Modeling and Simulation of Appearance

Lab #2 - Monte Carlo Direct Illumination

Julio Marco - juliom@unizar.es
Néstor Monzón - nmonzon@unizar.es

2024 - 2025

Disclaimer

- This lab **will be graded**. All the following ones, too.

- This lab **builds upon** what we did in **the previous one**.

- You need to install a patch (`patch_P2.zip`) from Moodle

    As always, run CMake and compile before running

- We will create our first Monte Carlo renderer with direct illumination.

Updating our Nori renderer

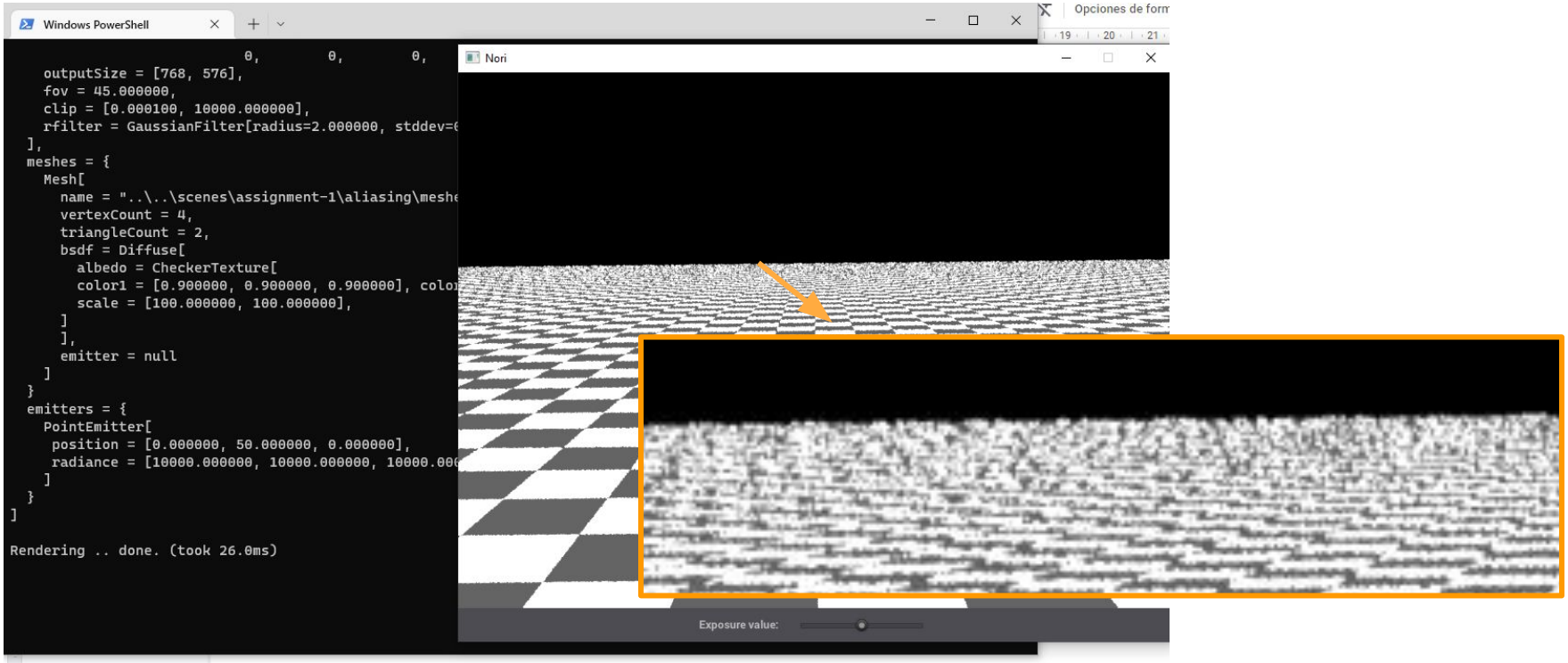
- After applying the patch and compiling everything again, we run:

`./nori ../../scenes/assignment-2/aliasing/aliasing.xml` (Linux)

`.\nori.exe ..\..\scenes\assignment-2\aliasing\aliasing.xml` (Windows)

Updating our Nori renderer

- After applying the patch and recompiling everything again, we run:



Updating our Nori renderer

- To avoid this noise, we can increase the number of samples per pixel used:

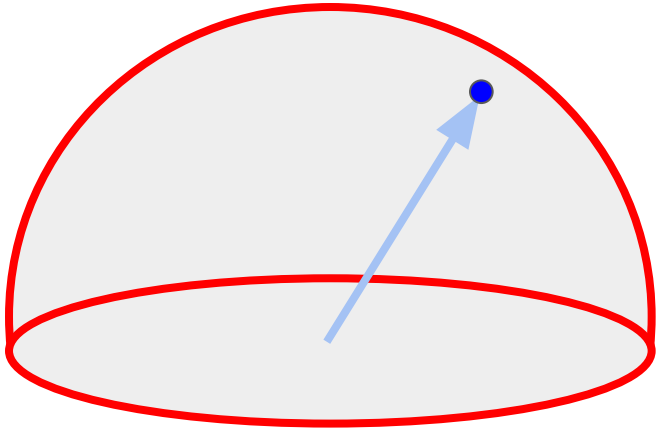
aliasing.xml

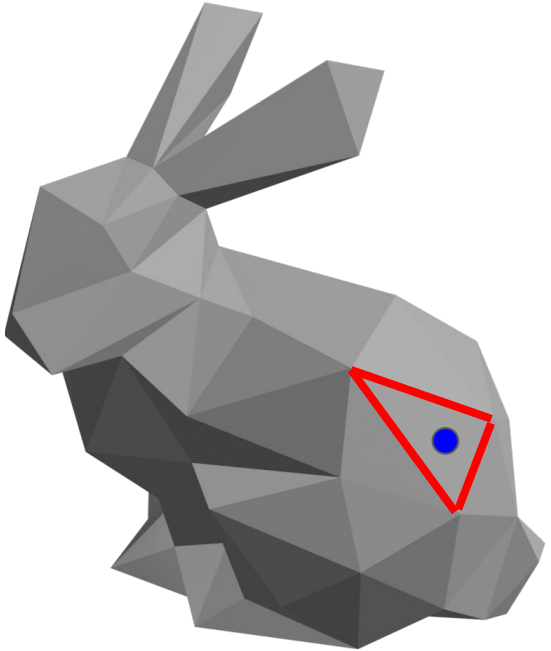
```
<!-- Independent sample generator -->
<sampler type="independent">
    <integer name="sampleCount" value="1"/>
</sampler>
```

- We can increase it to e.g., 64. It will take longer to compute, but will have much less noise:

Monte Carlo Sampling (40%)

- You are generating **samples** on various **domains** (planes, triangles, hemispheres…).





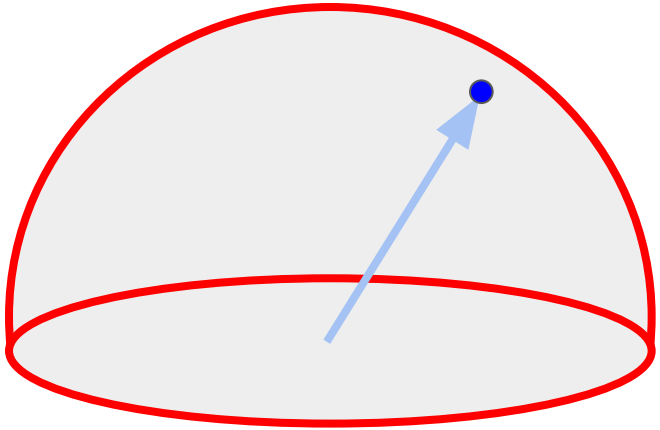
Monte Carlo Sampling (40%)

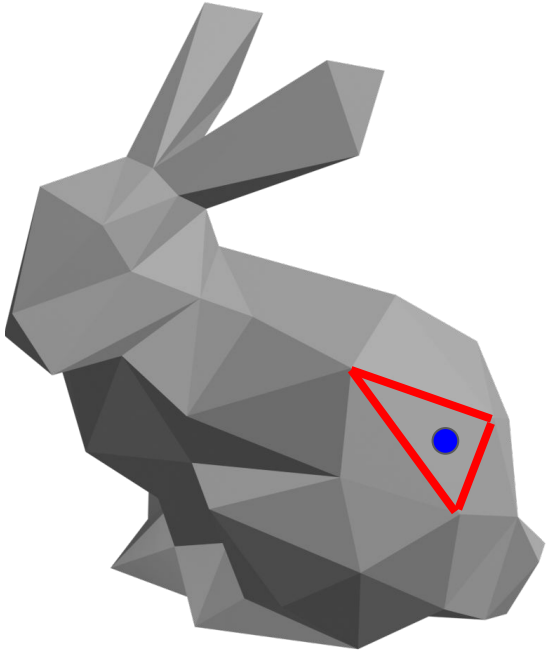
- You are generating **samples** on various **domains** (planes, triangles, hemispheres…).

- You thus have to implement:

    a) The PDF (Probability Distribution Function)

    b) The corresponding sample warping scheme



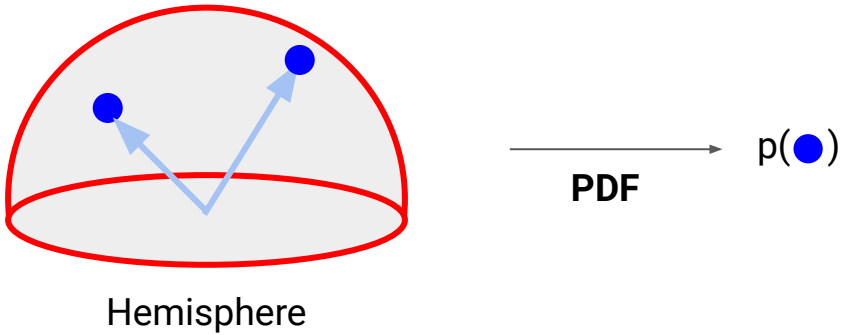


Monte Carlo Sampling (40%)

a) The PDF (Probability Distribution Function)

→ *What is the probability of a point being sampled in such domain?*

→ Note that p(x) = 0 for all points x outside the domain



Monte Carlo Sampling (40%)

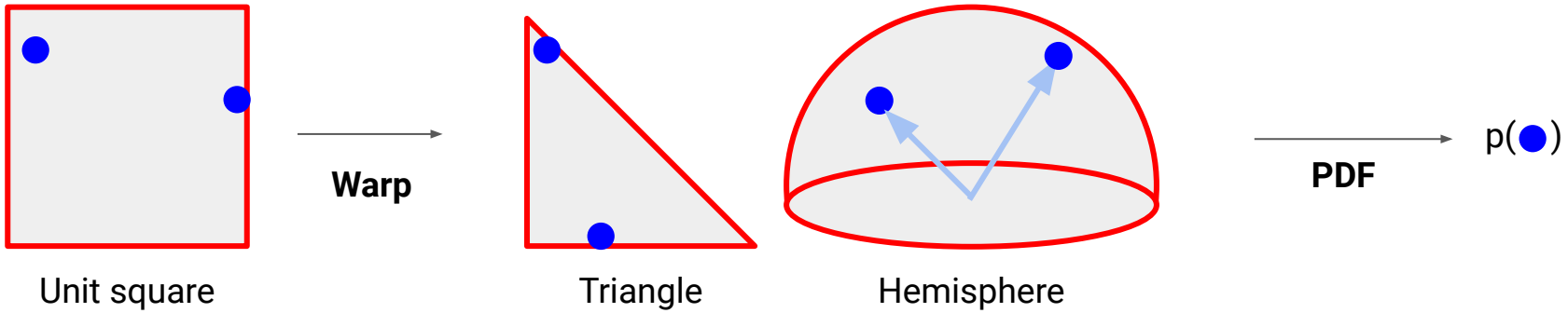
a) The PDF (Probability Distribution Function)

→ *What is the probability of a point being sampled in such domain?*

→ Note that p(x) = 0 for all points x outside the domain

b) The corresponding sample warping scheme

→ *Given a random point in a unit square, warp it to the corresponding shape.*

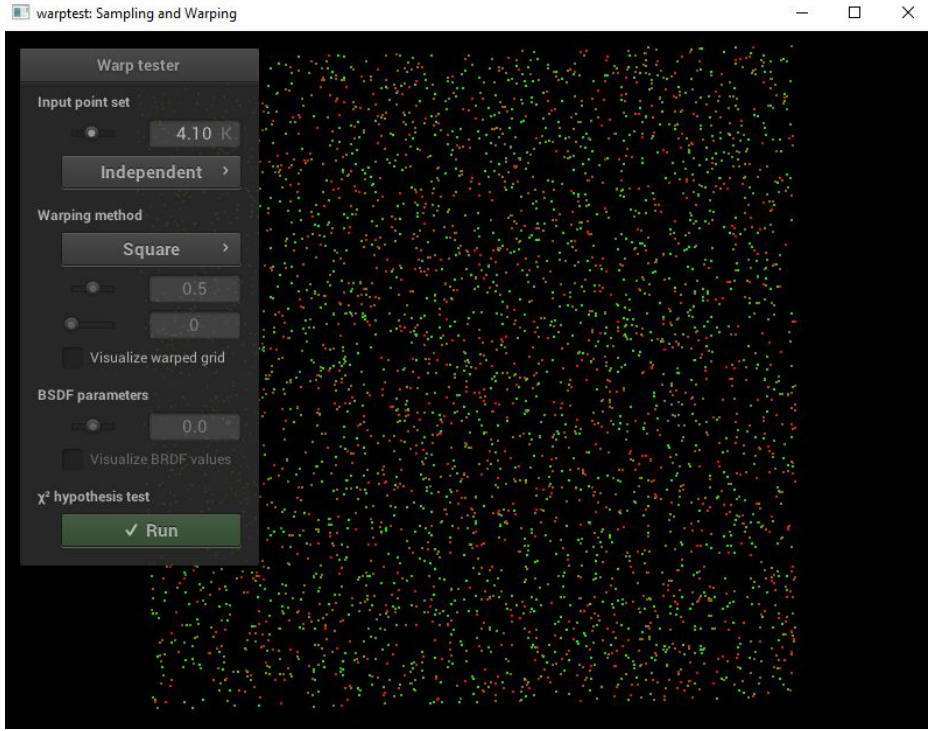


Monte Carlo Sampling (40%)

- How to validate it? → Run `./warptest` (Linux) or `.\warptest.exe` (Windows)

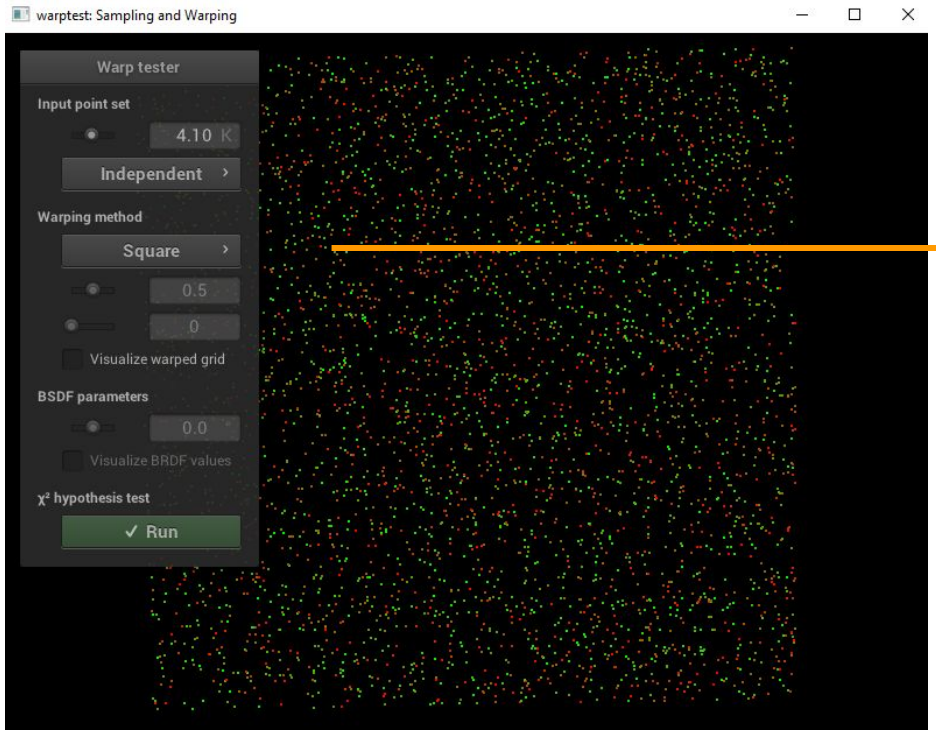
Monte Carlo Sampling (40%)

- How to validate it? → Run `./warptest` (Linux) or `.\warptest.exe` (Windows)



Monte Carlo Sampling (40%)

- How to validate it? → Run `./warptest` (Linux) or `.\warptest.exe` (Windows)

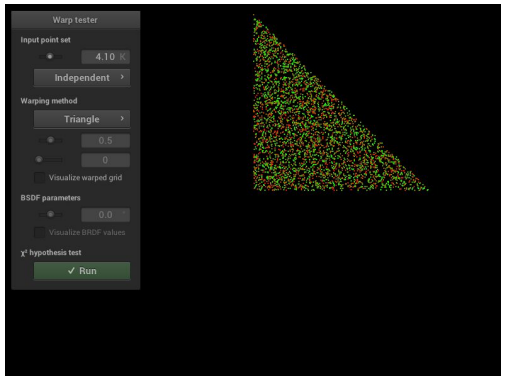


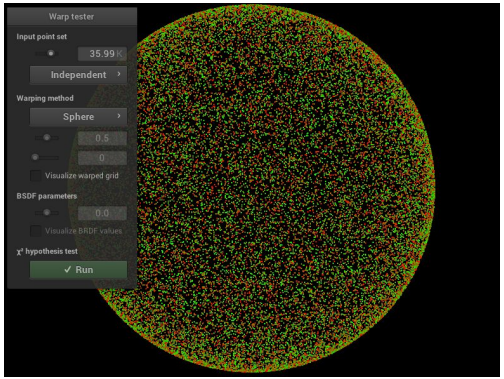
Here, you can change the domain.

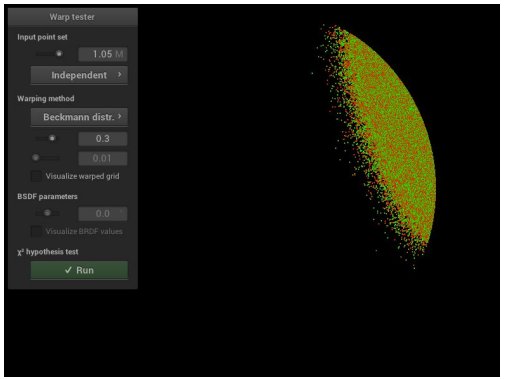
```cpp
Point2f Warp::squareToTent(const Point2f &sample) {
    throw NoriException("Warp::squareToTent() is not yet implemented!");
}

float Warp::squareToTentPdf(const Point2f &p) {
    throw NoriException("Warp::squareToTentPdf() is not yet implemented!");
}

Point2f Warp::squareToUniformDisk(const Point2f &sample) {
    throw NoriException("Warp::squareToUniformDisk() is not yet implemented!");
}

float Warp::squareToUniformDiskPdf(const Point2f &p) {
    throw NoriException("Warp::squareToUniformDiskPdf() is not yet implemented!");
}
```

Monte Carlo Sampling (40%)

- How to validate it? → Run `./warptest` (Linux) or `.\warptest.exe` (Windows)

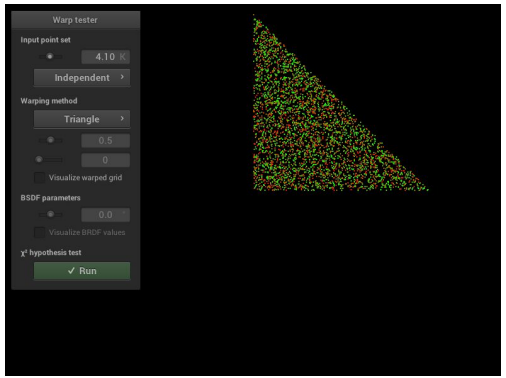


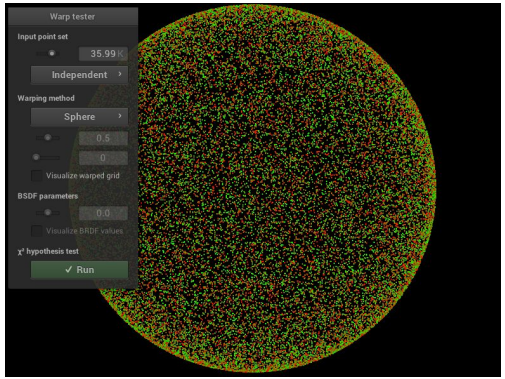


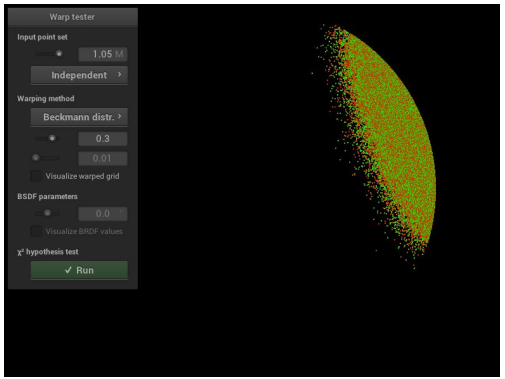


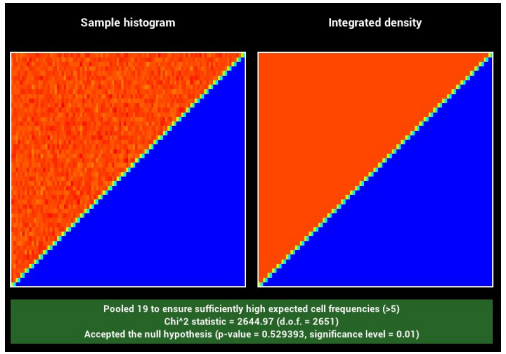
Monte Carlo Sampling (40%)

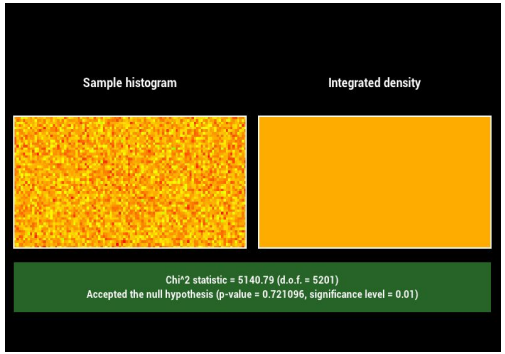
- How to validate it? → Run `./warptest` (Linux) or `.\warptest.exe` (Windows)

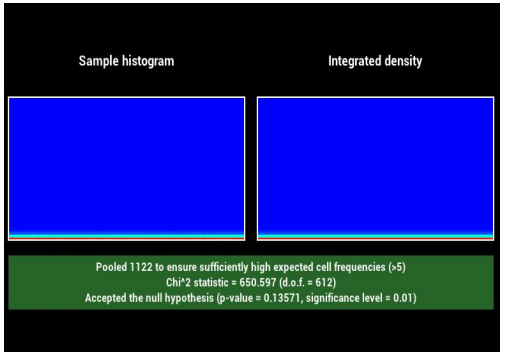












Direct Light – Emitter Sampling (60%)

We are now taking advantage of light sources (beyond pointlights).

1/ Integrator (20%)

2/ Mesh area light (30%)

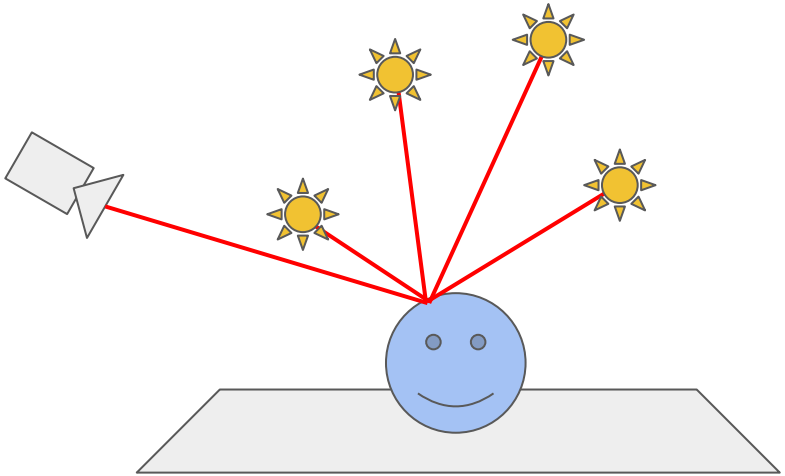
3/ Environment light (10%)

Integrator (20%)

- You should build your integrator on top of `DirectWhittedIntegrator` (prev. lab).

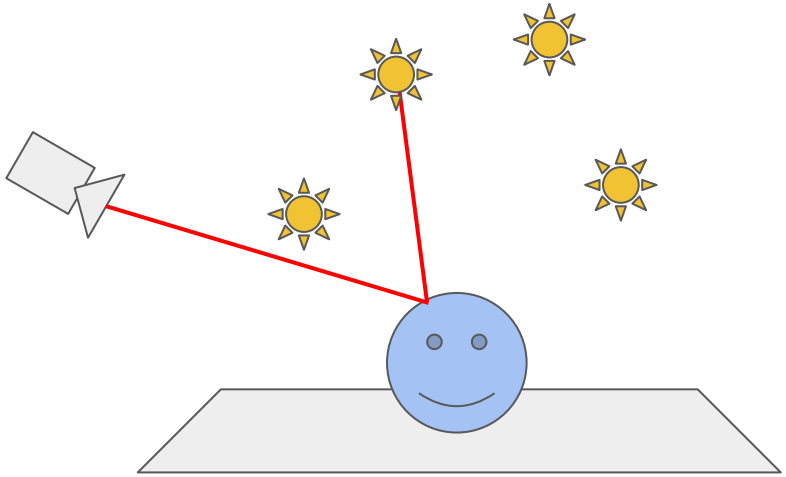
Integrator (20%)

- You should build your integrator on top of `DirectWhittedIntegrator` (prev. lab).



Whitted (prev. lab)

Each camera ray
loops through all lights



Emitter sampling (this lab)

Each camera ray
randomly samples one light

Integrator (20%) - Some tips

During rendering, NORI will approximate this integral with the Monte Carlo estimate

$$L_o(\mathbf{x}, \omega_o) \approx \frac{1}{N} \sum_{k=1}^{N} \left(L_e(\mathbf{x}, \omega_o) + L_e(\mathbf{r}(\mathbf{x}, \omega_i^{(k)}), -\omega_i^{(k)}) \, f_r(\mathbf{x}, \omega_o, \omega_i^{(k)}) \, \cos\theta_i^{(k)} \right), \qquad (7)$$

Nori takes care of this

You need to implement this in DirectEmitterSampling::Li

Nori will call that function multiple times

Mesh Area Light (30%)

- You are now implementing a new type of emitter: *area lights*.

    → Unlike point lights (prev. lab), area lights have a **finite** area.

Mesh Area Light (30%)

- TASKS:

    1/ Triangle sampling (`Mesh.cpp` and `Mesh.h`).

    2/ Area emitter (`area.cpp`).

    3/ Environment light (`environment.cpp`).

Mesh Area Light (30%)

- TASKS:

    1/ Triangle sampling (`Mesh.cpp` and `Mesh.h`).

        - Implement `Mesh::samplePosition`.

    2/ Area emitter (`area.cpp`).

    3/ Environment light (`environment.cpp`).

Mesh Area Light (30%)

- TASKS:

    1/ Triangle sampling (`Mesh.cpp` and `Mesh.h`).

    2/ Area emitter (`area.cpp`).

        - Fill in `AreaEmitter::eval`.

    3/ Environment light (`environment.cpp`).

Environment Light (10%)

- TASKS:

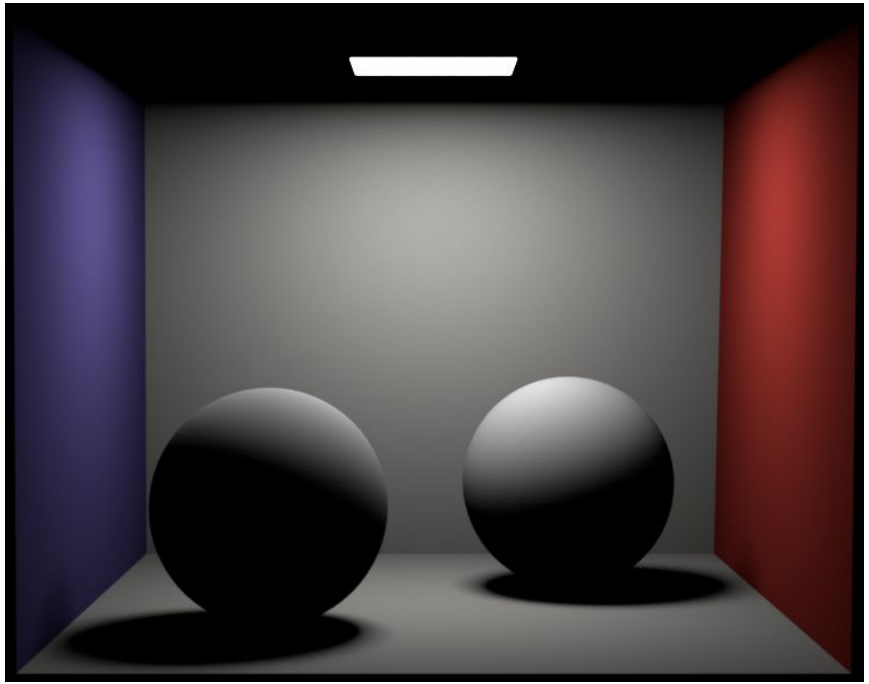
    1/ Triangle sampling (`Mesh.cpp` and `Mesh.h`).

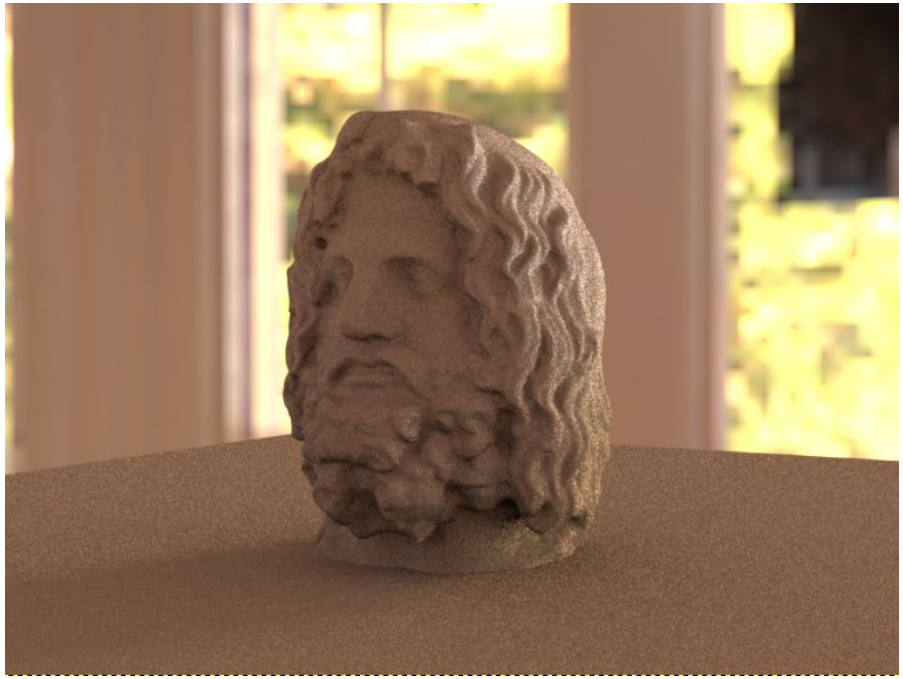
    2/ Area emitter (`area.cpp`).

    3/ Environment light (`environment.cpp`).

        - Fill in `EnvironmentEmitter::sample` and `EnvironmentEmitter::pdf`.

A couple of examples…





Final disclaimer

***** CAREFULLY READ THE REPORT *****

Do not work **sequentially;** read the whole instructions multiple times before implementing anything; some doubts may be answered at some point.