

Interactive Computer Graphics 2017 Fall Term Project

Speeding Up Ray-Tracing by BSMP and Scale Lights

R05942039 HungWei-Hsu

I. Introduction

Ray-tracing renders an image by tracing the path of lights emitted from pixels of the camera and the effects of encounters with virtual objects. Ray-tracing succeeds in simulating smooth and reflective surfaces. Yet, ray-tracing takes high temporal costs, which becomes a burden toward real-time rendering.

This term project proposes two improvements to reduce the time complexity of ray-tracing. First, BSMP (Binary Space Medium Partitioning) is proposed to improve the original BSP(Binary Space Partitioning) on conditions where objects are concentrated on certain partitions. Second, scale light is proposed to replace ray for better time efficiency on conditions where occlusion occurs among partitions. Finally, eleven images are tested to prove speed-up by 10% - 50%.

II. Related Works

1. Ray-Tracing

Ray-tracing renders images by emitting rays from camera pixels and simulate the effects of encounters with surfaces. Once a ray is incurred on some surface, three types of rays will be produced: (1) reflective light (2) refractive light (3) Shadow light. Then reflective rays and refractive rays are recursively emitted and weighted and summed as the pixel values on the camera.

The algorithm can be summarized as follows:

```
Color intensity(Ray r, int depth)
    IF depth >= 5
        RETURN Black;
    ELSE
        object = INTERSECT(r);
        IF no intersection
            RETURN background;
        ELSE
            Color reflect_color = intensity(reflective ray, depth + 1);
            Color refract_color = intensity(refractive ray, depth + 1);
            Color direct_color;
            Check for shadow
            IF (shadow)
                direct_color = ambient light;
            ELSE
                direct_color = phong illumination model;
            RETURN direct_color + reflect_color + direct_color;
```

From the pseudo code above, the time complexity of emission of one ray is $O(T(\text{INTERSECT}) * 2^{\text{MAX_DEPTH}})$. The bottleneck lies on finding the intersection of one ray with all objects. The simplest method is to enumerate all objects and check whether each object will intersect with the ray. On this condition, $T(\text{INTERSECT}) = O(\text{number of objects})$.

2. Binary Space Partitioning (BSP)

BSP is one method to reduce $T(\text{INTERSECT})$. The core idea is “Bounding Volume”. That is, we group all objects into several sets. Once a ray is emitted, we check on sets and remove those impossible sets and then find the intersection of the ray with possible sets. The set can be hierarchical.

BSP is to group objects by cut the space on each axis by half. That is, the space is cut by half **by the center value** on each axis and all objects is grouped into 2^D groups where D is the dimension of the space. The partitioning can be recursive. That is, in each group, we can further partition it into 2^D subgroups.

BSP is efficient when objects are averagely scattered on the space. However, since the grouping method is based on the spatial relationship independent of the objects distribution. Once objects are concentrated on one small partition of space and only few objects are outliers, BSP is prone to the problem of removing only groups of few objects with additional overhead of building and searching trees.

III. Binary Space Medium Partitioning(BSMP)

To solve the problem, we propose one improvement which separates each group by **the median of the objects** instead of **the center value** on the axis. We call this partitioning method Binary Space Medium Partitioning, or BSMP.

This partitioning method is based on the object distribution. It can ensure each group has similar number of objects. Once the objects are averagely scattered on space, BSMP is similar to BSP. Yet, once the objects lie primarily on specific partition, BSMP has better grouping methods than BSP.

For example, for the figure below, BSP can remove group of only two objects, BSMP can remove groups of 8 objects.

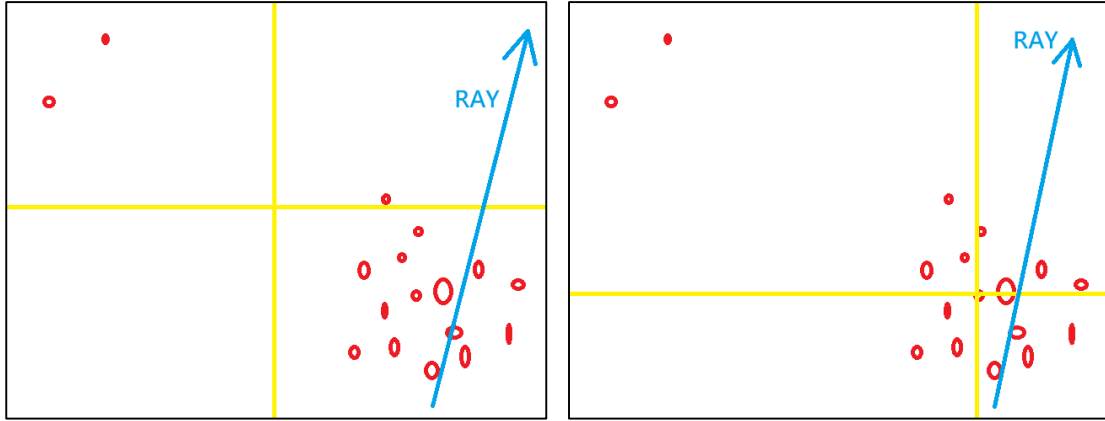


Fig1. Left: BSP, right: BSMP. Red circles are objects, blue arrow is the ray, yellow lines are boundaries of partitions.

IV. Scale Lights

Another improvement is the scale lights. In original ray-tracing, groups are intersected with one ray. However, if a ray will intersect multiple groups, it can intersect with nearer groups; once some groups are found intersections with objects inside, the farther groups need not be intersected.

Based on this facts, we emit the lights with segment instead of a ray. Then we find groups which have intersections with this segment and find intersection of “the ray” with objects inside. Once there are objects intersected with the objects, we return the intersection. Otherwise, we stretch the segment by 2 times length and find intersections again until the maximum segment length is reached.

To be noticed, we first find intersections of **groups with segments** and then find intersections of **objects with rays**. In addition, we mark every group with a flag searched. Once a group is searched, the flag is set as true. And next time even though the segment is found intersection with the group, the group will be skipped. These two steps are to prevent objects inside the same group from being checked twice.

The time complexity of scale lights is at most the time to search groups where the ray is intersected. Therefore, the time to search intersection with objects in scale lights is smaller than or equal to the time without scale lights. The overhead is the time to stretch the segment. Suppose the maximum segment length is 10^6 , the overhead is only 20 times of stretching. Therefore, it's a good deal overall.

The algorithm can be summarized as the pseudo code below:

```

Intersection emit(ray, bsmpt, initSegmentLength, maxSegmentLength)
    bsmpt.clear_search_flags()

    Segment seg;
    FloatingNumber segLength = initSegmentLength;
    seg.origin = ray.origin;
    seg.dest = seg.origin + ray.direction * segLength;

    Intersection intersect = bsmpt.find_intersection(ray, seg);
    WHILE segLength < maxSegmentLength AND intersect == NULL
        segLength *= 2;
        seg.origin = seg.dest;
        seg.dest = seg.origin + ray.direction * segLength;
    RETURN intersect

void bsmpt::clear_search_flags()
    this->search_flag = false;
    IF left_child is not NULL
        Left_child->clear_search_flags();
    IF right_child is not NULL
        Right_child->clear_search_flags();

Intersection bsmpt::find_intersection(ray, seg)
    IF not intersect(this) OR this->search_flag
        RETURN NO_INTERSECTION

    IF Left_child is not NULL AND Right_child is not NULL
        Intersection left_intersect = left_child->find_intersection(ray, seg);
        Intersection right_intersect = right_child->find_intersection(ray, seg);

        IF left_intersect.distance < right_intersect.distance
            RETURN left_intersect;
        ELSE
            RETURN right_intersect
    ELSE
        Intersection inter;
        FOR object IN this->objects
            Intersection tmp_inter = intersection(object, ray);
            IF tmp_inter.distance < inter.distance
                Inter = tmp_inter
        RETURN inter

```

TABLE I

	Baseline (RayTracing + BSP)	Segment_s2	%	Segment_s4096	%	Median	%
Fig 1	33.36	31.248	-6.33	22.556	-32.39	33.484	0.37
Fig 2	2375.06	1562.48	-34.21	1586.864	-33.19	6821.82	187.23
Fig 3	1961.64	1370.796	-30.12	1349.756	-31.19	1255.328	-36.01
Fig 4	412.88	274.492	-33.52	250.204	-39.4	178.028	-56.88
Fig 5	11.132	11.888	6.79	6.996	-37.15	10.88	-2.26
Fig 6	9.364	9.832	5	5.424	-42.08	9.288	-0.81
Fig 7	7.928	8.376	5.65	4.596	-42.03	7.924	-0.05
Fig 8	290.52	215.044	-25.98	190.72	-34.35	632.296	117.64
Fig 9	17.82	19.392	8.82	12.804	-28.15	18.208	2.18
Fig 10	15.188	14.96	-1.5	9.22	-39.29	14.78	-2.69
Fig 11	1062.484	733.924	-30.92	619.7	-41.67	745.084	-29.87
Total	6197.376	4252.432	-31.38	4058.84	-34.51	9727.12	56.96
Total – (Fig2, Fig 8)	3531.796	2474.908	-29.92	2281.256	-35.41	2273.004	-35.64

	Segment_Median_s2	%	Segment_Median_s4096	%
Fig 1	28.932	-13.27	24.128	-27.67
Fig 2	3499.58	47.35	5766.352	142.79
Fig 3	1018.364	-48.09	1298.964	-33.78
Fig 4	147.14	-64.36	235.3	-43.01
Fig 5	12.208	9.67	6.884	-38.16
Fig 6	9.76	4.23	5.336	-43.02
Fig 7	8.584	8.27	4.572	-42.33
Fig 8	412.908	42.13	399.164	37.4
Fig 9	19.492	9.38	12.612	-29.23
Fig 10	14.436	-4.95	9.14	-39.82
Fig 11	340.952	-67.91	506.456	-52.33
Total	5512.356	-11.05	8268.908	33.43
Total – (Fig2, Fig 8)	1599.868	-54.7	2103.392	-40.44

V. Experiments

We choose 11 scene to render. The rendered figures are as Table II. The elapsed time is listed in Table I. In scale lights, the maximum segment length is set as 40000. Once there is not intersection found with any group for segments, a ray will be emitted to prevent lost groups.

In addition, we observe that for fig2 and fig8, the balls are averagely scattered. In that conditions, BSMP will not help. On the contrary, the overhead to search medium will increase the temporal cost.

From the table, the scale lights can save around 30% temporal cost. Excluding fig2 and fig8, BSMP can save around 35%. Combining BSMP with scale lights can save around 54.7 %.

VI. Conclusion

We propose two improvements on our baseline RayTracing with BSP. One improvement BSMP is based on partitioning the space by the medium of the space instead of the center value. The other improvement scale light is to emit a segment to intersect groups instead one ray. We test our result on 11 images. Scale lights shows stable improvements around 30% decrease on temporal cost. BSMP depends on the data distribution. Together BSMP and scale lights show a improvements around 50 %.

VII. Website

https://github.com/HungWei-Andy/RayTracing_BSMP_ScaleLights

TABLE II

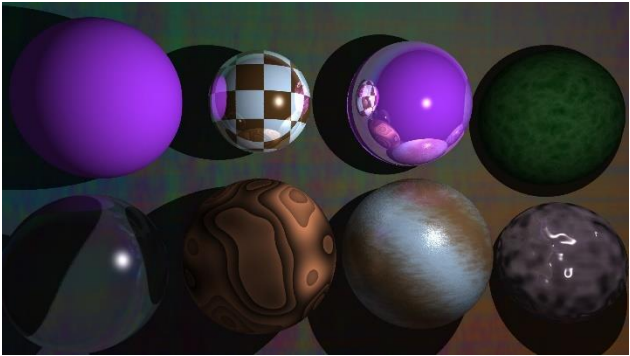
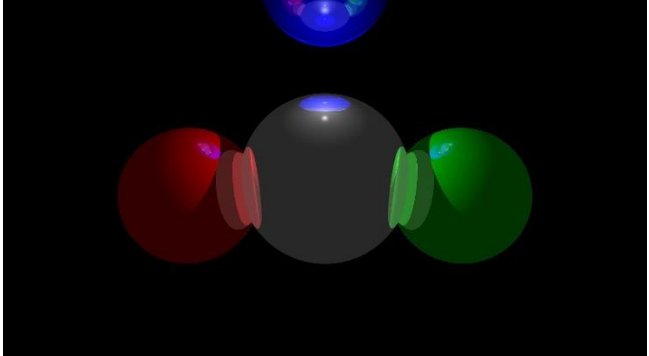
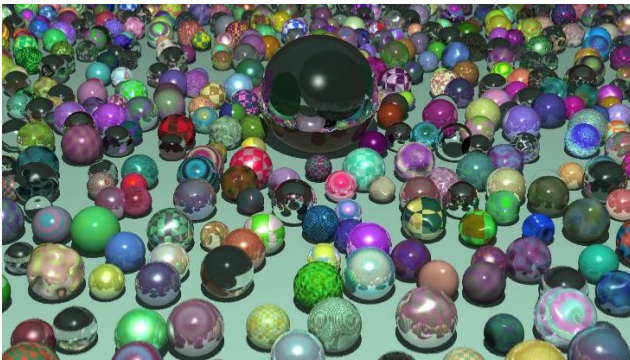

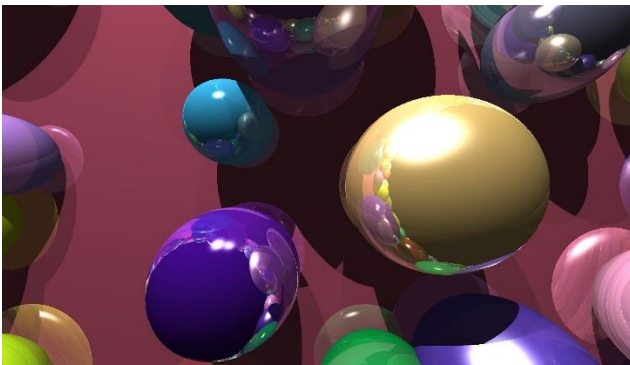



Fig1		Fig7	
Fig2		Fig8	
Fig3		Fig9	
Fig4		Fig10	

Fig5

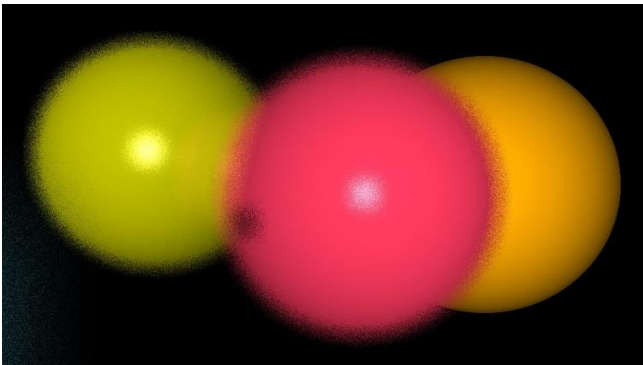


Fig11

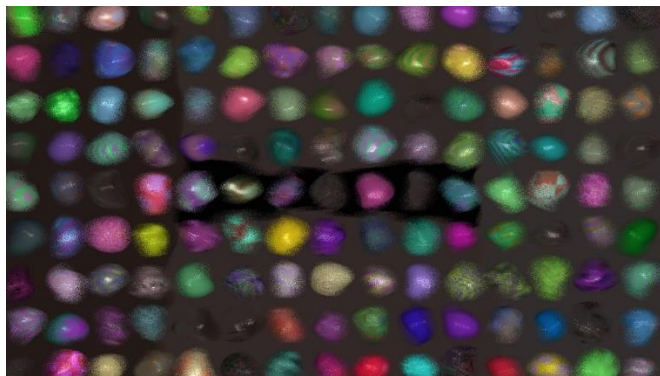


Fig6

