Projects for the Course Foundation of Computer Graphics

Project Development

- Choose one of the projects below. The choice is up to you.
- Each project should be deveesecute the project as you wishlop alone or in group. Note though that each project has a maximum number of participants. You cannot have more participant than the max. You could have less, but I discourage it.
- The final grade depends on the quality of the results and their presentation. I will not consider code "quality" nor execution times.
- It is possible to consult and utilize code found online. In that case, **the code should be cited** by clearly indicating it in the PDF. Note though that you cannot simply copy or link to an existing implementation.

Project Submission

- Your final submission must contain
  - code you have written
  - images demontrating the features
  - test ata
  - a PDF containing: (1) students names, (2) short description of what you have done, (3) commented results
- You must submit the project via Classroom, **in a single zip file**. Every member of the group must submit the same file.
- You must submit the project at least 5 days before the appello, to allow for a grade to be formulated.
- I will communicate the grade via Classroom and register it only after the oral exam.

Project Material

- You can write the project in either C++ or Python, or other languages mentioned below. For each project, I will indicate the possibility of using each language. But this should be considered an advise and you are free to execute the projects as you wish.
- In some projects, you must modify a renderer. I suggest you use either
  - Yocto/GL, using the setup done in class. Just download the code from Github and modify it. This is what we have done for many years in this class, so it is quite well tested.
  - Pbrt-v4, the new version of the code from the book Pbrt,
  - Mitsuba3, a research renderer that can be extended in Python (did not test myself though).

Projects: Rendering Systems

- **Intersection Primitives**: 1 person in C++
  - the rendering of points, lines and quads is now too approximate; in this project we will compute these intersections accurately
  - render points as spheres and lines as capped cones; intersection algorithms can be found here with names Sphere and Rounded Cone
  - render quads as bilinear patches as shown here
  - to integrate them in our code you have to (a) insert the new intersection functions in `intersect_bvh` in `yocto_bvh`, (b) change the functions `eval_position` e `eval_normal` to compute the new position and normal, or (c) change the data to return the intersection normal and position directly from the intersection call
  - make test files that show the difference between the methods (I would put a flag inside shapes that switches the intersection methods)
- **Path Guiding**: 2 people in C++
  - implement support for path giuding in our raytracer by using OpenPathGuidingLibrary
  - we might be able to quickly cover path guiding, but if not please see the introduction at SigCourse
- **Path Tracing in Rust**: 1 person in Rust
  - port our path tracer to Rust and render large scenes
  - use Rust libraries for IO
  - use nalgebra or nalgebra-glm for math support
- **Path Tracing in Julia**: 2 people in Julia
  - port our path tracer to Julia and render large scenes
  - use Julia libraries for IO
  - use StaticArrays for math support
- **Path Tracing in Metal**: 2 people in Metal
  - port our path tracer to Apple Metal using bindless rendering in Metal 3
  - Yocto contains a path tracer implementation for CUDA that may be helpful
  - I put this project here for those interested in GPU rendering, but *I do not know how feasible this project is*, so do this at you own risk

Projects: Rendering Algorithms

- **Shadow Terminator**: 1 person in C++ (max 24 points)
  - implement the shadow terminator fix shown in Raytracing Gem 2 - Chapter 4
  - show images that compare Yocto/GL to your solution
- **Better Sampling Sequence**: 1 person in C++
  - add a method to generate well distributed samples based on ZSampler
  - follow the implementation in pbrt-v4
  - demonstrate the method with direct illumination of area lights and environment maps
  - for environment maps, add support for stratified sampling following pbrt-v4
  - you can use small pieces of the pbrt code, but everything need to be well integrated in Yocto
- **Hair Shading**: 1 person in C++
  - add support for shading hairs following pbrt
  - if doing this project, contact the professor for test scenes
  - *you cannot use the pbrt code directly*, neither link to it nor cut and paste into your code; you need to carefully refactor this into Yocto and provide the proper abstractions
- **Layered Materials**: 1 person in C++
  - implement proper layered materials following here
  - you can take inspiration from the implementation in pbrt-v4
  - *you cannot use the pbrt code directly*, neither link to it nor cut and paste into your code; you need to carefully refactor this into Yocto and provide the proper abstractions
- **Volumetric Path Tracing I - Delta Tracking**: 2 people in C++
  - integrate delta tracking for smoke and clouds, based your implementation on pbrt
  - to integrate it, you have to implement a new `sample_transmittance()` that both samples the distances and returns the weight
  - implement volumetric textures by using nanovdb; you can get sample data from OpenVDB
- **Volumetric Path Tracing II - Modernized Volumetric Scattering**: 3 people in C++
  - implement proper volumetric scattering for heterogenous materials
  - implement volumetric textures by using nanovdb; you can get sample data from OpenVDB
  - implement a modern volumetric method from Algorithm 1 of Miller et al --- ignore most of the math here since it is not that helpful
  - you can use the implementation from pbrt-v4
  - also you have to add volumes to the renderer, which for now can be a hack to either hardcode a function that makes one or use example data from OpenVDB