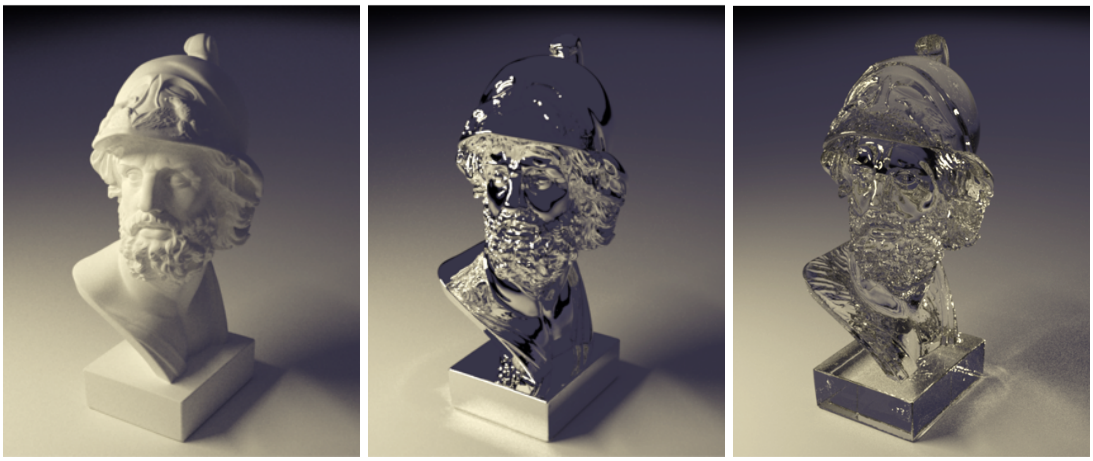

Assignment 4: Bonus and Projects

Deadline: 2022-07-26 23:59

Now that you have already implemented a basic path tracer, it's time to go further. And you can choose your own path. Only thing is, that we would like to see where you went :).

What this means, is that you can choose freely from many different tasks, some small, others super hard. And we want you to present your work in the best possible light: create your own scene. You are welcome to share your scene on the TUWEL forum, comment the scenes of others, and get feedback.

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

```
git checkout master
git pull
git merge submission3       # 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-2022/assignments.git
git pull upstream master
# resolve any merge conflict, or just confirm the merge.
git push
```

1 Create your own Scene (5 Easy Points - 30 Hardish Points)

Get artistic!

We will hold a competition for the best and most impressive scene. All of the entries will be presented in the hall of fame on our homepage.

You can prepare scenes by combining individual models (please make sure they are not copyrighted) and features that you implemented (mandatory or bonus) in custom Nori test scenes. Aim to get the most impressive renderings that you can manage!

Since scenes are quite subjective, we will grade along these criteria:

- Very few objects (bunnies, monkeys, etc.) arranged using the xml file (5 easy points)
- More complex arrangements, where you had to use Blender export, but no or only simple objects of your own design (10 - 20 points)
- Very complex scenes, with objects that you created yourself (30 points).

For all entries, please push the scene to git in your **last** commit. If you used Blender, also add a screenshot of the scene in Blender. Please use a reasonable sample count, so that the image is mostly noise free. Use one of these resolutions: 1280 × 720, 1920 × 1080, or 3840 × 2160.

If you don't want us to publish your scene, that's not a problem. Please write us a mail in that case.

2 Sampling and Appearance (3 Points)

These 3 points are something, that you really *should* do. They are easy enough, and definitely belong to a proper renderer.

Antialiasing (2 Points)

Before we get down to business, let's first get rid of aliasing in our renderings. Until now, we have only ever shot our rays through the center of the pixels. If you have a lower-resolution display or zoomed in on your renderings, you probably saw that they are somewhat jaggy because of this (look at sharp edges, like the bottom of the front box)! We

can quickly fix that by running minimalistic antialiasing for the whole pixel: in `main.cpp`, instead of shooting rays always through the pixel center, make it so that the rays can sample the full pixel width and height! Also make sure that your changes are stored in `pixelSample` and then passed to `block.put`. This will be important later.

2.1 Support for Filtering (1 Point)

When you fixed aliasing and computed output colors by integrating values over the whole pixel, you basically used a pixel-sized box filter. This is easy to implement, but really not a good choice for filtering: the box filter is sometimes jokingly referred to as the worst filter available. To get support for a few different filters, you need to implement the corresponding suppport in Nori. Once done, you should experiment with different filters and sample counts, to see what a difference they can make.

Apart from the theory behind it, which is not too complex, the **implementation** for supporting separable filters in a tiled renderer is not trivial (it's not that hard either), so we provide the missing code here:

```
void ImageBlock::put(const Point2f &_pos, const Color3f &value) {
if (!value.isValid()) {
/* If this happens, go fix your code instead of removing this warning ;) */
cerr << "Integrator: computed an invalid radiance value: "
<< value.toString() << endl;
return;
}

/* Convert to pixel coordinates within the image block */
Point2f pos(
_pos.x() - 0.5f - (m_offset.x() - m_borderSize),
_pos.y() - 0.5f - (m_offset.y() - m_borderSize));

/* Compute the rectangle of pixels that will need to be updated */
BoundingBox2i bbox(
Point2i((int)  std::ceil(pos.x() - m_filterRadius),
(int)  std::ceil(pos.y() - m_filterRadius)),
Point2i((int) std::floor(pos.x() + m_filterRadius),
(int) std::floor(pos.y() + m_filterRadius)));
bbox.clip(BoundingBox2i(Point2i(0, 0),
Point2i((int) cols() - 1,
(int) rows() - 1)));
```

```
/* Lookup values from the pre-rasterized filter */
for (int x=bbox.min.x(), idx = 0; x<=bbox.max.x(); ++x)
m_weightsX[idx++] = m_filter[(int) (std::abs(x-pos.x()) * m_lookupFactor)];
for (int y=bbox.min.y(), idx = 0; y<=bbox.max.y(); ++y)
m_weightsY[idx++] = m_filter[(int) (std::abs(y-pos.y()) * m_lookupFactor)];

/* Add the colour valuel after filtering to the current estimate.
* Color4f extends the Color3f value by appending a 1. Therefore,
* in the 4th component we are automatically accumulating the filter
* weight. */
for (int y=bbox.min.y(), yr=0; y<=bbox.max.y(); ++y, ++yr)
for (int x=bbox.min.x(), xr=0; x<=bbox.max.x(); ++x, ++xr)
coeffRef(y, x) += Color4f(value) * m_weightsX[xr] * m_weightsY[yr];
}
```

3 Bonus Tasks / Project (Loads of Points)

This is the main part of this assignment! There is a copious amount of additional bonus points up for grabs, check the following sections for details. In addition, you can also find papers or effects on your own and implement them. We will be generous with bonus points, but you definitely need to write a **list of effects, points, and the sources / paper that you used**. If you would like to go for something really ambitious but need an incentive, talk to us, we might be able to give you a project or similar.

Another thing to keep in mind: if you stick out of the crowd, it is likely we would recommend you for a PhD position either here or at one of the more specialised labs.

3.1 Adding a Microfact BSDF (10 Bonus Points)

Implement a more complex Microfacet material model, according to the steps outlined in Assignment 5, Part 1 found on the Nori webpage. This BSDF should give you a linear blend between a diffuse and a Torrance-Sparrow-based specular model. Note that some of the notes on the webpage do not apply: first, there is no default fresnel implementation in our framework; adding it is part of the assignment for implementing dielectrics. Second, the microfacet BRDF and its distribution will not be tested automatically on the server.

3.2 More Materials

The topic of materials does not stop at the microfacet model. There is a wide range of more complex aspects of objects' physical appearance, and the resulting rendering solutions can become very sophisticated. Background: Physics and Math of Shading by Naty Hoffman is a nice didactic introduction and contains a lot of in-depth information. It is a good read even if you don't want to implement anything! Naty also has course presentations hosted on Youtube for you to access. Our course book also has a lot of information on materials and even some code (but be aware that the notation and conventions might be different to what Nori uses).

Ideas for bonus tasks with materials:

- Textures (UVs are already being loaded, you'd need to extend the xml, 10 Points)
- Multilayer materials (e.g., clear coating paint, 5-10 Points, depending on the complexity of the implementation)
- Procedural materials, for instance marbles or glitter (5-10 Points)
- Support NEE + MIS with dielectric materials by implementing manifold next event estimation (100 Points)
- A physically correct gold BSDF (all effects described by Naty Hoffman, 20 Points)
- Fourier Basis BSDFs (from the book, 10 Points)
- Subsurface scattering (SSS, your own research, but we can give hints, 50 Points)

3.3 More Sampling

- Better sampling methods (5 points for stratified sampling, 10 points for a complete and artefact free Halton sampler implementation. Both should be implemented in the primary sample domain, i.e., replace the random number generator.)
- Adaptive sampling and reconstruction to get more out of the samples you take (e.g. MDAS, AWR. 50 Points)
- Path guiding (2 pass unbiased algorithm, first cast photons into the scene and record important 'incoming directions'. Then, use them for importance sampling. 300 Points)

- A basic path tracer on the GPU (CUDA, GLSL, HLSL... should be unbiased and support diffuse and specular BRDFs at least. 300 Points)

3.4 Other Ideas

- Nori already has a very simple tone mapper, but implementing a Reinhard operator or something similarly effective will give you 5 points.

- Light tracing (start the path from the light and connect it to the camera similar to NEE. You'd have to change stuff in the camera, scene, and main loop. The most important thing would be to hold an additional image block for the whole image plane, because the the ray can hit any pixel. We could give some more hints in TUWEL, just ask! 50 Points)

- Bidirectional Path Tracing (Light tracing is a good start, we can help in TUWEL. 150 Points)

- Participating Media (only physically based models, i.e., volumetric path tracing. Ask for reading material in TUWEL 300 Points)

3.5 Metropolis Light Transport (up to 1000 points)

Metropolis Sampling is the basis of the Metropolis Light Transport algorithm. If you have too much time on your hands and want to go down as a hero in the history of this lecture, feel free to attempt an implementation of it. Chapter 16 of the PBR book contains some introductory information, but you will have to spend additional effort researching the required backgrounds. 300 points for the primary sample space version, 1000 for the path space version. For something special like this you may take your time until the end of the summer (or even beyond). Contact us for details if you need further guidance or suggestions. Good luck!

Submission format

Put a short PDF or text file called `submission<X>` into your git root directory and state all the points that you think you should get. This does not need to be long. Also mention the code files, where you implemented something if it is not obvious.

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 fourth assignment, push to `submission4`.

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

- 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. You don't want to render high resolution images or complex scenes for testing. 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`.