędzy punktami można:
  - sprawdzić widoczność losowych punktów na sferze,
    - środek sfery znajdowałby się w aktualnym punkcie,
    - punkt na sferze = position + radius * reflect (sphere_sample, random),
    - wartość przesłonięcia += różnica pomiędzy tymi punktami (depth),

SSAO - podejście drugie

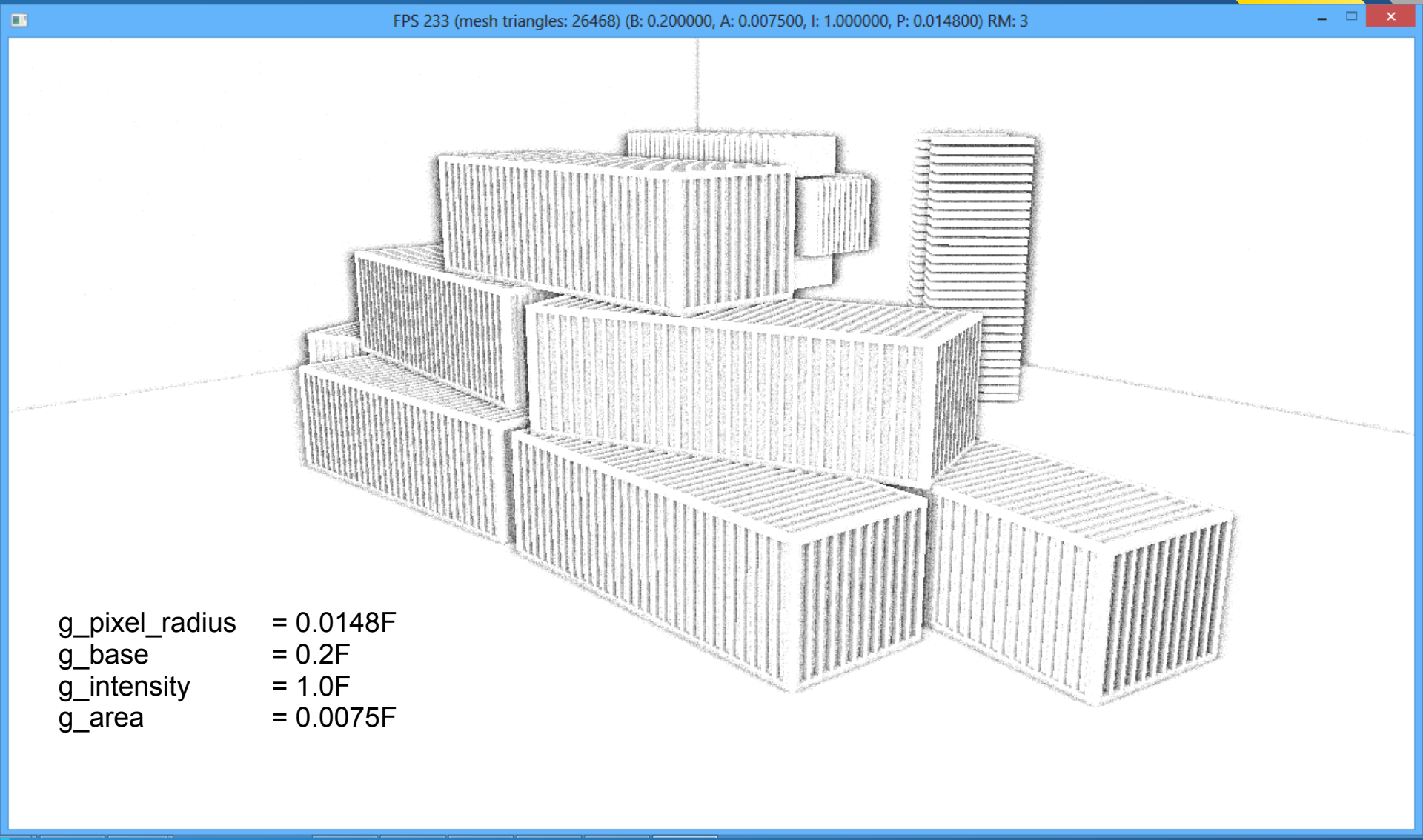


SSAO - implementacja

```
float compute_ssao_depth(float2 coord)
{
    const float falloff = 0.000001F;
    float3 random = normalize(sample_random_normal(coord).rgb);
    float depth = sample_depth(coord).r;
    float3 position = float3(coord, depth);
    float3 normal = sample_normal(coord);

    float radius_depth = g_pixel_radius / depth;
    float occlusion = 0.0F;

    [loop]
    for (int i = 0; i < 16; ++i) {
        float3 ray = radius_depth * reflect(g_sample_sphere[i], random);
        float3 hemi_ray = position + sign(dot(ray, normal)) * ray;

        float occ_depth = sample_depth(saturate(hemi_ray.xy)).r;
        float difference = depth - occ_depth;

        occlusion += step(falloff, difference) * (1.0F - smoothstep(falloff, g_area, difference));
    }

    float ao = 1.0F - g_intensity * occlusion * (1.0F / 16.0F);
    return saturate(ao + g_base);
}
```

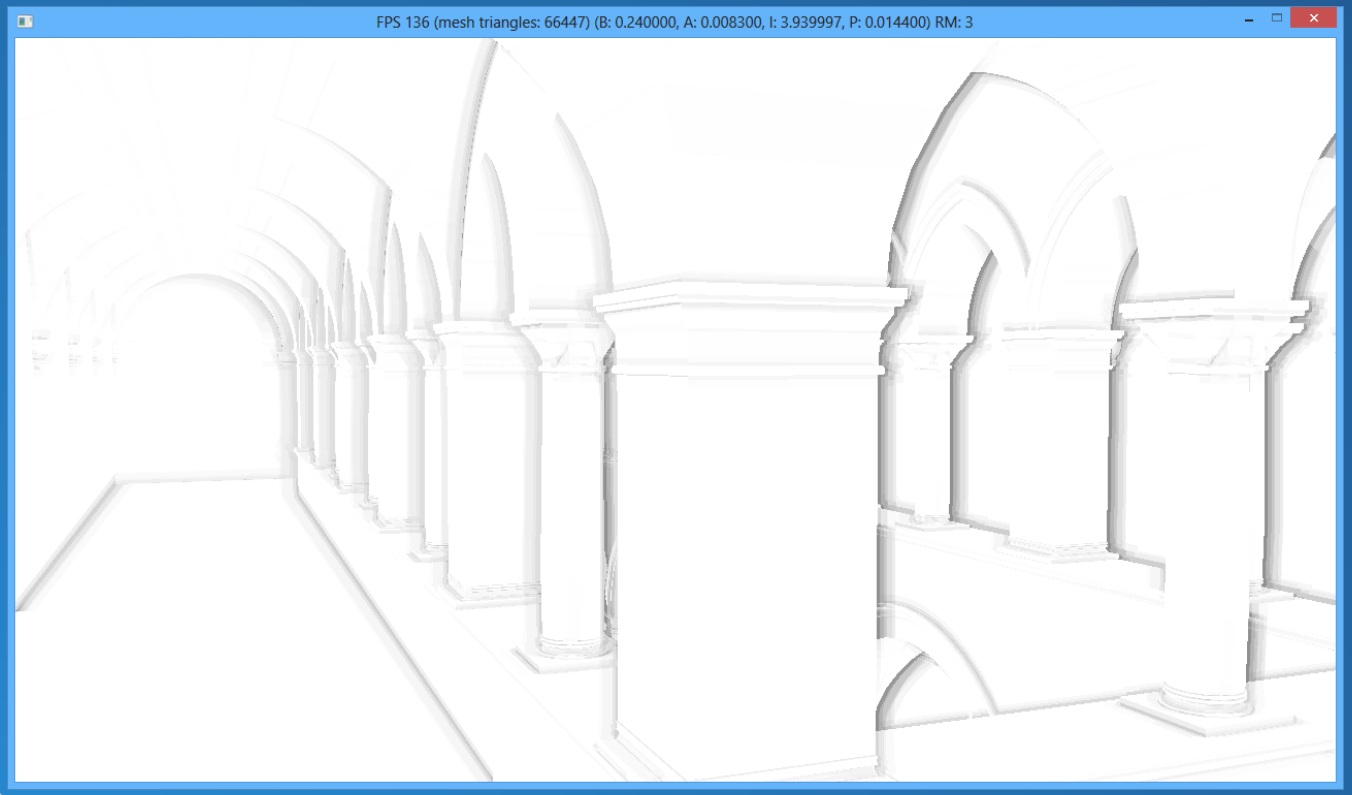
```
static float3 g_sample_sphere[16] =
{
    float3( 0.5381F,  0.1856F, -0.4319F), float3( 0.1379F,  0.2486F,  0.4430F),
    float3( 0.3371F,  0.5679F, -0.0057F), float3(-0.6999F, -0.0451F, -0.0019F),
    float3( 0.0689F, -0.1598F, -0.8547F), float3(-0.4776F,  0.2847F, -0.0271F),
    float3(-0.0146F,  0.1402F,  0.0762F), float3( 0.0100F, -0.1924F, -0.0344F),
    float3(-0.3169F,  0.1063F,  0.0158F), float3(-0.3577F, -0.5301F, -0.4358F),
    float3( 0.0103F, -0.5869F,  0.0046F), float3(-0.0897F, -0.4940F,  0.3287F),
    float3( 0.7119F, -0.0154F, -0.0918F), float3(-0.0533F,  0.0596F, -0.5411F),
    float3( 0.0352F, -0.0631F,  0.5460F), float3( 0.0560F,  0.0069F, -0.1843F),
};
```

SSAO - depth

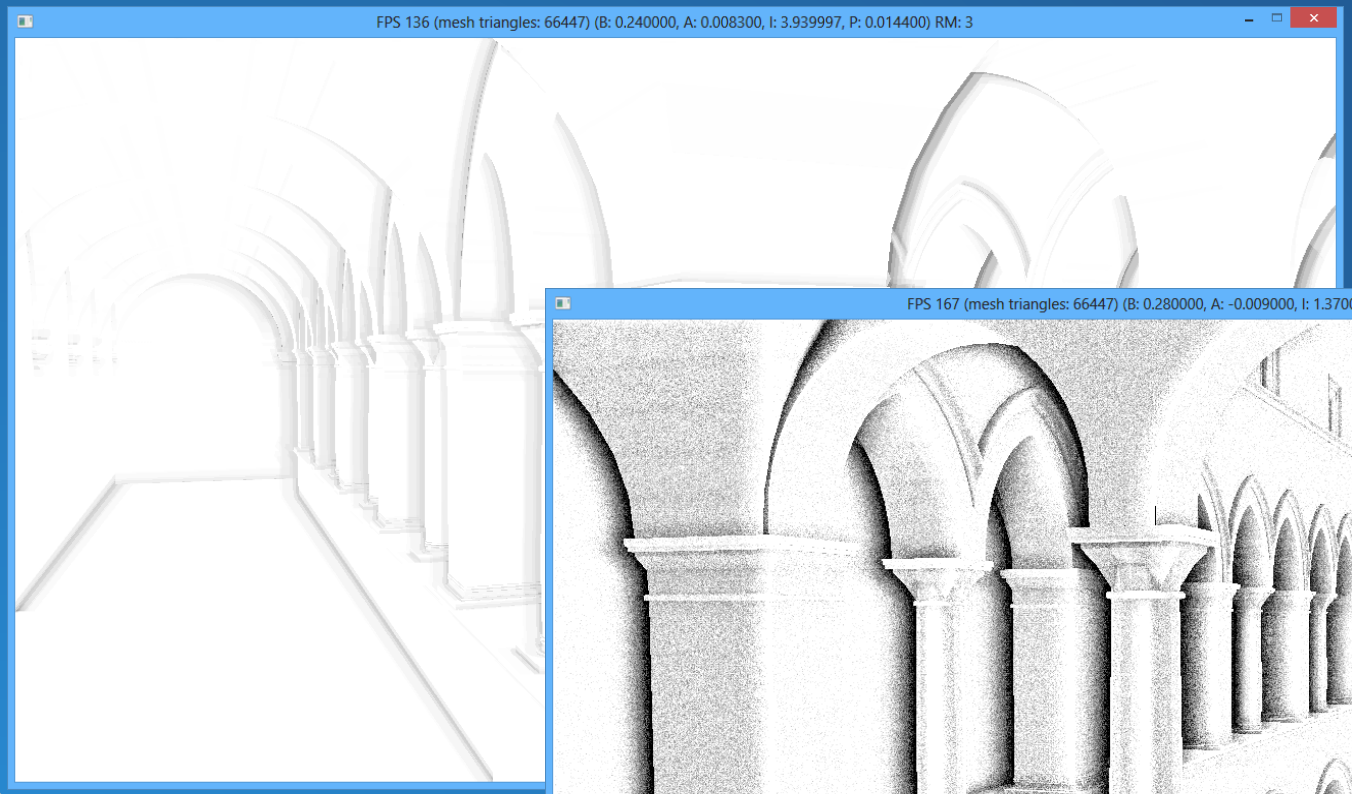


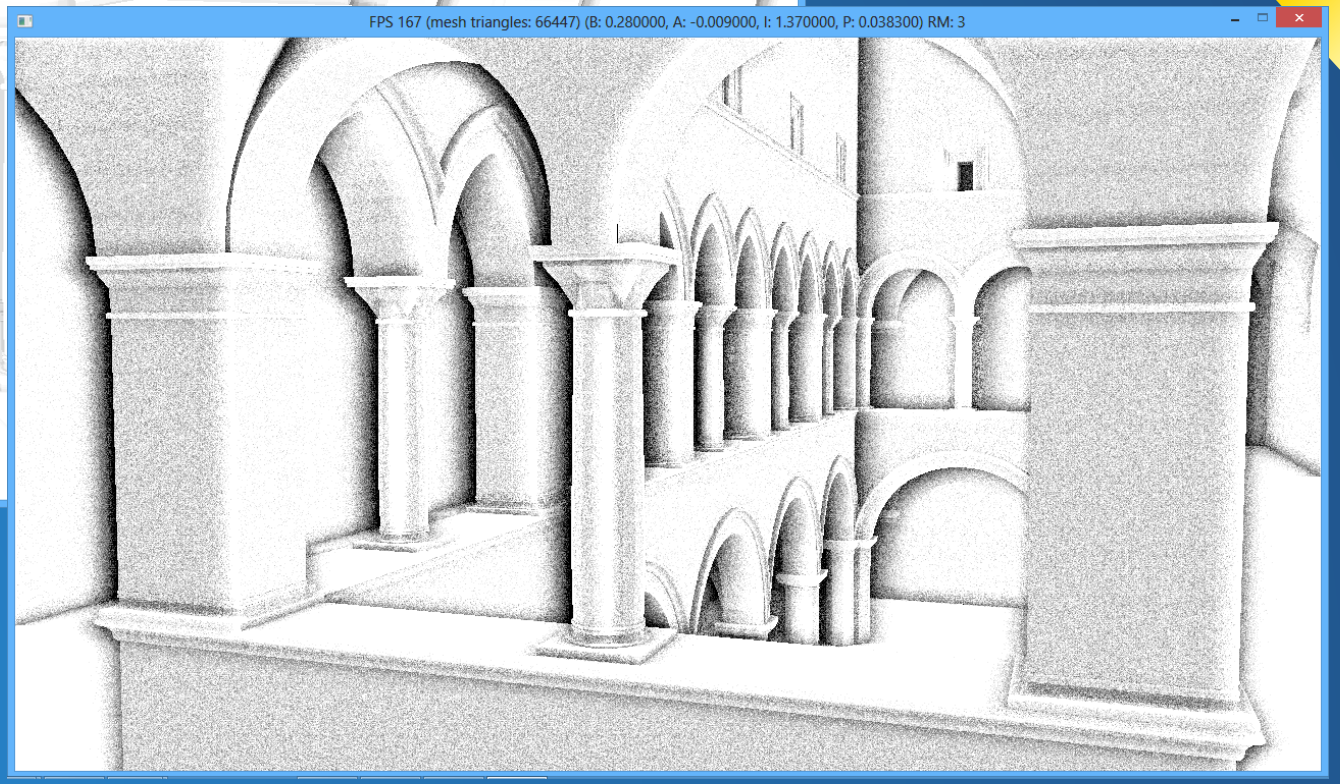
g_pixel_radius = 0.0148F
g_base = 0.2F
g_intensity = 1.0F
g_area = 0.0075F

Stary SSAO vs Nowy SSAO



Stary SSAO vs Nowy SSAO





Shader SSAO

Podsumowanie:

- sample_normal: 1 wywołanie
- sample_random_normal: 1 wywołanie
- sample_depth: 1 + 16 wywołań

W sumie: 19 wywołań

Poprawa jakości

- upsampling,
- blur,
- HBAO - Horizon-based Ambient Occlusion,
  - stworzona przez firmę NVIDIA,
    - (min. DX10, DX11),
  - miękkie, realistyczne cienie,

Poprawa jakości (SSAO)

Poprawa jakości (HBAO)

Materiały

- Finding Next Gen – CryEngine, Martin Mittring, Crytek GmbH. Advanced Real-Time Rendering in 3D Graphics and Games Course – SIGGRAPH 2007
- Five rendering ideas from Battlefield 3 / Need For Speed: The Run – SIGGRAPH 2011
- Screen Space Ambient Occlusion – Maurycy Chomicz – IGK2008