First, some high-level tips (regardless of project topic)

1. In my class, projects scored 20–30% lower on average than assignments. So work hard, start early.
2. Keep objectives small and achievable. Your project grade largely adheres to the 8 (or so) goals you set in your proposal. So write easy, achievable objectives. You can always extend and add features once you've checked the boxes. But not checking a box because you were too ambitious will lower your grade
3. Maximize visual quality with minimal effort.
    - Don't pick fancy but time-consuming or not visually rewarding features (new file formats, external libraries, low-level speed/memory optimizations).
    - Building photon mapping is a great project idea: you'll reuse most of your pipeline and get impressive effects with little coding and additional learning.

Now, a suggested order of objectives for your described interests.

The items can all be individual objectives or even several. You'll need to do a lot of reading on your own, especially for the last few.

1. **Offline renderer & basic path tracing**
    - Refactor the renderer to write a single .ppm image instead of to an interactive GLFW window (I recommend this viewer for .ppm files `https://github.com/Tom94/tev`).
    - Unidirectional path tracer. Yes, it is a good idea to get PT working first, these effects are important for photon mapping:
        - Emissive light sources
        - Monte Carlo integration over the BSDF (1/PDF weighting)
        - Recursive ray tracing with Russian roulette (e.g. terminate after ¿3 bounces)
        - Combine direct + indirect lighting
    - Also people often forget: jitter your camera-ray samples
2. **Basic photon mapping**
    - Phase 1:
        - Define a photon struct, emit photons from lights, trace paths, and store each diffuse hit in a KD-tree.
    - Phase 2:
        - For each diffuse surface, query photons within radius r and estimate density (using the BSDF, nearby photon flux/direction).

    (Good barebones implementation: `https://github.com/yumcyaWiz/photon_mapping`)

    At this point, you have a pretty solid project already. The rest are some ideas that could make your project very rigorous. I don't expect you to complete them all. Even a few will be very impressive. These are just ideas I came up with quickly to inspire you:
3. **Advanced extensions**
    - Homogenous Volumetric scattering in pathtracer. Supports very cool phenomena like god-rays, beautiful volumetric caustics (`https://blenderartists.org/t/volumetric-caustics-in-cycles/647570`). The theory is kind of complex, but it's easier than it sounds to implement once you understand it: In a participating medium, light is attenuated by two processes: absorption and scattering. Instead of tracing rays directly to the next surface, a distance to the next scattering/absorption event is sampled stochastically from an exponential distribution. If the sampled distance is shorter than the distance to the next surface, an absorption or scattering event occurs: if its absorption, the ray is terminated, if it scatters, the ray is attenuated

(Beer-Lambert law) and a new direction is sampled. This process repeats until the ray escapes the medium, hits a surface, or is terminated.

All mediums will have different scattering and absorption coefficients.

You can make the model simpler by assuming scattering only media.

- Progressive Photon Mapping (`https://cs.uwaterloo.ca/~thachisu/ppm.pdf`)
  - Multi-pass approach: trace fewer photons, estimate with a large radius, then re-trace photons and re-estimate with smaller and smaller radii over passes

  Why do this? Allows you to reach same level of visual fidelity while not requiring infinite number of photons to be stored in your KD-tree. i.e., saves memory. If you implement this, note how much memory you save for same quality of image.

  Okay, I said don't do things that are hard that don't make your image look nicer, but if you want to do the next extension within photon mapping, this is pretty crucial.

- Homogenous Volumetric scattering in Photon mapping:
  - You create a second volume photon map used to capture light interactions within mediums. During the photon tracing phase, for each photon, as it enters a medium, you sample next scattering/absorbtion event distance. If the photon scatters before hitting a surface, you store a scaled photon in the new volume photon map at that scattering location and sample a new direction. Repeat until the photon exits the medium or is fully absorbed. If hits a surface, proceed with normal photon mapping.
  - During the rendering phase, when a camera ray enters a medium, you simulate scattering events by sampling distances the same way. At each scattering point, you query the nearby photons from the volumetric photon map (within a radius) to estimate in-scattered radiance.

  (Good volume photon mapping implementation, with scattering only: `https://github.com/yumcyaWiz/volppm`)

Bonus features (not numbered because most of these can be done anytime)

- OpenMP parallelization for faster renders. I recommend doing this early so you don't wait forever for things to render
- New BSDFs:
  - Schlick's approximation for Fresnel refraction for glass (and water with a different refractive index)
  - Glossy surface reflections (microfacet models)
- Colored rays/photons store RGB flux per photon/ray:
  - Colored lights: emit different intensities per channel from light sources, blend color with surfaces
  - chromatic dispersion
    - use color dependent indices of refraction (R > G > B)
    - in a refraction BSDF, split light into the 3+ channels of different RGB (to simulate different wavelengths), randomly jittered near index of refraction
  - colored mediums (need volume **absorption** to be implemented)
    - define different absorption coefficients for each color channel. Can have colored glass/liquids, and lights that shine through it will filter into that color too!