Lab assignment 2 - `pbrt`

Javier Garea Cidre　　Martín Sande Costa

IGM 2019/2020

1 Hardware and OS

```
CPU: Intel i7-8550U (8) @ 4.000GHz
Memory:  8GB
OS: Ubuntu 18.04.3 LTS x86_64
```

2 Rendering times

2.1 `PNG` output

Total render time	**BVH**	**KD-Tree**
Integrator::Render()	99.84% (0:10:37.71)	99.87% (0:12:12.82)

Render time sliced by task	**BVH**	**KD-Tree**	
Light::Sample_*	13.63% (0:01:27.07)	12.78% (0:01:33.78)	**Triangle::Intersect**
Sampler::GetSample	13.41% (0:01:25.62)	12.76% (0:01:33.63)	**Accelerator::Intersect**
MIPMap::Lookup	11.10% (0:01:10.91)	12.05% (0:01:28.39)	**Light::Sample_***
Accelerator::Intersect	9.07% (0:00:57.94)	11.06% (0:01:21.15)	**Sampler::GetSample**
BDPT subpath generation	8.56% (0:00:54.70)	9.57% (0:01:10.25)	**MIPMap::Lookup**

Table 1: Render times for *png* output

2.2 Zenith view

Total render time	**BVH**	**KD-Tree**
Integrator::Render()	99.82% (0:11:22.78)	99.84% (0:12:57.97)

Render time sliced by task	**BVH**	**KD-Tree**	
Light::Sample_*	13.01% (,0:01:29.01)	13.96% (0:01:48.79)	**Accelerator::Intersect**
Sampler::GetSample	12.74% (,0:01:27.13)	12.32% (0:01:35.96)	**Triangle::Intersect**
Accelerator::Intersect	10.91% (,0:01:14.62)	11.57% (0:01:30.18)	**Light::Sample_***
MIPMap::Lookup	9.84% (,0:01:07.32)	11.04% (0:01:26.05)	**Sampler::GetSample**
BDPT subpath generation	8.39% (,0:00:57.41)	8.97% (0:01:09.92)	**MIPMap::Lookup**

Table 2: Render times for zenith view

2.3 Different light sources

Total render time	**BVH**	**KD-Tree**
Integrator::Render()	99.99% (0:14:02.66)	100.00% (0:17:42.71)

Render time sliced by task	**BVH**	**KD-Tree**	
Accelerator::Intersect	19.20% (0:02:41.82)	23.31% (0:04:07.69)	**Accelerator::Intersect**
Sampler::GetSample	11.24% (0:01:34.70)	20.44% (0:03:37.25)	**Triangle::Intersect**
BDPT connections	10.30% (0:01:26.80)	8.71% (0:01:32.52)	**Sampler::GetSample**
MIPMap::Lookup	10.25% (0:01:26.38)	7.97% (0:01:24.70)	**BDPT connections**
Triangle::Intersect	8.42% (0:01:10.97)	7.93% (0:01:24.27)	**MIPMap::Lookup**

Table 3: Render times for different light sources

2.4 Zenith view and different light sources

Total render time	**BVH**	**KD-Tree**
Integrator::Render()	99.99% (0:14:02.69)	100.00% (0:17:27.65)

Render time sliced by task	**BVH**	**KD-Tree**	
Accelerator::Intersect	20.24% (0:02:50.59)	24.34% (0:04:15.04)	**Accelerator::Intersect**
Sampler::GetSample	11.06% (0:01:33.20)	19.38% (0:03:23.03)	**Triangle::Intersect**
BDPT connections	10.23% (0:01:26.22)	8.71% (0:01:31.30)	**Sampler::GetSample**
MIPMap::Lookup	9.29% (0:01:18.28)	8.11% (0:01:24.96)	**BDPT connections**
Triangle::Intersect	9.00% (0:01:15.82)	7.42% (0:01:17.77)	**MIPMap::Lookup**

Table 4: Render times for different light sources and zenith view

3 Modifications

3.1 `png` output

In order to obtain the output render as `png` format, the output extension in the name of the output file must be changed. In this ocurrence `"string filename" "teapot-metal.png"`, instead of `"string filename" "teapot-metal.exr"`.

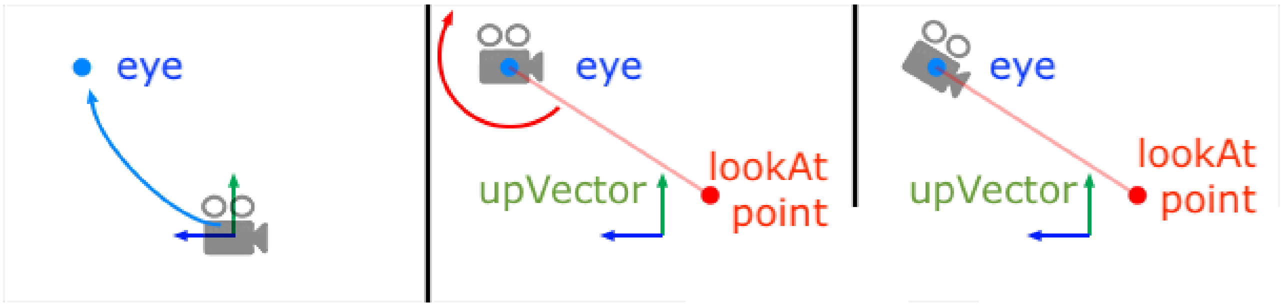
`pbrt` manual states:
"The ImageFilm uses the suffix of the given output filename to determine the image file format to use."

3.2 Zenith view

In order to obtain the output render from a zenith stand point the *LookAt* must be changed. It is formed of 3 components:

- eye
- look at point
- up vector - signals which of the 3 direction points upwards

The idea of the Look-At method is simple. In order to set a camera position and orientation, we need a point to set the camera position in space which we will refer to as the *eye*. We will also need a point that defines what the camera is looking at. We will refer to this point as the *look at point*



3.3 Different light sources

For this section we have deleted the `infinite LightSource` and added 3 `spot LightSources` and 1 `AreaLightSource`. Each one has a different light color and position.

4 Conclusions

Figure 1 shows a bounding volume hierarchy for a simple scene.
BVH$_s$ are more efficient to build than KD-trees, which generally deliver slightly faster ray intersection tests than BVH$_s$ but take substantially longer to build. This can be observed in section 2, where (BVH$_s$) render times are substantially faster than KD-tree. This also explains why using different acceleration data structures conveys in different task times.

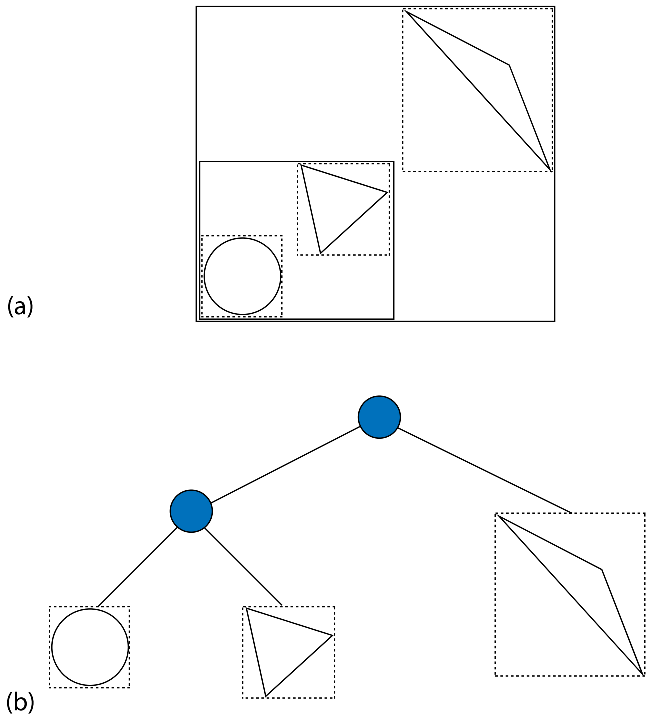


Figure 1: BVH simple scene

The KD-tree is built by recursively splitting the bounding box of the scene geometry along one of the coordinate axes. Here (figure 2), the first split is along the x axis. The left region is then refined a few more times. The details of the refinement criteria can all substantially affect the performance of the tree in practice — i.e., which axis is used to split space at each step, at which position along the axis the plane is placed, and at what point refinement terminate. This is in contrast to BVH$_s$, where each primitive is assigned to only one of the two subgroups after a split.

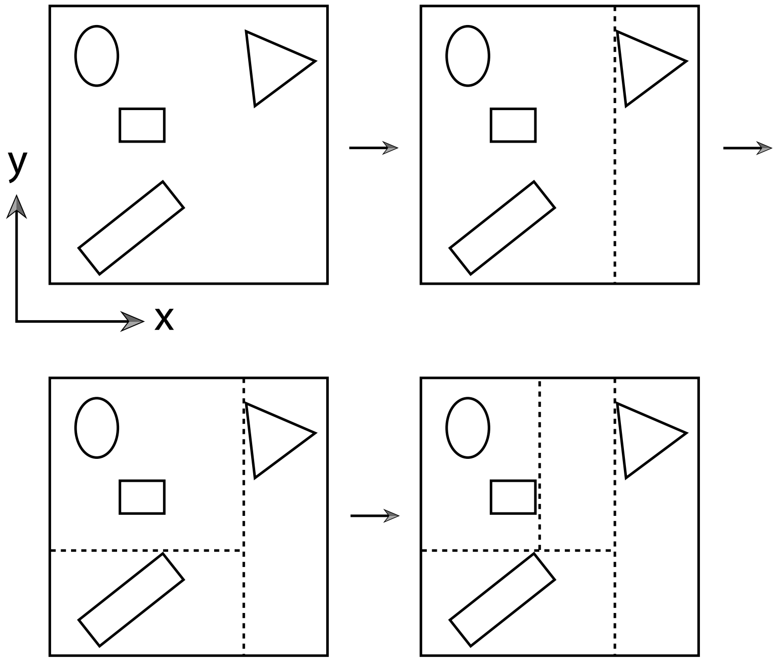


Figure 2: KD-Tree simple scene

All these acceleration data structures match two main schemas: Object Subdivision and Spatial Subdivision. The importance of the usage of these structure lies on the fact that not all rays in the image must be traced, meaning that the ray tracing depends on the object distribution in our model.