Project 3

Due on Friday, May 6

Introduction

Monte Carlo scattering computations appear in many different settings: designing nuclear reactors, planning radiation therapies in medicine, and rendering realistic materials in graphics. The basic picture is the same in each case. We think of the radiation as a stream of particles (electrons, photons, or neutrons, depending on the setting). As they travel through matter, these particles travel some random distance before interacting with an atom or molecule. Different interactions are possible depending on the context, including scattering, absorption, or fission. The nature of the individual particle interactions is random, but the distribution over possibilities can be measured.

In this project, you are given a simple Monte Carlo code to analyze a model of a laser shined on a homogeneous material. In the model, photons that enter the material travel a random distance (an exponentially distributed random variable) before hitting an atom. When photons from the laser hit an atom or molecule in the material, they are either

- *Absorbed* and converted into heat, or
- *Scattered* at a random angle.

Photons that are scattered back into the surface may also interact there, either reflecting back into the material or transmitting through the surface and back into the ambient environment. The Monte Carlo code more-or-less directly mimics these physical processes of scattering, absorption, and reflection over many randomly simulated particle trajectories. Based on the statistical behavior of these trajectories, the code approximates how the material absorbs or reflects the energy over time.

Your goal is two-fold:

1. Investigate the properties of the code and extend it in some minor ways.
2. Use the code together with techniques from earlier in the semester to study some interesting physics!

The code

The code `run_scatter.m` is available on CMS. It is based loosely on a C code written by Scott Prahl [1]:

`http://omlc.ogi.edu/software/mc/small_mc.c`

The code uses the same basic strategy as the more complete MCML code (Monte Carlo light transport in Multi-Layered tissues), and there is a reasonably comprehensible journal article documenting that code [1]. If you find this discussion too terse, I recommend looking at that article, or at a more general article on the more general MCNP code [2].

There are four parameters that describe the physics:

- μ_a is the expected number of absorption events per cm.
- μ_s is the expected number of scattering events per cm.
- g is a parameter that describes anisotropy in scattering. When $g = 0$, we have *isotropic* scattering: all directions of travel are equally likely after a scattering event. When g is close to 1, scattered photons are more likely to continue going in mostly the same direction. When g is close to -1, photons are more likely to scatter backward.
- n is the (relative) index of refraction. This is used to determine whether photons that encounter the surface get reflected, or not.

There are also several parameters that are used to control the algorithm details. On output, the code returns a histogram of the fraction of photons absorbed over different z ranges, as well as the fraction of photons that are reflected either specularly (i.e. just bouncing from the surface) or diffusely (i.e. after traveling through the medium).

After entering the medium at the origin headed straight in the positive z direction, a photon's trajectory is a sequence of random steps. The step lengths are sampled from an exponential distribution with mean equal to the mean free path ($(\mu_a + \mu_s)^{-1}$). If the step crosses the surface, the photon

[1] I would not regard this as a stellar example of C style. In particular, it bothers me to no end that the author assumes that the operating system will clear the global segment to zero at initialization. That said, it gets the job done, and saves me from reproducing the code for sampling the Henyey-Greeinstein distribution.

should either transmit through or reflect from the surface with a probability that depends on the angle of incidence and the index of refraction. Assuming that the current step is not the last, the photon is then scattered, which means it chooses a new direction by sampling from the Henyey-Greenstein distribution.

For computational purposes, we do not actually process absorption events by terminating a photon's trajectory. Instead, we assign to each photon a probability that it is still alive. This weight is initially $1 - r_{\mathrm{specular}}$, where r_{specular} is the probability that a photon would bounce from the surface before even entering the medium. At the end of each successive step, we update the histogram for the current location with the probability that the photon should actually have been absorbed, and reduce the weight on the photon by the same amount; then we pick a new direction and march forward. When the weight of a photon becomes sufficiently small, so that it no longer makes much difference, we can terminate the trajectory in an unbiased manner by a game of Russian roulette: if the photon survives (with probability p_{survive}), we increase the weight by a factor of $p_{\mathrm{survive}}^{-1}$ and keep going; otherwise, we decrease the weight to zero and move on to the next photon.

Though we have just described the computation in terms of the behavior of a single photon, our code actually runs several photon trajectories simultaneously (in a "batch"), and plays Russian roulette on an entire batch at once. We do this in order to play to MATLAB's efficient vector operations[2]. Otherwise, running getting good results is painfully slow.

Ignoring batching behavior, the basic algorithm is described in Figure 1.

[2] I thought about making you do this part, but decided it was sufficient to get you to walk through the exercise in the last two projects.

```
for each photon

  position  = [0; 0; 0]
  direction = [0; 0; 1]
  weight    = 1-rspecular

  while weight > 0

    s = exponential sample with mean mfp (mean free path)
    move in the current direction by a step s

    if new z < 0
      decrease weight by probability of transmission
      increase rdiffuse total by probability of transmission
      reflect new direction and position
    end

    add absorption probability to new location
    decrease weight by absorption probability
    if weight < wroulette
      Russian roulette: weight = weight/ps (prob ps) or 0 (prob 1-ps)
    end

    choose a new direction
  end
end

scale results by total number of trajectories and return
```

Figure 1: Pseudocode for basic MC simulation

Tasks

Unless you are doing a parameter study, please default to the parameters $\mu_a = 5$ cm^{-1}, $\mu_s = 95$ cm^{-1}, $g = 0.5$, $n = 1.5$, and $z_{\max} = 0.2$ cm.

1. Plot three or four sample trajectories using the default parameters. Repeat for $g = 0.9$. This will involve digging through the code to figure out what does what. You may attempt a three-dimensional plot, or you may just show something two-dimensional.

2. Currently, the code prints out the diffuse reflectance (the fraction of photons that enter the medium and then are scattered back out); but it does not print out error bars. Modify the code so that it does print error bars for this quantity. An error bar in this context is $\hat{\sigma}/\sqrt{n}$, where $\hat{\sigma}$ is the variance of one experiment and n is the number of experiments run; you can approximate $\hat{\sigma}$ by the sample variance. You may treat each photon simulation as one experiment, or you may choose to treat each batch as an experiment.

3. Plot the absorption profile (call it $\phi(z)$) in terms of absorption per centimeter as a function of the depth. Note that this is just the `heat` histogram scaled by bin widths. It will be useful to have routines to automatically plot this in both linear scales and on a semilogarithmic scale (logarithmic in the absorption density, linear in depth).

4. According to a diffusion approximation, the absorption profile in the absence of surface effects would be proportional to $\exp(-z/\delta)$, where
$$\delta = \frac{1}{\sqrt{3\mu_a(\mu_a + \mu_s(1-g))}}.$$
Use a semilogarithmic plot of the absorption profile to show graphically that for z large enough, the absorption really is approximately proportional to $\exp(-z/\delta)$.

5. Compare the absorption profile to the diffusion approximation for the parameters $\mu_a = 1$ cm^{-1}, $\mu_s = 100$ cm^{-1}, $g = 0.9$, $n = 1.0$, and $z_{\max} = 1.0$. This will take a longer than the previous simulation, because each trajectory runs for longer before it is terminated by the Russian roulette algorithm; can you explain why? Re-run with `opt.wroulette = 0.05;`

the code will run faster, but the plot may look noisy. What happens to the error bars on the diffuse reflectance?

6. Using your favorite choice of nonlinear least squares solvers (I suggest Gauss-Newton iteration), fit your computed absorption profile to a function of the form
$$\hat{\phi}(z) = C_1 \exp(-z/\delta_1) + C_2 \exp(-z/\delta_2).$$
To get initial guesses for δ_1 and δ_2, I suggest using the fact that for any z we can write
$$\hat{\phi}(z+2h) = \alpha\phi(z+h) + \beta\phi(z),$$
where
$$q(t) \equiv t^2 - \alpha t - \beta = (t - \exp(-h/\delta_1))\,(t - \exp(-h/\delta_2))\,.$$
We can estimate α and β by a linear system of the form
$$\begin{bmatrix} \phi(z+h) & \phi(z) \\ \phi(z+2h) & \phi(z+h) \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \phi(z+2h) \\ \phi(z+3h) \end{bmatrix},$$
where for $\phi(z)$ we use the histogrammed estimate of the density. Once we have α and β, we can solve a quadratic and take some logarithms to find δ_1 and δ_2; and once we have estimates for δ_1 and δ_2, we can solve a linear least squares problem to get initial estimates for C_1 and C_2.

Notes:

(a) In reality, the profile should have the form
$$\phi(z) = \sum_{j=1}^{\infty} C_j \exp(-z/\delta_j),$$
but as we will see, the first two terms form a perfectly adequate approximation.

(b) This sort of fitting of exponentials to data is sometimes called a *Prony problem* (or a modified Prony problem, since we have more data than coefficients). You may be able to find existing software for the problem; if so, you are welcome to use it. I just coded a little Gauss-Newton routine (with a separate update for the exactly computable C_j coefficients; also with line search, since I didn't want to have to fiddle with the results in the middle of more interesting experiments).

(c) In order to avoid being tricked by statistical noise, if you use the above trick for estimating δ_1 and δ_2, I suggest trying out several step sizes (usually you will want to try steps h to be an integer multiple of the bin width h_{bin}). You can compare the quality of the computed exponents for different steps by looking at the residual in a fit where only C_1 and C_2 are left free.

(d) I recommend doing the Gauss-Newton iteration in terms of the variables $\xi_1 = \exp(-h_{\text{bin}}/\delta_1)$ and $\xi_1 = \exp(-h_{\text{bin}}/\delta_1)$ rather than working directly with δ_1 and δ_2. It simplifies the algebra.

7. Once you have the fit, you should evaluate its quality. Report both the maximum absolute error and the maximum relative error. I suggest that you also plot the fitting error together with the absorption profile on a semilogarithmic plot so that it is easy to see the whole picture. Also verify that one of δ_1 or δ_2 should be close to δ. If δ_1 is close to δ, then δ_2 can be interpreted as a characteristic length for the layer where the surface effects are important.

8. Investigate the convergence of the parameters C_1, C_2, δ_1, and δ_2 as a function of the number of simulations. I am leaving this deliberately vague. You may show convergence plots, error bars, or whatever else you feel is appropriate.

Submission

You should submit a report that summarizes your work. This should at least include all the plots I requested and brief descriptions of your experiments, but feel free to include extensions of your own devising for up to two points of extra credit.

You should also submit your modified version of `run_scatter.m` that includes error bar computations for the diffuse reflection, a code `run_fit.m` that computes the two-exponential fit (possibly using `run_scatter.m` as a subroutine).

References

[1] L. Wang, S. Jacques, and L. Zheng. *MCML — Monte Carlo modeling of light transport in multi-layered tissues*. Computer Methods and Programs in Biomedicine, 47 (1995), pp. 131–146.
http://omlc.ogi.edu/software/mc/mcpubs/1995LWCMPBMcml.pdf

[2] J. Hendricks. *A Monte Carlo Code for Particle Transport: An Algorithm for All Seasons*. Los Alamos Science 22 (1994), pp. 32–43.
http://www.fas.org/sgp/othergov/doe/lanl/pubs/00326727.pdf