

```

//
// Fragment shader used in the paper "Omnidirectional Relief Impostors"
//
//
/*
    Numbering of the impostor vertices:

    p2-----p3
    |         |
    |         |
    |         |
    p0-----p1
*/

struct Impostor
{
    vec3 normal;          // impostor normal (world space)
    vec3 u;               // vector p1-p0 (normalized)
    vec3 v;               // vector p2-p0 (normalized)
    vec2 depthFactor;    // vec2((zfar-znear)/||p1-p0||, (zfar-znear)/||p0-p2|| )
};

uniform Impostor impostor;
uniform sampler2D sampler0; // RGBZ texture
uniform sampler2D sampler1; // normal map
uniform vec3 eye;          // viewpoint coordinates (world space)

varying vec3 coord;       // fragment coordinates (world space)

// ray-surface intersection code. Based on the paper by Policarpo, Oliveira, Comba,
// Real-Time Relief Mapping on Arbitrary Polygonal Surfaces, I3D 2005

void ray_intersect_binary_search(sampler2D sampler, vec2 dp, vec2 ds,
                                out float best_depth, out float best_depth_outside)
{
    const int linear_search_steps = 16;
    const int binary_search_steps = 7;
    float depth_step = 1.0 / float(linear_search_steps);
    float size = depth_step;
    float depth = 0.0;

    best_depth = 1.0;
    best_depth_outside = 0.0;

    // If extended polygons are supported (see Sect. 3.1)
    // (the texture covers only a portion of the support polygon
    // and gets texcoords beyond [0.0,1.0] range), we should move
    // to the initial position inside the valid range.

    // linear search
    // search from front to back for first point inside the object
    bool found = false;
    for ( int i=0; i<linear_search_steps-1 && !found ;i++)
    {
        depth += size;
        vec2 newTexel = dp+ds*depth;
        if (newTexel.s<=1.0 && newTexel.s>=0.0 && newTexel.t<=1.0 && newTexel.t>=0.0)
        {
            vec4 t = texture2D(sampler,newTexel);
            if (t.a <= depth) // inside
            {
                found = true;
                best_depth=depth; // store best depth
            }
        }
        else discard;
    }
    best_depth_outside = depth - size;

    // if intersection not found -> discard the fragment
    if (!found) discard;

    depth = best_depth;
}

```

```

// search around first point (depth) for closest match
for ( int i=0; i<binary_search_steps;i++)
{
    size*=0.5; // binary search
    vec2 newTexel = dp+ds*depth;
    vec4 t=texture2D(sampler,newTexel);
    if (t.a <= depth) // still inside
    {
        best_depth = depth; // best_depth inside
        depth -= 2.0 * size;
    }
    else
        best_depth_outside = depth;

    depth+=size;
}

}

void main()
{
    vec3 v = normalize(coord - eye);
    vec3 vp = normalize( vec3(dot(impostor.u,v),dot(impostor.v, v),-dot(impostor.normal,v)));
    vec2 startTexel = gl_TexCoord[0].st;
    vec2 dir = impostor.depthFactor * vp.xy / vp.z;
    float d_in, d_out;
    ray_intersect_binary_search(sampler0, startTexel, dir, d_in, d_out);
    vec2 st_in = startTexel + d_in*dir;
    vec2 st_out = startTexel + d_out*dir;
    vec4 color_in = texture2D(sampler0, st_in);
    vec4 color_out = texture2D(sampler0, st_out);

    // wall detection
    const float jump = 0.04; // greater value -> less cracks but more skin
    if ( (color_out.a - color_in.a ) > jump ) discard;

    vec2 st;
    vec4 color;
    if (abs(color_in.a - d_in) < abs(color_out.a - d_out))
    {
        st = st_in;
        color = color_in;
    }
    else
    {
        st = st_out;
        color = color_out;
    }

    vec3 surfaceNormal = texture2D(sampler1, st).xyz;
    surfaceNormal = surfaceNormal*2.0 - vec3(1.0, 1.0, 1.0);

    // correct z value can be computed here
    gl_FragColor = light(color, gl_NormalMatrix*surfaceNormal);
}

```