Lab 1
Due: XXX

This lab was designed after assignment 1 of Stanford's Image Synthesis class.

Objectives

- Get you familiarize with *pbrt*, the physically based rendering toolkit, an open-source software framework designed for physically based rendering.
- And other basic goals:
  - The concept of Literate Programming
  - The syntax of .pbrt files
  - To get you modify .pbrt files

Step 0: About PBRT

- PBRT
  - Textbook: https://www.pbr-book.org/3ed-2018/contents
- PBRT file format and example file.
  - A plain human-readable text format to describe the 3D scene to be rendered. It contains information about the geometry of objects in the scene, the materials of objects, lights, cameras, and rendering parameters.
  - A keyword-based syntax. Each line typically starts with a keyword followed by parameters. For example, keywords like "Camera" and "LookAt" are used to define cameras settings. Keywords like "translate", "rotate", and "scale" are used to apply transformations on the objects in the scene.
  - Comments in .pbrt file start with the '#' character. Comments are ignored by the PBRT renderer and are useful for adding notes or explanations to the scene description.
  - More scenes are available at scenes.

- PBRT has been developed and tested extensively on Linux, MacOS, and Windows; feel free to use whatever platform you're most comfortable with.

This lab is designed to motivate you to set up PBRT and render some test images.

Step 1: Build pbrt

- Please download the pbrt-v3 code from github with this command:

```
git clone --recursive https://github.com/mmp/pbrt-v3/
```

- PBRT uses cmake to build the system. You can find detailed instructions for setting up and compiling the PBRT on various platforms here.

Makefile builds (Linux, other Unixes, and Mac)

- Create a new directory for the build: `/path/to/pbrt-v3/build/`, change to that directory, and run `cmake [path to pbrt-v3]`. A Makefile will be created in the current directory. Next, run `make` to build pbrt. Depending on the number of cores in your system, you will probably want to supply make with the -j parameter to specify the number of compilation jobs to run in parallel (e.g. `make -j4` means run the make command with 4 parallel threads). Note: use `make -j4` on Linux/MacOS or `nmake /j4` on Windows.

  Follow these instructions:

```
cd pbrt-v3
mkdir build
cd build
cmake ..
make -j4
```

- Once you have successfully built PBRT, the final executables are ready in bin folder or Release or Debug folder depending on your compile mode. You can run PBRT explicitly from the command line by typing the full path to where the pbrt binary is located (for example, `./pbrt-build/pbrt`).

- (Optional) To be able to call the pbrt command from any folder, you can add full path of the folder to the execution environment on your OS. Then you can run the binary by only typing pbrt.
  - On Linux/MacOS, you can add the following command to your .bash_profile. The instruction is available here:

```
export PATH=/path/to/pbrt/bin/:$PATH
```

- On Windows, you need to go to the System's menu in Control Panel and add the full address of PBRT to your PATH environment variable. The instruction is available here.

Step 2: Rendering with PBRT

The lab1.zip contains starter files. After unzip the folder, you can render the image with the following command:

```
pbrt lighting.pbrt
```

After the rendering is done, it will print out statistics of the rendering process and the rendered image will be stored as lighting.exr. The output image's name is specified in the lighting.pbrt file under the Film directive.

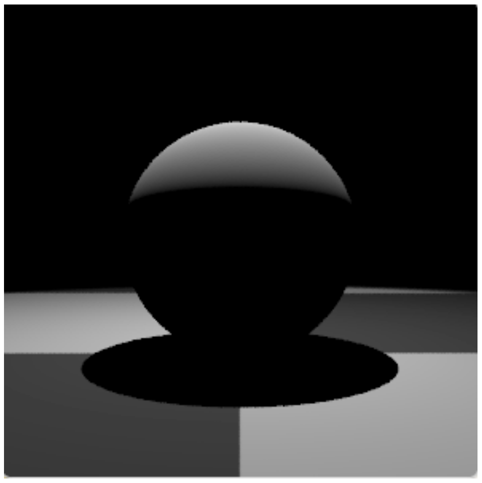
The EXR (OpenEXR) image file is a high-dynamic-range (HDR) image file format that is designed to store and exchange high-fidelity, high-dynamic-range images. EXR file has values that extend far beyond the typical 8-bit or 16-bit per channel range used in standard images. You can:

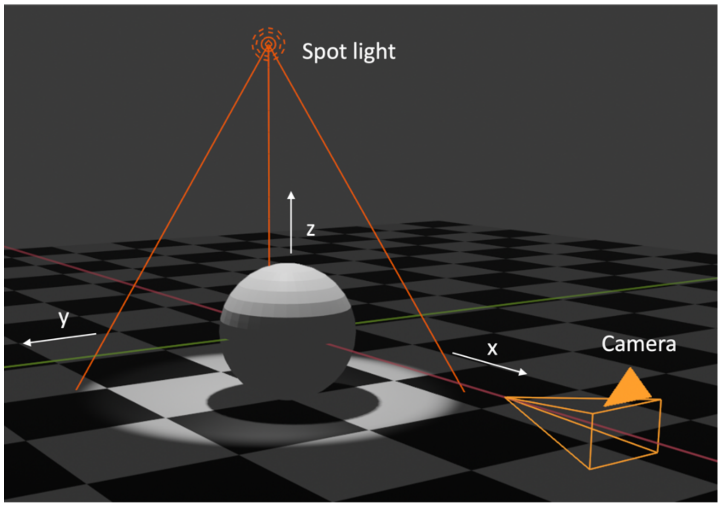
- Use the image processing/viewing tools which can do the conversion automatically (e.g. OS X's Preview, Adobe Photoshop, OpenEXR exrdisplay)
- Or use PBRT's built-in tool imgtool (built automatically when you built pbrt) to convert the image to the .png format.

```
imgtool convert lightings.exr lighting.png
```

We recommend converting to the .png format. Unless specified otherwise, please always convert your output images to .png for submission.

This is a correctly rendered image after you run `pbrt lighting.pbrt`:





The camera is located on the X-axis and is pointed at the center of the scene. A spotlight shines on top of the sphere and slightly behind the sphere, creating a hard shadow on the checkboard ground plane. The location of the camera and spotlight are illustrated in the image on the right.

.pbrt is editable with a simple text editor. Open `lighting.pbrt` with a text editor and go through the code and comments. Try to understand the parameters and syntax.

It would be helpful to go through the Camera, Lights, and Area Lights in pbrt-v3 file format manual before you start modifying the .pbrt scene file. Note that pbrt uses the **left-handed coordinate system**.

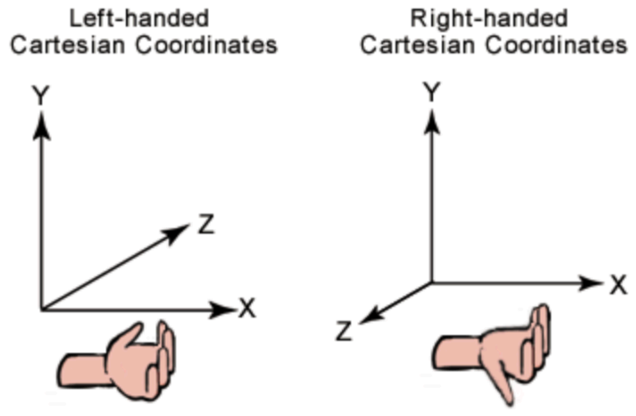


Image source: learn.microsoft.com

Changing Rendering Parameters to Speed Up Rendering

The image properties are defined under the Film:

```
Film "image" "string filename" ["lighting.exr"]
      "integer xresolution" [300] "integer yresolution" [300]
```

We can specify image properties such as output filename, extension, and resolution (with `integer xresolution` and `integer yresolution` parameters). Image resolution will affect the rendering speed. We recommend reducing the resolution when you are exploring and playing with parameters. You should use the original resolution to render the images for submission.

The number of ray samples that used to compute the value of each pixel also affects the rendering speed. More samples will slow down the rendering speed. The number of ray samples is specified in Sampler, for example:

```
Sampler "halton" "integer pixelsamples" [4]
```

To speed up your exploration process, you can use fewer samples. For submission, please render the image with at least 4 samples per pixel.

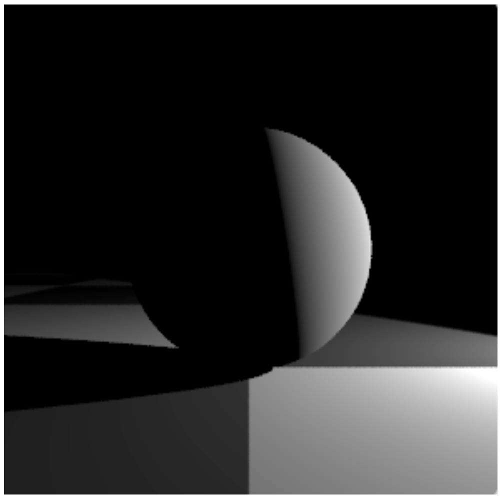
Step 3: PBRT Lighting

The main goal of this lab is to experiment with different lighting configurations, and material properties. We have six configurations.

Configuration 1: Spotlight from Right

In this configuration, we will move the spotlight so that it points up the cube from right. To do so, you will need to modify the spotlight parameters in `lighting.pbrt` file, in the spotlight section:

```
LightSource "spot" "color I" [50 50 50] "point from" [-0.5 0 4.1]
      "point to" [0 0 0] "float coneangle" [60]
```



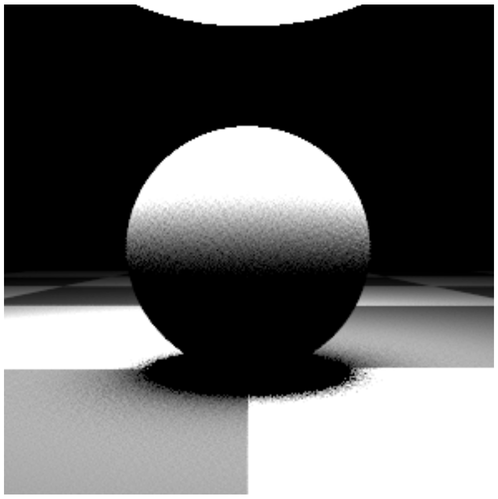
Your task is to find the right positions for point from and point to parameters of the spotlight to create a similar lighting above.

Configuration 2: Area Lighting from Top

The light generated by a spotlight originates from a single point. Light that is originated from a single point or a small area creates hard shadows, as shown in Configuration 1. Area lights can emit light from a larger region in space, resulting in a softer lighting of the object and a shadow with penumbra. The .pbrt scene file defines an area light as follows, which is initially commented out:

```
AttributeBegin
 AreaLightSource "area" "color L" [50 50 50]
 # adjust light source position
  Translate 0 6 0   # x y z
  Rotate 90 1 0 0  # angle x y z,  Note: this change the orientation of light source
 # define the shape of the arealight to be a disk with radius 1.5
 Shape "disk" "float radius" [1.5]
AttributeEnd
```

The shape of the area light is a disk with radius 1.5. Comment out the spotlight in Configuration 1 and modify the area light code above to generate the picture below:

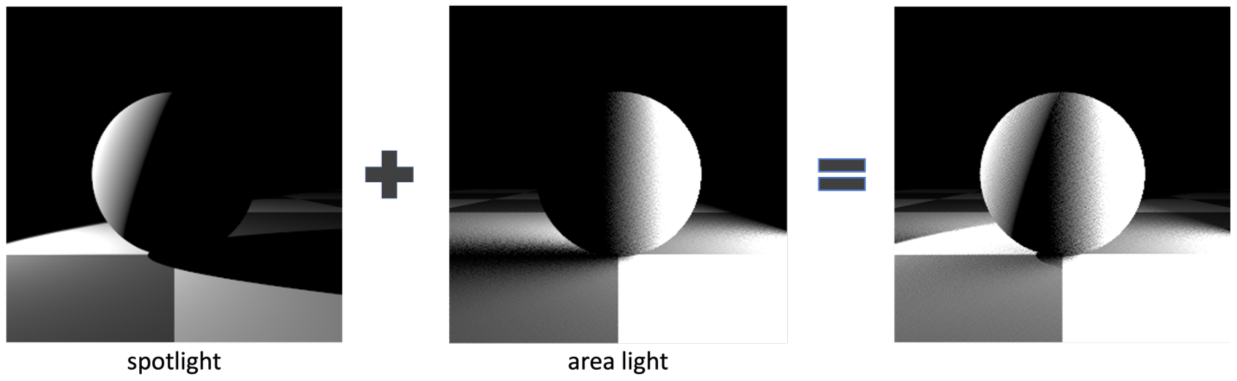


Note: you can increase the number of samples per pixel in Sampler to reduce noises in the rendered image.

Configuration 3: Two Light Sources

When a reflective surface is strategically placed opposite a light source, it can mitigate self-shadows on objects, promoting a more uniform illumination of the subject.

In this task, without changing the camera configuration, you will replicate this phenomenon by incorporating two light sources. First, illuminate the sphere object with a spotlight from left. Then, add a large area light on the right to give the effect of fill lighting on the dark side of the sphere. Render the scene with and without the fill lighting to observe the visual differences.

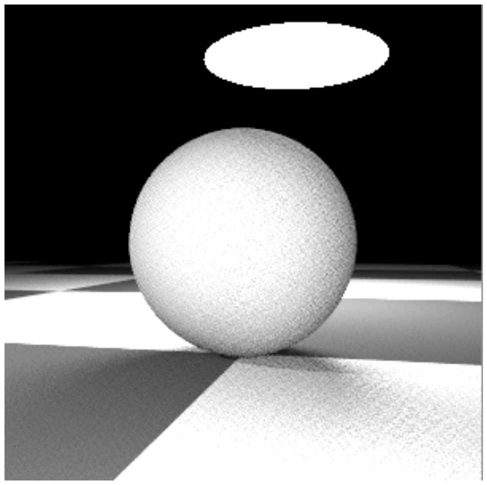


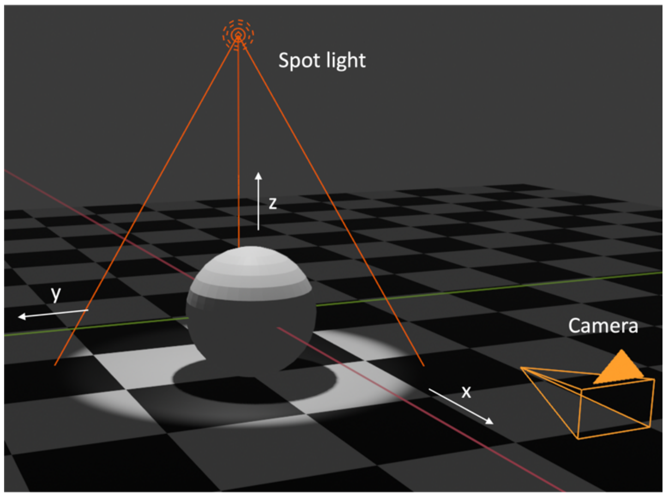
Configuration 4: Many Light Sources for Soft Lighting

Modify the lighting conditions to replicate the scene depicted below, in which three light sources are used in the scene. You will also need to change the camera configuration in this task. The LookAt directive specifies the position of the camera in world space, the location it is pointed at, and the up direction. Camera settings are defined in the pbrt scene file through the following lines:

```
LookAt 4.5 0 0   # eye
      0 0 0    # look at point
      0 0 1    # up vector
Camera "perspective" "float fov" 50
```

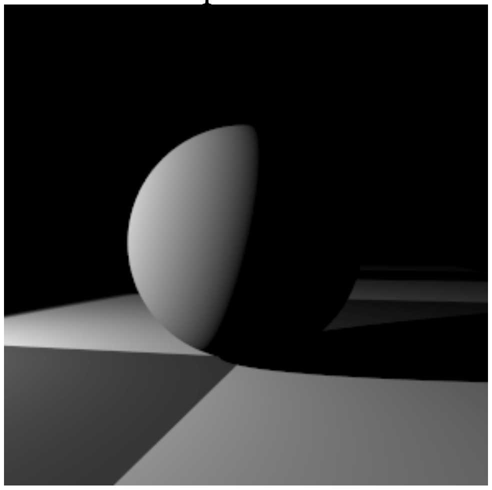
The original camera is placed on the X-axis and pointed towards the center of the scene. In this task, first, you need to change the camera settings in the LookAt directive. The camera is moved to see more right side of the scene, as illustrated below. Comparing the texture on the ground plane of the rendered image (left) with previous results, notice that the camera is moved.



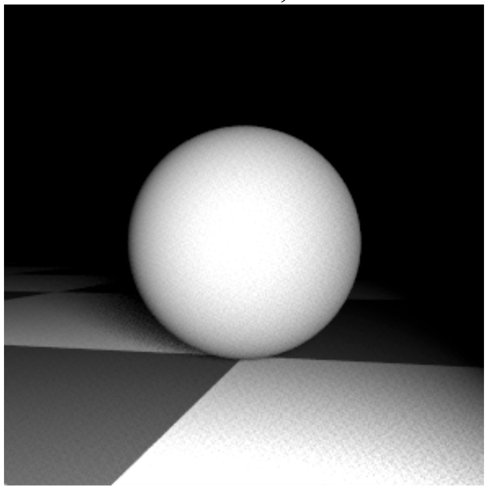


To help you find the right configuration of the three lights, the following three images show the separate effect of the three light sources:

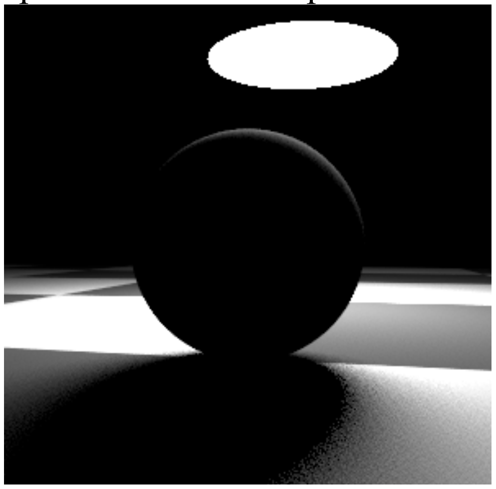
The main spotlight on one side of the sphere:



The fill light (area light) close to the camera, direct towards the object:



The background light that placed behind the sphere and facing towards the ground:



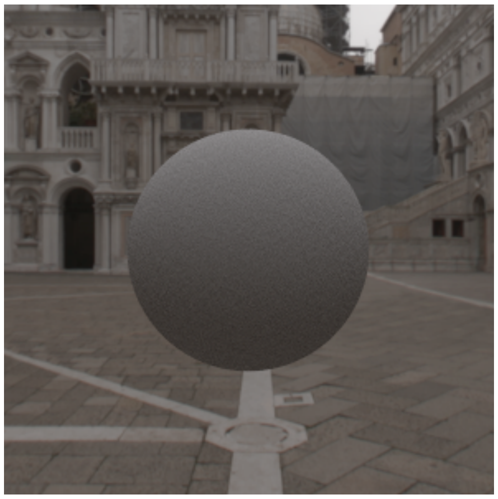
Configuration 5: Realistic Lighting

Manually placing lights is a time-consuming process and not very efficient in recreating the real environment. A popular technique to overcome this issue is to use a HDR "environment map" as a light source for the rendering. The environment map has captured light from all directions in a real-world environment and can simulate a real-world lighting for our model. The following environment map is provided for you in the assignment's zip file, at `textures/doge2_latlong.exr`:

Comment out all light sources, comment out the checkerboard ground plane, and uncomment out the environment map light:

```
LightSource "infinite" "string mapname" ["textures/doge2_latlong.exr"]
```

We will have the following rendering result:



Configuration 6: Material

The matte material is the simplest material in pbrt which describes a purely diffuse surface. Reflective materials have surfaces that bounce light back in a more organized and focused manner. This leads to the creation of highlights and sharper reflections, giving the object a polished and shiny appearance.

Your task in this configuration is to change the material of sphere into a reflective material, for example metal.

Step 4: Submission

Please try to do your best to mimic the lighting configurations in the example renderings, however we do not expect your images to be absolutely perfect matches. Credit will be given if the example renderings are reproduced to reasonable accuracy.

What to submit, a single zip file contains:

- Your solutions and rendered images for the six configurations in Step 3.
    - The scene file for each configuration with name `lighting_<CONFIG>.pbrt`. For example, the scene file for configuration 2 should be named as `lighting_2.pbrt`.
    - The images of each configuration. Please name them as `<CONFIG>_<NUM>.png`. For example, if you submit two images for configuration 4, you can name them `4_1.png` and `4_2.png`.
    - For your submission, use 300 x 300 image resolution and at least 4 samples per pixel. You can reduce the resolution and sample number if you simply want to test and explore pbrt's functionality.
- Be sure to submit any comments or remarks in a README.txt file. Also add information about any issued you encountered in README.txt file.