Adaptive Sampling

Steve Rotenberg

CSE168: Rendering Algorithms

UCSD, Spring 2014

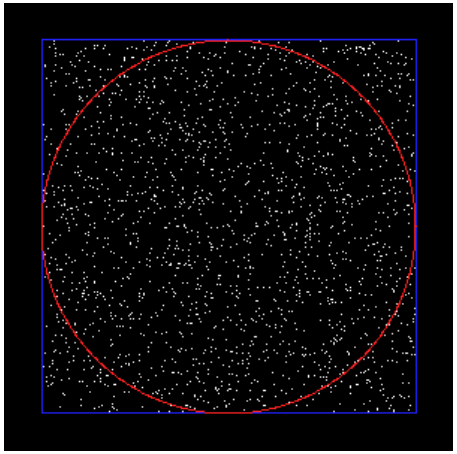
Monte Carlo Integration

- *Monte Carlo Integration* is the process of using random numbers to estimate the value of an integral

Estimating π

- Let's say that we want to use random Monte-Carlo integration to compute the value of pi
- We can do so by generating random 2D points in the [-1…1] interval and then counting how many of them are within a distance of 1.0 from the origin
- The ratio of the number points inside the circle to the total number of points will be approximately equal the ratio of the area of the circle (π) to the area of the square (4)

$$\frac{N_{inside}}{N_{total}} \approx \frac{\pi}{4} = 0.78539816 \ldots$$



Adaptive Sampling

Mean

- We can think of the previous process as estimating the value of a function by taking the average value of many estimates (or *samples*) of the function
- In the previous case, the value of each estimate was either 1.0 (inside) or 0.0 (outside), and they averaged out to π/4
- Given a set of *N* estimates x_i, we can compute the mean $\bar{x}$:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

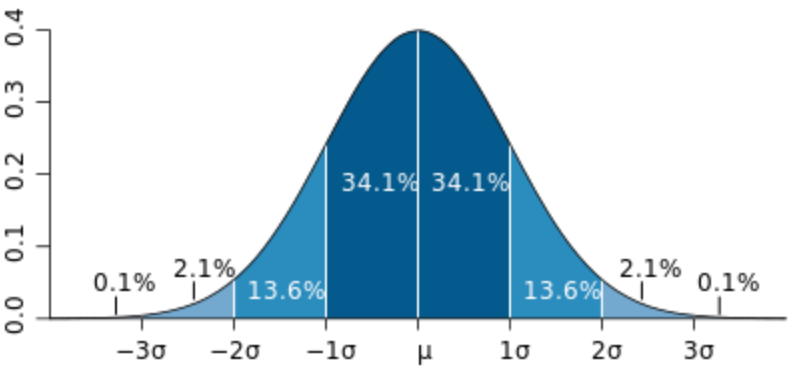
Variance

- We can also compute the *variance* of a set of samples
- The variance is the average of the squared sample deviations from the mean
- The variance measures how far a set of numbers is spread out
- A variance of 0 indicates that all the samples are identical. A low value indicates they will tend to be spread over a small area near the mean, and a large value indicates they will be spread over a large range

$$\mathrm{var}(x) = \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Standard Deviation

- Standard deviation σ is the square root of variance
- Its advantage is that it is in the same units as the original estimates, so if the x_i values are in percentages, for example, then σ is a percentage
- If the original set of samples has a Gaussian distribution, then we would expect that ~68.2% of the samples would be within $\pm\,\sigma$ of the mean
- Keep in mind, that our π estimation example wasn't a Gaussian distribution, and most of our ray tracing applications won't be either, but that's OK



Sample Standard Deviation

- The previous variance estimate is based on the idea that we have *all* of the relevant members of a set of numbers
- However, we're usually just working with a subset of possible values out of a much larger (or infinitely large) set
- To correct for this, it is better to use the *sample standard deviation*:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Standard Error

- The *standard error of the mean* or just *standard error* is a probabilistic bound on the estimation of the mean

$$err = \frac{s}{\sqrt{N}}$$

- If we take 100 samples of some function and compute the mean, we want to know how accurate our estimate is of the 'true mean'
- The standard error tells us the expected error range, but it isn't a guarantee
- Like the standard deviation, it is based on the assumption of a Gaussian distribution of samples
- Plus, it is not an absolute bound but a probabilistic one
- Still, it is a reasonable way to compute the expected error on many practical sample distributions

Error Estimation

- So now, we have a formula to estimate the error in our calculation of the mean

$$err = \sqrt{\frac{1}{N^2 - N} \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

Adaptive Sampling

- The idea behind *adaptive sampling* is to use the error estimate to determine when we've taken enough samples to reduce the error below some desired threshold
- To apply this to rendering, we can apply this on a pixel by pixel basis
- For each pixel, we start shooting rays, keeping track of the mean and variance of our estimate as we go, and using this to estimate the error
- When the error drops below some specified tolerance, we are done with that pixel
- It is best to start with a fixed number of samples (at least 10, but probably more) so that our initial error estimate is reasonable

Running Counts

- From a look at the variance formula, we see that we need to know the mean in order to compute the variance
- This would imply that we need to store all of the values of x_i to compute the mean and variance at any point
- However, by rearranging the equation, we can put it into a form that allows us to just keep running counts of the mean of x_i and mean of x_i^2 and adjust those with each new sample

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2 = \frac{1}{N}\sum_{i=1}^{N}{x_i}^2 - \left(\frac{1}{N}\sum_{i=1}^{N}x_i\right)^2$$

Running Counts

- This way, we don't need to store all the previous samples, and we just need to track two numbers (let's call them *m* (for mean) and *s* (for square))
- For each sample *i*, we compute the value of the function f_i and then update *m* and *s*

$$m \mathrel{+}= f_i$$
$$s \mathrel{+}= f_i^2$$

- At any time, we can then compute the mean variance as:

$$\text{mean} = m/N$$
$$\text{var} = s/N - \text{mean}^2$$

Running Counts

- The sample standard deviation is

$$s = \sqrt{\frac{N}{N-1}\text{var}}$$

- And the error is

$$err = \sqrt{\frac{1}{N-1}\text{var}}$$

Vectors

- As we are making estimates on colors, we really want to use vectors in this process
- In this case, our running count *m* is actually just a vector (color) and our running count for *s* is a scalar that sums up the magnitude squared
- The variance requires the mean squared, so we use the magnitude squared of the vector mean
- The variance, standard deviation, and error will therefore just be scalars

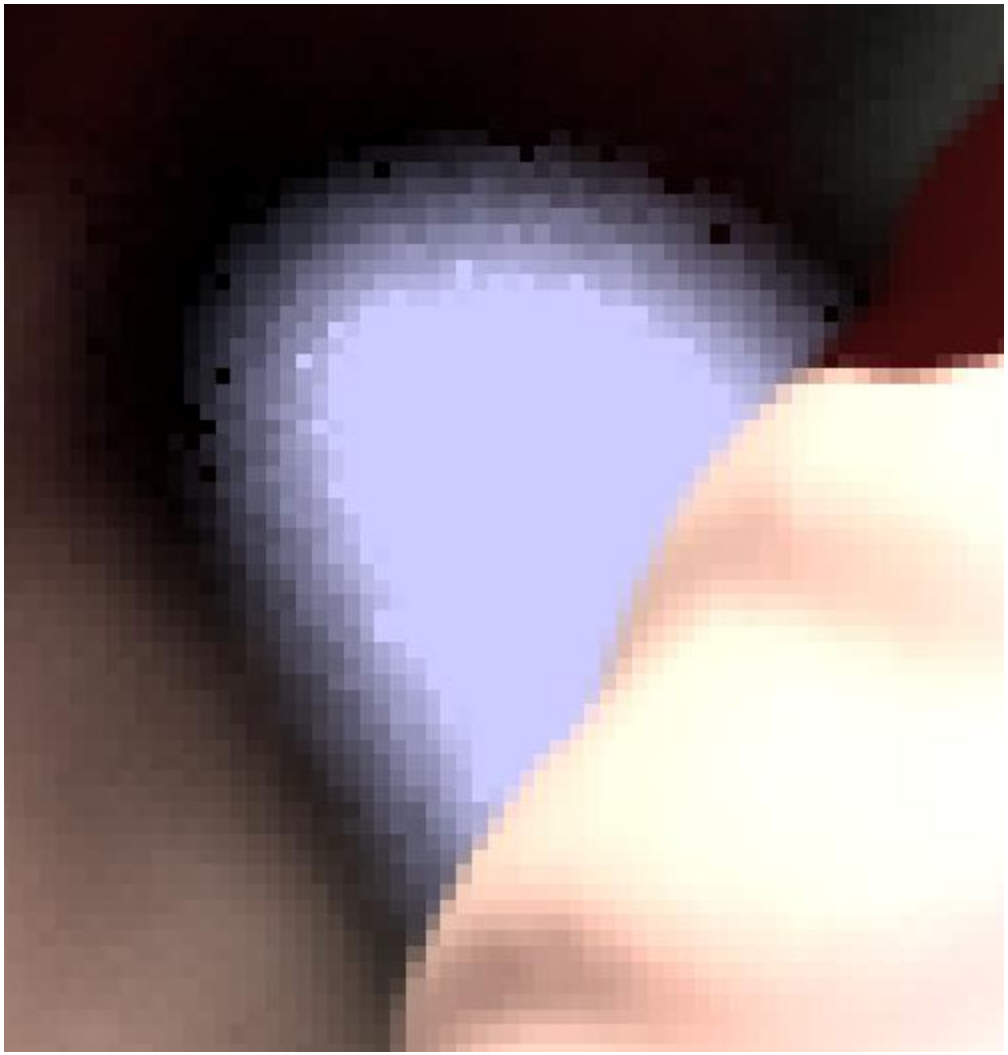
Adaptive Sampling

Adaptive Sampling Density

Adaptive Sampling

- Adaptive sampling is a nice technique that will usually improve render times
- However, it can suffer from some problems
- If the initial set of samples is small, this has a tendency to underestimate the error, leading to premature termination of the sampling
- This is mainly due to the fact that rendering does not generate a Gaussian distribution of values

Close-Up View



Importance Sampling

Mean Estimate

- We saw that when we are trying to estimate a function using Monte Carlo integration, our estimate after N samples is computed as:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Weighted Average

- A slightly more sophisticated approach is to take a *weighted average*

$$\bar{x} = \frac{\sum w_i x_i}{\sum w_i}$$

- Is there some set of weights w_i that will give us a better estimate than just taking the mean?

Importance Sampling

- The idea behind *importance sampling* is to attempt to select a weighted distribution of random numbers that will lead to a more accurate estimate of the function
- In order to do this however, we need to have some sort of idea of what the function might look like so we can select our weights

Monte Carlo Integration

- Let's say that we are trying to estimate an integral over some domain
- We'll choose a 1D function $f(t)$ over the domain [0…1] as an example, but this can be extended to n-D functions over any domain

$$z = \int_{t=0}^{1} f(t)dt$$

Monte Carlo Integration

$$z = \int_{t=0}^{1} f(t)dt$$

- We want to estimate *z*, so we choose a set of N random numbers ξ_i on the [0…1] interval and evaluate $f_i = f(\xi_i)$ for each one
- The estimate for the integral is then:

$$z \approx \frac{1}{N} \sum_{i=1}^{N} f(\xi_i)$$

Monte Carlo Integration

- Without knowing anything at all about $f()$, we can't really do much better than that
- However, if we have some sort of idea of the shape of $f()$, then we may be able to choose a set of weighted samples that do a better job
- For example, let's say that we know that $f()$ is actually the product of two other functions $g()$ and $h()$:

$$f(t)=g(t)h(t)$$

- Let's also say that we know $g(t)$ exactly, but have no idea what $h(t)$ is

Monte Carlo Integration

$$z = \int_{t=0}^{1} f(t)dt = \int_{t=0}^{1} g(t)h(t)dt$$

$$z \approx \frac{1}{N}\sum_{i=1}^{N} f(\xi_i) = \frac{1}{N}\sum_{i=1}^{N} g(\xi_i)h(\xi_i)$$

Importance Sampling

- If we choose a set of random numbers that match the distribution of *g*(), then we can estimate the function with:

$$z \approx \frac{\sum w_i h(\xi_i)}{\sum w_i}$$

- This will usually lead to a much better estimate, and allow us to take fewer samples
- This process is known as *importance sampling*, since we placing more samples where we think they are more important (i.e., where we expect the function to have a higher value)
- The question that remains is how do we generate a set of random numbers that matches the distribution of *g*()?

Rejection Sampling

- We want to generate a bunch of samples that match the distribution of $g(t)$ over the [0…1] interval
- One way to do this is with *rejection sampling*
- Let's say that the maximum value of $g(t)$ over the interval is g_{max}
- We start with a uniformly distributed random number ξ_i, but then choose a second random number δ_i
- If $g(\xi_i)$ is less than $g_{max}\,\delta_i$, then we keep ξ_i, otherwise, we reject it and try again
- We keep this up until we have a ξ_i that passes
- This is then used to evaluate $h(\xi_i)$ and contributes to our estimate
- This process may lead to a lot of rejections, which ultimately waste time, but this can be improved by using a tighter fitting bound than just g_{max}

Probability Density Function

- A more sophisticated way of generating these distributions uses the concept of a *probability density function* or *PDF*
- We start by normalizing $g(t)$ so that it represents 100% of the distribution

$$G(t) = \frac{g(t)}{\bar{g}}, \qquad \text{where} \quad \bar{g} = \int_{-\infty}^{\infty} g(t)dt$$

- $G(t)$ will match the shape of $g(t)$, but will be scaled so that:

$$\int_{-\infty}^{\infty} G(t)\, dt = 1$$

- Note that $g(t)$ does not need to be smooth or continuous, but it does need to be positive for all values of t

Cumulative Distribution Function

- We can then compute the *cumulative distribution function* or *CDF*:

$$C(t) = \int_{-\infty}^{t} G(t)dt$$

- The graph of $C(t)$ will go from 0 to 1 in a continuous and non-decreasing way as t increases
- Note that:

$$G(t) = \frac{d}{dt}C(t)$$

Random Distributions

- To generate a random number with the desired distribution, we choose a uniformly distributed random number ξ_i and find out where $C(\alpha_i) = \xi_i$
- α_i will then be a random number matching the original distribution of $g()$
- We can then evaluate $\bar{g}h(\alpha_i)$ and use it in our estimate

Random Distributions

- This is what is happening when we perform the BRDF sampling, discussed in a previous lecture, and implemented in project 3
- For the Ashikhmin BRDF, this technique was used to compute the sampling direction
- By using this technique, we can create a bunch of samples that estimate our function, while allowing each sample to have equal weight
- Using this approach should significantly reduce the noise in situations where we have irregular distributions (such as the sharp highlights on the BRDF)