CS7.302: Computer Graphics

Assignment 2 [100 points]

Deadline: 11:59 PM, 5th Feb 2024

Welcome to your second assignment. In this assignment, you will implement direct lighting with point and directional lights and texture mapping. For this assignment, you will build on top of an updated version of the base code, which includes a two-level BVH implementation for ray-intersection tests and some other bug fixes. Please pull the latest version of the code from the GitHub Repository. The scenes required for this assignment have also been uploaded to scenes repository so make sure to pull the latest version for that too. Finally, the Blender Addon for exporting the scene has been updated to support exporting Point lights and Directional Lights (Sun) from Blender. Download the latest addon version (v0.2), uninstall the previous version from Blender, and reinstall the latest version. For all of the questions in the assignment, use a diffuse BRDF, i.e $f(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) = \frac{c(\mathbf{x})}{\pi}$.

1 Background

Recall that the light transport equation which describes the shading at a point $\mathbf{x}$ is given as:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = \int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) V(\mathbf{x}, \omega_\mathbf{i})(\omega_\mathbf{i} \cdot \mathbf{n}) d\omega_\mathbf{i}. \tag{1}$$

Refer to slides of lecture 5 for the exact definition of each term.

In the case of direct lighting with directional lights, this integral changes into a discrete sum over all the directional lights as follows:

$$L_o^\mathcal{D}(\mathbf{x}, \omega_\mathbf{o}) = \sum_{\omega \in \mathcal{D}} L_i(\mathbf{x}, \omega) f(\mathbf{x}, \omega_\mathbf{o}, \omega) V(\mathbf{x}, \omega)(\omega \cdot \mathbf{n}), \tag{2}$$

where $\mathcal{D}$ is the set of all directional lights in the scene with directions $\{\omega_1, \omega_2, \cdots, \omega_n\}$. Similarly, for direct lighting with point lights, the integral changes into a discrete sum over point lights as follows:

$$L_o^\mathcal{P}(\mathbf{x}, \omega_\mathbf{o}) = \sum_{\mathbf{p} \in \mathcal{P}} \frac{L_i(\mathbf{x}, \frac{\mathbf{p}-\mathbf{x}}{|\mathbf{p}-\mathbf{x}|})}{|\mathbf{p}-\mathbf{x}|^2} f(\mathbf{x}, \omega_\mathbf{o}, \frac{\mathbf{p}-\mathbf{x}}{|\mathbf{p}-\mathbf{x}|}) V(\mathbf{x}, \frac{\mathbf{p}-\mathbf{x}}{|\mathbf{p}-\mathbf{x}|})(\frac{\mathbf{p}-\mathbf{x}}{|\mathbf{p}-\mathbf{x}|} \cdot \mathbf{n}), \tag{3}$$

where $\mathcal{P}$ is the set of all point lights in the scene with positions $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$. Finally, the total radiance due to both point and directional lights can be found by summing the results of the above equations:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_o^\mathcal{D}(\mathbf{x}, \omega_\mathbf{o}) + L_o^\mathcal{P}(\mathbf{x}, \omega_\mathbf{o}). \tag{4}$$

For both questions of the assignment, you are supposed to use the diffuse BRDF, i.e

$$f(\mathbf{x}, \omega_\mathbf{o}, \omega_\mathbf{i}) = \frac{c(\mathbf{x})}{\pi}, \tag{5}$$

where $c(\mathbf{x})$ is the color of the shading point $\mathbf{x}$. For section 2, you should use a constant value of $c(\mathbf{x}) = (1, 1, 1)$ for all objects in the scene. In section 3 the color at each point will be given by a texture map.

2 Direct Lighting with point & directional lights [50 points]

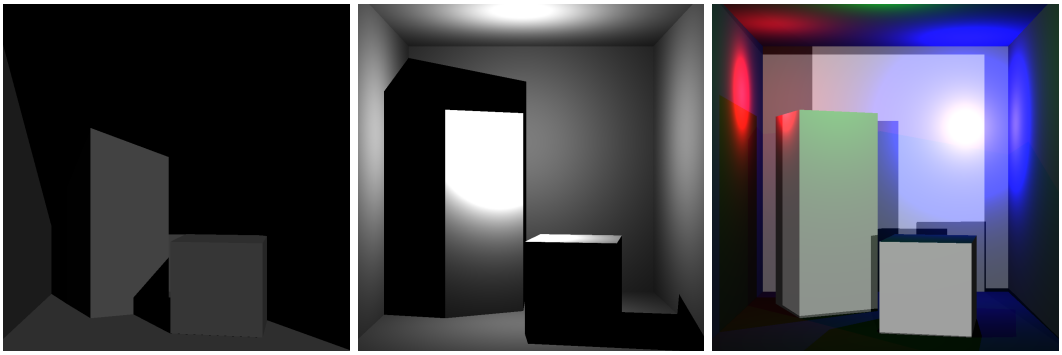

Figure 1: The Cornell Box scene rendered with a single directional light (left), a single point light (middle), and multiple colored point / directional lights (right)

.

In this question, you will implement direct lighting in the presence of directional and point lights. The question has been broken down into simpler sub-parts.

2.1 Define & load lights [10 points]

The information about the lights is stored in the scene **.json** file. Refer to point_light.json and directional_light.json for an example of how they are stored. In this question you are required to do the following things

- Fill the `Light` structure with the fields that are required for each type of light in light.h.
- Implement a load function in light.cpp which takes the `sceneConfig` as input and returns a `std::vector` of `Light` which you should then store in the `Scene` class.

2.2 Implement direct lighting with directional lights [20 points]

In this question, you must write the code to render shading of the scene due to directional lights (see eq. (2)). Modify the `Integrator::render` function to iterate over each directional light and sum the contributions due all directional lights in the scene.

2.3 Implement direct lighting with point lights [20 points]

Further modify the function to incorporate shading due to point lights (see eq. (3)). Sum the contributions from directional and point lights to get the final render.

3 Texture Mapping [40 points]

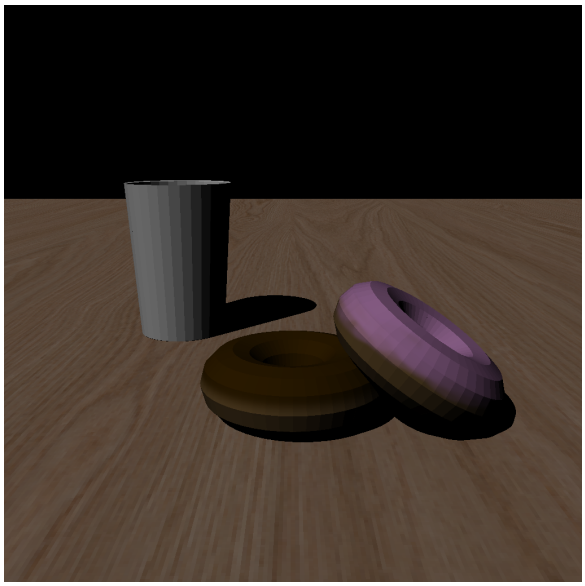

Figure 2: The textured donut scene rendered with nearest-neighbor interpolation

In this question, you will be implementing texture mapping in the renderer. Each object in the scene may have a texture attached to it, describing the diffuse color $c(\mathbf{x})$ at point $\mathbf{x}$. If the texture is missing for an object, then a constant diffuse color will be assigned to the object. You may use the `Surface::hasDiffuseTexture` to query whether an object has a texture associated with it. If the object doesn't have a texture associated with it, then access the constant diffuse color using the `Surface::diffuse` property.

All triangles of all objects have uv coordinates stored which may be queried using the `uv1,uv2,uv3` property of the triangle. Once the intersection point is found, implement barycentric interpolation to fetch the uv coordinates at the intersection point and store this in the `Interaction` struct.

For both of the following subquestions, you will need to use the `Texture::loadPixelColor` which returns the pixel color at integer coordinates x, y.

3.1 Nearest Neighbour fetch of textures [20 points]

Implement a new function `Texture::nearestNeighbourFetch` in texture.cpp which takes the uv coordinates as input and returns the color according to nearest neighbor interpolation strategy.

3.2 Bi-linear interpolation for texture fetches [20 points]

Implement a new function `Texture::bilinearFetch` in texture.cpp which takes the uv coordinates as input and returns the color according to the bilinear neighbor interpolation strategy.

Finally, use the color obtained from the above functions in the rendering loop to replace the $c(\mathbf{x})$ by the value returned from the texture lookup.

4 Your own scene with point lights and textures[10 points]

In this question, you must create your own scene in Blender, texture it, and render it in your renderer. Your scene should contain the following things:

- A simple model that you yourselves have created.
- The model should be UV-mapped. You may use the Smart UV Project tool in Blender to create the UV-Mapping.
- The model should be assigned a texture. The renderer only supports loading **.png** textures, so only use those.
- There should be one point light and one directional light in the scene.

Export the scene using the updated Blender Exporter and submit the scene in the final submission. Note that the scene won't be graded on its aesthetic quality but rather on the presence of all of the points noted above.

5 Submission

You need to submit the modified code along with a report. Add a command line parameter which will allow you to choose the method for performing texture interpolation. The renderer should be invoked as follows:

```
./render <scene_path> <output_path> <interpolation_variant>
```

The `interpolation_variant` will be an integer from $0 - 1$ where each integer is defined as follows:

- 0: Nearest-Neighbour Interpolation
- 1: Bilinear Interpolation

The report should consist of images for all scenes provided to you in the GitHub Repository alongside the time required for each to render. We have provided the ground-truth images rendered with a reference renderer in the **gt** directory for each scene. You may use these to verify the correctness of your code. Note that your code will be tested against different scenes that haven't been provided during evaluations. Make a **.zip** file with the modified code, report, and scene you created. You may delete the **.git** and **extern** folders from the code before submitting to avoid the file-size limit issues on Moodle.