Modeling and Simulation of Appearance

Assignment #1 - Nori preliminaries

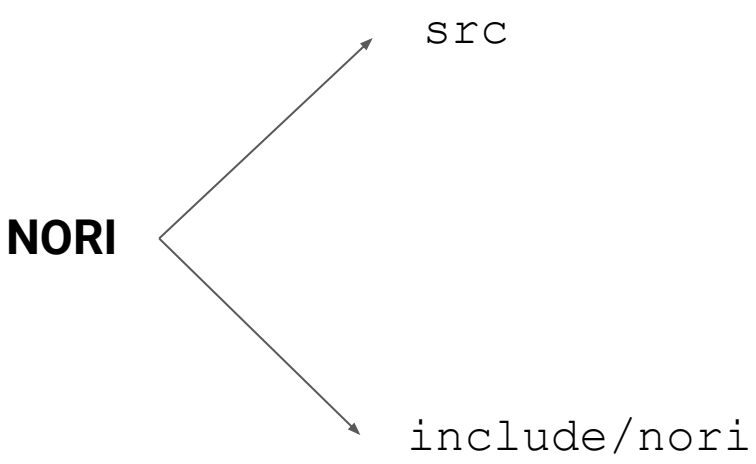
Julio Marco - juliom@unizar.es
Néstor Monzón - nmonzon@unizar.es

2024 - 2025

About this assignment

- We are learning:
    - How to **install and run** Nori (the rendering framework).
    - The **basics** of Nori.
    - How to **render** our very first scene.

Nori’s high-level overview



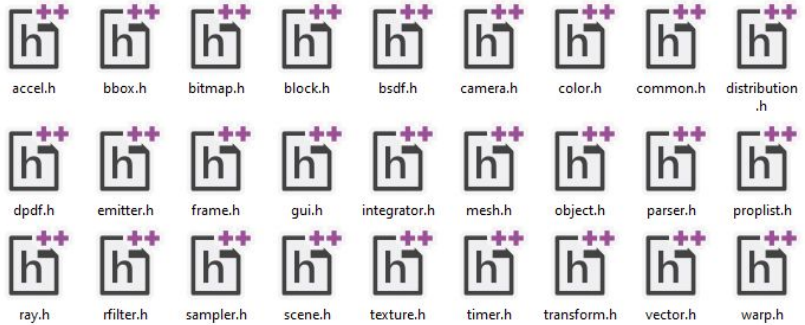
Nori’s high-level overview

NORI

`src`

`include/nori`

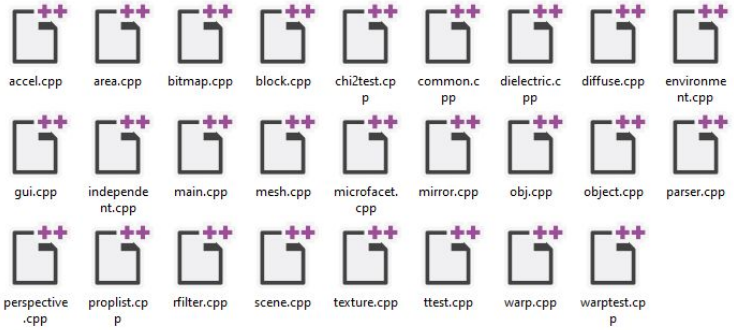
Contains:
- Headers and definitions.
- Documentation.



Nori’s high-level overview

NORI

`src`



`include/nori`

Contains:

- Implementations.

Nori’s high-level overview - Objects

- Nori’s most basic class is `NoriObject` (`include/nori/object.h`).
    - Everything **inherits** from it (Assignment #0 was worth it).

Nori’s high-level overview - Scene

- All the scene info is stored in the class `Scene` (`include/nori/scene.h`).
    - Emitters (light sources).
    - Geometries.
    - Materials.
    - Sampling routines.
    - Visibility computations.
    - …

Nori’s high-level overview - Scene

- All the scene info is stored in the class `Scene` (`include/nori/scene.h`).
    - Emitters (light sources).
    - Geometries.
    - Materials.
    - Sampling routines.
    - Visibility computations.
    - …

Task #1: Take a look into Object and Scene class.

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

`Ray3f` (`include/nori/ray.h`) → Ray-segment data

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

`Ray3f` (`include/nori/ray.h`) → Ray-segment data

`Color3f` (`include/nori/color.h`) → RGB values

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

`Ray3f` (`include/nori/ray.h`) → Ray-segment data

`Color3f` (`include/nori/color.h`) → RGB values

`Mesh` (`include/nori/mesh.h`) → Geometry representations

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

`Ray3f` (`include/nori/ray.h`) → Ray-segment data

`Color3f` (`include/nori/color.h`) → RGB values

`Mesh` (`include/nori/mesh.h`) → Geometry representations

`Intersection` (`include/nori/mesh.h`) → Information of intersection between a ray and the Scene (as a set of `Mesh`).

Nori’s high-level overview - Classes and structs

- Nori is built upon several classes that implement the basics of a renderer.

`Vector3f, Point3f, Normal3f` (`include/nori/vector.h`) → Basics of geometry

`Frame` (`include/nori/frame.h`) → Coordinates systems and computation

`Ray3f` (`include/nori/ray.h`) → Ray-segment data

`Color3f` (`include/nori/color.h`) → RGB values

`Mesh` (`include/nori/mesh.h`) → Geometry representations

`Intersection` (`include/nori/mesh.h`) → Information of intersection between a ray and the Scene (as a set of `Mesh`).

Task #2: Read the report and the particularities of each class (Section 5)

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

`Integrator (include/nori/integrator.h)`

  - Each rendering technique is referred to as integrator (i.e., a different approach for solving the rendering equation).

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

`Integrator` (`include/nori/integrator.h`) → Rendering algorithms

`Camera` (`include/nori/camera.h`) → Trace rays towards the scene

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

`Integrator` (`include/nori/integrator.h`) → Rendering algorithms

`Camera` (`include/nori/camera.h`) → Trace rays towards the scene

`Emitter` (`include/nori/emitter.h`) → Basics of all light sources

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

`Integrator` (`include/nori/integrator.h`) → Rendering algorithms

`Camera` (`include/nori/camera.h`) → Trace rays towards the scene

`Emitter` (`include/nori/emitter.h`) → Basics of all light sources

`BSDF` (`include/nori/bsdf.h`) → Basics of all appearance models

Nori’s high-level overview - Exposed interfaces

- Interfaces that will describe the rendering algorithms and scene properties:

`Integrator` (`include/nori/integrator.h`) → Rendering algorithms

`Camera` (`include/nori/camera.h`) → Trace rays towards the scene

`Emitter` (`include/nori/emitter.h`) → Basics of all light sources

`BSDF` (`include/nori/bsdf.h`) → Basics of all appearance models

Task #3: Read the report and the particularities of each class (Section 5)

Nori’s high-level overview - Scene file format and parsing

- Nori uses a XML-based scene description language.
    - As XML, it is based on tags.
    - Nori translates each tag to its corresponding object or property.

Nori’s high-level overview - Scene file format and parsing

- Nori uses a XML-based scene description language.
    - As XML, it is based on tags.
    - Nori translates each tag to its corresponding object or property.

```
<bsdf type="diffuse">
    <color name="albedo" value="0.5, 0, 0"/>
</bsdf>
```

Nori’s high-level overview - Scene file format and parsing

- Nori uses a XML-based scene description language.
    - As XML, it is based on tags.
    - Nori translates each tag to its corresponding object or property.

```
<bsdf type="diffuse">
    <color name="albedo" value="0.5, 0, 0"/>
</bsdf>
```



```
Diffuse(const PropertyList &propList) {
    m_albedo = propList.getColor("albedo", Color3f(0.5f));
}
```

Nori’s high-level overview - Scene file format and parsing

- Nori uses a XML-based scene description language.
    - As XML, it is based on tags.
    - Nori translates each tag to its corresponding object or property.

```
<bsdf type="diffuse">
    <color name="albedo" value="0.5, 0, 0"/>
</bsdf>
```



```
NORI_REGISTER_CLASS(Diffuse, "diffuse");
```

```
Diffuse(const PropertyList &propList) {
    m_albedo = propList.getColor("albedo", Color3f(0.5f));
}
```

Nori’s high-level overview - Scene file format and parsing

- Nori can have nested elements.

Nori’s high-level overview - Scene file format and parsing

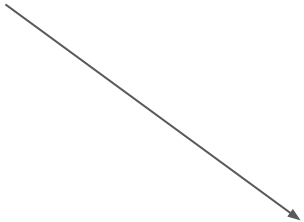
- Nori can have nested elements.

```
<mesh type="obj">
    <string type="filename" value="bunny.obj"/>

    <bsdf type="diffuse">
        <color name="albedo" value="0.5, 0, 0"/>
    </bsdf>
</mesh>
```

Nori’s high-level overview - Scene file format and parsing

- Nori can have nested elements.

```
<mesh type="obj">
    <string type="filename" value="bunny.obj"/>

    <bsdf type="diffuse">
        <color name="albedo" value="0.5, 0, 0"/>
    </bsdf>
</mesh>
```



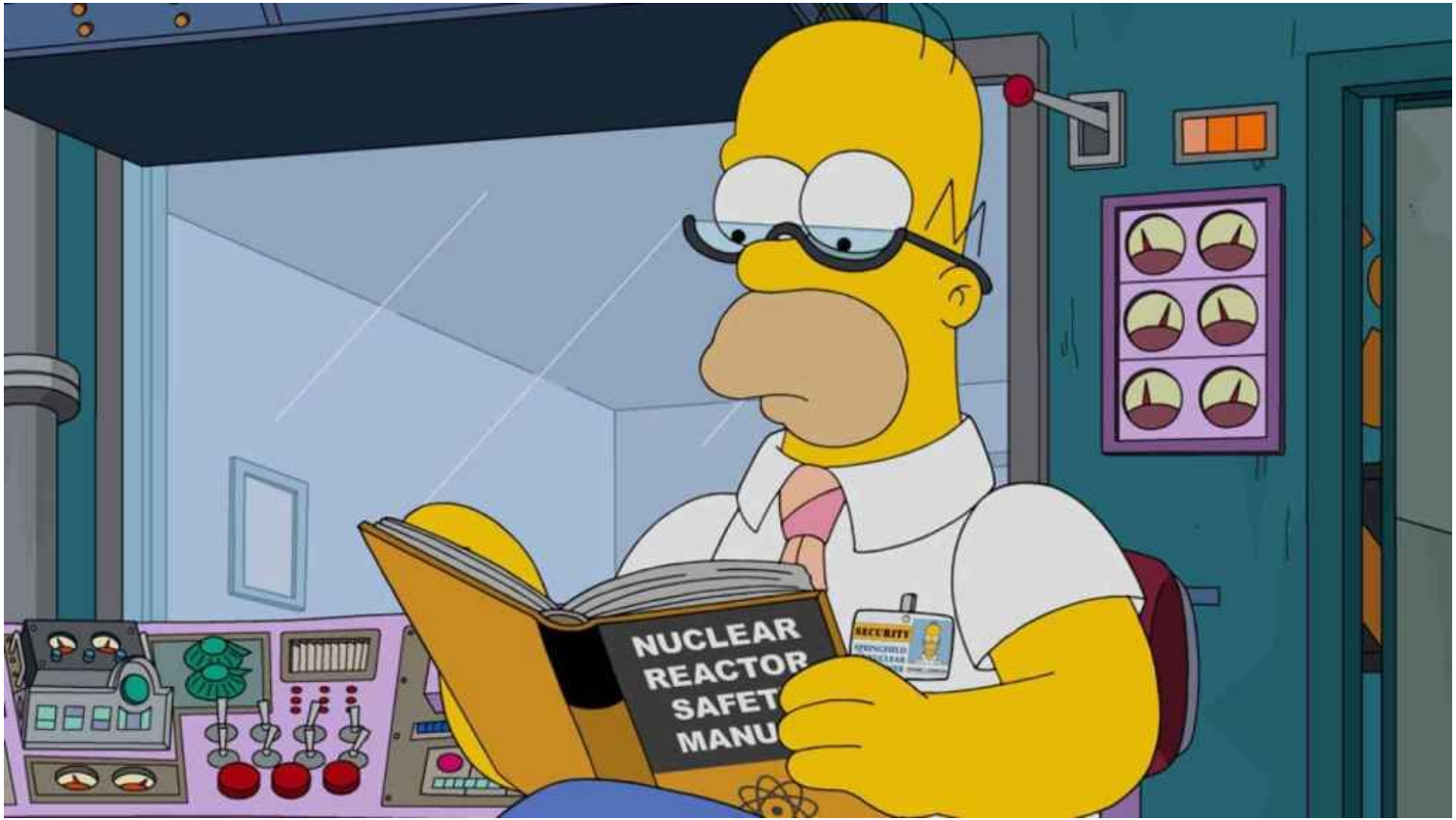
```
void Mesh::addChild(NoriObject *obj) {
    switch (obj->getClassType()) {
        case EBSDF:
            if (m_bsdf)
                throw NoriException(
                    "Mesh: multiple BSDFs are not allowed!");
            /// Store pointer to BSDF in local instance
            m_bsdf = static_cast<BSDF *>(obj);
            break;
    // ..(omitted)..
}
```

Nori’s high-level overview - Scene file format and parsing

- Nori uses a XML-based scene description language.

Task #4: Read the report and the different properties that you can pass within XML (Section 5.1)

Creating our first Nori class



Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

Task #5: Write the corresponding code (you can find it on page 9).

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

```
NormalIntegrator(constr PropertyList& props) {
    m_myProperty = props.getString("myProperty");
    std::cout << "Parameter values was: " << m_myProperty << std::endl;
}
```



```
protected:
    std::string m_myProperty;
```

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

```
std::string toString() const {
    return tfm::format(
        "NormalIntegrator[\n"
        "  myProperty = \"%s\"\n"
        "]",
        m_myProperty
    );
}
```

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

```
Color3f Li(const Scene *scene, Sampler *sampler, const Ray3f &ray) const {
    return Color3f(0, 1, 0);
}
```

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

```
NORI_REGISTER_CLASS(NormalIntegrator, "normals");
```


Links the class to the XML alias!

Creating our first Nori class

- We are starting with a dummy integrator to visualize **surface normals**.

- We create a new Nori object subclass in `src/normals.cpp`.

```
NORI_REGISTER_CLASS(NormalIntegrator, "normals");
```

```
153  /// Macro for registering an object constructor with the \ref NoriObjectFactory
154  #define NORI_REGISTER_CLASS(cls, name) \
155      cls *cls ##_create(const PropertyList &list) { \
156          return new cls(list); \
157      } \
158      static struct cls ##_{ \
159          cls ##_() { \
160              NoriObjectFactory::registerClass(name, cls ##_create); \
161          } \
162      } cls ##__NORI_;
163
164  NORI_NAMESPACE_END
```

Creating our first Nori class - Building

- We have to add our new class to our building system (`CMakeLists.txt`).

```
# The following lines build the main executable. If you add a source
# code file to Nori, be sure to include it in this list.
add_executable(nori
  # ...
  src/microfacet.cpp
  src/mirror.cpp
  src/normals.cpp
  src/obj.cpp
  src/object.cpp
  # ...
)
```


Task #6: Add your new .cpp file to CMake (you can check Section 6 of the report).

Creating our first Nori class - Building

- We have to add our new class to our building system (`CMakeLists.txt`).

```
# The following lines build the main executable. If you add a source
# code file to Nori, be sure to include it in this list.
add_executable(nori
  # ...
  src/microfacet.cpp
  src/mirror.cpp
  src/normals.cpp
  src/obj.cpp
  src/object.cpp
  # ...
)
```


This will have to be done for every single new class that you implement in Nori.

Creating our first Nori class - Building

- We have to add our new class to our building system (`CMakeLists.txt`).

- You can now **compile** your project.

- Creates a `nori` (or `nori.exe`) executable in the `build` directory.

Creating our first Nori class - Scene

- We are creating a very simple scene in XML.

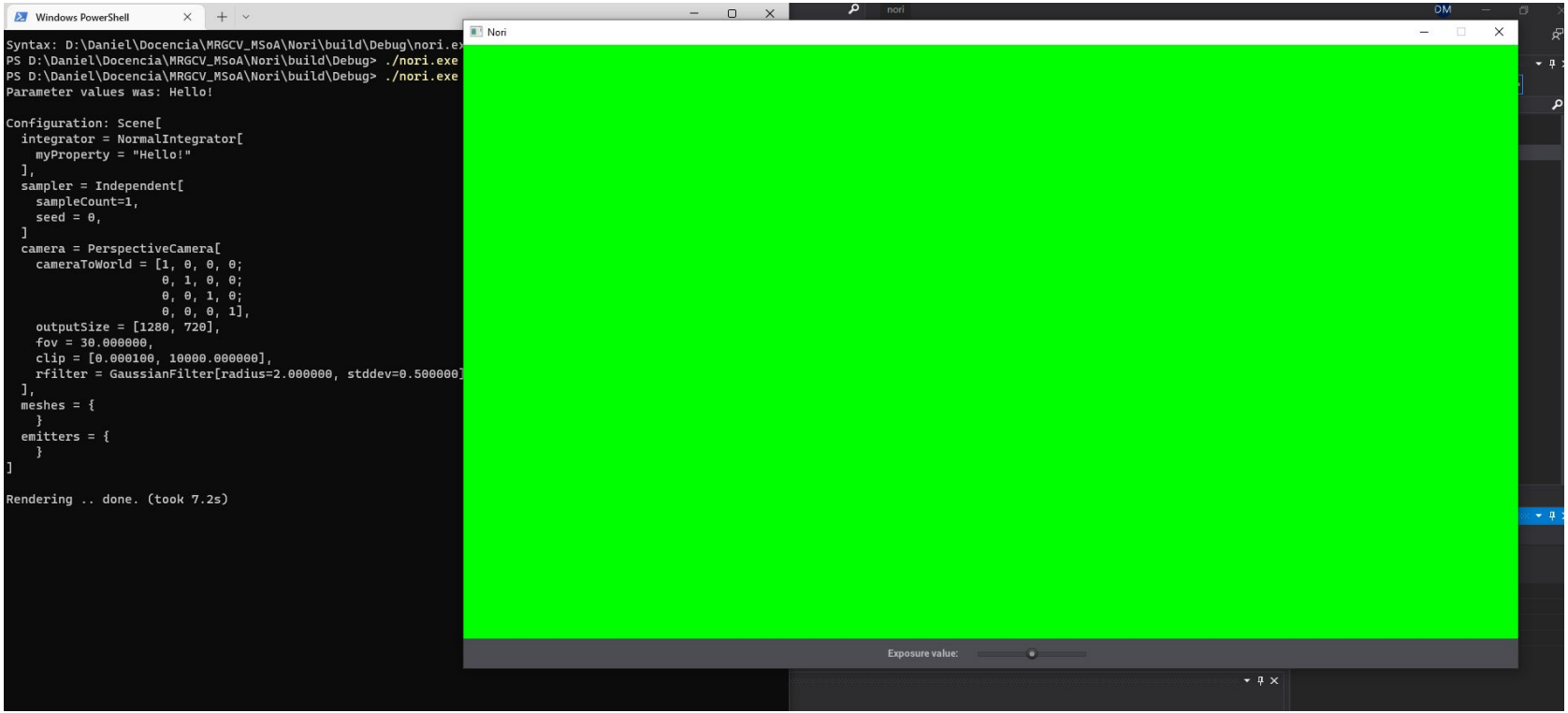
Task #7: Create a test.xml scene following the code on the report (page 10-11).

Creating our first Nori class - Running

- **Run** Nori with the scene .xml file as parameter.

Creating our first Nori class - Running

- **Run** Nori with the scene .xml file as parameter.



Creating our first Nori class - Tracing rays

- Build a more sophisticated integrator (changing `normals.cpp`).

Task #8: Change your normals.cpp file as stated in page 12 in the report.

Creating our first Nori class - Tracing rays

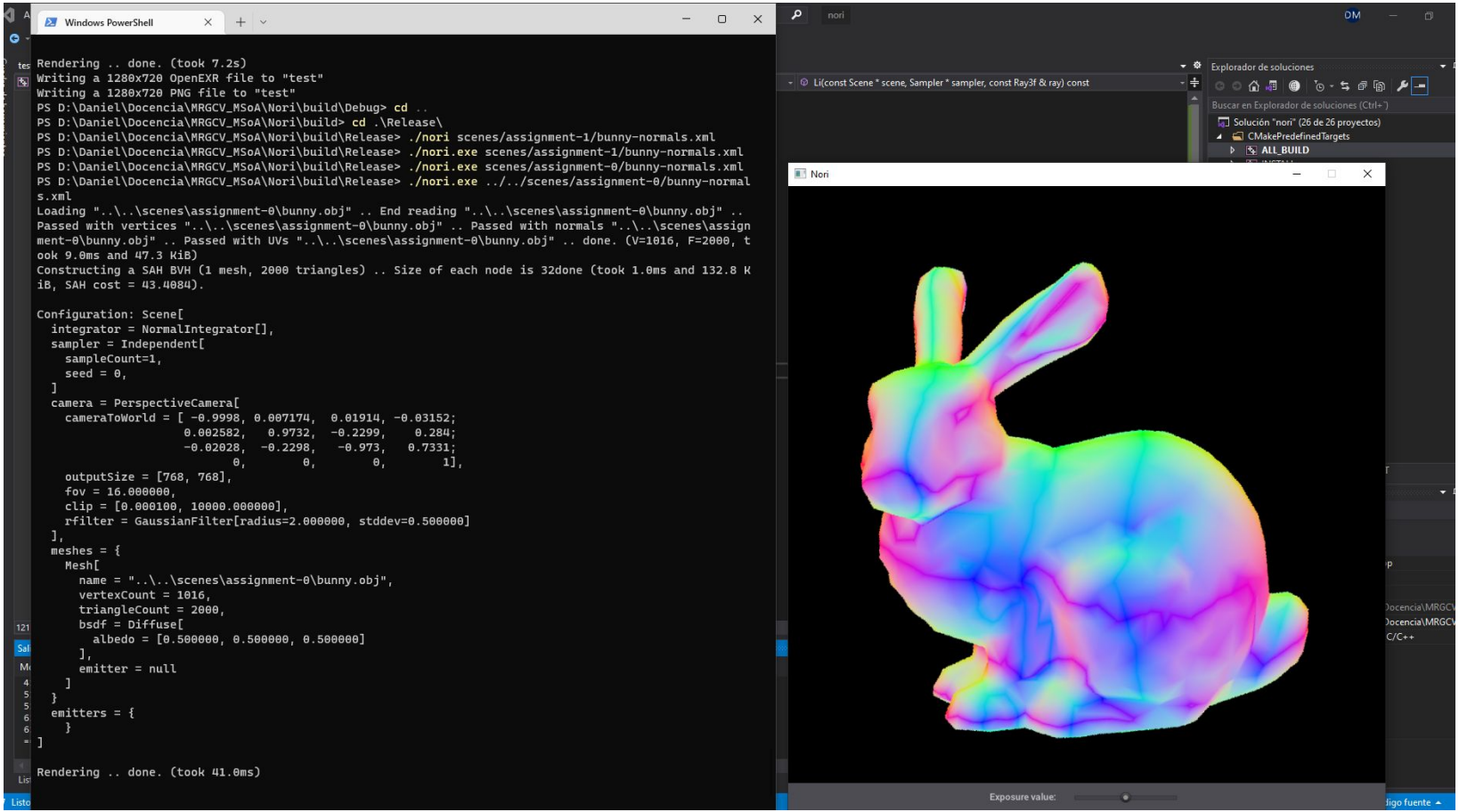
- Build a more sophisticated integrator (changing `normals.cpp`).

Task #8: Change your normals.cpp file as stated in page 12 in the report.

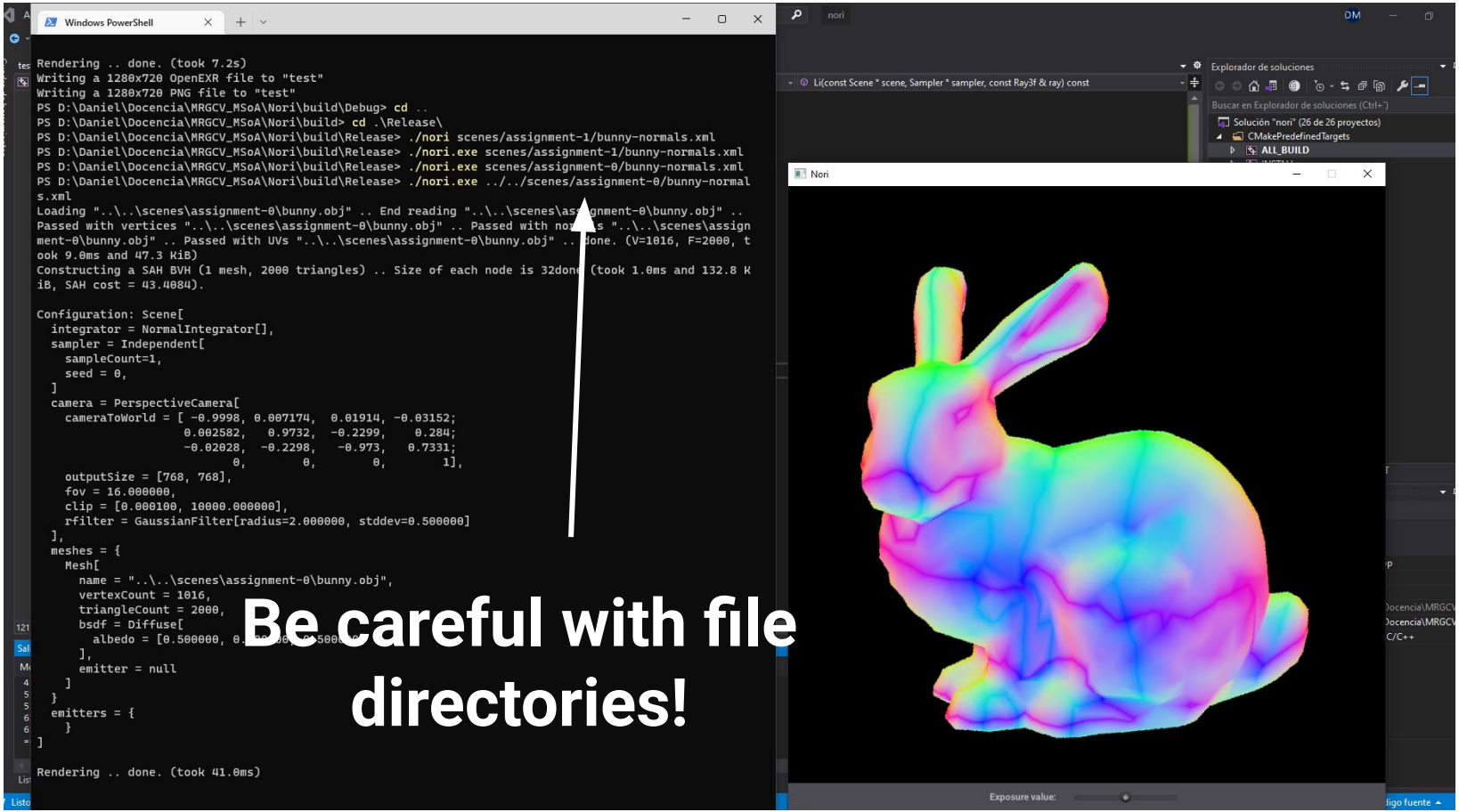
- Invoke Nori with the provided bunny XML file.

Task #9: Run Nori with the bunny XML.

Creating our first Nori class - Running



Creating our first Nori class - Running



Adding direct light from point sources

- Now we let you work at your own pace.

Adding direct light from point sources

- Now we let you work at your own pace, **but you should get here:**

What to submit

- This lab is **not evaluable**.

- Yet, you have to submit:

    - `.exr` and `.png` outputs for `bunny-normals.xml`.

    - `.exr` and `.png` outputs for `serapis-whitted.xml`.

    - `.exr` and `.png` outputs for `Serapis` with your `DepthIntegrator`.

- In a single `.zip` file named as `p1_NIP1_NIP2.zip`.

    Deadlines:

    * Group 2: October 16, 2024 at 23:59.

    * Group 1: October 21, 2024 at 23:59.