k-DOP Clipping: Robust Ghosting Mitigation in Temporal Antialiasing, Supplementary Documentation

Julius Ikkala
julius.ikkala@tuni.fi
Tampere University
Tampere, Finland

Tuomas Lauttia
tuomas.lauttia@tuni.fi
Tampere University
Tampere, Finland

Pekka Jääskeläinen
pekka.jaaskelainen@tuni.fi
Tampere University
Tampere, Finland

Markku Mäkitalo
markku.makitalo@tuni.fi
Tampere University
Tampere, Finland

k-DOP clipping shader code

 GLSL source code for performing k-DOP clipping:

```
const int neighborhood_size = 9;
vec3 kdop_clipping(
    vec3 cur_color,
    vec3 prev_color,
    vec3 colors[neighborhood_size]
){
    const float epsilon = 1e-5f;
    const int axis_count = /* Number of k-DOP axes (k/2) */;
    const vec3 axes[] = vec3[](
        // k-DOP axes go here
    );

    vec3 dir = prev_color - cur_color;
    float near = -1e9f, far = 1e9f;
    [[unroll]] for(int a = 0; a < axis_count; ++a)
    {
        vec3 axis = axes[a];
        vec2 extent = vec2(1e9f, -1e9f);

        [[unroll]] for(int n = 0; n < neighborhood_size; ++n)
        {
            float t = dot(colors[n], axis);
            extent.x = min(t, extent.x);
            extent.y = max(t, extent.y);
        }

        extent += vec2(-epsilon +epsilon);

        float proj_pos = dot(cur_color, axis);
        float inv_dir = 1.0f / dot(dir, axis);
        float t0 = (extent.x - proj_pos) * inv_dir;
        float t1 = (extent.y - proj_pos) * inv_dir;

        near = max(near, min(t0, t1));
        far = min(far, max(t0, t1));
    }
    if(near <= far && (near > 0.0f || far > 0.0f))
    {
        float t = clamp(near > 0.0f ? near : far, 0.0f, 1.0f);
        return cur_color + t * dir;
    }
    return cur_color;
}
```

The following variant implements k-DOP variance clipping:

```
const int neighborhood_size = 9;
vec3 kdop_variance_clipping(
    vec3 cur_color,
    vec3 prev_color,
    float gamma,
    vec3 colors[neighborhood_size]
){
    const int axis_count = /* Number of k-DOP axes (k/2) */;
    const vec3 axes[] = vec3[](
        // k-DOP axes go here
    );

    vec3 dir = prev_color - cur_color;
    float near = -1e9f, far = 1e9f;
    [[unroll]] for(int a = 0; a < axis_count; ++a)
    {
        vec3 axis = axes[a];
        vec2 moments = vec2(0);

        [[unroll]] for(int n = 0; n < neighborhood_size; ++n)
        {
            float t = dot(colors[n], axis);
            moments += vec2(t, t*t);
        }

        float proj_pos = dot(cur_color, axis);
        moments /= vec2(neighborhood_size);
        float mu = moments.x;
        float sigma = sqrt(moments.y - mu*mu);
        vec2 extent = vec2(
            min(mu - gamma * sigma, proj_pos),
            max(mu + gamma * sigma, proj_pos)
        );

        float inv_dir = 1.0f / dot(dir, axis);
        float t0 = (extent.x - proj_pos) * inv_dir;
        float t1 = (extent.y - proj_pos) * inv_dir;

        near = max(near, min(t0, t1));
        far = min(far, max(t0, t1));
    }
    if(near <= far && (near > 0.0f || far > 0.0f))
    {
        float t = clamp(near > 0.0f ? near : far, 0.0f, 1.0f);
        return cur_color + t * dir;
    }
    return cur_color;
}
```

Both variants combine the construction of the k-DOP with its intersection, as this allows us to not store a temporary array of

extents. These functions were used in the measurements of the paper, but are not necessarily optimal.

Precalculated k-DOP axis sets

The k-DOP axes were precalculated using a tool, whose source code is available at https://github.com/vga-group/taa-kdop-optimizer.

The following listing is the 32-DOP axis set used in the paper:

```
vec3(1, 0, 0),
vec3(0, 1, 0),
vec3(0, 0, 1),
vec3(0.820081, 0.456727, -0.344773),
vec3(0.540295, 0.829202, 0.143195),
vec3(0.255800, 0.841084, -0.476597),
vec3(-0.406935, -0.389062, 0.826459),
vec3(-0.826708, -0.382923, -0.412219),
vec3(0.260942, -0.577482, 0.773578),
vec3(0.254398, 0.637821, 0.726957),
vec3(0.310900, -0.728083, -0.610930),
vec3(0.798513, -0.556827, -0.228738),
vec3(0.673383, -0.163602, -0.720964),
vec3(-0.813922, 0.369658, -0.448201),
vec3(0.477650, -0.853722, 0.207384),
vec3(-0.554854, -0.041550, -0.830910)
```

It was computed such that the first three axes are the X, Y and Z vectors. With them alone, the method matches AABB neighborhood clipping. For the rest of the axes, a simulated annealing-like process was used to optimize them for smallest bounded volume, given minimum and maximum extents of -1 and 1. This effectively finds the tightest-fit k-DOP axes around a contained sphere.

Zeroed components can help with performance, assuming loops are unrolled and an optimizing compiler is used that can simplify the dot product of a constant vector containing zeroes. For a more lightweight general option, here's a similarly produced 22-DOP set which contains more zeroed entries:

```
vec3(1, 0, 0),
vec3(0, 1, 0),
vec3(0, 0, 1),
vec3(0.000000, 0.664104, -0.747640),
vec3(-0.656496, -0.656139, 0.372149),
vec3(-0.664251, 0.000000, -0.747509),
vec3(-0.664672, 0.000000, 0.747135),
vec3(0.000000, 0.664287, 0.747478),
vec3(0.657415, -0.657637, -0.367857),
vec3(-0.656571, -0.657954, -0.368796),
vec3(0.656800, -0.655635, 0.372501)
```

Two scene-optimized sets were used in the paper. They were generated by selecting a set of random 3x3 neighborhoods from a screenshot, then applying a simulated annealing-like process to find axes that minimize the average k-DOP volume around those neighborhoods.

This 16-DOP set was optimized for the "grass" scene:

```
vec3(-0.663315, 0.729578, -0.166518),
vec3(0.575024, -0.084585, -0.813753),
vec3(-0.158262, 0.489372, -0.857594),
vec3(0.256166, -0.038665, -0.965859),
vec3(0.640264, -0.436421, -0.632138),
vec3(-0.813891, 0.466034, 0.346978),
vec3(-0.831598, -0.215117, -0.512025),
vec3(0.488190, -0.812612, 0.318327)
```

And this 16-DOP was optimized for the "asphalt" scene:

```
vec3(0.997167, -0.054820, 0.051500),
vec3(0.043682, -0.727663, 0.684542),
vec3(-0.451553, 0.808335, -0.377751),
vec3(-0.003001, 0.615645, -0.788018),
vec3(0.130192, -0.063137, -0.989476),
vec3(0.775079, -0.584236, -0.240668),
vec3(0.589837, 0.171644, -0.789070),
vec3(0.898010, -0.436338, -0.056459)
```