CSc 305 - Spring 2020
Introduction to Computer Graphics
Assignment 4
University of Victoria

Due: April 12, 2020 at 23:55. **Late assignments will not be accepted.**

1 Overview

The purpose of this assignment is to give you the opportunity to continue exploring and expanding your ray tracing system. The end goal is to produce the best possible image with your choice of implemented features.

Unlike previous assignments, all of the marks for this assignment are based on optional features (outlined in the next section). However, your assignment must have all of the following to be considered valid.

- A virtual camera system,
- point lights,
- diffuse shading,
- multisampling,
- out-of-gamut colour handling, and
- a variety of geometries.

These features define a common baseline for all implementations. If your submission does not contain at least these features, you will receive a mark of zero. You are encouraged to add as many features of the features in the next section as you can, even if they exceed to maximum number of points.

2 Specification

The marks for this assignment will be provided through the implementation of the features outlined below. The final rendered image must be no smaller than 1000×1000 pixels. The success of your final rendered image is based on whether all of your implemented features are visible and not purely on your implementation. You are therefore encouraged to experiment with different colours, sizes, and positions to ensure that all your geometries are visible in the final render.

Features

For full marks, your assignment must implement some of the features below. Note that some features are worth more marks than others (based on their difficulty). You may implement as many features as you want, but your final mark will be capped at 100%. For tasks that require you to take timings, you may use the provided `Timer` class contained in the `atlas/core/Timer.hpp` header.

- **Parallelization:** For all features in this category, you will use the standard threading library provided by the STL. Note that the three options in this section are mutually exclusive (it is not possible to implement more than one). You will also need to provide a comparison of timings between a regular render (without the use of parallelization) and with the parallel implementation.
  - [1 mark] Basic parallelization: divide the image into a grid of $n \times n$ sections, which we will call *slabs*. This number can be equal to the maximum number of threads the system supports. Each thread is then assigned a slab and the output is rendered normally.
  - [2 marks] Intermediate parallelization: divide the image into a grid of $n \times n$ slabs, where n is larger than the maximum number of threads the system supports. The slabs are then distributed across the threads.
  - [3 marks] Advanced parallelization: divide the image into slabs based on the distribution of objects in the scene. Sparse areas will have fewer slabs, whereas dense areas will contain a larger number of slabs. The slabs are then distributed across threads ensuring to maintain an even load balance.
- **Accelerator Structures:** Note that all features in this category are mutually exclusive (it is not possible to implement more than one). In order to showcase these features, your scene most contain *at least* 10 primitives. You will also need to provide a comparison of timings between a regular render (without the accelerator structure) and with the structure enabled.
  - [1 marks] Implement regular grids. See [Suffern 2007, chapter 22]
  - [2 marks] Implement a bounding-volume heirarchy (BVH). See [Pharr, Jakob, and Humphreys 2016, chapter 4.3].
  - [3 marks] Implement KD-trees. See [Pharr, Jakob, and Humphreys 2016, chapter 4.4].
- **Reflections:** For all features in this category, your final render must contain objects employing the reflection type. If you implement both, then you must have at least one object with mirror reflection and one with glossy reflection.
  - [1 mark] Implement mirror reflections. See [Suffern 2007, chapter 24]
  - [1 mark] Implement glossy reflections. See [Suffern 2007, chapter 25]
- **Global Illumination:** Note that all features in this category are mutually exclusive (it is not possible to implement more than one). Be aware that, with the exception of path tracing, these features will be considerably harder as they will require reading, understanding, and implementing a research paper. It is recommended that you only attemp these if you have enough time.

  - [1 mark] Implement simple path tracing. See [Suffern 2007, chapter 26].
  - [2 marks] Implement Bidirectional path tracing. See [Lafortune and Willems 1993] and [Veach and L. Guibas 1995].
  - [3 marks] Implement photon mapping. See [Jensen and Christensen 1995].
  - [4 marks] Implement Metropolis light transport. See [Veach and L. J. Guibas 1997].

- **Transparency:** The two types of transparency are mutually exclusive (so you will not receive the marks for simple transparency if you implement realistic transparency).
  - [1 mark] Implement simple transparency. See [Suffern 2007, chapter 27].
  - [2 marks] Implement realistic transparency. See [Suffern 2007, chapter 28].

- **Textures:** For all features in this category, you must have at least one object showcasing the type of texture that you implemented.
  - [1 mark] Implement regular texture mapping. See [Suffern 2007, chapter 29].
  - [1 mark] Implement procedural textures. See [Suffern 2007, chapter 30].
  - [1 mark] Implement noise-based textures. See [Suffern 2007, chapter 31].

- **Miscellaneous:**
  - [1 mark] Implement shadows. See [Suffern 2007, chapter 16].
  - [1 mark] Implement ambient occlusion. See [Suffern 2007, chapter 17].
  - [1 mark] Implement area lights. See [Suffern 2007, chapter 18].
  - [1 mark] Render a mesh.
  - [1 mark] Implement a scene graph. For this task, note that you must have a hierarchy that is *at least* two levels deep.

All references mentioned here have been posted on conneX and are freely available through the UVic library.

You will be provided with an assignment bundle that will contain the following:

1. A full CMake setup for the assignment similar to the one seen in the labs,
2. a `main.cpp` file containing a function to save the image to a file as seen in the labs and an empty main, and
3. a blank `assignment.hpp` header.

Your submission must include the following:

1. One `main.cpp` file containing your implementation.
2. One `assignment.hpp` file containing the definitions of any auxiliary data structures, functions, etc that you may require. Note that you may leave this file blank if necessary.

3. One `README.txt` file containing a list of all features that you have implemented. If you choose to implement any accelerator structures or parallelization options, then the file must also contain the two timings between the regular render (without the feature) and the render with the feature enabled.

4. One `render.bmp` file containing your submission for the competition.

Please note that the full assignment bundle does not need to be provided in your submission, only the individual files listed above. For the purposes of this assignment, you will be allowed to submit a single archive containing all of the aforementioned files.

3 Evaluation

This assignment is worth 14% of your final grade and will be marked out of 10. Your code will be compiled and run in ECS 354 and will be graded according to the following rubric:

Parallelization	**Marks**
Basic parallelization	1 mark
Medium parallelization	2 marks
Advanced parallelization	3 marks
Super advanced stuff	4 marks
Accelerator Structures	**Marks**
Implement regular grids	1 mark
Implement a BVH	2 marks
Implement KD-trees	3 marks
Reflections	**Marks**
Implement mirror reflections	1 mark
Implement glossy reflections	1 mark
Global Illumination	**Marks**
Implement simple path tracing	1 mark
Implement bidirectional path tracing	2 marks
Implement Metropolis light transport	3 marks
Implement photon mapping	4 marks
Transparency	**Marks**
Implement simple transparency	1 mark
Implement realistic transparency	2 marks
Textures	**Marks**
Implement regular texture mapping	1 mark
Implement procedural texture	1 mark
Implement noise-based textures	1 mark
Miscellaneous	**Marks**
Implement shadows	1 mark
Implement ambient occlusion	1 mark
Implement area lights	1 mark
Render a mesh	1 mark
Implement a scene graph	1 mark

Your implementation is expected to follow best practices for object oriented design in C++ (using a similar level of object integration as the code you have seen during the labs). If your code does not use an object-oriented design, or exhibits bad style, you may be deducted up to two marks from your grade.

Your code must compile and run correctly in ECS 354. If your code cannot be compiled or if it crashes during the demo, you will receive a mark of zero. If you do not submit the `README.txt` or

`render.bmp` you will receive a mark of zero.

References

Jensen, Henrik Wann and Niels Jørgen Christensen (1995). "Photon maps in bidirectional Monte Carlo ray tracing of complex objects". In: *Computers & Graphics* 19.2, pp. 215–224.

Lafortune, Eric P and Yves D Willems (1993). "Bi-directional path tracing". In:

Pharr, Matt, Wenzel Jakob, and Greg Humphreys (2016). *Physically Based Rendering: From Theory to Implementation*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 0128006455.

Suffern, Kevin G. (2007). *Ray tracing from the ground up*. 1st ed. Wellesley, Mass: A K Peters.

Veach, Eric and Leonidas Guibas (1995). "Bidirectional estimators for light transport". In: *Photorealistic Rendering Techniques*. Springer, pp. 145–167.

Veach, Eric and Leonidas J Guibas (1997). "Metropolis light transport". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 65–76.