Monte Carlo Methods and Area Estimates

CS3220 - Summer 2008
Jonathan Kaldor
Monte Carlo Methods

- In this course so far, we have assumed (either explicitly or implicitly) that we have some clear mathematical problem to solve
- Model to describe some physical process (linear or nonlinear, maybe with some simplifying assumptions)
Monte Carlo Methods

• Suppose we don’t have a good model for the overall process, though (or we wish to validate our model against the process). How can we go about this?

• Suppose we can imitate an experiment (trial, etc). Can we use this to draw conclusions about the overall process?
Monte Carlo Methods

- If we can simulate the experiment on a computer, we can change the data inputs and observe the effects on the results.
- In particular, if the experiment involves some random chance, we can run it a bunch of times and accumulate statistics on the outputs.
Monte Carlo Methods

• When we simulate a process on a computer that involves random chance, that is known as a Monte Carlo simulation.

• One simulation run: particular choices for each of the random choices.
Uses

- Whenever the underlying model is unknown / difficult to compute
- Whenever the inputs are unknown (or are known with a large amount of uncertainty)
Applications

- Particle Physics
- Physical Chemistry
- Finance
- Computer Graphics
Physically Based Rendering

- Rendering an image requires computing the light arriving at each point in the camera.
- Light can arrive at the camera after bouncing any number of times - we can look at one set of bounces as a path of light.
- Model can be expressed as a complex differential-integral equation that can be very difficult to evaluate.
Physically Based Rendering

- We can simulate it using a Monte Carlo approach:
  - Simulate a particular path for light
  - Do this a bunch (a bunch!) of times
  - In the limit, we will end up with the average distribution of light in the scene
[Moon and Marschner, 2006]
A Simple Example

- Suppose we want to verify some basic rules about probability with rolling a die.
- In particular, what are the odds of rolling two 1s in a row?
- Basic probability: \( \frac{1}{36} \) (\( \frac{1}{6} \) chance of rolling the first 1, another \( \frac{1}{6} \) chance for the second 1), or \( p = 0.0278 \) chance.
A Simple Example

- Monte Carlo experiment: roll a die twice, count how many times we roll 1 twice in a row. Divide the number of occurrences by the total number of trials to give us the probability of occurrence.

- For 100 trials: 0.0200
  For 1000 trials: 0.0310
  For 10000 trials: 0.0293
  For 100000 trials: 0.0278
A Simple Example

- We can already see some general properties of Monte Carlo simulations
- Simulating each trial is relatively quick
- ... but we typically need a lot of trials to converge to the correct answer (with only 4 digits of precision to boot!)
Computing Random Numbers

- Monte Carlo simulations require a good source of randomness
- What is randomness to begin with?
  - Intuitively: no pattern in the numbers
  - May also have some constraints on distribution (either uniform or normally distributed)
Computing Random Numbers

• On (almost all) computers, the best we can do is a \textit{pseudorandom} sequence

• Called so because the process for determining the next number is entirely deterministic, although the resulting sequence “looks” random

• We typically can provide a seed which controls the initial starting point
Computing Random Numbers

• That’s not to say that the pseudorandom sequences we get are not random
• They can pass several tests of randomness
• Much more common problem: misuse of random numbers by programmers that makes them less random
Computing Random Numbers

- An example: suppose we want to generate random numbers uniformly distributed in a circle with unit diameter.

- Uniformly distributed: probability of landing in a particular region depends only on the area of the region (equal area regions will have approximately equal numbers of points inside).
Computing Random Numbers

- Here are two algorithms

  1.) Generate two random numbers $r_1, r_2$ uniformly in the unit square. If $\sqrt{r_1^2 + r_2^2} \leq 1$, return the numbers; otherwise, repeat.

  2.) Generate two random numbers $r_1$ and $r_2$ uniformly. Return $r_1/2 \cos(2\pi r_2)$ and $r_2/2 \sin(2\pi r_2)$
Computing Random Numbers

- These algorithms look like they will both generate points randomly in the circle... but their characteristics are quite different
50% of points

Not 50% of area!

50% of points
Computing Random Numbers

- When using randomness in our algorithms, we need to be careful that we are using it correctly.
- In particular, that we don’t destroy randomness or distribution properties of our sequence when manipulating them.
Estimating Areas Via Monte Carlo

• The previous discussion leads to a method for estimating the area / volume of an arbitrarily shaped object

• We only need to be able to test whether or not a point is inside the object
Estimating Areas Via Monte Carlo

- Procedure: generate a uniformly distributed random point in some enclosing rectangle of area $A$ and test whether it is inside or outside the object.

- After $n$ trials, we have $p$ of our points inside our object. The estimated area is then $p*\frac{A}{n}$.
Estimating Areas Via Monte Carlo
Estimating Areas Via Monte Carlo

- Why do we need uniformly distributed points in order to get a good estimate of the area / volume?
Estimating Area Via Monte Carlo

- We can use this to compute an estimate for the value of $\pi$ in a similar way as well
- 10,000 trials: 3.1144
  100,000 trials: 3.1385
Estimating Integrals Via Monte Carlo

• We can also use Monte Carlo simulation to estimate the value of integrals

\[ \int_0^1 f(x) \, dx \approx \frac{1}{N} \sum f(x_i) \]

where we have \( N \) uniformly distributed random points in \([0, 1]\)
Estimating Integrals Via Monte Carlo

- Note that this is only over integral bounds from 0 to 1. In general, we have an integral over a to b

- $\frac{1}{(b-a)} \int_{a}^{b} f(x) \, dx \approx \frac{1}{N} \sum f(x_i)$

- Moving the weight to the other side, we get:
  $\int_{a}^{b} f(x) \, dx \approx (b-a) / N \sum f(x_i)$
Estimating Integrals Via Monte Carlo

- This can be directly extended to multiple integrals:

\[
\int_{c}^{d} \int_{a}^{b} f(x,y) \, dx \, dy \approx \frac{(b-a)(c-d)}{N} \sum f(x_i, y_i)
\]
Estimating Integrals Via Monte Carlo

- In general, this is an extremely slow way of getting “good” estimates (at least in terms of error)
- For integrals, the error is typically decreased with the square root of $N$ (that is, the error is around $1/\sqrt{N}$), which is usually nowhere near good quadrature algorithms)
Estimating Integrals Via Monte Carlo

- The benefit of Monte Carlo is with higher dimension multiple integrals, and with extremely complex integrals (like those in rendering)
- It also provides an easy (but slow!) ground truth to compare against approximations
- Rendering!