Efficient Rendering of Rounded Corners and Edges for Convex Objects

Simon Courtin[1], Sébastien Horna[1], Mickaël Ribardière[1], Pierre Poulin[2], and Daniel Meneveaux[1]

[1] Univ. Poitiers, CNRS, XLIM, UMR 7252, Poitiers, France
[2] LIGUM, Dept. I.R.O., Université de Montréal

Abstract. Round edges and vertices are a source of multiple light effects, but their representation is often neglected because they introduce complex geometric meshing. This paper presents a new method for managing thin rounded edges without explicitly changing the underlying geometry and the associated complex meshing, so as to produce their visual effects with ray tracing and path tracing. Our method relies on cylinders and spheres positioning, associated with a detection and acceleration structure that makes the process more robust and efficient than existing bevel shaders. Moreover, using simple smooth surfaces rather than polygonal meshes also makes it possible to generate close views of the surfaces with a much better visual quality and much less memory consumption. Our results present the produced effects, as well as comparisons with existing industrial software shaders.

1 Introduction

This document presents additional material, including images produced with commercial software, comparisons between existing methods and our approach, as well as some implementation details and data structure.

2 Images produced with commercial engines

Figure 1 illustrates the results obtained with several existing bevel methods found in commercial software. Geometric subdivision is mostly employed because the resulting geometry can be used in any rendering engine.

Figure 1.a corresponds to Catmull-Clark subdivisions, that highly increase the geometric complexity. Figure 1.b-d show various artifacts visible with normal interpolation methods (bevel shaders), even with objects as simple as cubes. Bevel shaders do not account for geometric changes, and the object silhouette are not modified. The sampling strategies used to interpolate edge normals from other faces do not correctly handle adjacency, as illustrated in Figure 1.c. Finally, Figure 1.d demonstrates that highlights are also prone to artifacts, since interpolated normals do not correspond to the real object surface.

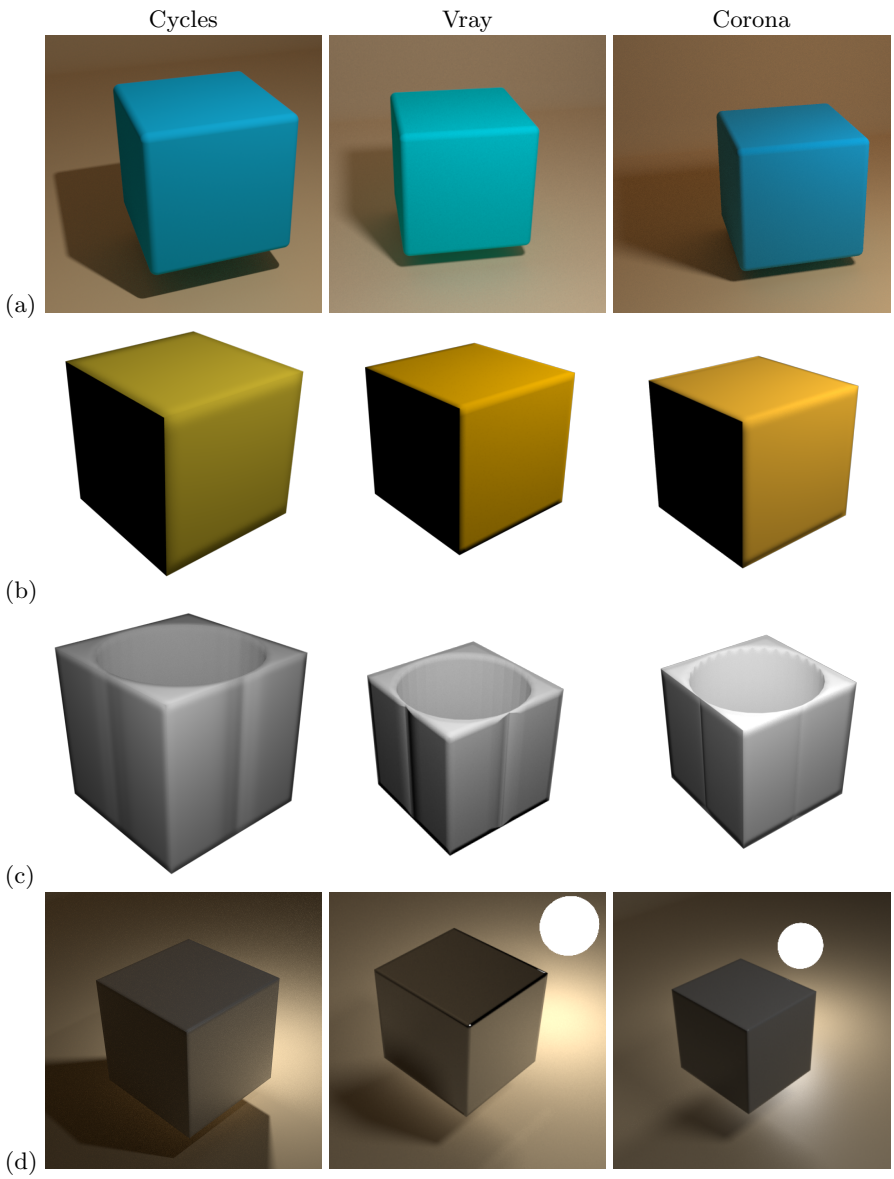
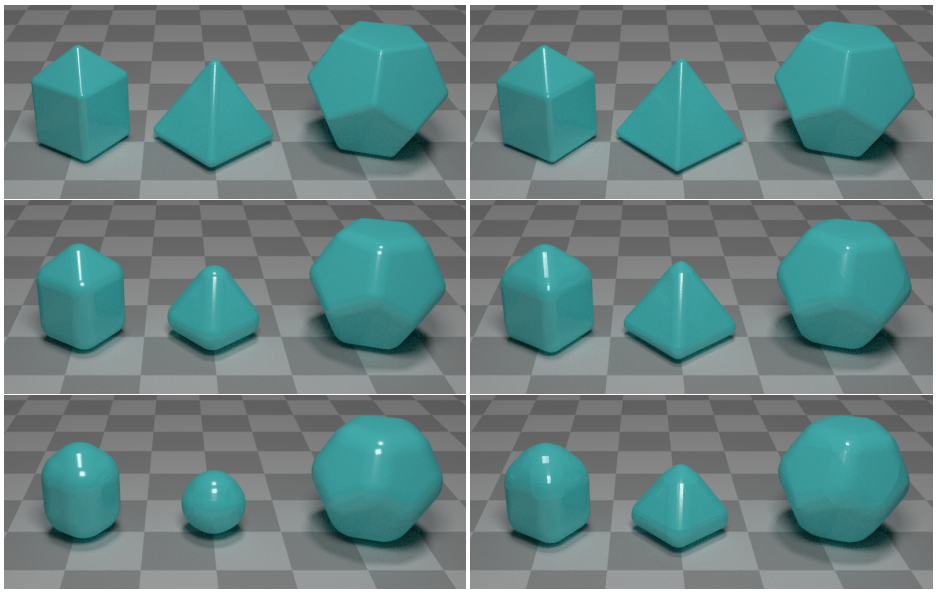


Fig. 1: Rounding and bevel shaders with *Cycles, Vray* and *Corona*. (a) Rounded corners based on subdivions surfaces; (b) Bevel shaders do not change the existing geometry; (c) Artifacts due to bevel shaders with thin geometric configurations; (d) Glossy objects and missing highlights with bevel-shaders.

3 Rounding comparisons

Figure 2 shows comparisons of rendered rounded corners between our method and mesh subdivisions. As illustrated, our system ensures a correct appearance even with high radius values.



(a) Our method **(b)** Subdivision surfaces

Fig. 2: Comparisons between our method and Catmull-Clark subdivisions (from *Blender* software), with chamfer sizes set to 0.2, 0.5 and 0.8.

With 10 mesh subdivisions and thin bevels, the rounding effect appears very smooth. With our method, smoothness is ensured with any rounding radius value (Figure 2.a). When bevel sizes increases, the obtained subdivision faces become visible (Figure 2.b).

4 Data structure and algorithms

With our method, the rendering system only requires testing detection cylinders. The actual chamfer intersection is processed thanks to cylinders and spheres intersections.

Listing 1 provides the main idea used for using detection cylinders based on a given intersection point I.

```
// Input:
vec3 I;   // given 3D point
vec3 P1p; // P1' the first detection cylinder axis point
vec3 V12; // Axis direction : V12=P2'-P1'
float r2; // rd*rd, rd being the detection cylinder radius

// Computation:
float p=dot(V12,I-P1p); // project I on axis = cos(alpha)
if(p>0.0 && p<1.0) { // between P1' and P2'
    float d2 = (1-p^2) // =sin(alpha)^2, distance to the axis
    if(d2 < r2) {
        // I belongs to the bevel area
        return hit_rounding_cylinder_data
    }
}
```

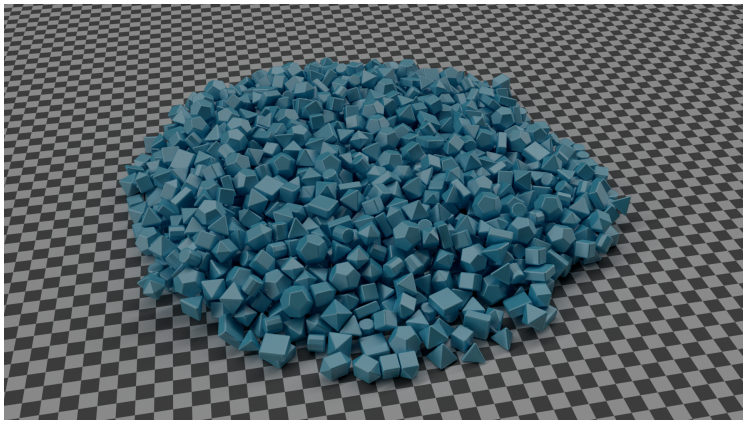
Listing 1: Method for testing if an intersection on object surface is in detection cylinder

The data structure stored for each edge is provided in Listing 2. Rounding Spheres centers are not explicitly stored since all the necessary information is already contained in the cylinders data structure: Rounding cylinders axes are defined by sphere centers and the same radius is employed for all the edges of a given object.

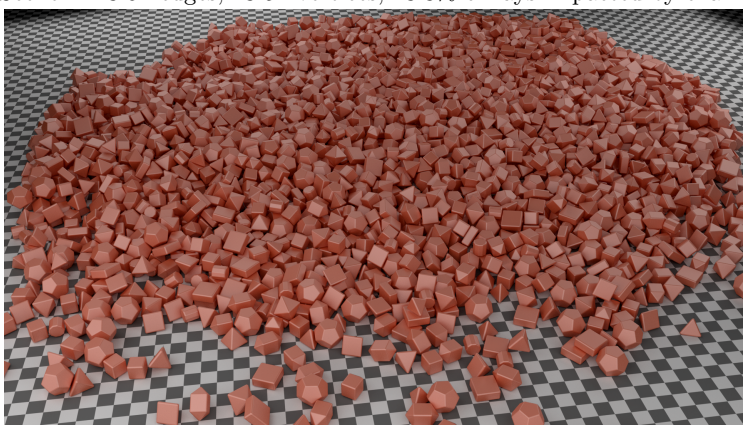
```
struct cylinder {
    float3 v1;
    float3 v2;
    float radius;
    /*precomputed values*/
    float3 v1v2;
    float3 dir;
    float r2;
    float lenght2;
};

struct sphere {
    float3 center;
    float radius;
};

struct edge{
    cylinder* detect;
    cylinder* chamfer;

    //corner
    corners* corners[2];
};

struct corner{
    float3 position;
    edge[] adjacent_round_edges;
};
```

Listing 2: Cylinder, Sphere, Edge and Corner data in our rounding method

5 Additional Results



(a) Scene 1: 43.6k edges, 23.5k vertices, 23.3% of rays impacted by chamfers.

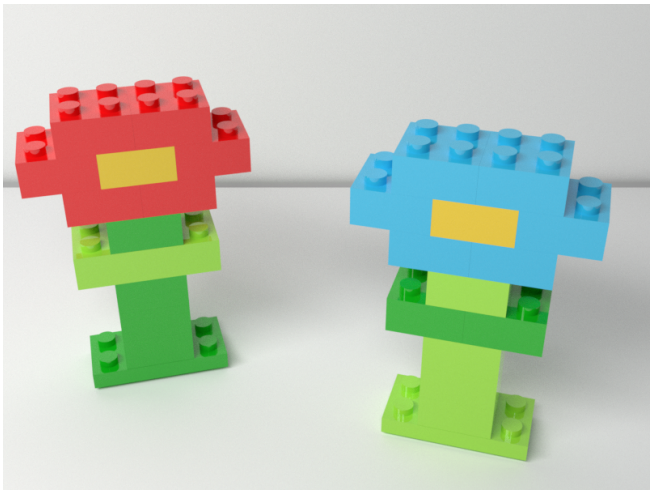


(b) Scene 2: 106k edges, 57.5k vertices, 30.4% of rays impacted by chamfers.

(c) Scene 3: 259k edges, 140k vertices, 29.2% of rays impacted by chamfers.

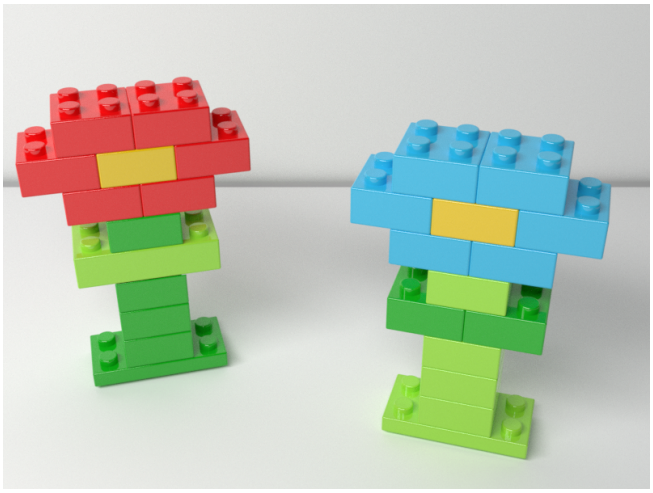
Fig. 3: Three test scenes composed of polyhedrons rendered with our method.

Fig. 4: Blocks: 182k edges, 122k vertices, 1.89% of rays impacted by chamfers.



(a) No chamfer

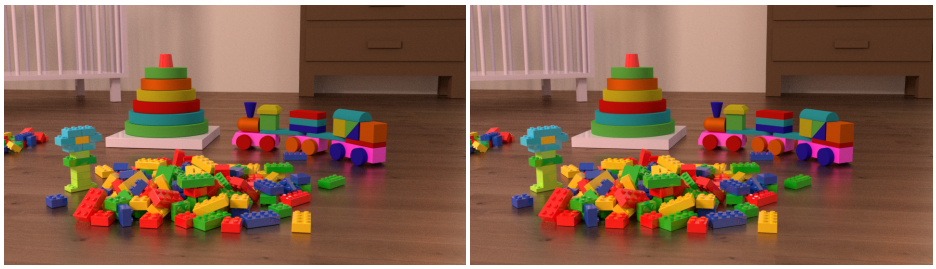
(b) Photograph



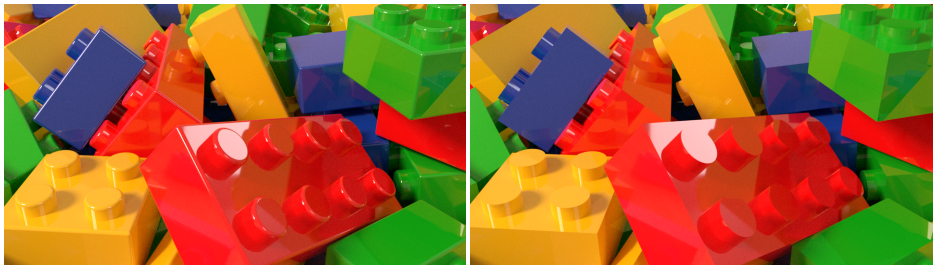
(c) Our rendering method

Fig. 5: Visual comparisons of real and virtual objects with and without rounded corners: (a) Virtual objects with sharp edges; (b) Photo of real play blocks; (c) Same virtual objects with our rounded spheres and cylinders. Notice the top red and cyan blocks on the two structures, with their darker adjacent rounded edges.

(a) Frame 139.



(b) Frame 200.



(c) Frame 432.

Fig. 6: Frames from Blocks scene video. Left: Frame with rounded edges and corners (our method); Right: Same frame without rounding.