Efficient accelerated ray tracing of point-based objects

Ali Beddiaf*
Department of computer science
University Hadj Lakhdar of Batna
05000 Batna, Algeria
Email: alibeddiaf@gmail.com

Med Chaouki Babahenini
LESIA Laboratory, Department of computer science
University Mohamed Khider of Biskra
07000 Biskra, Algeria
Email: chaouki.babahenini@gmail.com

***Abstract*—In this paper, we present a ray tracing technique that uses a set of accelerations in behalf of rendering point-based scenes efficiently. Our model has been accelerated by geometrically optimized intersection computations in addition to the use of bounding volumes and spatial subdivision. the main goal is to get the highest possible frame rate and this by reducing the intersections computation time.**

***Keywords*—Point cloud, efficient rendering, ray tracing, bounding volumes, spatial subdivision.**

I. Introduction

The problem of illumination is considered as the final stage in the rendering process for any graphics project. It concerns the interactions between the objects of the scene with the light sources. This may be done locally where the surface of the objects contains itself information about the portion of the light that it receives and the final image will be simply the projection from the object space to the screen space like flat, Gouraud and Phong shading. Or globally, here more complicated illumination has to be computed because both direct and indirect shading are taken into account. Ray tracing is one of the techniques that simulates the real radiant intensity exchange between surfaces of the scene.
Point as a primitive of representation has been introduced firstly by Levoy and Whitted [1]. Then an important local illumination model has been proposed by Zwicker et al. [2] where they replaced each point by a splat. The efficiency of this last has been proven, however for more realistic rendering, a global illumination model must be found. Various approaches based ray tracing have been proposed in this context [3] [4] but we focus in this work on the efficiency of the splat-based ray tracing implementation by fitting the classical techniques of spatial subdivision, bounding volumes and geometrical optimizations to the particular case of point clouds.

II. Related work

Phong splatting has been proposed initially by Botsch et al. [5] in order to extend surface splatting introduced by Zwicker et al. [2], with the acoustic effects (specular reflection, transparency and shininess). These acoustic effects were generated through what Botsh et al. [5] call the normal field. Each splat has to be associated with additional information: normal field. This last has to be computed in preprocessing step basing on the principal curvatures of subsets of point cloud. The result will be: the central normal, two principal directions $(\vec{u}, \vec{v})$ and two coefficients (α, β), for each splat!. So the Phong splatting approach is used as part of a local illumination model where the final image is simply projected to the screen space. Recently the normal field has been reused for the global illumination case by Linsen et al. [3], and the main goal was still to give a normal as accurate as possible along the splats surface of the object. Linsen et al. [3] have proposed a splat-based ray tracing for point clouds where they reuse the normal field technique as a preprocessing step. The way they weight the attributes (normals) of the overlapped splats is the same of Schaufler et al. [4]. The computations of the ray-splat intersection have been sped up through an octree data structure.
Another data structure has been used to accelerate the rendering of the point-based objects by Rusinkiewicz et al. [6], it is a bounding sphere hierarchy. The context in which this technique has been proposed was the local illumination (rasterization).

III. Proposed model

Note that the input data in our illumination model is simply discrete points with only low density. Each point has a position, a normal and an initial size. Each one will represent one circular splat.

A. Intersections computation

As mentioned by previous works [3], the intersection computation in the ray tracing algorithm on splats needs some adaptation because the splats are overlapped. Therefore, there is a set of wrong intersections (Fig 1) that must be ignored during the computing of the intersections ray/splat, as follows:

a) Wrong intersection
b) Wrong reflection
c) Wrong refraction

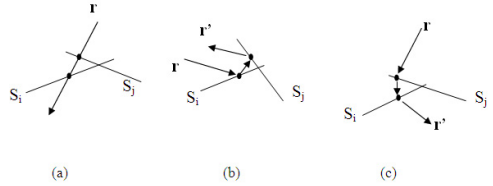


Fig. 1. Wrong intersections: (a) intersection (b) reflection (c) refraction.

978-1-4799-3824-7/14/$31.00 ©2014 IEEE

B. Bounding volumes

Ray/object intersection is one of the first problems often encountered when we want to make a ray tracer. One of the simplest and fastest intersections is the ray/sphere one. In this purpose, we use a sphere as bounding volume of the splat (Fig 2). With this way, we propose to compute the intersection

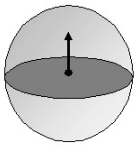


Fig. 2. Sphere bounding a splat.

at two levels:

1. Ray/sphere intersection
   This will avoid the ray/splat intersection computation when this test fails (Fig 3).
2. Ray/splat intersection
   When the incoming ray intersects the sphere, here we look if the ray also intersects the splat (Fig 3).

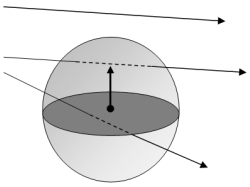


Fig. 3. Rays may intersect the sphere, the splat, both or no one.

C. Ray/sphere intersection

We consider the ray R(t) formulated in parametric way, as follows:

$$R(t) = (o, \vec{d}) = o + t\vec{d} \tag{1}$$

Where o is the origin of the ray, $\vec{d}$ is its direction and $t > 0$ (Fig 4).

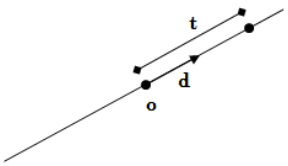


Fig. 4. Parametric ray.

Also, the implicit equation of a sphere is:

$$(S_x - c_x)^2 + (S_y - c_y)^2 + (S_z - c_z)^2 = r^2 \tag{2}$$

Where $c(c_x, c_y, c_z)$ is the center of the sphere, r is its radius, and $S(S_x, S_y, S_z)$ are points that lie on the sphere surface.

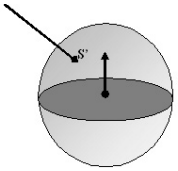


Fig. 5. Implicit sphere.

The first intersection $S' = R(T')$ between the ray and the sphere (Fig 5) will be found through the solving of system of equations in order to find T' that is given as following:

$$T' = min(max(t0, 0), max(t1, 0)) \tag{3}$$

Where t0 and t1 are:

$$\begin{aligned} t0 &= \frac{-B - \sqrt{B^2 - 4AC}}{2A} \\ t1 &= \frac{-B + \sqrt{B^2 - 4AC}}{2A} \end{aligned} \tag{4}$$

A, B and C are constant:

$$\begin{aligned} A &= d_x^2 + d_y^2 + d_z^2 \\ B &= 2(d_x(o_x - c_x) + d_y(o_y - c_y) + d_z(o_z - c_z)) \\ C &= (o_x - c_x)^2 + (o_y - c_y)^2 + (o_z - c_z)^2 - r^2 \end{aligned} \tag{5}$$

D. Ray/splat intersection

In the case of ray/sphere intersection finding (called S'), we test if this ray intersects the splat or not. Starting from the intersection point S', we look for the intersection point P' between the ray $(S', \vec{d})$ and the plane P of the splat. Then, if the distance between P' and c is less or equal than r, here we can say finally that the ray $(o, \vec{d})$ intersects the splat on the point P' (Fig 6).

In order to compute the intersection ray/plane, we let $P(x, y, z)$ the points set that lies on the plane defined by the point c (sphere/splat center) and the normal $\vec{N}$. The following equation must be verified:

$$\forall p \in P : (p - c).\vec{N} = 0 \tag{6}$$

Similarly, by solving a system of equations (equations 1+6), the solution P' will be:

$$\begin{aligned} P' &= S' + T'\vec{d} \\ T' &= \frac{(c - S').\vec{N}}{\vec{d}.\vec{N}} \end{aligned} \tag{7}$$

E. Geometrical optimizations

In addition to the ray/splat intersection computation avoidance, the bounding sphere has another major interest which is the ability of rejecting a large set of incoming rays that does not have a geometrical chance to hit the sphere, without computing the intersections.

a) Is the ray origin inside the sphere? (Fig 7)
b) The ray origin is outside the sphere. Is the distance t0 negative? If yes, there's no intersection (Fig 8).

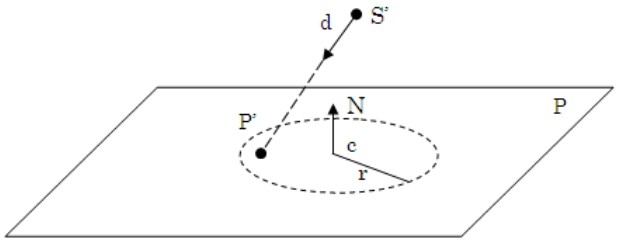


Fig. 6. A plane.

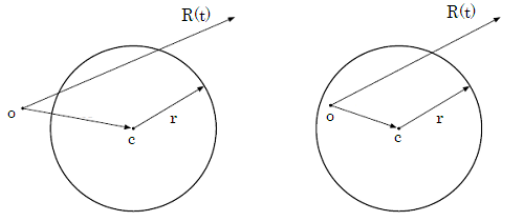


Fig. 7. Ray origin is outside the sphere (left) and inside the sphere (right).

c) The ray origin is outside the sphere. Is the square distance t1 negative? If yes, there's no intersection Otherwise, there's a ray/sphere intersection (Fig 9).

Similarly with the ray/plane (ray/splat) intersections, we apply a geometrical optimization to avoid probably ray/plane intersection computations.

a) Is the ray perpendicular with the plane normal? If yes, there's no intersection (Fig 10).

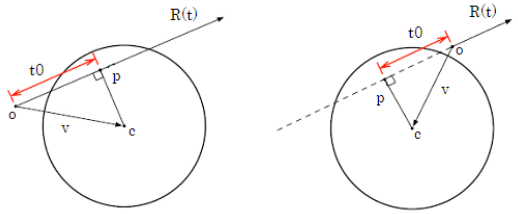


Fig. 8. Distance t0 is positive (left) and negative (right).

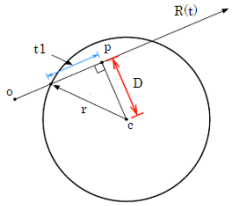


Fig. 9. Distance t1 have to be positive when the ray intersects the sphere.

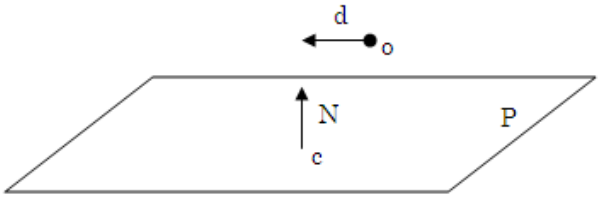


Fig. 10. Ray is perpenduclar with the plane normal.

F. Normal interpolation

As mentioned, the splats have an initial size and an overlap will appear between this splats. The way we estimate the normal of the intersection is the same to Schaulfer's way [4], in another words, the final normal will be the interpolation of the overlapped splats normals (Fig 11). This interpolation is weighted according the distance between the intersection points and their splats centers, as follows:

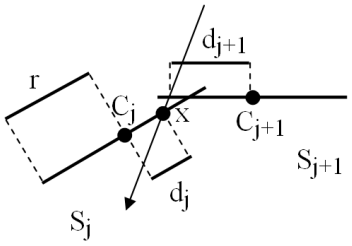


Fig. 11. Polynomial interpolation of the normal.

$$Normal_x = \sum_{i=0}^{n} \frac{Normal_i.d_i}{r}$$
$$d_i = (r - c_i) \tag{8}$$

G. Spatial subdivision

Although the bounding volume and the intersection detection, the rendering time is still disappointing. For this reason we apply spatial subdivision to further improve the intersections computation time. Many subdivision schemes have been used to accelerate the ray tracer such as: uniform grid, octree, BSP, Kd-Tree. However the performance of these schemes depends finally to scene complexity (free and occupied regions). We choose for our work the uniform subdivision scheme because of its fast traversal algorithm (Fig 12); however a more complicated scheme could be applied.

We fix the threshold of subdivision of the grid depending to the radius of the splats, precisely the cell size will be twice of the splats radius. We justify this choice by two reasons; the first one is to reduce the time of grid filling (one splat could belong at most to no more than eight cells), and the second one is to avoid having cells occupied by a lot of splats (Fig 13).

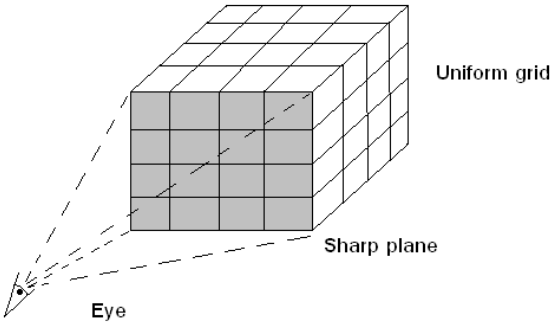


Fig. 12. Uniform subdivision of the scene.

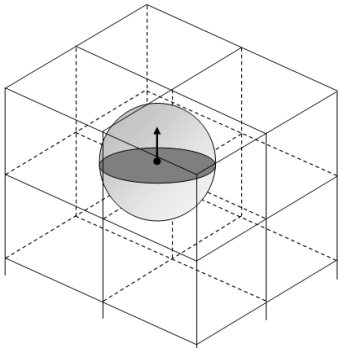


Fig. 13. Eight cells filled by one splat.

TABLE I. RENDERING TIME OF THREE OBJECTS USING DIFFERENT ACCELERATIONS.

Objects	Sphere	Bunny	Rabbit
All the accelerations	3sec.	7sec.	5sec.
Without uniform grid	17sec.	5min. 20sec.	9min. 10sec.
Only bounding sphere	20sec.	6min. 10sec.	10min. 54sec.
Without any acceleration	1min. 40sec.	1h. 10min.	54min. 40sec.

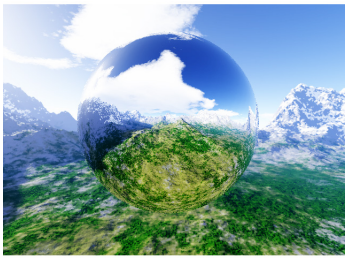
IV. EXPERIMENTAL RESULTS

As depicted in Fig 14 the splat-based ray tracing (with only one-object and one-bounce) has generated images whose dimension is 640x480 pixels.

The following table (Table I) summarizes the different configurations used to evaluate our ray tracer in matter of rendering time by applying the accelerations. It's also shown graphically in Fig 15.

We can see clearly how much the bounding sphere has sped up the rendering time in the three scenes compared to the basic ray tracing, however the geometrical optimization does not change significantly the performance. Also, the spatial subdivision further accelerated the rendering till having a rendering time only between 3 and 7 seconds for the three objects. Comparing the basic ray tracer with the accelerated one, we can observe that the latter is faster than the former 33, 600 and 656 times according to the rendering time of the sphere, the bunny and the rabbit respectively. We can also see that the basic ray tracing of the bunny (35945 points) took more time compared to the rabbit (70658 points) and this is justified by the large occupied regions according to the current point of view (Fig 14).

(a)



(b)

(c)

Fig. 14. Images produced by our ray tracer: (a) A bunny. (b) A mirror sphere. (c) A rabbit.

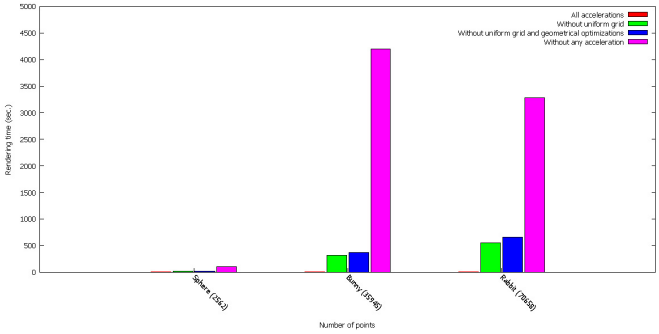


Fig. 15. Graph showing the rendering time of three different objects using different accelerations.

V. CONCLUSION AND FUTURE WORK

We have proposed an efficient ray tracing approach for splat-based objects by fitting the techniques of spatial subdivision, geometrical optimizations and bounding volumes to the particular case of the 3D objects represented by point clouds. We can say that the proposed accelerations made the

ray tracer many times faster than the basic one. In perspective, we recommend the evaluation of the proposed accelerated ray tracing for more complex scenes (many objects) with multi-bounce. Despite all the proposed accelerations of the intersections computation, this work stays open for GPU-based accelerations [7].

REFERENCES

[1] M. Levoy and T. Whitted, *The Use of Points as a Display Primitive*, ser. UNC report. University of North Carolina, Department of Computer Science, 1985.

[2] M. Zwicker, H. Pfister, J. van Baar, and M. H. Gross, "Surface splatting." in *SIGGRAPH*, 2001, pp. 371–378.

[3] L. Linsen, K. Muller, and P. Rosenthal, "Splat-based ray tracing of point clouds," *Journal of WSCG*, vol. 15, no. 1-3, p. 5158, 2007.

[4] G. Schaufler and H. W. Jensen, "Ray tracing point sampled geometry," in *In Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, 2000, pp. 319–328.

[5] M. Botsch, M. Spernat, and L. Kobbelt, "Phong splatting," in *Proceedings of the First Eurographics conference on Point-Based Graphics*, ser. SPBG'04. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2004, pp. 25–32.

[6] S. Rusinkiewicz and M. Levoy, "Qsplat: a multiresolution point rendering system for large meshes," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 343–352.

[7] S. Kashyap, R. Goradia, P. Chaudhuri, and S. Chandran, "Real time ray tracing of point-based models," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games: Posters*, ser. I3D '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:1.