The exercises will take place in room G40 in Mühlenpfordtstrasse 23. Your y-account is sufficient to login and access all tools. Ctrl+Alt+T gives you a terminal and g++ is your GNU C++ compiler.

Each week you must complete the assignments and hand in your commented source code for the practical tasks, as well as your solutions to the theoretical tasks (with drawings/formulas). Please use different colors in your drawings and also make sure that formulas are recognizable in your source code. Your group must present the completed assignments on **Friday, 15.12.2017, 9:45**.

To keep presentation time short keep (a copy of) the original scene to generate the result shown below.

### 6.1 KD-Tree Acceleration (70 Points)

So far our ray tracer has only used bounding boxes for reducing the number of ray/primitive intersections. This works well for small scenes, but for larger ones we need a data structure to speed up the process of finding the first hit of a ray with the primitives. In the lecture you will learn about kd-trees. Have a look at `scene/fastscene.cpp` and implement the missing parts. When determining the split dimension use the dimension in which your local bounding area is the widest. When determining the split position, use the **median** of the minimum bounds of the primitives. If everything is implemented correctly, the result should still look like the image below but render a lot faster.

![KD-Tree Acceleration Result](image_url)
6.2 KD-Tree visualization (20 Points)

To check the correctness of your splitting, implement a new renderer in renderer/kdtreerenderer.cpp. This renderer should utilize the function FastScene::countNodeIntersections to create a map, which contains the number of splitting planes each viewing ray must pass to traverse the whole scene. Then colorize this map as follows: 0 intersections → green, maximum number of intersections → red and for in-between values interpolate between those extremes. If you have done everything correctly your result_kd.png should look like this:

![result_kd.png](attachment:image.png)

6.3 Parallelization (10 Points)

Some of you have already done it. Now you can get points for it: Parallelize the outer loop in SimpleRenderer::renderImage. If you are not familiar with std::thread or similar constructs, a simple solution would be to look at OpenMP and its loop parallelization capabilities. If you choose to try OpenMP, don’t forget to activate the necessary compiler flags in your CMakeLists.txt. Render your image again and look at the CPU utilization with htop. Also compare execution times with the time command or similar mechanisms.

http://bisqwit.iki.fi/story/howto/openmp/