ADVANCED GRAPHICS – 2018/2019

January 31st – 13.30 – 15.30 – RUPPERT-BLAUW

*Please write clearly. Please do not ask for clarification during the exam. If you find a question unclear or ambiguous: write down how you interpret the question, then answer it. For each question you can score up to 10 points, your grade is points * 9 / 70 + 1.*

IMPORTANCE

1. When using Russian roulette, the recommended formula for survival probability is:

$$P_{survive} = min(\,1, max(\,red, green, blue\,)\,)$$

An alternative survival probability is:

$$P_{survive} = min(\,1, (red\; + \;green\; + \;blue)\;/\;3\,)$$

Explain why and how the first formula is better.

The first one has the interesting effect that (0.8,0,0) becomes (1,0,0) if the path survives. The second one has the problem that color components may become arbitrarily large, which shows up as fireflies.

2. Explain why the following techniques can be seen as importance sampling techniques:

a) Next Event Estimation

NEE: sending 50% of the rays to the lights, while the lights typically do not take up 50% of the hemisphere accounts for the fact that the lights do contribute disproportionally to the irradiance.

b) Russian Roulette

RR: focusing effort on paths that contribute significantly, while suppressing less relevant paths.

c) For dielectrics: basing the probability of generating a reflection on the Fresnel term

Fresnel: if the reflection is important, we sample it more often.

Note, to be sure: I am not asking for a description of the techniques.

ACCELERATION STRUCTURES

3. Why is it often advantageous to have empty leaves in a kD-tree, and why do we not have these in a BVH?

Rays traversing empty space do not test primitives, which is good. So splitting a leaf may prevent primitive tests. In a BVH, this happens implicitly, as the bounds enclose the primitives, leaving empty space between the bounds.

4. We know that a BVH constructed using the SAH can be traversed more efficiently than one constructed using midpoint splits. But, why is this so?

The SAH leads to a tree that consists of AABBs with a low summed surface area, which directly affects the probability of a ray hitting these nodes.

BIAS AND APPROXIMATIONS

5. The solid angle of a planar area light source over the hemisphere of point p is approximately

$$\frac{A_{light}\cos\theta_{light}}{d^2},$$

The projection of a flat surface on the curved hemisphere is always smaller than the area of the flat surface itself. The error can be arbitrarily large for very large lights.

where d is the distance between p and the light, A_{light} is the surface area of the light source and $\cos\theta_{light}$ is the dot product between the light normal and a normalized vector from the light to p.

By now, it is well-known that this is an approximation. Why is this an approximation?

LIGHT TRANSPORT

6. A programmer optimizes a path tracer for a GPU in the following unconventional manner.

   The path tracer evaluates one path for each pixel. For these paths, a single light source is randomly selected out of a large set of lights. This randomly chosen light source is the same for all pixels.

   a) After evaluating one sample per pixel, is the output of the path tracer *unbiased*? Motivate your answer.

   b) After taking (and averaging) many samples, is the output of the path tracer unbiased? Motivate your answer. The output is unbiased in both cases. Even though the result will look silly for a long time, the expected value of the estimator equals the correct result, even after 1 sample.

7. Consider the following path tracer pseudo code:

```
Ray ray = generatePrimaryRay()
vec3 thoughput = { 1, 1, 1 }
vec3 energy = { 0, 0, 0 }
loop:
        IntersectionInfo intersection = scene.Intersect( ray )
        if (intersection.NoHit()) return energy
        if (intersection.HitLight())
                // we hit a light; set path energy and terminate
                energy += throughput * intersection.EmissiveColor
                return energy
        else
                // continue the random path
                vec3 R = NewRandomDirection( intersection.normal )
                throughput *= intersection.diffuse_color
                ray = new Ray( intersection.position, R )
```

   Assume that the code produces correct output, and that the scene consists of pure Lambertian (i.e. diffuse) surfaces, plus area lights. The distribution of the random directions produced by 'NewRandomDirection' may or may not be uniform.

   Under these assumptions:

   Write down the pdf that is implicitly used in the random bounce code.

dot(R,N)/PI. This is the pdf we used for cosine-weighted random directions. The dot is countered by the dot for the conversion of radiance to irradiance and the division by PI in the BRDF, so in this specific situation, things get really simple.

May the Light be with you!

Don't forget to feed the Caracal.