UCSD CSE 168 Assignment 5:

Final Project

The final project will be a project of your choice. You can implement some non-trivial rendering algorithms, or render some interesting scenes in torrey or your own renderer. It can also be a rendering-related research project with a new algorithm or system. Below we provide some options for you to think about, but feel free to come up with your own ones.

Grading. The project will be graded based on three aspects: technical sophistication, artistic sophistication, and report quality. You can have a project that does not have much technical component, but it renders beautiful images. Alternatively, you can have a project that produces programmer arts, but is highly technical. To encourage people to pursue ambitious projects, the more ambitious the project is, the less requirement that is to the completeness of the project. For example, if you aim to work on a research idea that is novel enough to be published at SIGGRAPH, we will not judge you using the standard of a technical paper. It is sufficient to show your work and the initial prototypes and experiments you develop, even if the initial result does not show practical benefit yet. The key is to show how much you have learned in the process.

Logistics. Before 5/19, please let us know what you plan to do for the final project by submitting a brief report to Canvas. Schedule a chat with us if you have any question. We will have a checkpoint at 6/5: send us a brief progress report in a few pages on Canvas (ideally provide results and images) describing what you did and what you plan to do next. Let’s have a maximum of two people in a group.

1 Ideas

The general recommendation is to add some features to your renderer. Try to come up with some cool scenes to demonstrate the features you added. Following are some features that you can add to torrey. For projects marked with very easy or easy, you might need to implement a few of them.

Bump/normal mapping (very easy). One way to add more details to surfaces is to use a bump map or a normal map: textures that modify the shading normal. Read PBRT for detail about bump mapping and add it to your renderer.

Anisotropic BRDFs (very easy). The BRDFs we implemented in the homework are all isotropic, but real world materials often exhibit anisotropic highlights. Read PBRT and implement some anisotropic BRDFs.

Environment maps/image-based lighting (easy). These are infinitely far away light sources with directionally varying intensity. Our background in torrey is actually already secretly a constant color environment map. Read PBRT for some references.

Mipmapping (easy). Currently we lookup our textures using a bilinear interpolation. A better way is to prefilter the textures. Read Chapter 10.1 and Chapter 10.4 of PBRT and implement texture filtering in your renderer. Show some scenes where it makes differences.

Curve geometry (easy). It is common to use Beizer curves, instead of triangles, to represent hair and fur or other thin geometry. Read PBRT and implement ray curve intersection in your renderer.

Nonlinear motion blur (easy). The bonus motion blur discussed in the RTNW book only supports translation and does not handle rotation and scaling. Read PBRT and implement motion blur that supports rotational and scaling motion.
**Tonemapping (medium).** Implement advanced tonemapping algorithms and compare their efficiency. A few good candidates are: Reinhard’s photographic tone mapping, Durand’s bilateral filtering, Drago’s adaptive log mapping.

**Texture cache (medium).** In production rendering, the textures may not totally fit in the memory. Therefore, we need to read textures from hard disk on demand and cache them in the memory. Read Matt Pharr’s note on texture caches and implement it in your renderer.

**Realistic camera (medium).** It is possible to simulate the whole lens system in a ray tracer. This allows us to reproduce distinctive characteristic of real cameras e.g., depth of field, vignetting, spherical aberration, etc. Read PBRT for how to implement a lens system in a renderer and implement it in your renderer.

**Low-discrepancy sampling (medium).** In production rendering, independent random sampling is rarely used. In practice, we always apply stratification to our random number sequences. Pick a low-discrepancy sampling sequence in PBRT, e.g., Halton/Hammersley, \((0, 2)\), or Sobol, and implement it in your renderer.

**Irradiance caching (medium).** Implement irradiance caching or its variants.

**(Progressive) photon mapping (medium).** Implement photon mapping or its variants.

**Path guiding (medium).** Implement some version of path guiding. The 5D tree from Eric Lafortune might be a good start.

**Light baking and precomputed radiance transfer (medium).** Use your path tracer to bake directionally dependent global illumination into textures or light probes, and load it in a real-time renderer.

**Denoising and adaptive sampling (medium).** Implement a Monte Carlo denoiser and/or add adaptive sampling features to it.

**Vector graphics renderer (medium).** Implement a 2D vector graphics renderer that supports closed Bezier curves and strokes.

**Layered BSDFs (medium).** Implement Weidlich and Wilkie’s Arbitrarily Layered Micro-Facet Surfaces for rendering layered BSDFs.

**Watertight ray triangle intersection (medium).** Our current code does not ensure a robust intersection between rays and triangles and can be inaccurate due to floating point cancellation error. Read PBRT for a more robust strategy for handling intersection.

**Multiple-scattering volume rendering (medium-hard).** Read Homework 2 of CSE 272 and implement it.

**Order-independent transparency (medium-hard)** Implement a real-time renderer that supports transparency. Compared different strategies: depth peeling, linked list sorting, and/or moment-based order independent transparency.

**Tessellation-free displacement mapping (hard).** A higher-fidelity way to add details to surfaces compared to normal mapping is displacement mapping. Instead of perturbing normals, we perturb positions. It is possible to apply displacement mapping on-the-fly without preprocess the displacement map into a detailed mesh. Read Tessellation-Free Displacement Mapping for Ray Tracing and implement the method.

**Moment shadow mapping (hard).** Implement moment shadow mapping in a real-time renderer.
Bidirectional path tracing (hard). Implement Bidirectional Path Tracing.

Fourier BSDF (hard). Implement Fourier BSDF for rendering measured BSDFs.

GPU rendering (hard). Port torrey to modern GPU hardware (CUDA/OpenCL/OptiX/Vulkan/DirectX).

Level of details (hard). Read Pixar’s Stochastic Simplification of Aggregate Detail and implement the method.