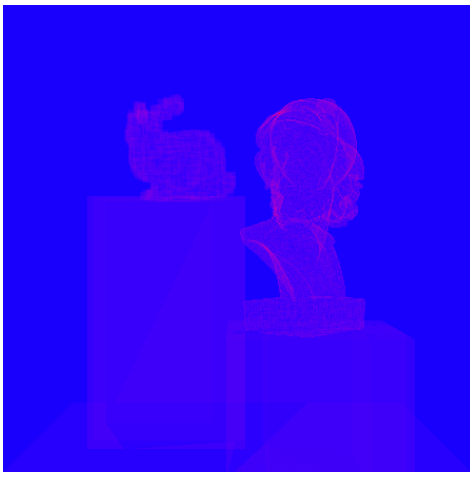

# Assignment 2: BVH Building & Traversal

Deadline: 2021-05-12 23:59

In this task, you will implement a spatial acceleration structure inside the Nori rendering framework. This will speed up rendering, which is crucial for complex scenes with a lot of geometry. Proper acceleration structures reduce the number of intersection tests during traversal (see above, red = high, blue = low) and can cut render time, e.g., from 40 minutes to 6 seconds.

**We have updated the `assignments` repository. Please merge all upstream changes before starting to work.**

```
git checkout master
git pull
git merge submission1       # just to be sure
git push                    # just in case something fails, make a backup
git remote add upstream git@submission.cg.tuwien.ac.at:rendering-2020/assignments.git
git pull upstream master
# resolve any merge conflict, or just confirm the merge.
git push
```

## BVH Building + Traversal (up to 25 points)

Use the information provided in the lecture and accompanying slide set to build a reasonably fast BVH. Implementing the BVH requires adding two behaviors: BVH building and BVH traversal. Both of these changes can be made in the `accel.h` and `accel.cpp` exclusively. We have added the `intersection integrator` to Nori. Using it in your scene will produce a color-coded output (red=high, blue=low, compare with image on page 1), showing you how many ray-triangle intersections were performed during traversal in each pixel. This should be a good indicator how your acceleration structure is doing in terms of ray tracing performance. Depending on your effort with BVH building, we will award points as follows:

**Spatial median split** max. 10 pts

**Object median split** max. 15 pts

**Sweep SAH split** max. 20 pts

Since there are not that many meshes in our test scenes, it suffices to build a separate BVH for each mesh and perform traversal on them one after the other.

Using the fast traversal presented in the lecture (checking nearest node first) gives 5 points. You should also try to optimize for the overall tracing performance for rendering detailed models like the Ajax. Caching pre-computed, high-quality BHVs is of course not allowed and will not be useful, since we will run our own tests on your code. We will make a competition of the solutions that required the least amount of time to build and trace arbitrary scenes! The results will be published in a hall-of-fame on the rendering website. We can anonymise your result if you prefer not to leave a trace. Write a mail to Adam in that case.

## Bonus Tasks

There are also several options for earning additional points. The information to get started on these can be found in the lecture slides or by checking the references on the last slide:

**Adding a k-d tree variant** +10 pts

**Implement dual-split trees** (click for link) +10 pts

**Build an über BVH over the per-mesh BVHs (layered)** +5 pts

### Create a faster solution than our reference builder +20 pts

### Use the provided `tbb` library to do multi-threaded BVH building +10 pts

If you choose to implement one or multiple of the above, please make sure to submit along with your code a `report.pdf` in the root directory of your repository that explains what you did and how we can enable your alternative solutions. For testing your layered BVH, you can download a scene from the internet (e.g. blendswap), and use the rudimentary blender export plugin ("ext/plugin"). If you are tired of waiting for your results or going for the high-score, the last bonus task will be very interesting as it can quickly speed up building by $\times 10$ or more, depending on your CPU!

## Submission format

To store or submit your code, please use our own, institute-hosted submission Gitlab `https://submission.cg.tuwien.ac.at`. You will receive a mail with your account and assignment repository as soon as they are ready. The `master` branch is for development only. You should push there while you are experimenting with the assignment and don't want to lose your work. Once your solution works and you believe it is ready to be graded, please use the branch `submission<X>` where `<X>` is the assignment number. E.g., in order to submit your solution for the first assignment, push to `submission1`.

If you push to a `submission` branch, the server will trigger automatic compilation and some testing for your code. You can track the state of new submissions being processed on the GitLab page for your repository under "CI/CD > Pipelines". If a stage fails, click on it to receive additional output and system information from the executing server. If everything worked, you will shortly find a report with your test results in the "CI/CD" pipeline section, when checking the artifacts of the "report" stage. You can submit multiple times until the deadline, but don't clog the system by, e.g., using the submission server for debugging. The last submission that was pushed before the deadline counts, regardless of the results from automatic testing. They are only meant for your convenience and to provide some automated feedback.

**Please make sure to NOT add unnecessary files (project folders, temporary compiler results), as your application will be created from your code and CMake setup only**. Examples of files that are usually relevant:

- **changed or added** CMakeLists.txt files
- **changed or added** code files (.h, .cpp)
- **changed or added** test cases if you want to show off advanced solutions

Make sure to keep the directory structure in your submitted archive the same as in the framework.

## Words of wisdom

- Remember, that you need at least 15 points for each assignment!
- The framework is using Eigen under the hood for vectors and matrices etc. Be careful when using `auto` in your code (Read here why).
- The lecture slides should you provide with the information you need to implement a simple or elaborate BVH.
- If you are having trouble with performance, consider changing the resolution and/or number of samples for your test cases.
- If you have questions, please use TUWEL, but refrain from posting critical code sections.
- You are encouraged to write your own test cases to experiment with challenging scenarios.
- Tracing rays is expensive. If your acceleration structure is not yet working, you don't want to render high resolution images or complex scenes. You may also want to avoid the `Debug` mode if you don't actually need it (use a release with debug info build!).
- To reduce the waiting time, Nori runs multi-threaded by default. To make debugging easier, you will want to set the number of threads to 1. To do so, simply execute Nori with the additional arguments `-t 1`.