



Assignment 3: Importance Sampling

Deadline: 2021-06-02 23:59

In this assignment you will extend the Monte Carlo rendering system from the last assignment with importance sampling of various functions and next event estimation. In the above image, we see what these methods can do: the left scene is rendered with uniform hemisphere sampling. In the center, we use cosine-weighted hemisphere sampling (importance sampling). On the right, we use next event estimation to perform surface sampling in a recursive path tracer. Images were rendered with the same number of samples per pixel (32).

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

```
git checkout master
git pull
git merge submission2      # 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
```

We also provide a reference implementation for assignment 2, you can download it from TUWEL.

1 Sample Warping (7 easy points, 9 bonus points)

Random numbers are often generated uniformly in the range between 0 and 1. We can combine multiple such random numbers to sample cartesian domains uniformly, but different distributions are needed, e.g., to get uniform distribution in a non-cartesian domain (for recursive rendering, we need to sample the hemisphere for instance), or for importance sampling techniques. This task can be fully solved in `warp.cpp`.

The process of *changing* an existing distribution is called warping. In this assignment, you will start with easily obtainable, canonic random inputs, and convert them to new, useful distributions. The input to all warping functions are two uniformly distributed $([0, 1])$ random numbers, and the output are samples on the target domain. The input is always a 2D vector with values of two canonical random variables ξ_1, ξ_2 .

In many cases, we may have an existing sample and need to obtain its PDF value for a given sampling strategy, thus a method to produce the PDF from input samples is also required. The input is always a sample x , 2D or 3D, for which a PDF value $p(x)$ value should be computed.

To visualize and check your implementations, we will be using the `warptest` executable, which is part of the Nori framework. You should complete several of the warping functions that it tests. For an introduction on how to use `warptest` and what each distribution is supposed to do, please refer to the Assignment 3, Part 1 from the Nori home page (<https://wjacob.github.io/nori/>). **Note that our scoring system is different, please find it below.** `SquareToUniformHemisphere` is already there, some of you were already cleverly using it in the first assignment to do uniform hemisphere sampling.

squareToTent 2 points, test your basic Monte Carlo sampling knowledge, **bonus**

squareToUniformDisk 3 points, **required**

Sampling: Use the input canonic variables to generate samples (r, θ) in polar coordinates where $r \in [0, 1)$ and $\theta \in [0, 2\pi)$, such that they are uniformly distributed when transformed to Cartesian coordinates (x, y) . Return the current sample (x, y) at the end of the function body.

PDF: The input is a 2D vector with a sample location in Cartesian coordinates (x, y) on a square, with $x, y \in [-1, 1)$. Return the proper value for the corresponding result from the uniform distribution PDF $p(x, y)$ on the disk. Note: For a uniform distribution, the PDF is constant. Just make sure that the sample location is valid!

squareToUniformSphere 2 points, can use it to implement spherical lights, **bonus**

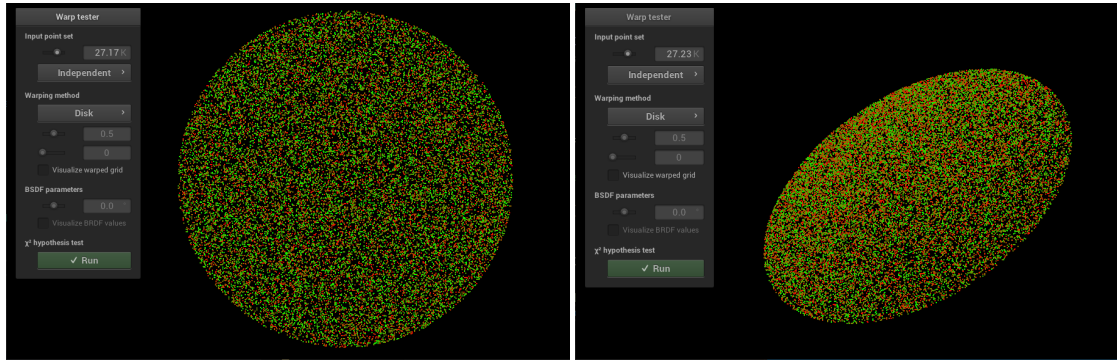
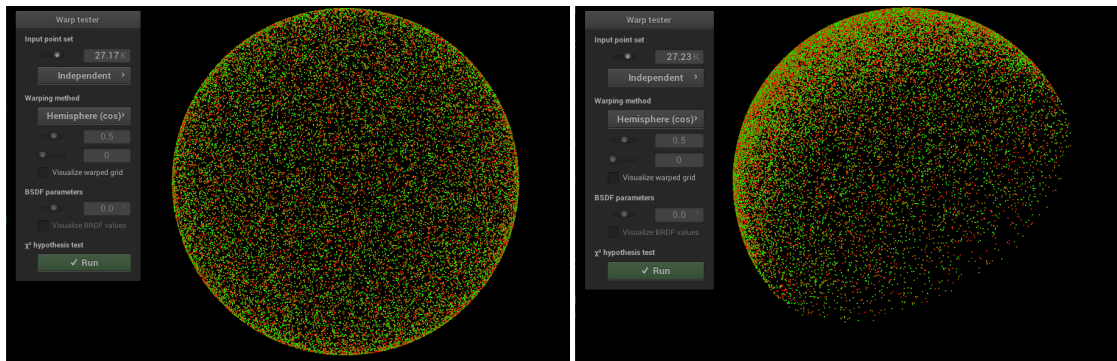


Figure 1: Reference solutions for uniformly distributed samples on the unit disk

squareToCosineHemisphere 4 points if inversion method, 1 point if Malley's, **required**

Sampling: The input is a 2D vector sample that holds values of two canonical random variables ξ_1, ξ_2 . Use them to generate samples (θ, ϕ) on the unit hemisphere such that they have a distribution proportional to $\cos(\theta)$ (i.e., more samples the closer we get to the pole of the hemisphere) and convert them to ω with the transformation for spherical coordinates. Return the sample (x, y, z) at the end of the function body.



PDF: Input is a 3D vector with a sample location ω on the unit **sphere** in Cartesian coordinates (x, y, z) with all values in range $[0, 1)$ and $\sqrt{x^2 + y^2 + z^2} = 1$ (z is up). Return the appropriate result for the PDF value $p(\omega)$. Compute and return the appropriate result for a PDF with distribution $p(\omega) \propto \cos(\theta)$.

squareToBeckmann 5 points, used for materials (needed in last assignment!), **bonus**

2 Importance Sampling (18 points, 20 bonus points)

Use cosine-weighted hemisphere samples for diffuse materials (8 easy points)

Use the cosine-weighted hemisphere sampling method, as described in the lecture. First make sure that your direct lighting and path tracing integrators use the diffuse BSDF class appropriately, then extend the diffuse BSDF with cosine-weighted hemisphere sampling. Ideally, you can reuse your warping solutions from the first part of this assignment! The BSDF should switch between using cosine-weighted and uniform hemisphere sampling, depending on the value of the `use_cosine` flag provided by each object's material (default: false). Note that this affects both the sampling and PDF computation! Confirm for yourself that cosine-weighted hemisphere sampling can reduce the noise in your scenes. To test this, compare the output of the test scenes that end in `uniform` with the ones that end in `cosine`. The latter use cosine-weighted hemisphere sampling and should give slightly cleaner results.

Bonus: Multiple Importance Sampling for Direct Lighting (10 points) Implement MIS between hemisphere sampling and light surface sampling using the balance heuristic in your direct lighting integrator. Whether or not MIS is used should be parameterizable via boolean `mis_sampling` (default false) in the test files. Choose between the two sampling strategies with equal probability, generate the sample using the chosen method and compute the sample's probability with both methods. Make sure that you use the surface's BSDF to generate the hemisphere samples to benefit from cosine-weighted hemisphere sampling if it is enabled by a material. Then use the equations from the lecture to compute the proper MIS weight. Return the contribution that you would get with the chosen method, multiplied by the MIS weight and a (simple!) compensation term for the choice you made when picking one method over the other. You should use the balance heuristic, simpler heuristics will count but not give full points. You can test MIS on the `ajax-2lights_dl*.xml` scenes, where you should be able to observe the following: the small area light is better suited for surface sampling, while the larger one is better with cosine-weighted hemisphere sampling, but MIS can give you the best of both worlds. Feel free to also explore ideas that we didn't describe here (rays that miss are black by default, but you could use a sky colour or an environment map). These things do not go unseen :)

Next Event Estimation (10 points, 10 bonus points) Implement next event estimation (NEE) for your diffuse path tracer, according to the lecture slides. It should be active depending on a boolean `nee` in the test file (default false). That is, on every bounce, you create one light surface sample and try to connect to compute direct lighting with surface sampling. Another ray is then sent out to retrieve indirect light in the next bounce. Make sure that you use the BSDF to generate the indirect sample to benefit from cosine-weighted hemisphere sampling of indirect light on materials that use it. If you implement NEE, be careful not to erroneously count the emittance twice (i.e., first when doing the light surface

sampling and then when hitting a light source randomly). To get a correct image, the emitant surface points that your ray hits should only be considered on the first intersection or if the last material did not support hemisphere sampling (e.g., mirrors!). For all other light, the illumination is computed via direct lighting, i.e., one bounce in the future (hence, "next event"). For further details, please see the lecture slides. Just as a heads-up: implementing NEE will dramatically improve the quality of your renderings! In combination with spatial acceleration structures, you should now be able to render impressive scenes fast! To test this, compare the output of the test scenes that end in uniform or cosine with the ones that end in nee. The latter use next event estimation and should give significantly cleaner results.

If you pay close attention, you may see that NEE is just a very special case of multiple importance sampling! The way that NEE was described in the lecture was using the 0/1 heuristic of MIS: at the first hit, we choose hemisphere sampling for light sources, afterwards we use light surface sampling. Hence, we can say that we use MIS weights for both techniques that are either 0 or 1, depending on the recursion depth. Instead of this 0/1 heuristic, you may also implement NEE with the balance heuristic (10 bonus points). This is a somewhat challenging exercise, not so much in code, but to get to the correct solution mathematically and making sense of it. It is mostly for you to test your limits on combining multiple complex concepts!

3 Further Bonus Points

Metropolis-Hastings for Cosine-Weighted Hemisphere Sampling (10 points) The Metropolis-Hastings algorithm is an advanced and very essential achievement of computer science. It actually enables you to importance-sample functions whose distribution you don't know! However, depending on the problem you pose it, it might not be the most efficient method for getting a solution. For this task, familiarize yourself with the Metropolis-Hastings algorithm and Metropolis Sampling (Wikipedia and PBR book, chapter 13). Use the Metropolis-Hastings algorithm to perform cosine-weighted hemisphere sampling without relying on a closed-form solution. How does it compare to your previous solution for cosine-weighted hemisphere sampling?

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. For something special like this you may take your time until the end of the semester (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 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

- If you are having trouble with performance, consider changing the resolution and/or number of samples for your test cases.
- The warp tests only check if the samples you generate match the corresponding PDFs you define. Best start with the PDFs and then try to match them with sampling.
- Hemisphere sampling, next event estimation and MIS are all methods for integrating the same integral. Given enough samples, they all should converge to the same result.
- 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`.