Efficient Rendering of Rounded Corners and Edges for Convex Objects

Simon Courtin, Sébastien Horna, Mickaël Ribardière, Pierre Poulin, Daniel Meneveaux

Efficient Rendering of Rounded Corners and Edges for Convex Objects

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

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

Abstract. Many manufactured objects and worn surfaces exhibit rounded corners and edges. These fine details are a source of sharp highlights and shading effects, important to our perception between joining surfaces. However, their representation is often neglected because they introduce complex geometric meshing in very small areas. This paper presents a new method for managing thin rounded corners and edges without explicitly modifying the underlying geometry, so as to produce their visual effects in sample-based rendering algorithms (e.g., ray tracing and path tracing). Our method relies on positioning virtual spheres and cylinders, associated with a detection and acceleration structure that makes the process more robust and more efficient than existing bevel shaders. Moreover, using our implicit surfaces rather than polygonal meshes allows our method to generate extreme close views of the surfaces with a much better visual quality for little additional memory. We illustrate the achieved effects and analyze comparisons generated with existing industrial software shaders.

Keywords: rounded edges · bevel · chamfer · shading · implicit surface representation.

1 Introduction

The realism of computer-generated images can be greatly improved with the representation of detailed features, such as dust, hair, or imperfect surfaces [12]. However, such thin structures are often neglected because (i) their impact affects only small portions of image pixels, (ii) their representation highly increases scene complexity in terms of geometry and appearance modeling, and (iii) their associated processing results in high complexity, both in terms of computation time and memory requirements.

This paper focuses on the rendering of thin rounded corners and edges. As shown by previous authors [15], managing the effects resulting from rounded corners and edges improves the observed realism. They produce specular highlights and appearance changes that improve shape perception. Real objects almost never exhibit perfect sharp edges, and their borders appear as brighter or darker than the rest of the surface, as illustrated in Figure 1. Such rounded edges are often due in real life to imprecisions or desired intentions caused by the manufacturing process, e.g., in material cutting, moulding, sculpting, etc., or due to the wear of surfaces in their physical environment.

(a) No chamfer **(b)** Photograph **(c)** Our method

Fig. 1: Visual comparison of real and virtual objects with and without a rounding operation: (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.

Rounding operations have been studied for a long time in geometric modelling, and many software packages offer robust operations [1,2]. Beveling operations rely on explicit chamfering [10,17] or on subdivision surfaces [4,7,16]. The process generates a polygonal mesh that approximates rounded chamfers. However, this is usually not applied to all objects of a 3D scene because of the high complexity associated with the resulting mesh. In addition, even with very detailed meshes, some information is lost on rounded shapes because of the discretization process, thus resulting in limited shading effects. Mesh discontinuities may also be visible depending on the observer point of view and lighting configurations.

The rendering of round edges can also be rendered using bevel shaders, to avoid generating explicit geometry. For instance, Saito et al. [15] propose a rendering system dedicated to rasterization, where reflected radiance is integrated analytically on the curvature of rounded spheres and cylinders. Three rendering passes are performed, in which corners and edges are processed separately, and added to the final image. Tanaka et al. [18] extend this method with cross-scanline [15], thus reducing artifacts on thin curves. More recently, Wei et al. [19] propose an extension for GPU real-time rendering. These three methods have proven efficient, but their extension to global illumination remains complex since illumination on edges is performed locally and mapped directly on the final image. Several industrial software packages have an implementation of bevel shaders for managing rounded edges (e.g., Cycle [3], V-Ray [9], and Corona [8]). However they are mostly based on interpolation of normals, that is unfortunately not robust to many simple configurations (several examples are illustrated in the additional material). These methods remain unsuitable for handling the shrinking due to chamfering, and they fail with thin structures.

This paper focuses on the visual appearance of rounded corners and edges without explicitly generating a detailed mesh. We aim at defining a method that can be integrated efficiently in a ray tracing renderer (and more generally in a path tracing renderer), with an analytic description of spheres for corners and cylinders for edges. The goal is to be able to naturally generate smooth-to-sharp highlights within the rendering process, including correct geometry (volumetric shrinking on corners and edges) and normals. Our method consists in placing

automatically spheres on corners and cylinders on edges of convex objects. Our method does not modify the original geometry representation (it is not invasive), since it operates in parallel with the existing data and rendering process. We propose a new structure that handles the geometric modifications due to the rounding operation, while explicitly taking an analytic representation of smoothness with spheres and cylinders, ensuring C^1 continuity at the junctions of surface and rounded feature. We also define a structure that serves for the detection of rounded corners and edges, and for acceleration with topological links. The main contributions of our work can be summarized as follows:

- Automatic positioning of rounding spheres and cylinders to generate a C^1 smooth continuous surface, that accounts for accurate and fast ray-object intersection in many geometric cases. The resulting shape and rendering process account for volume shrinking due to chamfering.
- A geometric structure that allows for efficient detection and intersection of rounded corners and edges, including a graph for managing vertex and edge adjacencies useful for handling rays at grazing angles.
- A noninvasive structure, defined in parallel with the existing 3D data and accelerating structure, that integrates well in a sampling-based rendering system. The structure has a small impact both on memory and computation cost.

Our method has been integrated in the *Cycles* rendering system and in *Blender* [5,6]. The achieved results illustrate configurations with many rounded objects in virtual environments. Applying our rounding operations and updating our detection structures are performed in only a few seconds. It has been used in path tracing [11,14], and results illustrate substantially visible differences and quality improvements compared to existing bevel shaders.

This paper is organized as follows. Section 2 presents an overview of our method. Section 3 provides technical details of rounding geometry positioning as well our detection and acceleration structure. Section 4 describes the ray intersection process with our structure. Section 5 discusses the implementation techniques and the achieved results. Section 6 concludes and presents insights into future investigations.

2 Overview

The goal of this work is to introduce smooth chamfer effects on object corners and edges, with as few geometric primitives as possible, and efficient ray-object intersection. With our method, the original scene geometry is not modified, which makes it easy to edit chamfer radii, without much impact on a potentially complex mesh geometry. The efficiency of our approach comes from a detection and acceleration structure, located on the chamfer boundaries.

Figure 2 illustrates the general idea of the structure construction and on the rendering process. First, each edge in the original geometry is processed with its two adjacent faces to determine the positions of a rounding cylinder and a detection cylinder (Figure 2(a)). When a ray is traced in the 3D scene, the resulting intersection is with the original geometry (Figure 2(b)). This intersection

is tested with the detection cylinders (Figure 2(c)) using a Kd-tree structure. If the intersection lies within one or more detection cylinders, the associated interior rounding cylinder is used for ray intersection (Figure 2(d)). Finally, the origin of a reflected or shadow ray is placed at the correct point on the original geometry.

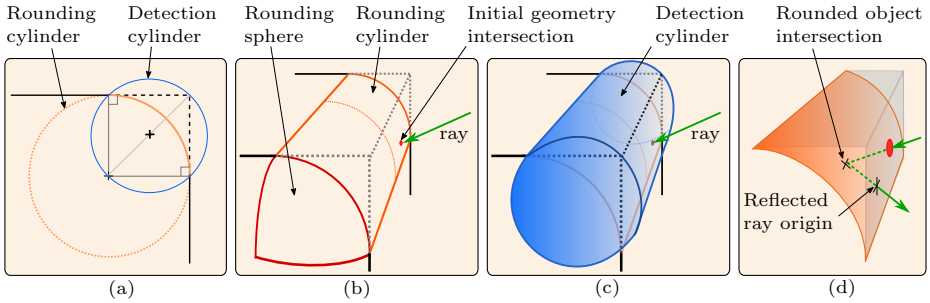


Fig. 2: (a) A rounding cylinder and a detection cylinder are associated to a rounded edge. (b) The ray tracing process intersects the original geometry of the 3D scene. (c) The corresponding intersection point is used to determine if its position lies within a detection cylinder using a Kd-tree structure. (d) If the intersection point lies within a detection cylinder, the intersection is computed with the corresponding rounding cylinder. If a reflected or shadow ray follows this intersection point, its origin is placed accordingly on the original geometry.

Note that in some cases, the ray does not intersect the rounding geometry (spheres and cylinders). The ray should continue its path through the scene, as explained in Section 4. We introduced a topological structure to allow rays to travel through rounding spheres and cylinders, and efficiently perform the intersection tests.

3 Rounding and Detecting

Positioning rounding spheres and cylinders at corners and edges of polygonal meshes has already been addressed [15,18]. This section briefly recalls the main principles and provides the notations used in the remainder of this article.

Using spheres for corners and cylinders for edges presents several advantages: (i) they correspond to smooth analytic primitives with well-known ray intersection processing, and (ii) continuity between surfaces, spheres, and cylinders is straightforwardly ensured for an identical radius [13].

Figure 3 illustrates the positioning methodology for cylinders. Let us consider a rounding cylinder of radius r (user-defined), associated with an edge (v_1, v_2), shared by two faces of normal $\boldsymbol{N_a}$ and $\boldsymbol{N_b}$ respectively. Let $\boldsymbol{E} = v_2 - v_1$ ($\boldsymbol{E'} = \boldsymbol{E}/|\boldsymbol{E}|$). The cylinder axis crosses the line defined by the half-vector direction $\boldsymbol{H} = -(\boldsymbol{N_a} + \boldsymbol{N_b})/|\boldsymbol{N_a} + \boldsymbol{N_b}|$. The axis of the rounding cylinder is defined by two points $p_1 = v_1 + d\boldsymbol{H}$ and $p_2 = v_2 + d\boldsymbol{H}$, with $d = r/\sin\alpha$.

As illustrated in Figure 4, rounding a corner consists in placing a sphere with the same radius r (as the cylinder), centered at the intersection between all

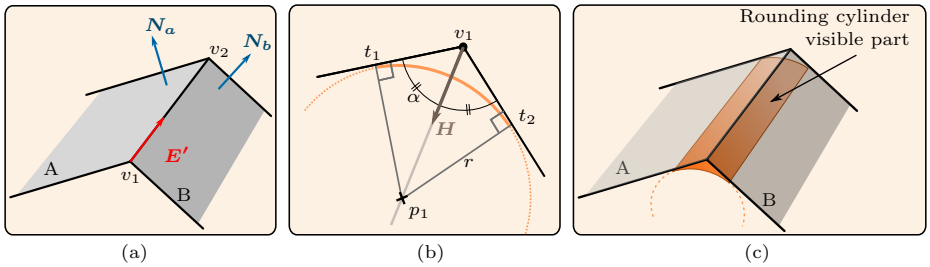


Fig. 3: A rounding cylinder for an edge (v_1, v_2) is associated with adjacent faces A and B. (a) A and B have respective normals N_a and N_b, $\boldsymbol{H} = -(N_a + N_b)/|N_a + N_b|$. (b) p_1 (resp. p_2) is defined from vertex v_1 (resp. v_2) and $\boldsymbol{H}$; (c) The visible area of the rounding cylinder defined by the axis (p_1, p_2), and the cylinder radius r is defined by the user.

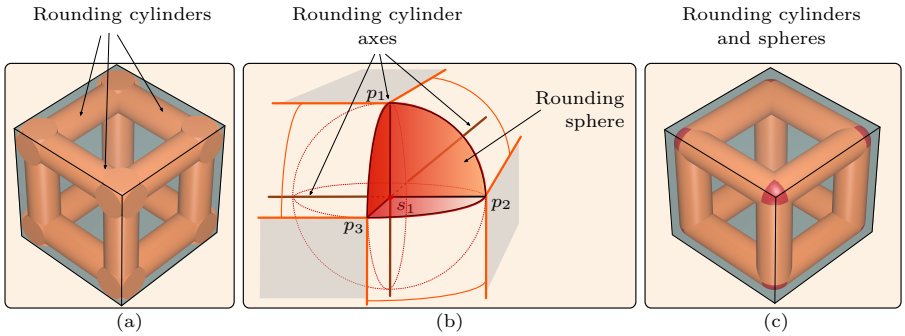


Fig. 4: (a) Set of rounding cylinders for a cube. (b) Axes of cylinders cross at a point, defining a rounding sphere center (s_1 is the sphere center associated with p_1, p_2, and p_3). (d) Final configuration with rounding spheres and rounding cylinders.

axes of adjacent rounding cylinders [13]: p_1 and p_2 are respectively replaced by the sphere centers s_1 and s_2, thus ensuring C^1 continuity between the rounding spheres, cylinders, and surfaces.

This representation is suitable for any convex object, whatever the number of edges incident to a vertex, as illustrated in Figure 5, for rounding spheres and cylinders having all the same radius. The rendering system can be applied from both sides of the chamfer (i.e., observed from outside or from inside a rounded object).

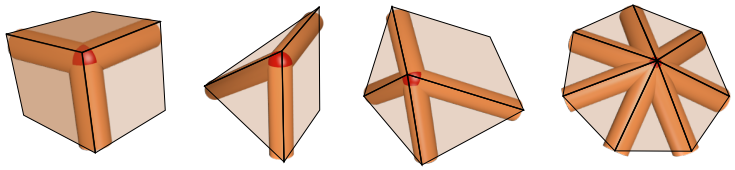


Fig. 5: Various configurations of positioning rounding cylinders, with different number of edges incident to one vertex.

Our goal is to perform robust and efficient ray intersection tests for ray/path tracing applications. The rounding primitives are placed *inside* an object geometry, that would not be reached by (exterior) rays. This is why an additional structure is mandatory for determining the intersection point and associated normal on the rounding primitive.

3.1 Detection Cylinders

The structure defined in this section aims at finding the potential rounding cylinder for a given intersection point on the original geometry. On a polygon, the area concerned by a chamfer corresponds to the region between a face edge and the beginning of the rounding cylinder. We propose to define a *detection cylinder* that precisely delimits this region (see Figure 6).

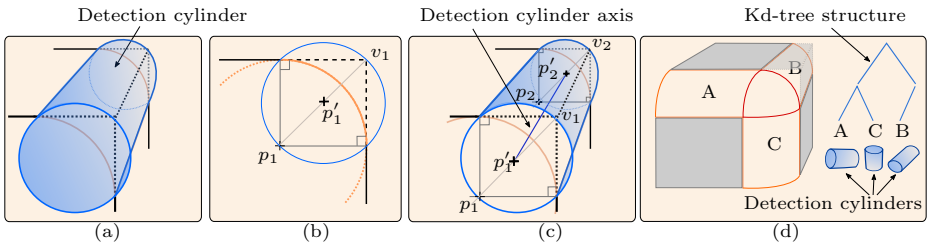


Fig. 6: Detection structure associated with three bevel areas: (a) A detection cylinder delimits the chamfer area. (b) The axis is placed so as to precisely fit the chamfer borders. (c) The representation in 3D. (d) Detection cylinders are organized into a Kd-tree.

The detection cylinder axis associated with an edge (v_1, v_2) is defined by two points $p'_1 = (v_1 + p_1)/2$ and $p'_2 = (v_2 + p_2)/2$ (as illustrated in Figure 6(c)). The detection cylinder radius r_d is equal to $|v_1 - p'_1|$.

One detection cylinder is associated to each edge; its central axis is parallel to the edge, and they are of equal length. These detection cylinders precisely delimit the region corresponding to the chamfer on the adjacent faces. Therefore, when a ray intersects a face at a 3D point I, a first test is performed to determine if I is within a detection cylinder (see Figure 7(a)). If so, the associated rounding cylinder is considered for intersection with the ray. An adjacency graph is also defined for managing grazing ray directions. A rounding cylinder is linked to its two end rounding spheres, and a rounding sphere is linked to all its connected rounding cylinders.

This detection structure has several advantages: (i) It completely contains and exactly fits the bevel region. (ii) It is defined by one axis and a radius. (iii) Determining if an intersection point is inside a cylinder is straightforward and fast. (iv) The adjacency graph allows finding the chamfer intersection along a series of rounded edges and corners traversals without renewing the search in the Kd-tree.

3.2 Kd-tree Structure

Because we aim at managing scenes with a large number of rounded edges, an acceleration structure is mandatory. We have chosen an organization of detection cylinders based on a Kd-tree, which is faster in this case than a bounding volume hierarchy (BVH) since the number of tests is smaller. We use a classical binary split along the longest axis, but any heuristic can be used (surface area heuristics, middle cut, etc.). The leaves contain the set of detection cylinders. This choice favors a fast construction with a balanced tree. In our implementation, a Kd-tree is constructed for each object or a group of simple objects, so as to benefit from moderate tree depths.

Our structure is independent from any path tracing structure, since it does not rely on the same geometry. When a ray-object intersection I is identified, it is located within the Kd-tree structure, and for each detection cylinder belonging to the corresponding Kd-tree leaf, the algorithm tests if I lies in the corresponding volume. If the test is positive, the intersection with the enclosed rounding cylinder can be performed, as explained below.

4 Ray tracing Process

Let us consider a scenario in path tracing, and the corresponding ray intersection process. The following method can be applied for paths issued from the observer or from light sources. Thus, it could be used in any modern rendering method based on stochastic paths construction: Photon mapping, bidirectional path tracing, Metropolis light transport, etc. For a given intersection point I on the original geometry, our method determines if such an intersection exists, and if so, computes the intersection point and associated normal off the rounding primitive.

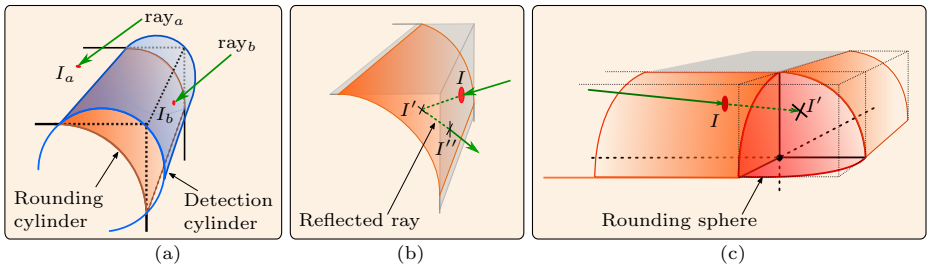


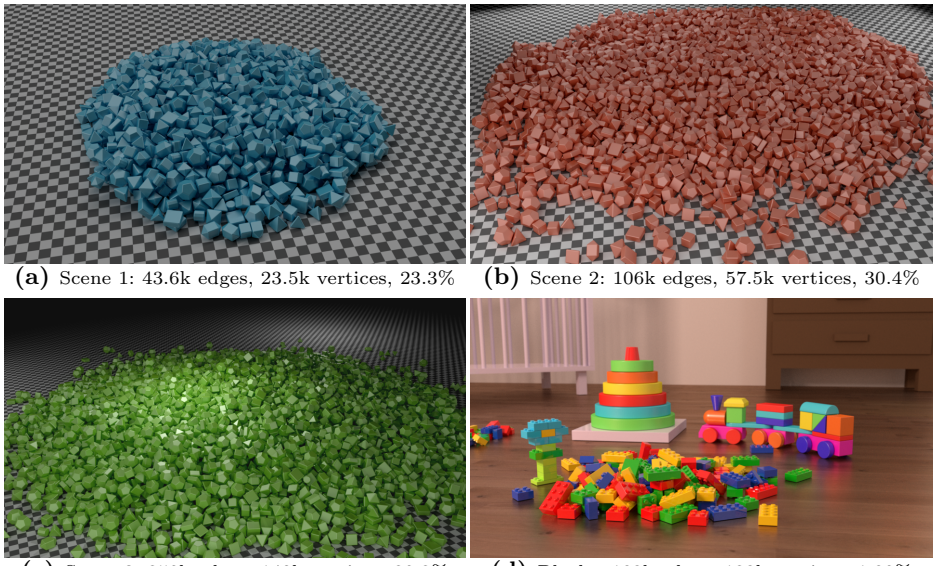
Fig. 7: Ray-object intersection using the original mesh. (a) With ray$_a$, the intersection I_a is located on the original geometry and outside all detection cylinders: the ray tracing process can be continued. For ray$_b$, I_b lies inside a detection cylinder and the intersection has to be tested with the rounding cylinder. (b) A ray intersects the rounding cylinder in I', and the reflected ray origin is set to I'' on the original mesh; (c) The ray does not hit the rounding cylinder, but the rounding sphere in I'.

Figure 7 illustrates the possible cases for an intersection point I. The Kd-tree identifies if I is located within a detection cylinder. If no detection cylinder is identified, I is directly used as the intersection point for the global illumination process (Figure 7(a)). Otherwise, every rounding cylinder containing I is used as a geometric primitive for ray intersection. Three configurations are possible:

- An intersection is found on the rounding cylinder, between the limits defined by s_1 and s_2. In this case, the resulting intersection point and its normal are used for further processing instead of I (Figure 7(b)).
- An intersection is found on the rounding cylinder between the limits defined by s_1 and p_1 (resp. s_2 and p_2). This area corresponds to a corner, and the rounding sphere is tested to determine the potential intersection (Figure 7(c)).

Given an intersection point I' on a rounding primitive (sphere or cylinder), the associated normal is computed. Since I' is located beneath the object surface, secondary rays must start on the original geometry mesh. They are first defined by their origin I' and a direction D fixed by the path tracer, but actually traced from an origin I'' located on the mesh surface. If no intersection is found on the rounding structure, the ray carries on its traversal in the adjacency structure.

5 Implementation and Results



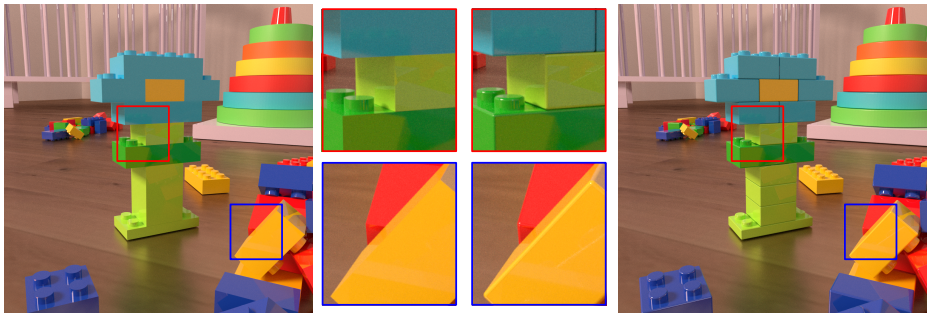
(a) Scene 1: 43.6k edges, 23.5k vertices, 23.3% **(b)** Scene 2: 106k edges, 57.5k vertices, 30.4%

(c) Scene 3: 259k edges, 140k vertices, 29.2% **(d)** Blocks: 182k edges, 122k vertices, 1.89%

Fig. 8: Number of rounded sphere/edge and percentage of rays impacted by chamfers for four test scenes. (a) Cylinders generated in 1.5 seconds, Kd-tree in 17 milliseconds; (b) cylinders generated in 3.6 seconds, Kd-tree in 44 milliseconds; (c) cylinders generated in 9.7 seconds, Kd-tree in 111 milliseconds; (d) cylinders generated in 9.5 seconds, Kd-tree in 0.004 milliseconds.

The results presented in this section have been computed on an Intel Core 17-4790 3.60 GHz processor, with 8 threads and 16 GB of RAM. The code is completely integrated into *Blender* [5] and *Cycles* renderer [6], in order to provide fair comparisons with existing bevel shaders and modifiers. All images and statistics are for a resolution of 1920×1080 with 512 paths per pixel.

Figure 8 illustrates the main scenes used in this paper, with worst cases corresponding to Figure 8(a)-(c), where many corners and edges are defined on small objects. In practice, the user selects a set of objects in *Blender*, and our process automatically places the rounding and detection cylinders for all the corners and edges, as well as build a Kd-tree structure per object.



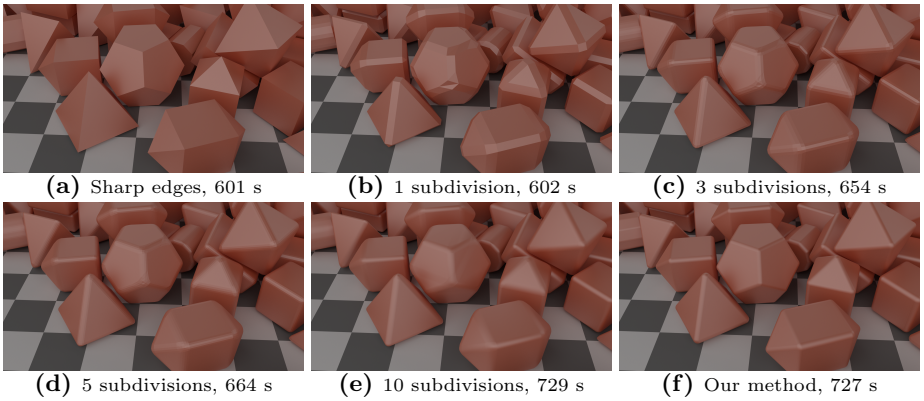
(a) 661 s (b) 691 s

Fig. 9: Without rounded edges, blocks look flat, contrary to the appearance of real blocks. The total time difference between the two images is less than 5%. In this example, 2.65% of rays are affected by rounded edges.

The data structure associated with corners and edges is provided in the supplementary material. Note that data for rounding spheres are not explicitly stored since all the necessary information is already contained in the cylinders data structure: rounding cylinders axes are defined by the two associated sphere centers.

When a ray-object intersection occurs with the original object, the object index is returned by the rendering system with the corresponding data. The index is used for accessing the correct Kd-tree and its associated rounding spheres and cylinders. Figure 9 shows the visual importance of rounded edges. Without rounded edges (Zoomed-in red frame), the individual blocks blend together, contrary to the clear demarcations (darker at junctions, brighter on highlights, between real blocks). Note also the highlights on the rounded knobs on top of each block. Even when observed from a distance (Zoomed-in blue frame), the edge representation greatly affects the perception of object shapes. The left image looks much flatter.

Figure 10 compares our method with standard rounding meshes, with various levels of subdivision for the rounded corners and edges. The rendering time for our method in this image corresponds to a subdivision level of 10, but with much less memory consumption (as shown in Table 1). In addition, our method is not prone to edge flickering when the light source (or viewpoint) moves, as shown on the video from the supplementary material.



(a) Sharp edges, 601 s **(b)** 1 subdivision, 602 s **(c)** 3 subdivisions, 654 s

(d) 5 subdivisions, 664 s **(e)** 10 subdivisions, 729 s **(f)** Our method, 727 s

Fig. 10: Close-up view of Scene 2 from Figure 8(b) for a comparison between our method and different levels of mesh subdivisions. Rounded corners and edges affect 28.9% of rays in this image.

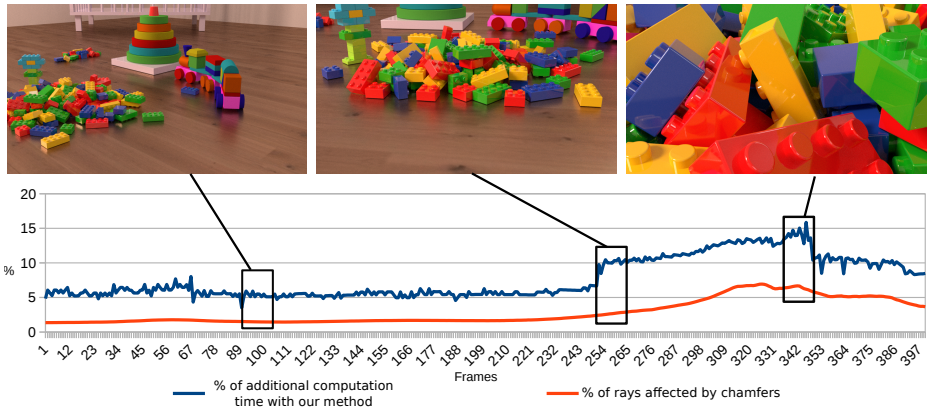


Fig. 11: Variations of computation time for a path traced animation of a moving viewpoint in the scene Blocks (Figure 8d. The red curve indicates the percentage of rays that hit chamfers, while the blue one shows the computation time increase.

Table 1 compares computation time and memory requirements for a number of configurations: our method, including all the rounding spheres and cylinders, detection cylinders, and the Kd-tree acceleration structure has only a minor impact on memory. In terms of computation time, the worst case we have observed concerns Scene 2, for which almost 30% of rays are affected by bevel corners and edges. In this case, our method is as fast as a subdivision of level 10, but the memory consumption is much lower.

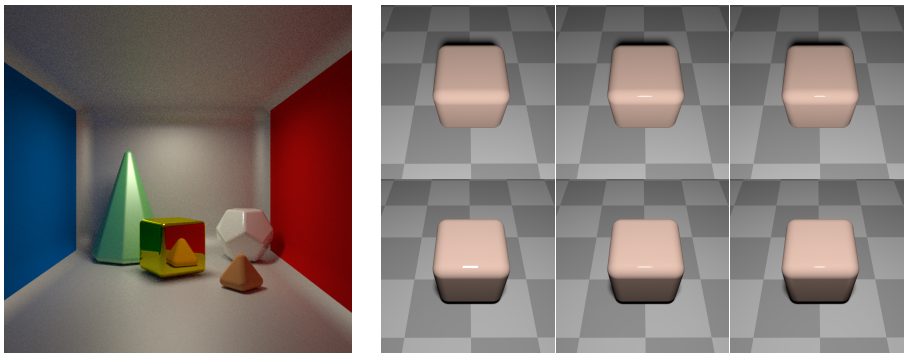
Using rounding cylinders instead of sharp edges requires additional computation time: respectively 3.2%, 1.9%, 34.5%, and 3.4% for Scenes 1, 2, 3, and Blocks. Our method is slightly faster than a subdivision of level 10 for the meshing method, but with far less memory requirements, and with a smoother appearance. The computation time highly depends on the number of rays that are affected by chamfers. This is why computation time increases very differently

for our tested scenes or depending on the viewpoint as illustrate in Figure 11 abstract from the video attached in additional material. Figure 12 illustrates internal rounded corners and comparisons between meshed chamfers and our method.

	Scene 1			Scene 2			Scene 3			Blocks		
	# tri	time	MB	# tri	time	MB	# tri	time	GB	# tri	time	MB
No rounding	42k	386s	128	103k	639s	478	252k	656s	2,42	276k	1010s	122
Our method	**42k**	**493s**	**132**	**103k**	**823s**	**487**	**252k**	**882s**	**2.44**	**276k**	**1050s**	**138**
1 subdivision	108k	396s	143	267k	686s	516	664k	695s	2.51	507k	1022s	182
3 subdivisions	364k	436s	201	909k	716s	662	2.26M	746s	2.86	1.01M	1029s	299
5 subdivisions	777k	452s	296	1.93M	746s	900	4.85M	793s	3.48	1.54M	1045s	439
10 subdivisions	2.49M	509s	690	6.24M	839s	1889	15.6M	909s	5.96	2.9M	1068s	780
Bevel shader	42k	628s	128	103k	1106s	478	252k	1148s	2.42	276k	1585s	122

Table 1: Number of triangles, rendering time, and memory consumption for the four test scenes of Figure 8.

Memory consumption with our method is much lower than mesh subdivision. Bevel shaders also have very little impact on memory, but the rendering quality fails in many cases. This is due to the normal interpolation, not always adapted to geometric configurations, and that does not account for volume removal due to chamfering.



(a) Internal corners (b) Flat mesh (c) Our method (d) Norm interp.

Fig. 12: (a) A Cornell box with rounded corners, containing objects chamfered with our method. The rounded corners and edges of the Cornell box itself seen from the interior. (b), (c), and (d) Two frames from a video provided in supplementary material. (b) Subdivision of level 10 without interpolation of normals. (c) Our method. (d) Subdivision of level 10 with interpolation of normals.

Our methods offers a simple control on the chamfer radius, with a constant the visual quality when observed from any viewpoint in the scene. It also can be easily integrated in an existing rendering system based on ray sampling.

6 Conclusion and Future Work

This paper presents a method dedicated to the efficient rendering of rounded corners and edges on convex objects with ray tracing based renderers. The process consists in automatically positioning rounding spheres and cylinders of a

given radius within objects. It does not affect the object original geometry and does not require much change to the rendering system.

A detection and acceleration structure allows us to determine whether an intersection point is located on a rounded corner or edge. The actual intersection on the rounded primitive is performed only if required, making the method robust, with a small memory consumption compared to regular and subdivision meshes, while offering a simple control.

Our method does not account for nonconvex objects because convexity changes around a corner imply complex changes of curvature that cannot be handled by a simple sphere.

In the future, we aim at improving the method with arbitrary geometric configurations while maintaining a C^1 continuity. Some technical improvements could also reduce computation times, because for instance parts of the method have not been implemented using SSE instructions.

References

1. Autodesk: 3DS Max chamfer modifier. www.help.autodesk.com
2. Blender Foundation: Bevel geometry tool in Blender. www.docs.blender.org/manual
3. Blender Foundation: Blender shader for round edges. www.docs.blender.org/manual
4. Blender Foundation: Subdivision tool in Blender. www.docs.blender.org/manual
5. Blender Foundation: Blender. www.blender.org (2018)
6. Blender Foundation: Cycles. www.cycles-renderer.org (2018)
7. Catmull, E., Clark, J.: Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design **10**(6), 350–355 (1978)
8. Chaos group: Corona round-edges shader. www.coronarenderer.freshdesk.com
9. Chaos group: V-RAY round-edges shader. www.docs.chaosgroup.com.
10. Chiyokura, H.: An extended rounding operation for modeling solids with free-form surfaces. In: Computer graphics 1987, pp. 249–268. Springer (1987)
11. Kajiya, J.T.: The rendering equation. In: ACM Siggraph Computer Graphics. vol. 20, pp. 143–150. ACM (1986)
12. Loubet, G., Neyret, F.: Hybrid mesh-volume LoDs for all-scale pre-filtering of complex 3D assets. In: Computer Graphics Forum. vol. 36, pp. 431–442. Wiley Online Library (2017)
13. Max, N.: Cone-spheres. In: ACM SIGGRAPH Computer Graphics. vol. 24, pp. 59–62. ACM (1990)
14. Pharr, M., Jakob, W., Humphreys, G.: Physically based rendering: From theory to implementation. Morgan Kaufmann (2016)
15. Saito, T., Shinya, M., Takahashi, T.: Highlighting rounded edges. In: New Advances in Computer Graphics, pp. 613–629. Springer (1989)
16. Stam, J., Loop, C.: Quad/triangle subdivision. In: Computer Graphics Forum. vol. 22, pp. 79–85. Wiley Online Library (2003)
17. Szilvasi-Nagy, M.: Flexible rounding operation for polyhedra. Computer-Aided Design **23**(9), 629–633 (1991)
18. Tanaka, T., Takahashi, T.: Precise rendering method for exact anti-aliasing and highlighting. The Visual Computer **8**(5), 315–326 (1992)
19. Wei, L.Y., Shi, K.L., Yong, J.H.: Rendering chamfering structures of sharp edges. The Visual Computer **31**(11), 1511–1519 (2015)