

Spring 2007
22C:251 Advanced Computer Graphics (Image Synthesis)
Assignment 3

Due: Before class on Thursday, March 8th

Goal: Parallelize your ray/path tracer to get a 2–8x speedup. Implement explicit direct lighting for your path tracer to reduce the number of samples required for image convergence.

Note: You may implement the entire assignment using Rays, and not RayPackets. However, students planning on writing a interactive ray tracer should strongly consider updating their packet code to work with at least the parallelized implementation.

Problem 1: (25 points) Parallelize your ray or path tracer. You should be able to specify the number of threads to use on the command line (e.g., `raytrace.optimized -np 4 buddhaScene.txt`).

- You may select any method for distributing pixel work between multiple CPUs.
- I encourage you to have parallel implementations of both the OpenGL and batch modes, but you are only required to implement one or the other. If your semester project will be interactivity, you must implement a multi-threaded OpenGL display. If your semester project will be realism, you must implement a multi-threaded batch mode.
- You must benchmark your results on 1, 2, 4, and 8 CPUs on the machine `dpm1h045.divms.uiowa.edu`.
- Your results should scale nearly linearly in number of CPUs.
- Note: `dpm1h045` currently has hyperthreading enabled, which makes it appear as if it has 16 CPUs. This means 2 threads may be scheduled for the same CPU core, which results in non-linear speedups. I'll disable this shortly, but if you aren't getting a 8x speedup, try running with 16 threads to guarantee you're using all the CPUs.

Groups must also implement a variety of load balancing schemes to find which ones give better performance and scalability. Make this choice a command-line parameter.

1. Using N threads, each thread traces every N^{th} pixel.
2. Using N threads, each thread traces every N^{th} row.
3. Using N threads, the screen is split into N chunks.
4. Using N threads, the screen is split into M chunks ($M \gg N$), each thread processes one at a time before asking for another.
 - Test this with a variety of chunk sizes to find the “ideal” block size.

Problem 2: (25 points) Implement explicit direct lighting in your path tracer. To do this, you will need to:

1. Modify your `PathtracedLambertianMaterial` class (I suggest making a copy with a new name).
2. Your new material class should shoot two random rays per intersection, one to a random point on a light and one to a random ray anywhere in the hemisphere.
3. Add a flag called `countEmittedLight` that determines if a particular ray should count emitted light (see Figure 13.6 in “Realistic Ray Tracing”). You should either initialize this flag correctly in the `Ray` constructor (and `Ray::Reset()` if you have one) or search your code for anywhere you create a new `Ray` and set the flag correctly then.
4. Implement an “area light” class. This will have a pointer to a *Primitive* (or an *Object* if you’re willing to think carefully about the implementation).
5. You will need to be able to query the light for a random point (e.g., implement a method `GetRandomPoint()`).
6. You will need to be able to query the light for its surface area, so you can find the probability ($\frac{1}{area}$) of sampling any given point (e.g., implement a method `GetArea()`).
7. You will need to be able to query the light for the surface normal at your random point. You may use the standard `Primitive::ComputeNormal()` method, or you may wish to implement a new method.
8. Test your code on the Cornell Box (with the correct quadrilateral light source) using at least 100 samples per pixel. Compare this with the same number of samples using naive path tracing (e.g., only one random ray per intersection that counts both emitted light and reflected light).
9. Also create another interesting test scene. Post both your results for this scene and the scene file on your web page so other students may use it.

If you are working in a group, you must add logic to handle multiple light sources using explicit direct lighting. What I mean is that you should only shoot one light ray per intersection, even if you have 1000 lights. I encourage you to design your “interesting scene” from above to have multiple lights. Implement three scheme for this:

- It picks a light to sample (uniformly) randomly from the lights in the scene.
- It picks a light to sample randomly from the scene, where each light’s probability is weighted by its surface area.
- It picks a light to sample randomly from the scene, where each light’s probability depends on the solid angle it subtends from the intersection point. I encourage you to compute the solid angle of the light exactly (perhaps write `AreaLight::GetSolidAngle(Point &fromHitPoint)`), but you may approximate it.