Path tracing in Julia

Andrea Princic
1837592

Valerio Venanzi
1852473

March 25, 2023

Introduction

This project is about porting the path tracer seen in class to the Julia language. Using Yocto/GL as the base, we ported almost all of its features to Julia, paying attention to what is important to write efficient Julia code.

Implementation

We followed a bottom-up methodology, starting from the basic data structures, scene I/O and BVH, and then moving to a simple naive tracer that only supported intersections.
We then started implementing materials and textures, until the naive tracer was fully functional. After the naive was completed we moved to the more advanced path tracer, which supports light sampling and volumes.
We also implemented almost the same CLI interface as Yocto/GL.

Optimizations

Data structures

We managed to use immutable data structures wherever possible, which allows Julia to optimize the code better. We started by using mutable structs almost everywhere until the naive tracer was completed, and then we started rearranging things to make them immutable. This resulted in a significant speedup.

Allocating memory

After playing with Julia for a while, we realized that allocating memory is very slow compared to other languages. The problem with allocating memory comes out especially when dealing with BVH, since every time we need to compute an intersection, we allocate a stack to traverse the BVH.
First we tried to reduce the maximum depth from 128 to 32, but almost no speedup was achieved. Then we had the idea to use a pre-allocated array to store the stack directly into the main function, and pass it to the sampling function which passes it to the intersection function. We allocate one stack

for each thread, and then we use the thread ID to access the correct stack. We do the same for the volume stack.

Results

Unfortunately we weren't able to load HDR images, because apparently their values get automatically clamped to $[0, 1]$ by the Julia image library. We decided not to spend too much time trying to understand why and how to fix this, since it's not a problem with our code and the result is still correct. Following are some images we rendered.

Figure 1: naive sampler struggles to find light from the environment while light sampler finds it easily

Figure 2: there is no difference between naive and light sampling since the only light source is the environment

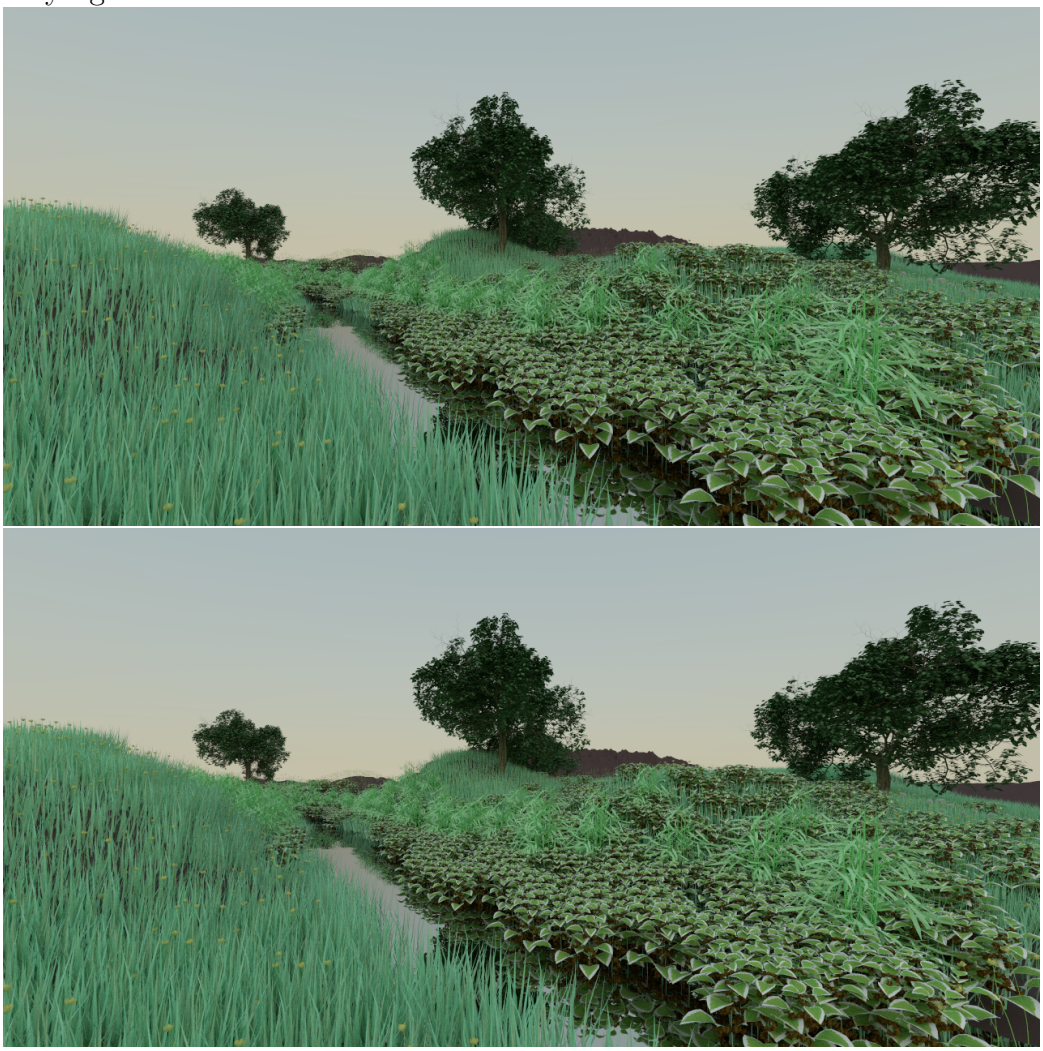

Figure 3: naive sampling doesn't work with volumes

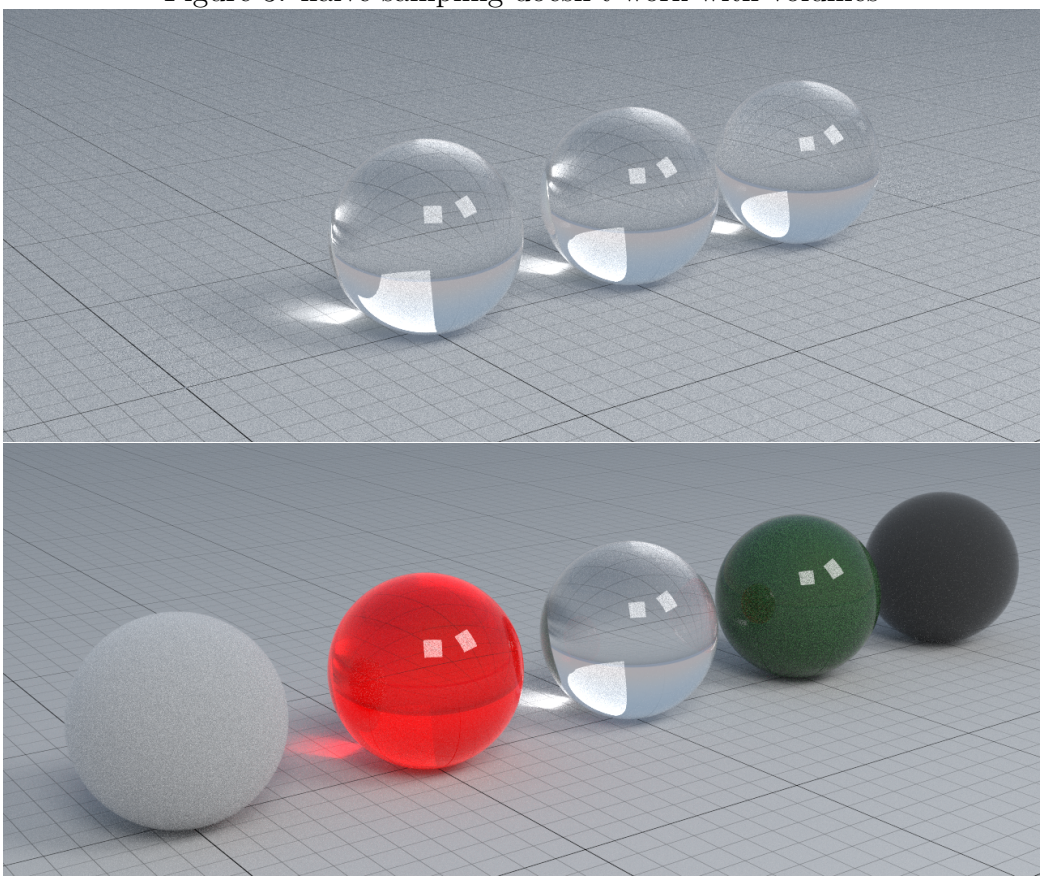

Figure 4: HDR images get automatically clamped, producing a result that looks “cold” with respect to the Yocto/GL version