着色像素点 1 对某个像素点进行着色,ShadingEngine

输入: 采样 *sample* 信息 sac, 像素 *pixel* 信息 pc, 着色 *shading* 信息 shc, 着色点 *shading_point* 信息 sp

输出: shading_result

```
function SHADE_HIT_POINT(sac, pc, shc, sp)
    shading_result.main.a ← sp.a [ material 的 alpha map]
    material ← sp.get_material()
    if shading_result.main.a > 0 then
        surface_shader ← GETSURFACESHADER(material)
        shading_result.main.rgb ← surface_shader. EVALUATE(sac, pc, shc, sp)
        APPLY_ALPHA_PREMULT(shading_result)
    end if
    return false
end function

function GETSURFACESHADER(material)
    if has_diagnostic_surface_shader then
        return diagnostic_surface_shader
    else
        return material.m_surface_shader
    end if
end function

function PHYSICALSURFACESHADER::EVALUATE(sac, pc, shc, sp)
    basis ← sp.get_shading_basis()
    shading_components ← shc.get_lighting_engine. COMPUTE_LIGHTING(sac, pc, shc, sp)
    if m_lighting_samples > 1 then
        for 1 : mighting_samples − 1 do
            sp.set_shading_basis(basis)
            shading_components ← shc.get_lighting_engine. COMPUTE_LIGHTING(sac, pc, shc, sp)
        end for
        shading_components/ = m_lighting_samples
    end if
    if has_npr then
        NPRSurfaceShaderHelper. EVALUATE(sac, shc, sp)
    end if
    return shading_components.m_beauty.rgb
end function

function PTLIGHTINGENGINE::COMPUTE_LIGHTING<B_NEXT_EVENT_ESTIMATION>(sac, shc, sp)
    [path_length, shading_components] ← path_tracer. TRACE(sac, shc, sp)
    return shading_components
end function
```

着色像素点 2 路径追踪,PathTracer::trace

```
function PathTracer::trace(sac, shc, sp)
    vertex.path_length ← 1
    vertex.scattering_modes ← ALL
    vertex.throughput ← 1
    vertex.shading_point ← sp
    vertex.prev_mode ← Specular
    vertex.prev_prob ← DiracDelta(−1)
    diffuse_bounces ← glossy_bounces ← specular_bounces ← iterations ← 0
    while true do
        ray ← vertex.shading_point.get_ray()
        vertex.outgoing = −ray
        if vertex.shading_point.hit_surface() then
            material ← vertex.get_material()
            if material then
                object_instance ← vertex.shading_points.get_object_instance()
                if PassThrough(vertex, sac) then
                    next_ray ← ray
                    next_ray.ori ← ray.hit_point
                    intersector ← shc.get_intersector()
                    next_shading_point ← intersector. trace(next_ray, vertex.shading_point)
                    vertex.shading_point = next_shading_point
                    Continue
                else
                    NormalBounce(vertex, material)
                end if
            end if
        end if
    end while
end function

function PassThrough(vertex, sac)
    if vertex.path_length > 1 then
        alpha ← vertex.shading_point.get_alpha()
        if sac.random() >= alpha || alpha <= 0  then
            return true
        end if
    end if
    return false
end function
```

着色像素点 3 正常处理,NormalBounce

```
function NormalBounce(vertex, material)
    vertex.edf ← material.edf
    vertex.bsdf ← material.bsdf
    vertex.bssrdf ← material.bssrdf
    vertex.cos_on ← dot(vertex.outgoing, vertex.shading_normal)
    EmittedLightContribution(vertex)
    if RussianRoulette(sac, vertex) then
        ProcessHit(vertex)
    end if
end function

function EmittedLightContribution(vertex)
    if vertex.edf & vertex.cos_on then
        emitted_radiance ←AddEmittedLightContribution(vertex)
        emitted_radiance* = vertex.throughput
        shading_components. AddEmission(vertex.path_length, emitted_radiance)
    end if
end function

function RussianRoulette(sac, vertex)
    if vertex.path_length <= rr_min_path_length then
        return true
    else
        scattering_prob ← vertex.throughput
        if scattering_prob < sac.random() then
            return false
        else
            vertex.throughput/ = scattering_prob
            return true
        end if
    end if
end function
```

着色像素点 **4** 正常处理,ProcessHit

```
function ProcessHit(sac, vertex)
    if ContinueProcess() then
        if IsSubSurface then
            ProcessSubSurface()
            UpdateVertex()
        end if
        if IsBSDF then
            ProcessBSDF()
            next_shading_point ← intersector.trace()
            vertex.parent_shading_point = vertex.shading_point
            vertex.shading_point = next_shading_point
        end if
        return true
    end if
end function
```