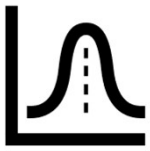


Rendering: Importance Sampling

Bernhard Kerbl

Research Division of Computer Graphics
Institute of Visual Computing & Human-Centered Technology
TU Wien, Austria

With slides based on material by Jaakko Lehtinen, used with permission

Welcome back to this lecture on Rendering. Our topic today is an extremely important one, that is, Importance Sampling

Today's Goal

- Improve the efficiency of Monte Carlo with importance sampling
- Understand how we can produce custom distributions in simple 1D, 2D and 3D domains by warping simple, uniform random variables
- Learn how we can transform samples between cartesian and non-cartesian domains (e.g., from polar (θ, ϕ) to XYZ vectors)
- Understand how we can incorporate these steps into path tracing

Importance sampling will enable us to significantly raise the effectiveness of Monte Carlo integration, compared to how we did it up until now.

Until now, we used only uniform sampling for the hemisphere. We gave you a few simple functions for generating and compensating for samples on the hemisphere surface.

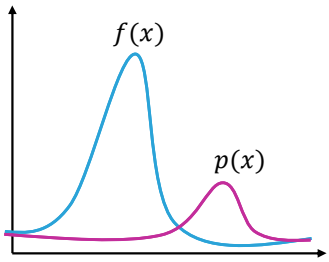
This time, not only are we going to derive how these functions came to be, we are also going to learn about the mathematical background and the tools that we need so that we can derive arbitrary sampling strategies, because some of them will be much better than uniform sampling. We will show you how to do this in 1D, 2D and even 3D on the hemisphere, and all that we will need as input will be simple random variables, that every modern computer can produce. Since the hemisphere is a special domain, we will also have to

look at how we can come up with sampling strategies that achieve a particular distribution on these non-trivial domains. And lastly, after sitting through all the related math, you will quickly see how this ties nicely into our path tracer.

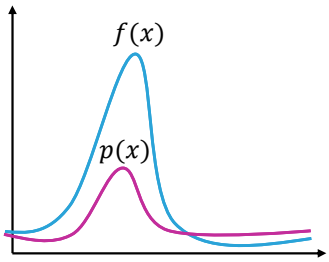
Importance Sampling

- All these things sound tedious… why do we need to create samples from arbitrary distributions? In different domains even?

- When we sample, e.g., the hemisphere, we can use any PDF we like



Bad sampling (high variance)



Importance sampling (low variance)

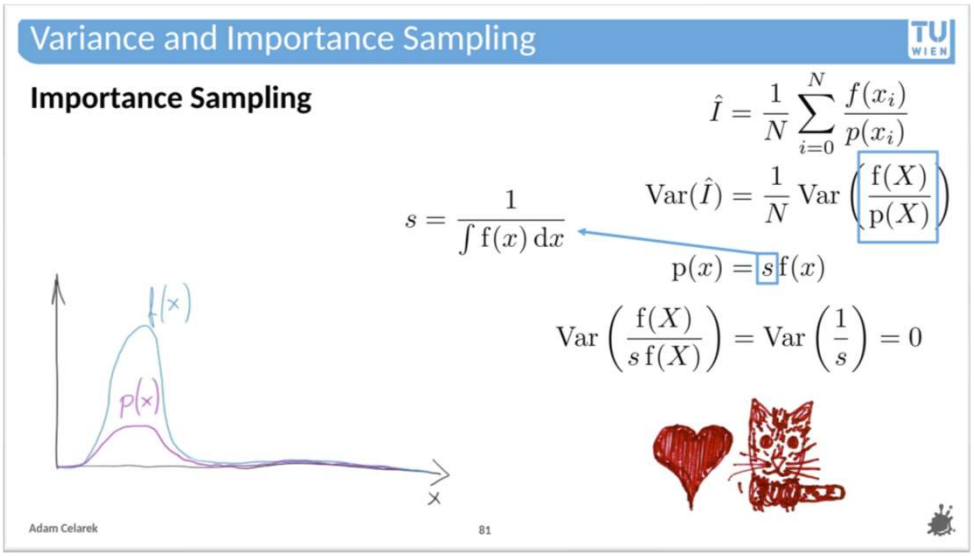
- We know the selection of the proper $p(x)$ as importance sampling

No beating around the bush, there will be a wall of math between you and the end of this lecture. But we cannot simply stick with uniform sampling, because choosing appropriate sampling distributions is a key ingredient to make Monte Carlo integration perform better.

Whenever we try to integrate a function over a domain, such as the hemisphere, we are free to choose the sample PDF, or distribution, as long as we can make samples in the domain that follow this distribution. And ideally, if we have the choice, we will try to make an informed decision, so that the sample distribution mimics the function itself as closley as possible. Becuase when we do that, that is what we call importance sampling, and we already derived that it has immense benefits for the estimation quality of Monte Carlo.

Importance Sampling

- Remember: if possible, you want a PDF $p(x)$ that mimics $f(x)$!



Rendering – Importance Sampling 4

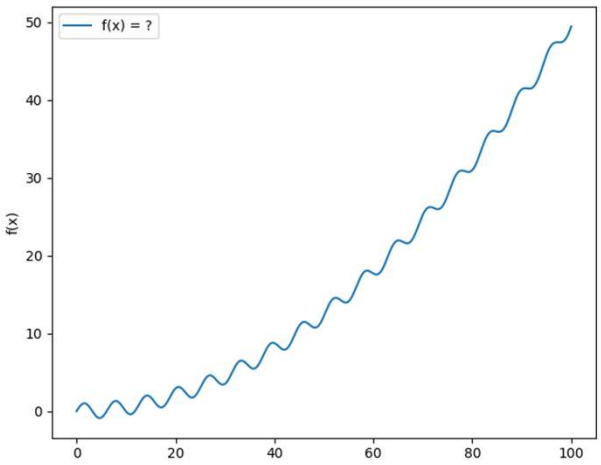
In the Monte Carlo lecture, we saw that the variance of the Monte Carlo estimator for integrating a function f(x) is lowest, actually 0, when the sample distribution PDF, p(x), is proportional to the actual function we are trying to integrate. Lower variance means less noise, and the faster we get rid of the noise in our renderings, the faster the image quality improves.

Now, if we knew what f(x) was and could normalize it, then we could just use that and have an optimal distribution. But this is a paradox: if we were able to normalize f(x), that would imply we know its definite integral. If we knew the definite integral of f(x), then we wouldn't need Monte Carlo to approximate it in the first place. To illustrate this for our special case, it means if we had perfect and total knowledge of how light bounces through a scene, we wouldn't need path tracing to come up with an image. But in our case and many others, this perfect knowledge of f(x) is

unobtainable. However, we can usually make good approximations, and find distributions p(x) that are reasonably close to f(x). Let’s see an example of how this could work.

Monte Carlo Integration with Importance Sampling

- Let's look at an application for importance sampling in practice
- Consider a target function $f(x)$
- You want to compute its integral, but have no closed-form solution or can only measure $f(x)$ ad-hoc?
- Clearly, a case for Monte Carlo

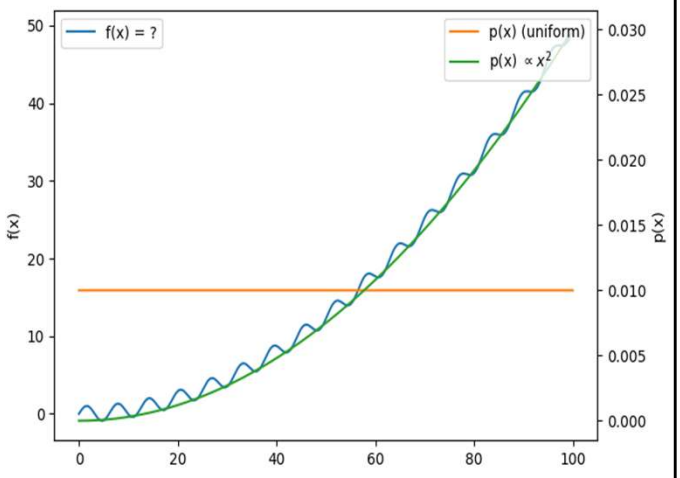


Here's an unknown function, f(x). We want to compute ist integral in a certain range, but we have no closed-form solution. What can we do about it?

Clearly, this is a case for Monte Carlo. With any reasonable sampling strategy, the more samples we will throw at it, the better our approximation will get.

Monte Carlo Integration with Importance Sampling

- If we take another look, the shape of this function seems familiar…
- It appears to be quite close to x^2!
- We already know that uniform sampling of $f(x)$ is only **one** way to do Monte Carlo integration…
- Let's try instead with $p(x) \propto x^2$



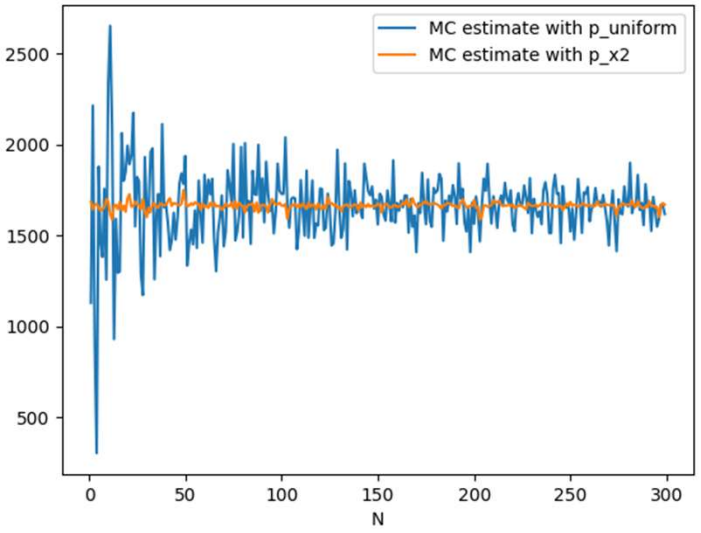
But hold on, the shape of this function actually looks quite familiar. There are a few fluctuations in it, but ist base shape is actually VERY close to x squared.

We already did monte carlo integration with uniform sampling, so we may now try our hand at using something else. WE are going to go ahead, and try to compute the integral of the function in two different ways: once with uniform sampling and once with a strategy where samples are distributed proportionally to x squared. Let‘s see how the versions compare.

Uniform vs Importance Sampling (Python)

integrate_mc(0, 100, N, f, **p_uniform**, **gen_uniform**) *vs* integrate_mc(0, 100, N, f, **p_x2**, **gen_x2**)

- Both methods converge towards the same result
- But the importance-sampled method converges quicker!
- Let's see what the code behind it looks like..



Here we see, up top, the two invocations that we did in python to launch the process. Both methods integrate in the range from 0 to 100, and both get to use a varying number of samples to do so. On the right, we plot what result the two methods come to with each given number of samples N.

Clearly, both methods converge to the same result, around 1700. But the method that uses importance sampling got there much much quicker. Imagine, how much we could improve our rendering procedures if we could also make our Monte Carlo integration that much more effective!

Let's see how we did that in code, and maybe this is all straight forward and we can reproduce something similar immediately…

Uniform vs Importance Sampling (Python)

integrate_mc(0, 100, N, f, **p_uniform**, **gen_uniform**) *vs* integrate_mc(0, 100, N, f, **p_x2**, **gen_x2**)

```
def integrate_mc(a: float, b: float, N: int, f, p, gen):
    X = gen(a, b, N)
    estimates = f(X)/p(X, a, b)
    result = estimates.sum() / N
    return result

def p_uniform(x, a: float, b: float):
    return 1/(b-a)

def p_x2(x, a: float, b: float):
    b3 = ((b**3)/3)
    a3 = ((a**3)/3)
    return x**2/(b3-a3)
```

```
def gen_uniform(a: float, b: float, N: int):
    xi = np.random.rand(N)
    return xi * (b - a) + a

def gen_x2(a: float, b: float, N: int):
    xi = np.random.rand(N)
    b3 = (b**3)
    a3 = (a**3)
    return (a3+xi*(b3-a3))**(1.0/3.0)
```

By the end of the day, this should make sense to you!

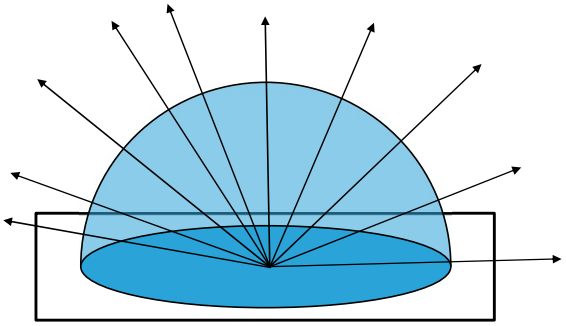
Ok, the main loop looks ok, this is just a basic monte carlo integration procedure, which takes a method for generating samples and sums them up, weighted by the PDF and eventually computes a mean. We saw this quite a few times already. But whats interesting are the functions we are passing in for sampling according to x squared. Those are the functions p_x2 and gen_x2 here, which take uniform random variables and the targeted integration interval as parameters.

And… they don‘t look straightforward. The computations in them are not too complicated, but the question is, why do we cube the ranges of the integration interval, why the division by 3, why a cube root in the generation function? These things look somewhat arbitrary. And in comparison, x^2 is a rather simple distribution, so what might the sample

generators for other functions look like? The good news is that, if you pay close attention to the methods we describe in the first half of this lecture, you should be able to come up with functions like these, and much more complicated ones, yourself, which gives you the skills to apply your own importance sampling solutions in the future.

Importance Sampling on the Hemisphere

- Before, we did uniform hemisphere sampling, and it worked
- But perhaps we can also use importance sampling here?
- Can we perhaps importance-sample the *rendering equation*?



- The hemisphere is a peculiar domain. Sampling it with arbitrary distributions is a little bit more complex…

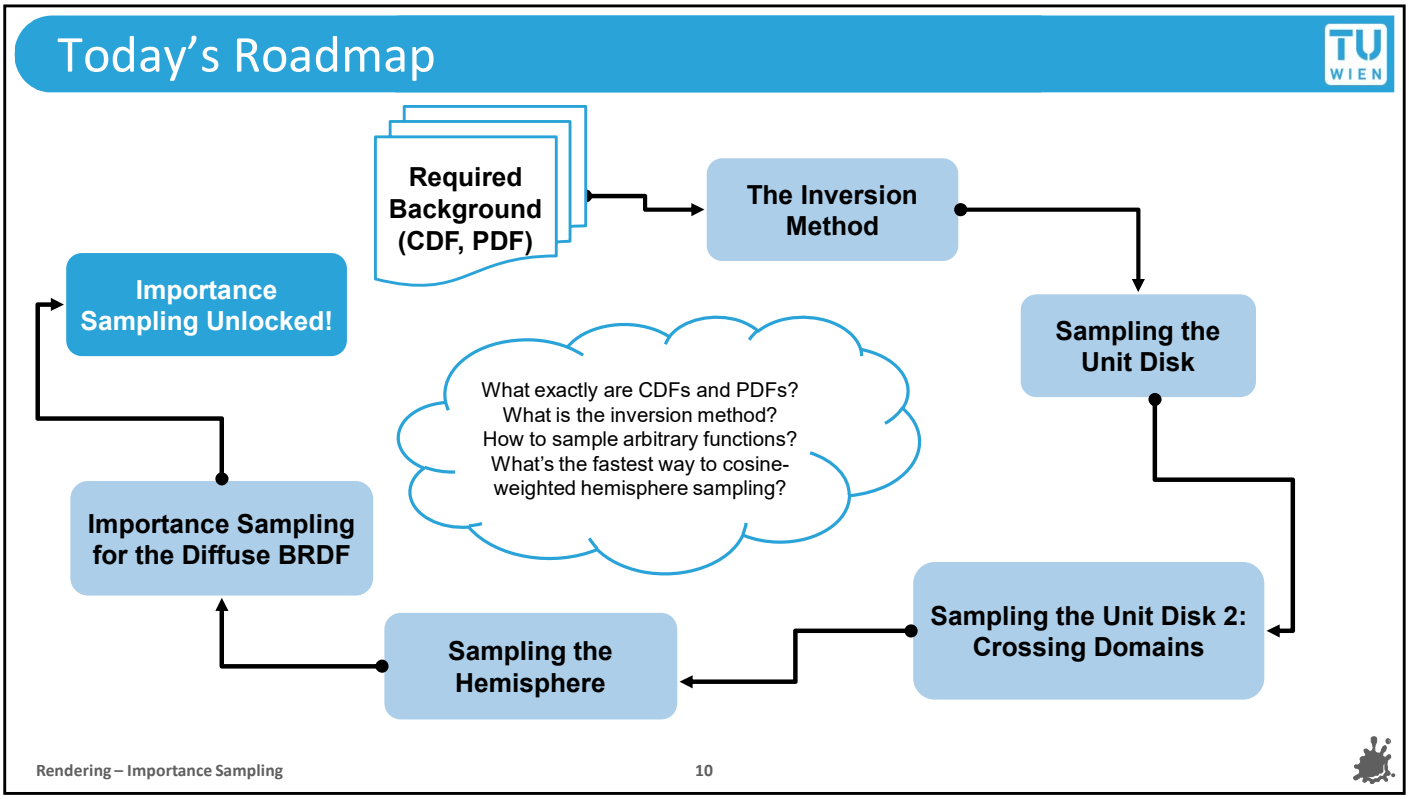
Once we know how to do this, we are going to approach the hemisphere. It is a little special, so first of all, we will have to analyze this domain and see how we can integrate over its surface. For those who were wondering before, we will also derive the solution for uniform hemisphere sampling in the process.

But imagine what we could do if we unlock importance sampling on the hemisphere as well. Perhaps we could try to importance sample the rendering equation itself and immediately get clean, noiseless images with just a few samples? We will talk about what exactly is and isn't possible when we get there. But, just as a heads-up, we will be able to clearly improve the quality of our renderings in our path tracer from. The fact of the matter is that, the solution to do this will only involve a few lines of code, so if you don't care about the

background or about understanding the math that makes importance sampling tick, we will actually provide you with a shortcut to the corresponding solutions in the second half of this lecture.

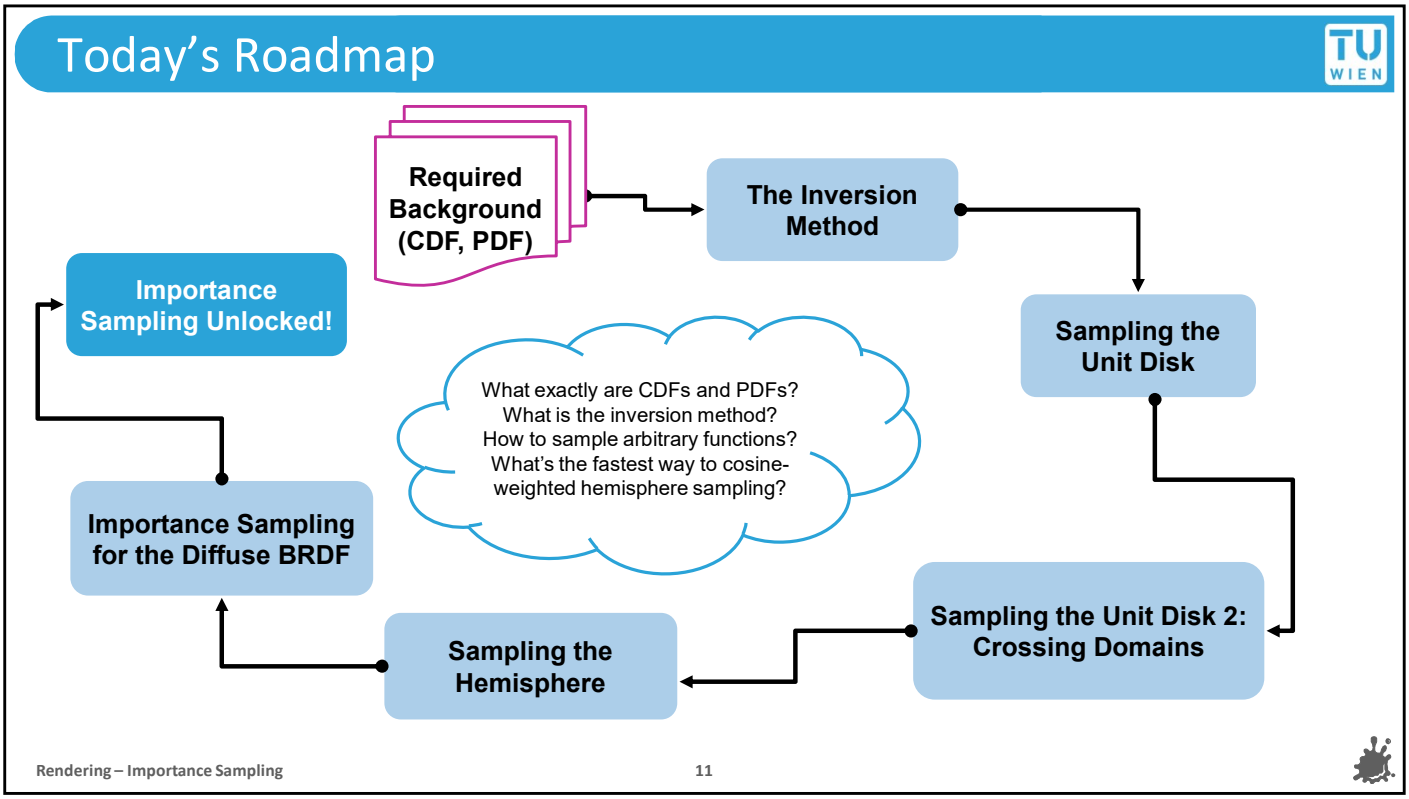
But we still hope that you are willing to also pay attention to the theoretical background in case you ever need to dig deeper than importance sampling only the most basic functions.

Ok, let’s get to it!



Here we have today's roadmap. We will begin by looking at some of the background that we need, talk about the PDF again and related concepts, which will lead us directly to the inversion method. Once we have seen how the inversion method can be used to make samples from arbitrary distributions, we will try to make them not only for 1D and 2D functions, but we will also see how we can sample a more demanding domain, that is the unit disk. We will do this in two ways, once with geometric reasoning and once with the formal method for transforming samples between cartesian and non-cartesian domains. After we have mastered the unit disk, we will dare to move on to the hemisphere and derive the solution for uniform sampling in this new domain. Finally, we will combine the inversion method with the transformation of samples to achieve importance sampling on the hemisphere, and we will look at a concrete example for exploiting it in a path

tracer to improve the image quality.



First off, we will look at some basic concepts that we need, then we will introduce the CDF, the cumulative distribution function, and drill down into the formal definition of the PDF which have already been using. But before we can do this, let's start with a quick recap of random variables.

Discrete Random Variables

- In daily life, we are mostly confronted with *discrete* random results
  - A coin flip
  - Toss of a die
  - Cards in a deck

- Each possible outcome of a random variable is associated with a specific probability p. Probabilities must sum up to 1 (100%)

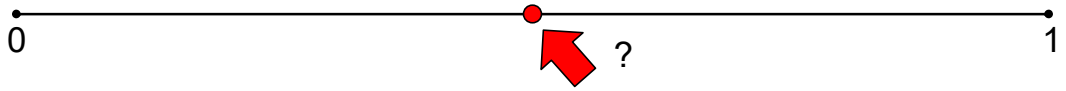
- E.g., a fair die: $X \in \{1,2,3,4,5,6\}$ and $p_1 = p_2 = \cdots = p_6 = \frac{1}{6}$

Continuous Random Variables

- A continuous random variable X with a given range $[a, b)$ can assume any value X_i that fulfills $a \leq X_i < b$
- Working with continuous variables generalizes the methodology for many complex evaluations that depend on probability theory
- There are infinitely many possible outcomes and, consequently, the observation of any specific event has with vanishing probability
- How can we find the probabilities for continuous variables?[2]

Cumulative Distribution Function (CDF)

- For continuous variables, we cannot assign probabilities to values



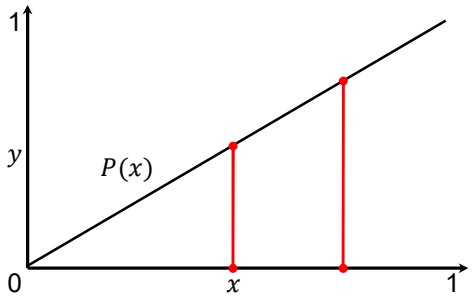
If X can take on any value with equal probability, what is the probability of $X = 0.5$?

- The *cumulative distribution function* (CDF) lets us compute the probability of a variable taking on a value *in a specified range* [2]

- We use notation $P_X(x)$ for the CDF of X's distribution, which yields the probability of X taking on any value $\leq x$

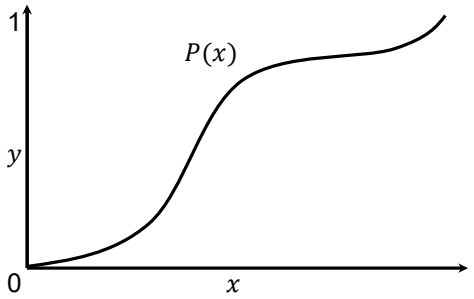
Probability for a Range with CDF

- $P_X(b) - P_X(a) = Pr\{a \leq X_i \leq b\}$
- Read as: *the probability of X taking on any value from* 0 *to b, minus the probability of X taking on any value from* 0 *to a*
- Example: uniform variable ξ generates values in range [0, 1):
  - $P_\xi(x) = x$
  - $P_\xi(0.75) - P_\xi(0.5) = 0.25$



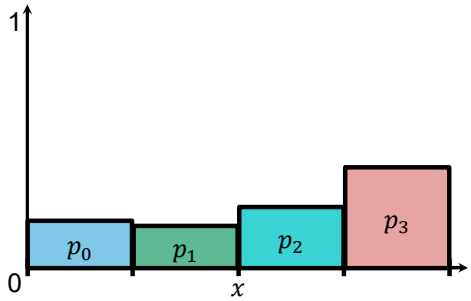
Properties of the CDF

- CDF is bounded by [0, 1] and monotonic increasing
  - Probability of **no** outcome is 0, the probability of **some** outcome is 1
  - Die: Rolling a number between 1 and 6 cannot be less probable than rolling a number between 1 and 5
- CDFs can be applied for discrete and continuous random variables
- How do we compute the CDF?

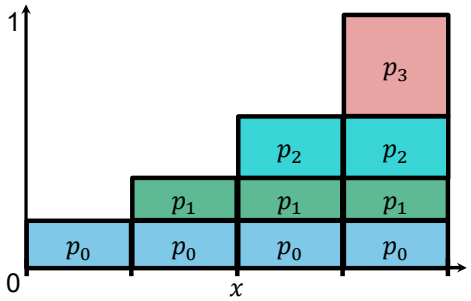


Computing the CDF for Discrete Random Variables

- Determine the limits $[a, b]$ of your variable X
- For each outcome, find its probability $p_a, \ldots, p_b$
- The CDF of that variable is then a function $P_X(x) = \sum_{i=a}^{x} p_i$



Outcome Probabilities



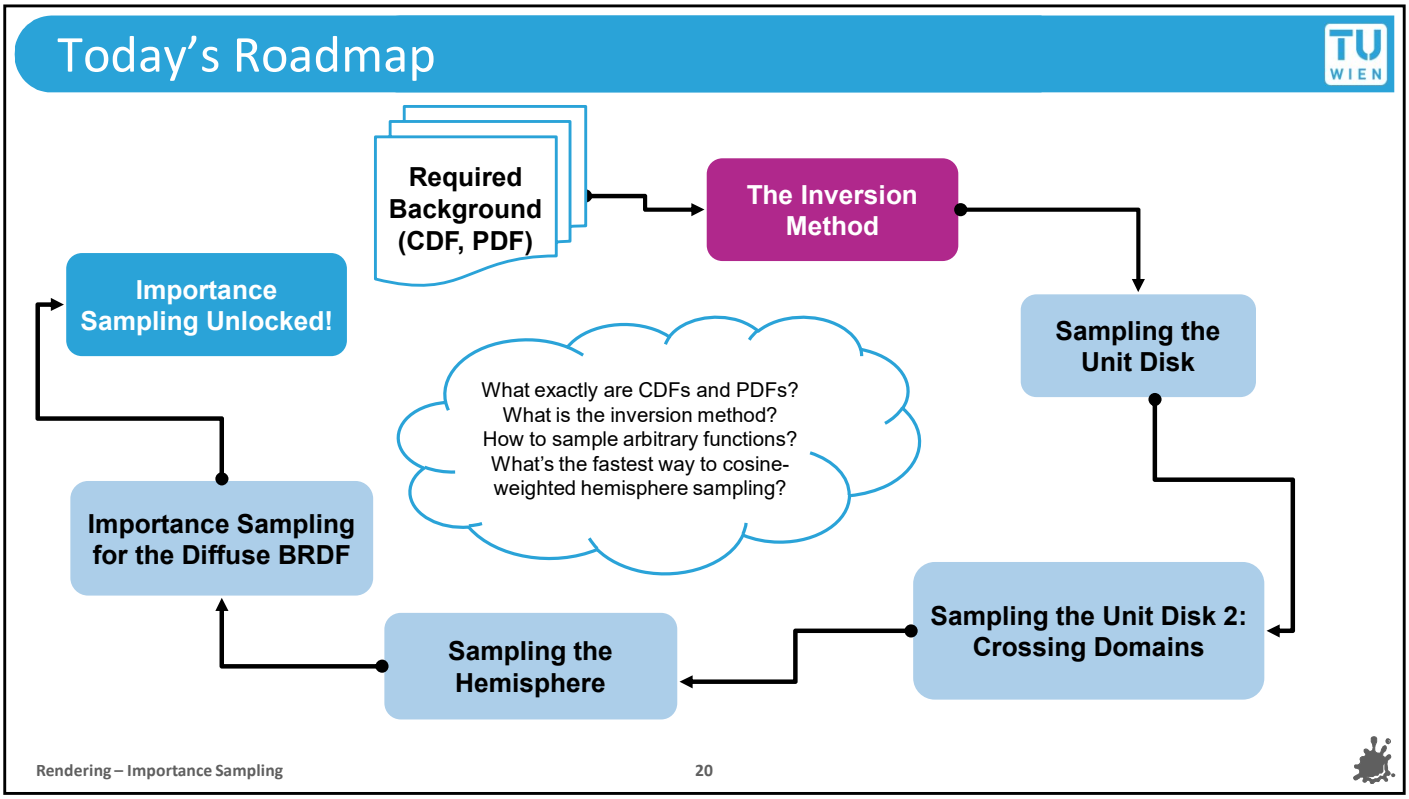
Cumulated Probabilities (CDF)

Probability Density Function (PDF)

- The PDF $p(x)$ is the derivative of the CDF $P(x)$: $p(x) = \frac{dP(x)}{dx}$
- For a PDF $p(x)$, $P(x) = \int p(x)\,dx$ and $\int_a^b p(x)\,dx = P(b) - P(a)$
- $p(x)$ must be positive everywhere: a negative value would mean we can find $[a, b]$ such that $\int_a^b p(x)\,dx$ has a negative probability
- $p_X(x)$ can be understood as the **relative** probability of $X_i = x$. I.e., if $p_X(a) = 2p_X(b)$, then $X_i = a$ is twice as likely as $X_i = b$

Notes about the PDF

- Notation may **look** like probability, but PDF values can be >1!
- For both discrete and continuous variables, we can reference "$p(x)$" to describe their distribution
- **Summary**: for a continuous variable X with a known, integrable PDF, we can find the CDF and probabilities of X landing in a given *range*
- …is this actually helpful?



This brings us to our next big topic in today's agenda, that is, the inversion method. We will see how incredibly useful the our new-found knowledge about the CDF is and how we can use the inversion method to exploit it and let us perform importance sampling with a wide range of target functions in 1D and 2D.

Creating Variables with Custom Distributions

- By discovering the CDF, we have found a powerful new tool
- The function is often invertible: this means, we can generate random variables with a desired distribution!
- Rationale: Since the CDF is monotonic increasing, there is a unique value of $P_X(x)$ for every x with $p_X(x) > 0$
- More informally, if we plot a 1D CDF, any x value that X can take on has a unique, corresponding coordinate on the y-axis

Basic Sampling with Canonical Random Variables

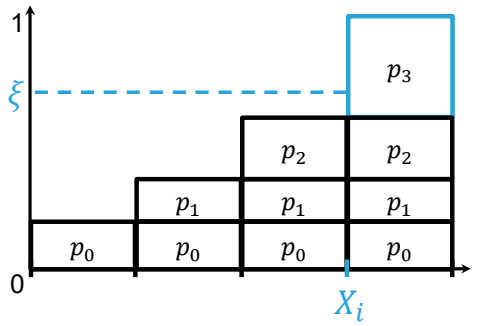
- We want to generate samples for a custom distribution from a random input that we can easily obtain in a computer environment
- Our assumed input is the **canonical random variable** ξ:
  - continuous (i.e., a **real** data type)
  - with **uniform** distribution
  - in the range **[0, 1)**
- Goal: warp samples of ξ to ones distributed according to some $p(x)$

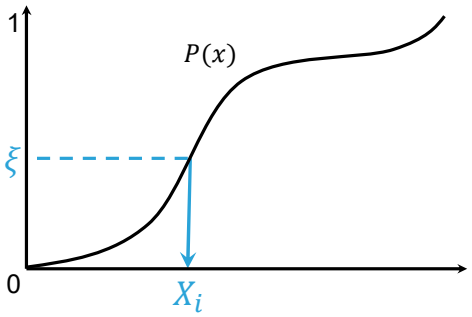
The Canonical Random Variable

- Our assumed default input variable
- PDF for ξ is 1 in range $[0,1)$ and 0 everywhere else
- CDF for ξ is linear
- Important property: we have equality $P(\xi_i) = \xi_i$

The Inversion Method

- For discrete variables: we draw ξ and iterate event probabilities
- When their sum first surpasses ξ, we have found X_i
- For continuous variables: exploit P_X's bijectivity and use its inverse!
- We can use canonic ξ to compute $X_i = P_X^{-1}(\xi)$ according to $p_X(x)$





And we can exploit this property in order to achieve what we wanted to do before, namely to simulate a random variable with a given PDF/CDF. And the way to do that is by using a process called the Inversion Method. If we have a random variable X with some distribution, we can draw a sample from our canonical random vraible XI and consider it as the output of X's CDF to find the value of X that would have generated it. In the discrete case, we can simply iterate over the outcome probabilities of X and stop at the event where their sum surpasses our sample of XI. For the continuous case, we can use the bijectivity of the CDF and take its inverse. So the value of random variable X that would produce the cumulative probability given by our sample Xi is found via the inverse of X's CDF.

Example: Exponential Distribution

- Used mainly for estimation of time intervals between two events
- The probability of a value decreases exponentially
- Needs additional parameter λ, often called *rate parameter*
- We can find its PDF and CDF in most probability text books
  - $p(x, \lambda) = \lambda e^{-\lambda}$
  - $P(x, \lambda) = 1 - \mathrm{e}^{-\lambda x}$, $P^{-1}(x', \lambda) = -\frac{\log(1-x)}{\lambda}$

Warping Uniform To Exponential Distribution

```
def warp_expx(X, lambda: float):
    return –np.log(1.0 – X) / lambda

LAMBDA = 0.5

samples_uniform = np.random.rand(N)
samples_exp_ref = np.random.exponential(1.0/LAMBDA, N)
samples_exp_gen = warp_expx(samples_uniform, LAMBDA)

h1, h2, h3 = histograms(0.0, 1.0, 20, samples_uniform, samples_exp_ref, samples_exp_gen)

show_histogram(h1)
show_histogram(h2)
show_histogram(h3)
```

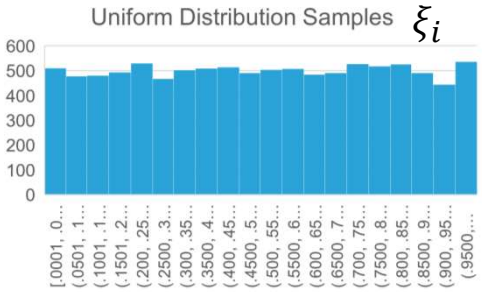

What we are doing here, is, we are basically putting into practice what we just saw, we are computing the inverse of the CDF and thereby simulating an exponential distribution by warping the outputs of a uniform distribution.

And in order to evaluate this, we are going to write a simple python program that uses two different arrays of random numbers from the numpy library, one with with an exponential distribution for reference and one with uniform distribution.

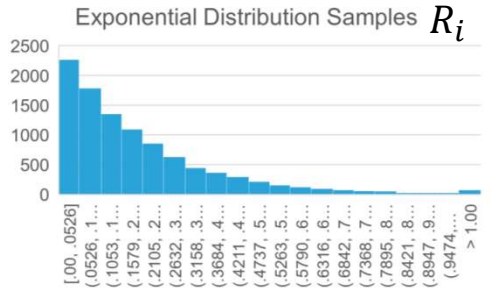
And what we are trying to do here is, we record all the sample from the uniform and the reference distribution so we can plot them later and, last but not least, we also make a new array of values from the result we get when we warp the uniform samples with the inverse of the CDF which we computed on the last slide.

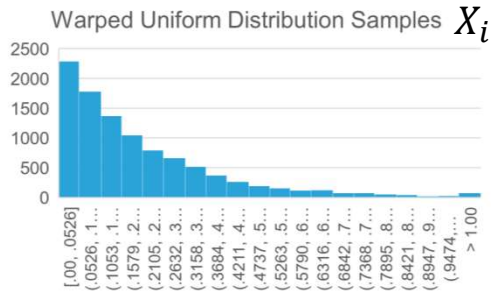
Warping Uniform To Exponential Distribution

- Histograms of generated samples



$X_i = P_X^{-1}(\xi_i)$





Mix Multiple Random Variables

- Let's add another variable and combine them for 2D output
- In doing so, we are extending our sampling *domain*
- The sampling domain is defined by
  - The number of variables, and
  - Their respective ranges
- Think of the domain as a space with the axes representing variables

So now that we saw that we can simulate individual random variables form their PDFS, lets move to 2D. We can generate some basic 2D distributions by combining the output of two random variables. By doing so, we are extending our sampling "domain". When we talk about multidimensional sampling, the domain defines the number of random variables we are using and their respective range. For the sake of visualization, we will plot the domain as a space where the axes represent the individual variables.

Joint PDF

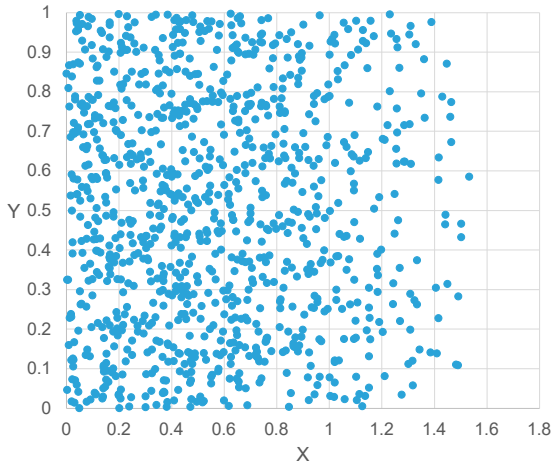
- If multiple variables are in a domain, the **joint** PDF probability density of a given point in that domain depends on all of them
- In the simplest case, with independent variables X and Y, the joint PDF of their shared domain PDF is simply $p(x, y) = p_X(x)p_Y(y)$
- We can sample independent variables in a domain by computing and sampling the inverse of their respective CDFs, separately

In a domain with multiple random variables, every possible outcome is a combination of the values that the random variables take on. Therefore, we define the "joint" PDF for any such combination that defines a point in this domain. In the simplest case, we have independent variables X and Y, and if they are independent, then the joint PDF for a point in their shared domain is simply the product of the separate PDFs for each coordinate. In this case, we can simulate a 2D random variable by computing the inverse of their individual CDFS and using the inversion methods on them with outputs from two canonical random variables.

Inversion Method Examples in 2D

- 2D with $Y = \xi$. For X, use $X \in [0, \frac{\pi}{2})$ and $p(x) = \cos x$
- $P_X(x) = \int p(x)\, dx = \int \cos x\, dx = \sin x$
- $P_X^{-1}(\xi) = \sin^{-1}(\xi)$

```
def gen_cosx(a: float, b: float, N: int):
    xi = np.random.rand(N)
    return np.arcsin(xi)

def p_cosx(x, a: float, b: float):
    return np.cos(x)

X_i = gen_cosx(0, 1, 1000)
plot(X_i, np.random.rand(1000))
```

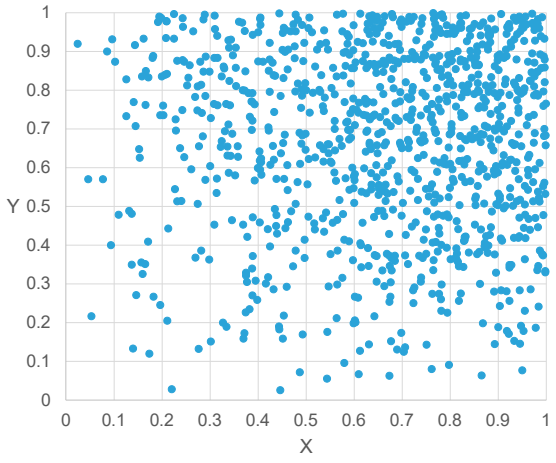


Here we see an example where we use a random variable for the Y axis that mimics the canonical random variable and another for the X axis where the probability is distributed according to the cosine of x. Since the cosine is 1 for input 0 and 1 for input pi/2, we can see that the density of the samples decreases with increasing x. The sampling of this 2D variable is achieved by inverting their CDFs separately and warping samples from two canonical random variables, as shown in the code below.

Inversion Method Examples in 2D

- X and Y in range $[0,1)$
- For both variables, $p(v) = 2v, P(v) = v^2, P^{-1}(\xi) = \sqrt{\xi}$

```
def gen_2v(a: float, b: float, N: int):
    xi = np.random.rand(N)
    return np.sqrt(xi)

def p_2v(v, a: float, b: float):
    return 2*v

X_i = gen_2v(0, 1, 1000)
Y_i = gen_2v(0, 1, 1000)
plot(X_i, Y_i)
```

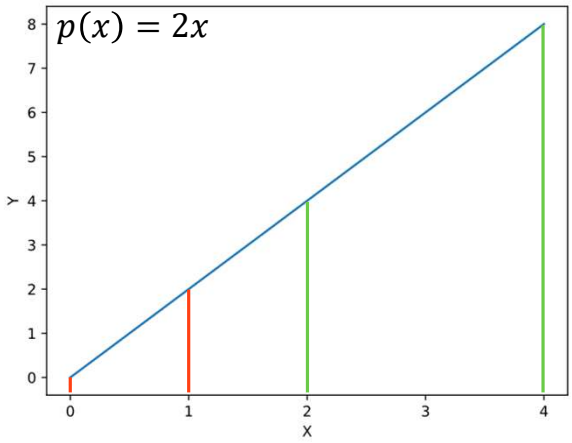


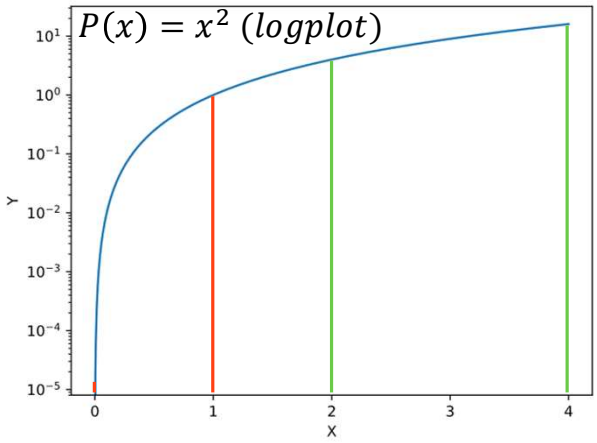
Here we have another small example with variables X and Y both in the range from 0 to 1, and for both variables we are using the same linear PDF.

Again, we can sample this distribution by computing the CDF, inverting it and using samples from different canonical random variables as input. It may be difficult to tell visually, but the joint PDF tells us that there are approximately 4 times as many samples in the upper right corner than there are at the center of the plot.

Choosing a Different Range

- Let's pick a slow-growing portion of the distribution function
- Compared to $[0,1)$, cumulative density only doubles in $[2,4)$

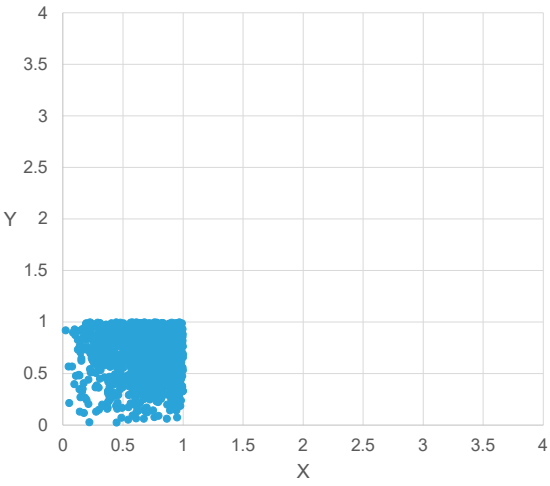




So let's do a little variation on the last setup, and instead of having variables in the range from 0 to 1, we actually want them to replicate the behavior of a linear PDF in the range from 2 to 4. This means that we would see a smaller change in the density, because the relative growth of the CDF is much smaller.

Inversion Method Examples in 2D

- Try X and Y in range $[2,4)$
- For both variables, $p(v) = 2v, P(v) = v^2, P^{-1}(\xi) = \sqrt{\xi}$
- Nothing happens.
- How can we adapt variable ranges?
- Something is missing!



So if we apply the same recipe as before… nothing happens. If we plot a higher range, we see that the samples are the same as before, which is no surprise because our CDF and inverse CDF are also identical. So apparently we are missing some steps if we want to applyt the inversion method to arbitrary setups. Let's look at the missing steps right now.

Restricting the PDF / CDF

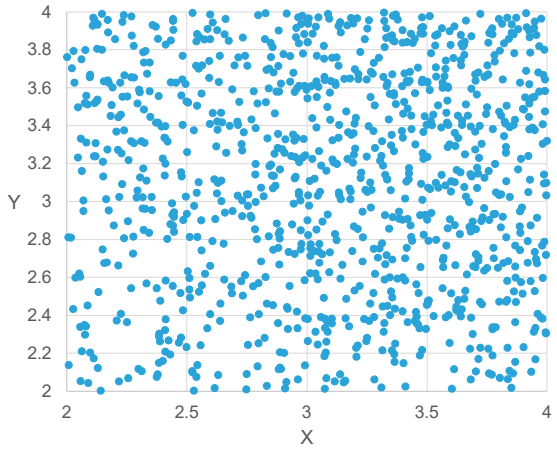
- Consider a given range from a to b for a variable and a candidate PDF $f(x)$ with the desired distribution shape
- If $\int_a^b f(x)\,dx \neq 1$, $f(x)$ is not a valid PDF for this variable

- *The probability that the result is one of all possible results* $\neq 100\%$
- To fix this, we compute the proportionality constant $c = \int_a^b f(x)\,dx$ and compute a valid $P(x) = \frac{F(x)}{c}$ and $p(x) = \frac{f(x)}{c} \propto f(x)$

First, what we need is to ensure that we enforce the necessary on the PDF and CDF we are using restrictions. For now, let's talk about the distribution we want our variable to have as the PDF CANDIDATE f(x). Remember that we said that a CDF is bounded from above by 1. That means that, if we have a random variable with a specified range, the integral of the PDF over that range must equal to 1. If this is not the case, it corresponds to saying that the probability of the variables outcome being one of all possible outcomes is not 100%, which is a paradox. To fix this, we can compute proportionality constant by finding the value that f(x) integrates to over the full range and divide f(x)'s antiderivative by it to get the CDF. This ensures that CDF adheres to its necessary restrictions while making the PDF p(x) proportional to the candidate function f(x).

Restricting the PDF / CDF

- For range $[a, b]$ where $a \neq 0$, we add a constant offset $k = -P(a)$



- Try $X, Y \in [2,4)$ and $f(v) = 2v$ again

- We compute $c_Y = c_X = \int_2^4 2v\, dv = 12$ and add $k = -\frac{4}{12}$ to get:

$$P(v) = \frac{v^2-4}{12},\ P^{-1}(\xi) = 2\sqrt{3 \cdot \xi + 1}, p(v) = \frac{2v}{12}$$

If we are looking at ranges that do not start with 0, we also need to add a constant offset to the CDF. If our variable ranges from a to b, then we subtract the value that the CDF would produce for input a. This is valid because we defined the CDF to be the PDFs antiderivative, and remember that antiderivatives can include arbitrary constant factors, because those fall away when we differentiate. With this in mind we try our setup from before again, but this time apply the proportionality constant and the correct offset. As we can see in the updated plot, this now produces samples with the expected growth in density and in the expected range.

The Inversion Method, Completed

- Find a candidate function $f(x)$ with the desired distribution shape
- Choose the range $[a, b]$ in $f(x)$ you want your variable to imitate
- Determine the indefinite integral $F(x) = \int f(x)\, dx$
- Compute the proportionality constant $c = F(b) - F(a)$
- The CDF for the new variable X is $P_X(x) = \frac{F(x)-F(a)}{c}$
- Compute the inverse of the CDF $P_X^{-1}(\xi)$
- Use $P_X^{-1}(\xi)$ to warp the samples of a canonic random variable so that they are distributed with $p(x) = \frac{f(x)}{c}$ in the range $[a, b)$

So finally, we have the inversion method for simulating random variables with custom distributions completed. To summarize, let's go over the individual steps we need to do once more.

First, we identify a candidate function f(x) that we want for the distribution of values in our random variable,

Second we choose the range inside the candidate function that our variables distribution should imitate

We then compute the indefinite integral, or antiderivate of the candidate PDF to get a candidate CDF

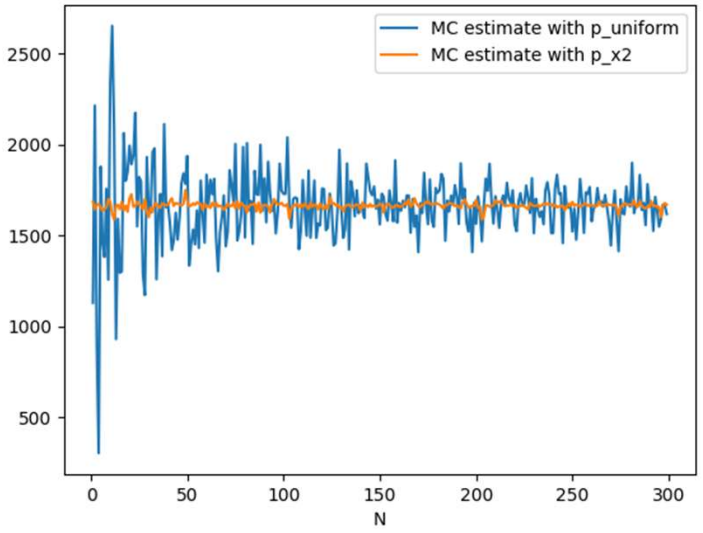
We find the proportionality constant and the final CDF by subtracting the offset and dividing by the c

We then inver the CDF and from now on we can use the inverse to compute warped samples with the desired distribution by using samples from the canonical random variable as input

Deriving the $p(x) \propto x^2$ Sample Generation Functions

integrate_mc(0, 100, N, f, p_uniform, gen_uniform) **vs** integrate_mc(0, 100, N, f, p_x2, gen_x2)

```
def integrate_mc(a: float, b: float, N: int, f, p, gen):
    X = gen(a, b, N)
    estimates = f(X)/p(X, a, b)
    result = estimates.sum() / N
    return result

def p_uniform(x, a: float, b: float):
    return x/(b-a)

def p_x2(x, a: float, b: float):
    b3 = ((b**3)/3)
    a3 = ((a**3)/3)
    return x**2/(b3-a3)
```



Now that we have completed the inversion method, it is a good time to look back at the example that we looked at in the beginning. Remember how we found that sampling a particular function with a distribution proportional to x squared worked much better than uniform sampling. But the code that we used to do it was somewhat unintuitive. Now that we have mastered the inversion method, let‘s look at it once more.

Deriving the $p(x) \propto x^2$ Sample Generation Functions

integrate_mc(0, 100, N, f, p_uniform, gen_uniform) **vs** integrate_mc(0, 100, N, f, p_x2, gen_x2)

```
def integrate_mc(a: float, b: float, N: int, f, p, gen):
    X = gen(a, b, N)
    estimates = f(X)/p(X, a, b)
    result = estimates.sum() / N
    return result

def p_uniform(x, a: float, b: float):
    return 1/(b-a)

def p_x2(x, a: float, b: float):
    b3 = ((b**3)/3)
    a3 = ((a**3)/3)
    return x**2/(b3-a3)
```

$$F(x) = \frac{x^3}{3}, c = \frac{b^3 - a^3}{3},$$

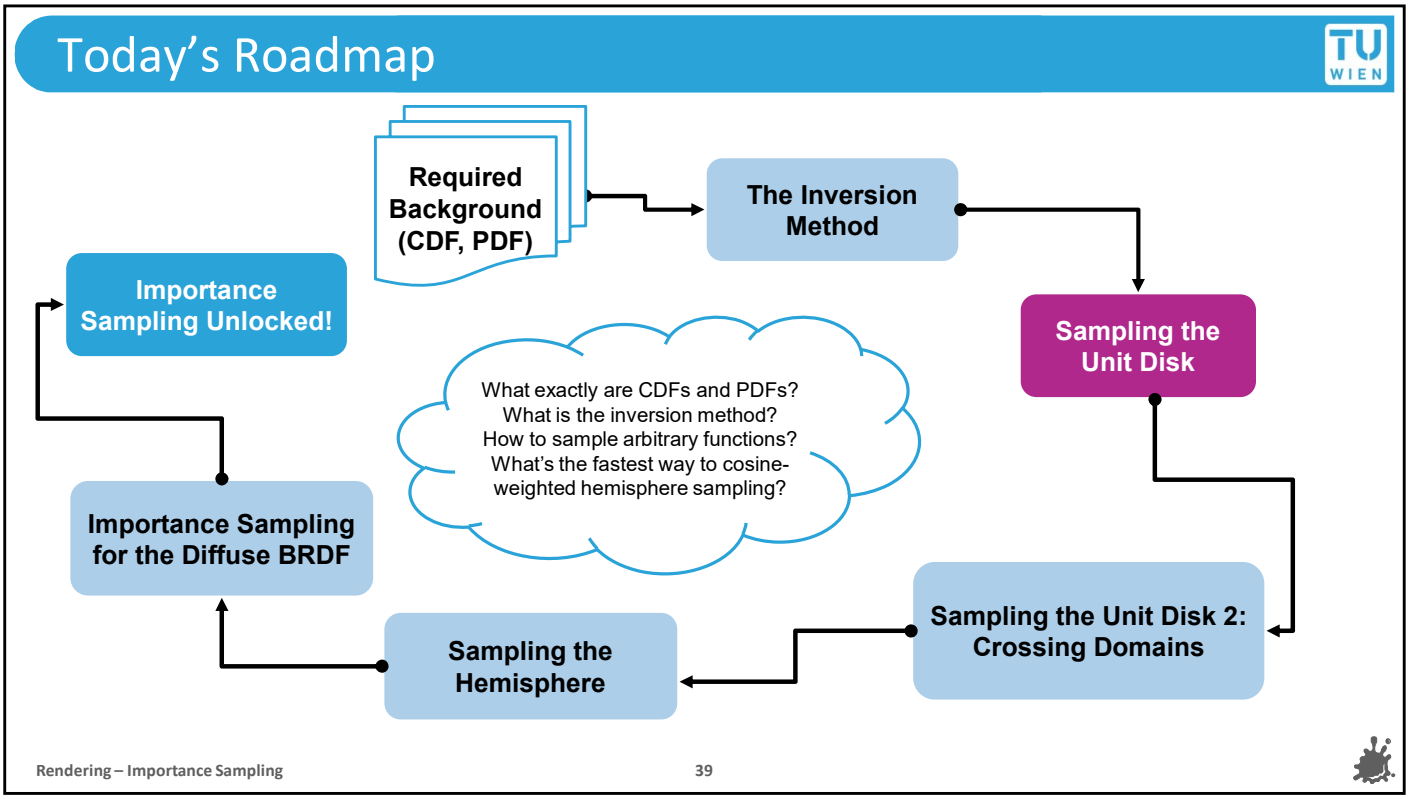
$$p(x) = \frac{x^2}{c},$$

$$P_X^{-1}(\xi) = \sqrt[3]{a^3 + \xi(b^3 - a^3)},$$

```
def gen_uniform(a: float, b: float, N: int):
    xi = np.random.rand(N)
    return xi * (b - a) + a

def gen_x2(a: float, b: float, N: int):
    xi = np.random.rand(N)
    b3 = (b**3)
    a3 = (a**3)
    return (a3+xi*(b3-a3))**(1.0/3.0)
```


And you should confirm for yourself that, in fact, when we follow the recipe for the inversion method, we arrive at the solution that is implemented by these functions. We first compute the indefinite integral, then the proportionality constant, and then use it to get a valid PDF and a CDF, which we then invert. The function p_x2 gives the code for a valid PDF, and the function gen_x2 produces samples, according to the inverse of the CDF. I hope that you can appreciate how cleanly these mathematical concepts that we just developed can be applied in code to give us a desired behavior, but if not that‘s ok too :)

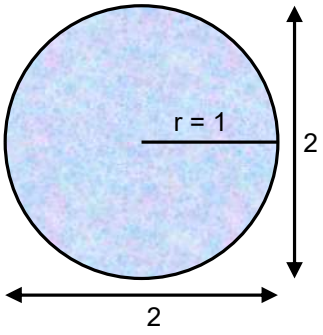
That already takes care of the first big leap for today. But now we are going for a real challenge: solutions for arbitrary sampling strategies with non-cartesian domains!



But before we do this directly on the hemisphere, we are going to start small and first try our hand at something a little simpler: the unit disk.

Sampling a Unit Disk

- Imagine we have a disk-shaped surface with radius $r = 1$ that registers incoming light (color) from directional light sources
- As an exercise, we want to approximate the total incoming light over the disk's surface **area**
- We integrate over an area of size π
- We will use the Monte Carlo integral for that



Before we move to the hemisphere, we start off with a smaller step in the right direction, that is sampling the unit disk.

Imagine that we have a disk-shaped surface, and we can measure the light that arrives at each point as an RGB color.

Now in order to get a measure of the full light that the disk receives, we want to sample and integrate over the surface.

To start off, we will want to generate uniformly distributed samples in 2D, but this time there is a catch: we want to distribute samples evenly

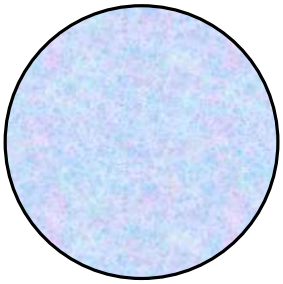
On the surface of a unit disk with radius 1 instead of a rectangular area. For rectangular areas, we saw that we can just take two independent uniform variables and sample them separately.

It is not so simple for the unit disk.

For now, we know that the surface area of the unit disk, is of size pi, so we will keep that information in mind.

Uniformly Sampling the Unit Disk

- If we can manage to uniformly sample the disk, then we can compute the Monte Carlo integral as a simple average $\times \pi$
- By drawing uniform samples in x and y, we cannot cover the area precisely
- Inscribed square: information lost
- Circumscribed square: unnecessary samples

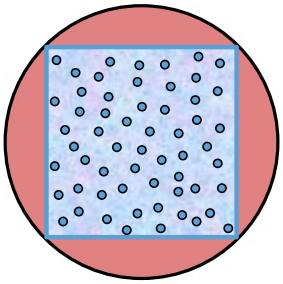


From the Monte Carlo lecture, we know that if we can uniformly sample a domain, then the Monte Carlo integral is a simple average times the volume of the domain.

But if we simply draw samples from uniform variables in x and y, we cannot cover the circle of the disk precisely.

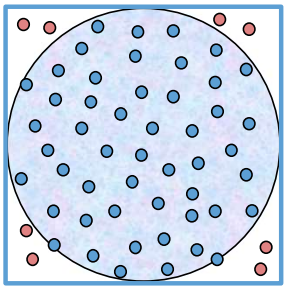
Uniformly Sampling the Unit Disk

- If we can manage to uniformly sample the disk, then we can compute the Monte Carlo integral as a simple average
- By drawing uniform samples in x and y, we cannot cover the area precisely
- Inscribed square: information lost
- Circumscribed square: unnecessary samples



Uniformly Sampling the Unit Disk

- If we can manage to uniformly sample the disk, then we can compute the Monte Carlo integral as a simple average
- By drawing uniform samples in x and y, we cannot cover the area precisely
- Inscribed square: information lost
- Circumscribed square: unnecessary samples



If we want to enclose the unit disk with cartesian coordinates, we can look at x and y coordinates from -1 to 1.

But, inside that 2x2 area, the unit disk only occupies only an area of size of pi out of the total 2x2 area.

So we will necessarily be placing redundant samples that land outside the disk.

Since there is no light measured outside of the disk, those samples are worthless.

Back to the Unit Disk

- We do not want to waste samples if we can avoid it
- Instead, find a way to generate uniform samples on the disk
- Create samples in a non-cartesian domain: 2D polar coordinates
  - Polar coordinates defined by radius $r \in [0,1)$ and angle $\theta \in [0,2\pi)$
  - Transformation to cartesian coordinates:
    $x = r\cos\theta$
    $y = r\sin\theta$

So second attempt to sample the unit disk. Lets find a way to generate inherently uniform samples without throwing them away.

There are other ways to sample a 2D plane. We have a coordinate system that is also 2-dimensional but where we can easily limit points to a disk of given extent. We're talking about polar coordinates.

Polar coordinates around a given origin are defined by a radius r and angle theta in the range from 0 to 2pi. For the unit disk, we will use a radius of 1.

The transformation from polar coordinates to cartesian coordinates is as follows: x = r times the sine of theta and y is r times the cosine of theta.

Uniformly Sampling the Unit Disk?

- Convert two ξ to ranges $[0, 1), [0,2\pi)$ to get polar coordinates

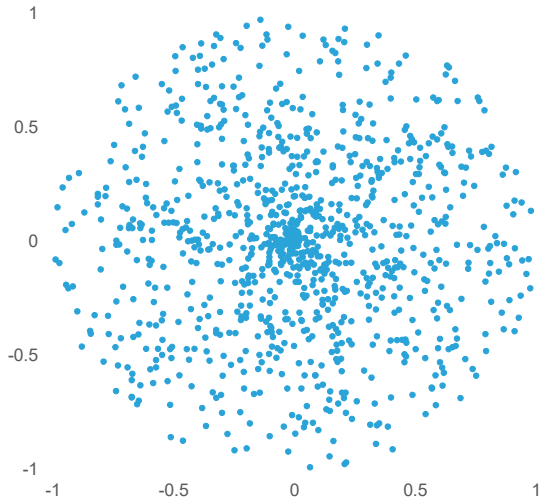
- Convert to cartesian coordinates

```
void sampleUnitDisk()
{
	std::default_random_engine r_rand_eng(0xdecaf);
	std::default_random_engine theta_rand_eng(0xcaffe);

	std::uniform_real_distribution<double> uniform_dist(0.0, 1.0);

	for (int i = 0; i < NUM_SAMPLES; i++)
	{
		auto r = uniform_dist(r_rand_eng);
		auto theta = uniform_dist(theta_rand_eng) * 2 * M_PI;
		auto x = r * sin(theta);
		auto y = r * cos(theta);

		samples2D[i] = std::make_pair(x, y);
	}
}
```


So this seems like a good solution: We generate uniform samples in polar coordinates, where samples can be naturally limited to a disk by a maximum parameter for r and then convert them to cartesian coordinates to get points in x,y! So let's try this right away.

Clumping

- We successfully sampled the unit disk in the proper range
- However, the distribution is not uniform with respect to the area
- Samples clump together at center
- Averaging those samples will give us a skewed result for the integral!



Except, not exactly, because, unfortunately, uniformly distributed samples in polar coordinates are NOT also uniformly distributed in cartesian coordinates, Since we wanted to sample the surface area uniformly, this distribution will not do because samples clump together at the center. And the fact of the matter is, if sample distribution and PDF don't match, that is, if we treat a non-uniform distribution as if it were uniform during monte carlo integration, we will get a wrong result for our integral. So how do we fix this?

You can pause the video here and take five or ten minutes to try and come up with a solution for yourself if you like.

Uniformly Sampling the Unit Disk: A Solution

- The area of a disk is proportional to r^2, times a constant factor π
- If we see the disk as concentric rings of width Δr, the j inner rings up to radius $r_j = j\Delta r$ should contain $\left(\frac{r_j}{r}\right)^2 N$ out of N total samples
- Conversely, the i^{th} sample should lie in the ring at radius $r_i = r\sqrt{\frac{i}{N}}$
- Since ξ is uniform in $[0, 1)$, we can switch $\frac{i}{N}$ for ξ_i to get $r_i = r\sqrt{\xi_i}$

Ok, so let's take a step back, think about the issue again and try to solve this smartly.

We know that the area of a disk is r squared times pi

Consider a sample budget of N samples and try to distribute them evenly over the surface of the disk

If we interpret the disk as many, slim concentric rings of width delta r, then the inner j rings up to radius j times delta r should contain rj over r squared times N of the grand total N samples that we are using

We can invert this figure to find that, if we want to distribute samples uniformly, given the radius r, then the i/th sample should lie in the ring that covers the radius r times square root of i over N.

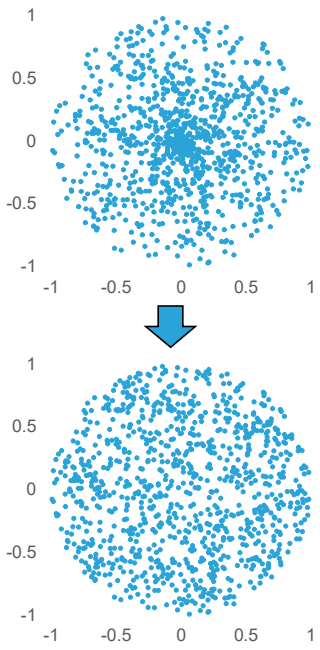
Since we know that the order of samples along our domain axes doesn't matter in monte carlo integration,

we could swap the coordinate i/N for a canonic random variable, and we get r_i = r times sqare root of the output from a canonical random variable.

We don’t need to make a change for theta, because we saw that the distribution doesn’t vary with the angle.

Uniformly Sampling the Unit Disk: A Solution

- It works, and it is not even a bad way to arrive at the correct solution
- However, for more complex scenarios, we might struggle to find the solution so easily
- With the tools we introduced earlier (and a few new tricks), we can formalize this process for arbitrary setups!



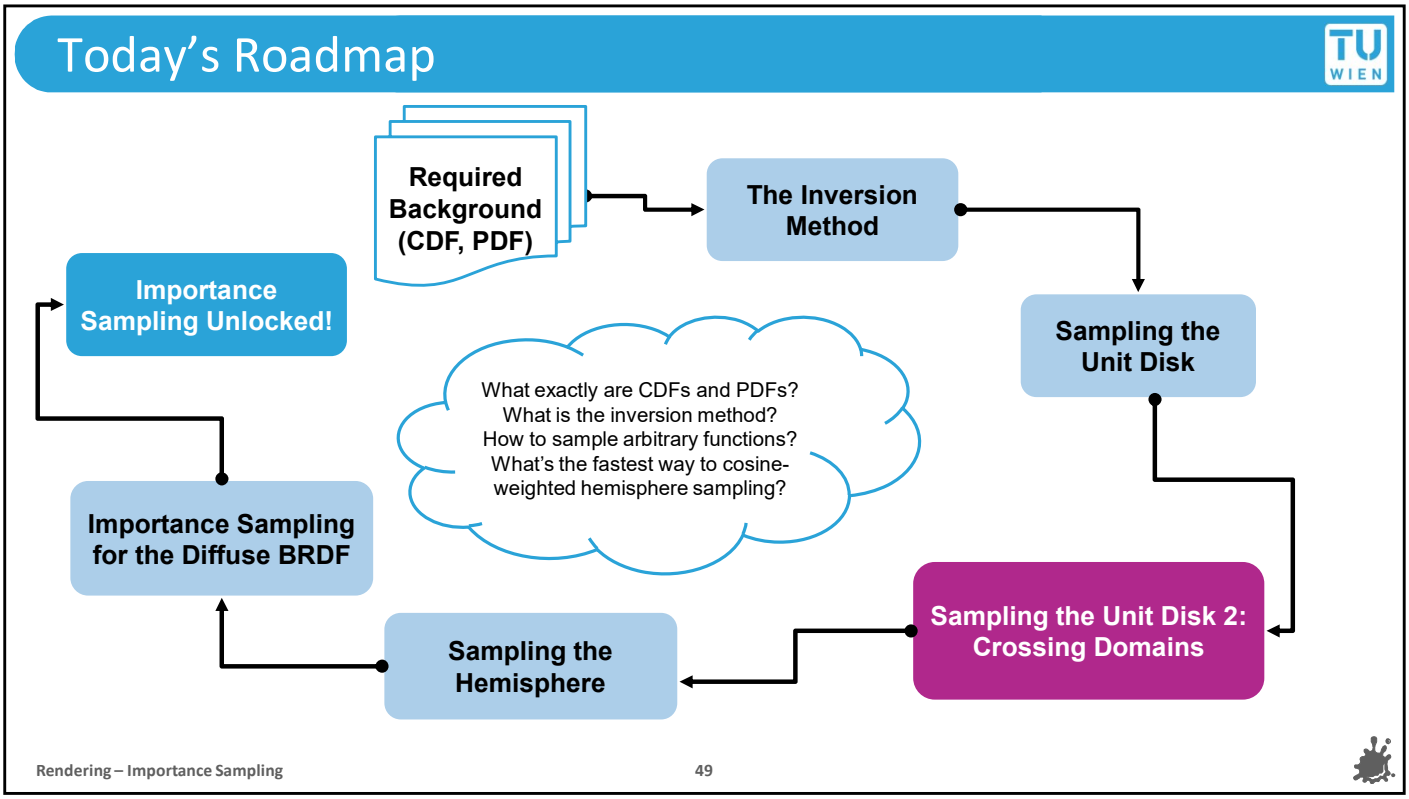
So let's make this change. Instead of uniformly sampling r, we compute it as the square root of a canonical random variable times r.

It works indeed, and it is not even a bad way to arrive at the correct solution

However, if you pursue rendering into its professional domain, you will find that you won't be able to derive the solution for other sampling domains so easily.

You can take this solution and apply it and it will work fine. But we will show you a more formal solution so that you also have the necessary tools if you have to solve more complex tasks than this. So what follows is a formal method to achieve uniform sampling of

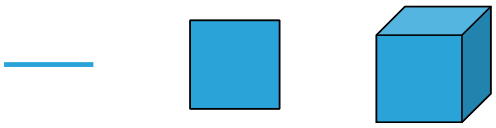
random variables with under transformations.



This brings to our next stop, part 2 of the chapter for sampling the unit disk. This time, we will do it by mathematically analyzing and compensating for the effect that a transformation of samples from one domain to the other has on their distribution.

Another Look at the PDF

- We saw samples being “warped”: we can interpret the inversion method as a spatial transformation of uniform samples
- Let’s treat the space in the input domain like a grid of infinitesimal *hypercubes*: segments in 1D, squares in 2D and cubes in 3D[5]



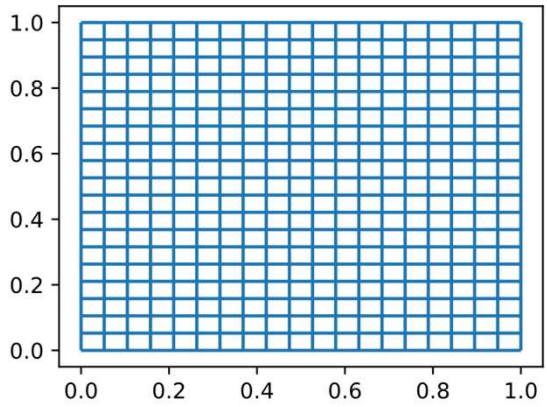
- If we warp a domain where each variable is ξ to one with joint PDF p_D, then $\frac{1}{p_D}$ is the change in volume of the hypercubes after warping

So let’s take a step back for a moment and look at the PDF from a different angle. We saw 2D samples being “warped” into a new, 2D distribution, and we can interpret this warping as a transformation of the space from the input domain to a target distribution. If we convert uniformly spaced input samples with the inversion method, we usually don’t get uniformly spaced outputs.

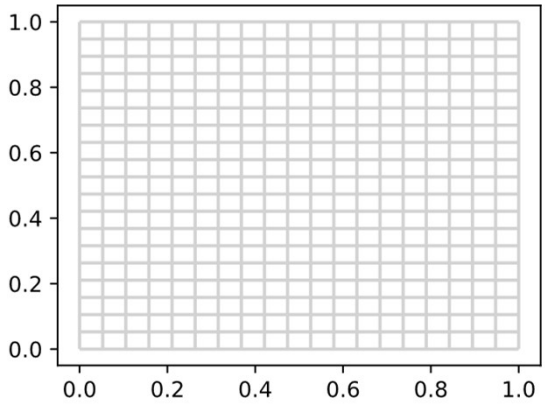
Let’s treat some tiny, regular interval in the input domain as an infinitesimal hypercube: a hypercube is a line in 1D, a square in 2D and a cube in 3D. If we warp the inputs of canonic variables as we did before to samples of a distribution with a joint PDF pD, then we can express the change in the volume of these hypercubes by 1 over pd.

Visualizing the PDF in 2D

- The left represents our inputs and the right our target distribution
- This time, we warp **grid** coordinates with the inversion method



ξ_1, ξ_2

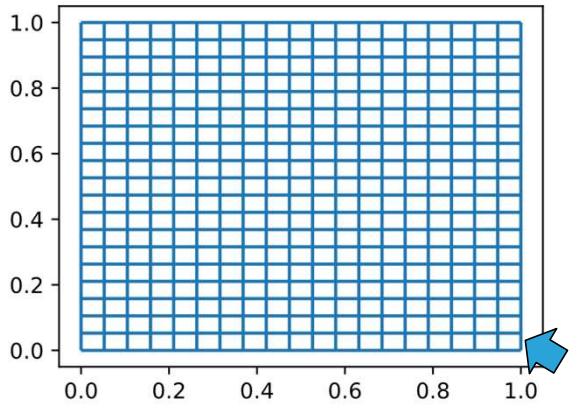


$Y = \xi_2$ and $X \in [0,1), p_X(x) = 2x$

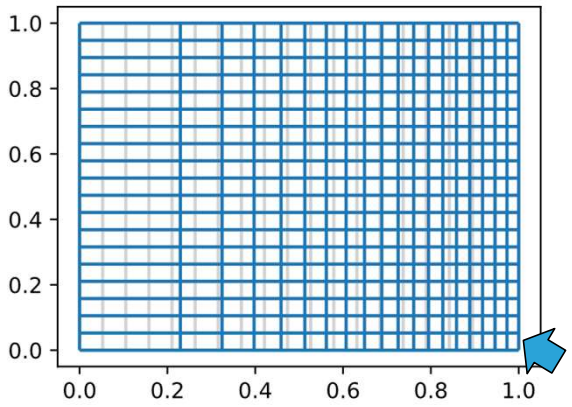
Here we have on the left our input variables, Xi 1 and Xi 2, and we plot a regular grid over the entire 2D input domain. On the right-hand side, we will plot the same grid coordinates after we warp them with the inversion method to produce the distribution given below, which is uniform for y coordinates and linear for x coordinates.

Visualizing the PDF in 2D

- The areas of all 2D hypercubes (grid cells) are scaled by $\frac{1}{p_X(x)p_Y(y)}$
- $p_X(x) = 2x$, cells on the right at $(1, y)$ are half their original width



ξ_1, ξ_2



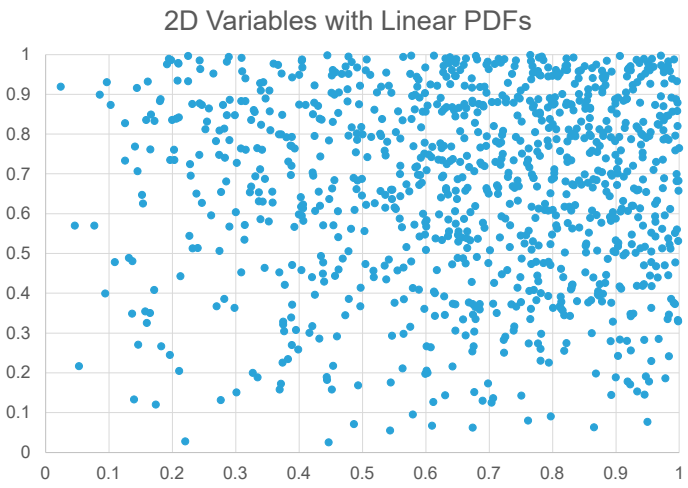
$Y = \xi_2$ and $X \in [0,1), p_X(x) = 2x$

And we can see that if we do that, the areas of all 2D hypercubes are scaled by the facter 1 over px(x).

Note that, after the transformation, the rectangles on the very right are approximately half as wide as the original ones

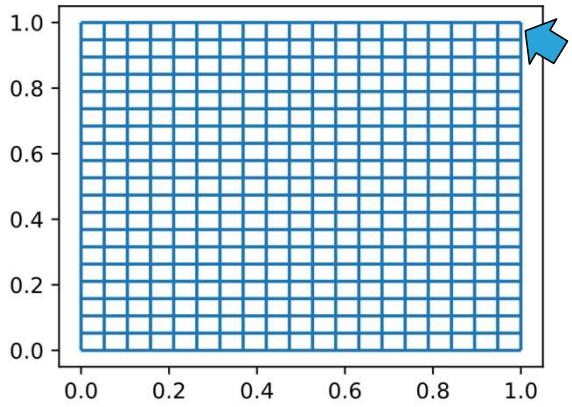
Visualizing the PDF in 2D

- Earlier, we saw samples $X, Y \in [0,1)$ with $p_X(x) = 2x, p_Y(y) = 2y$

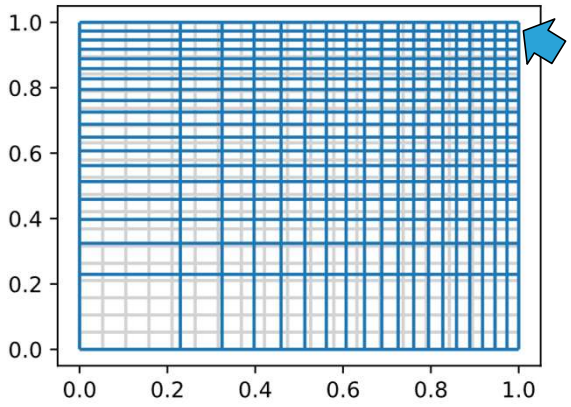


Visualizing the PDF in 2D

- In this 2D setup, we have joint PDF $p(x,y) = p_X(x)p_Y(y) = 4xy$
- Space near point $(1,1)$ is compressed down to $\frac{1}{4}$ of its original size



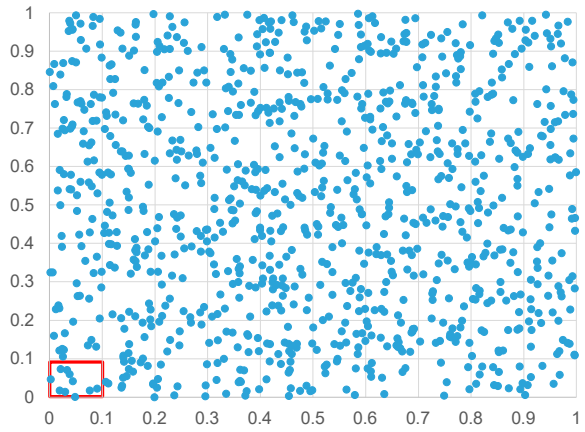
ξ_1, ξ_2



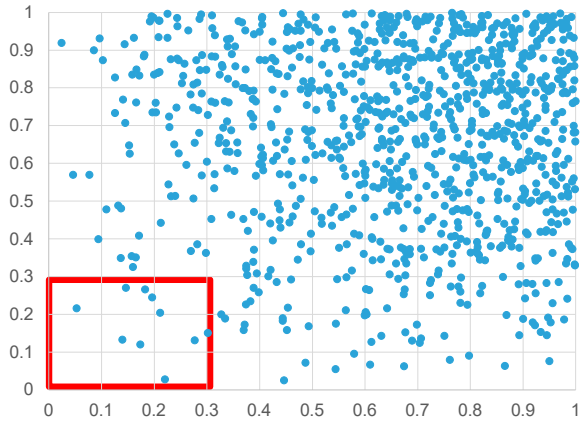
$X, Y \in [0,1), p_X(x) = 2x, p_Y(y) = 2y$

Visualizing the PDF in 2D

- This PDF compresses space at higher values of x, y, dilates at lower
- If space shrinks or grows, samples in it become denser or sparser



ξ_1, ξ_2



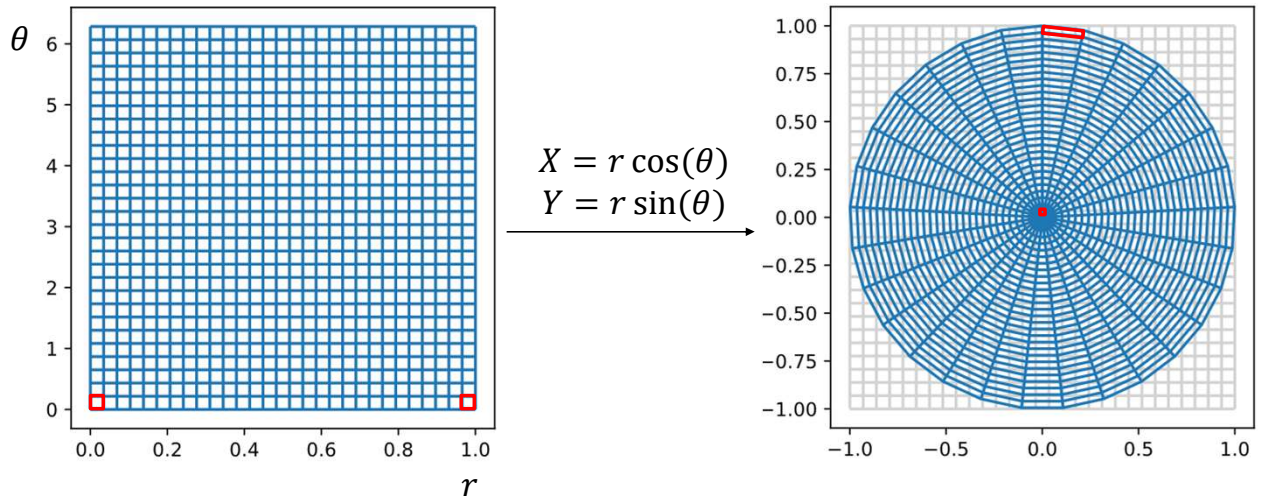
$X, Y \in [0,1), p_X(x) = 2x, p_Y(y) = 2y$

And if we plot once again the samples that are generated by the canonical random input variables and again after the transformation to a distribution with the desired PDF, we can see that the space that was occupied before by a small hypercube has now been expanded, and with it the distance between the samples in this region.

So in summary, an alternative way to interpret the PDF is to take its reciprocal and treat 1/ the pdf as the change of the volume for infinitesimally small regions at given points.

Polar To Cartesian Coordinates

- Let's transform a regular grid from polar to cartesian coordinates



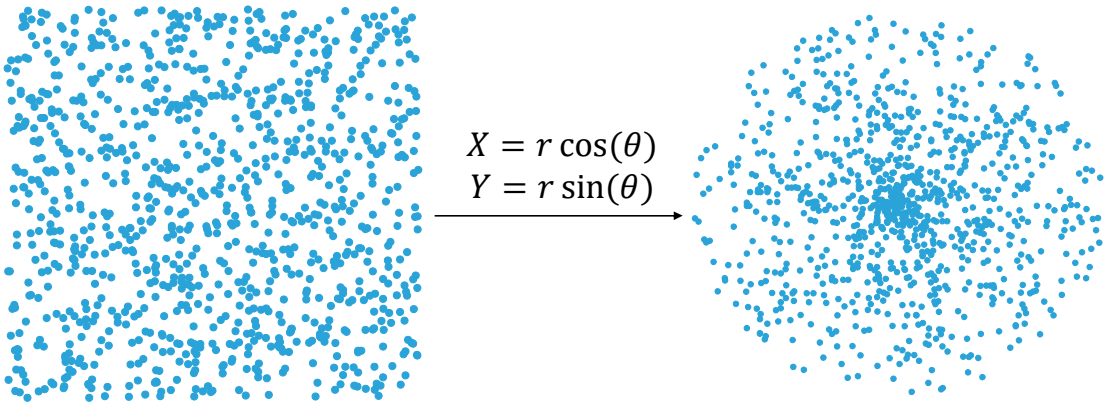
Right away, let us convert a regular grid from polar to cartesian coordinates. What happens if we do that?

We can see that some squares in the grid are squished together, while others are expanded, depending on where they are in polar coordinates.

Remember when we interpreted the reciprocal of the PDF as a transformation of hypercube volume? So we can see that the transformation from polar to cartesian coordinates seems to have a clear effect on the PDF. And we can make this even more clear.

First Attempt to Learn the PDF

- Take 100k samples, transform and see in which square they end up

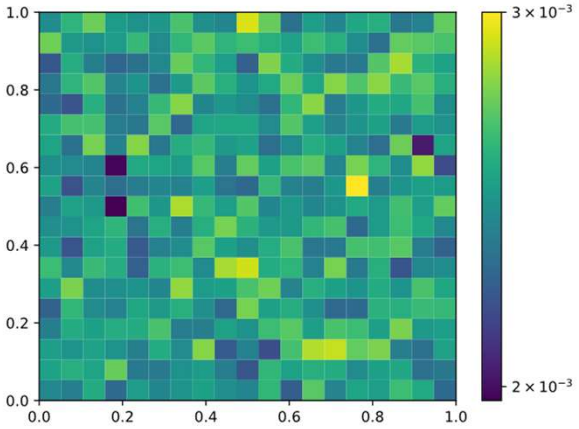


Here is the same transformation, but now instead of a grid with uniformly created random points from polar coordinates to cartesian coordinates.

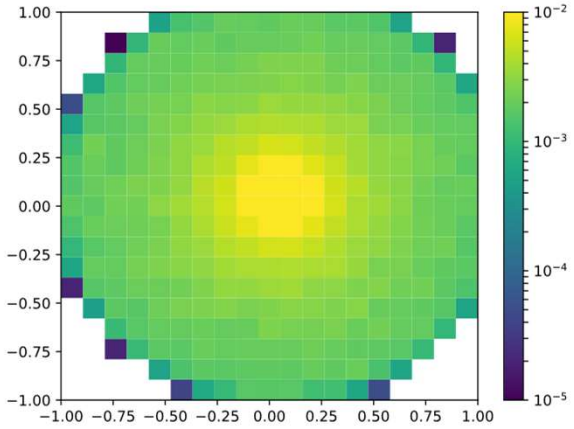
We already know and see that this distribution doesn't look uniform, but we can still visualize it a bit more clearly

First Attempt to Learn the PDF

- Take 100k samples, transform and see in which square they end up



ξ_1, ξ_2



$X = \xi_1 \cos(2\pi\xi_2), Y = \xi_1 \sin(2\pi\xi_2)$

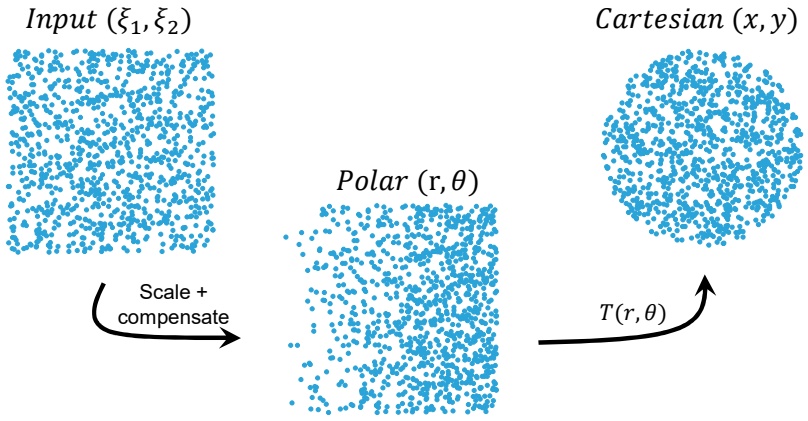
The PDF in the source and target domain for a particular sampling strategy after is actually quite easy to visualize. All you have to do is subdivide your target domain into bins, generate a number of samples and then count how many of them end up in each box. Optionally you can normalize the counters to get the actual density values for each bin. Note that by making it discrete in this way, what we see is technically a probability MASS function now rather than a probability density function, but for an infinite number of bins, they are the same. What we see confirms what we saw earlier: after transforming uniform samples in polar coordinates to cartesian ones, the density is much higher at the center of the disk.

So we know we need the transformation from polar to cartesian coordinates to make samples inside a disk.

But we also know that this transformation changes our sample distribution!

Knowing the PDF

- If we know the effect of a transformation T on the PDF, we can
  - Use it in the Monte Carlo integral to weight our samples, or
  - Compensate to get a uniform sampling method **after** transformation



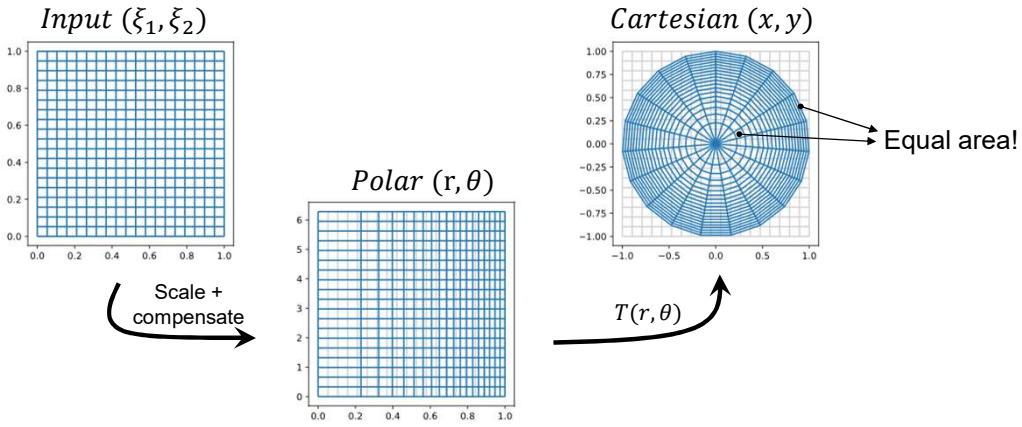
Now our goal is to find out exactly what this change to the PDF is when samples are being transformed. If we can quantify this effect, we could

Use it weight samples during monte carlo integration, because we know the actual PDF, or

Compensate the sample generation in polar coordinates so that they will be uniform AFTER the transformation

Knowing the PDF

- If we know the effect of a transformation T on the PDF, we can
  - Use it in the Monte Carlo integral to weight our samples, or
  - Compensate to get a uniform sampling method **after** transformation



We are going to go with the second option.

Computing the PDF after a Transformation

- Assume a random variable A and a **bijective** transformation T that yields another variable $B = T(A)$
- Bijectivity implies that $b = T(a)$ must be either monotonically increasing or decreasing with a
- This implies that there is a unique B_i for every A_i, and vice versa
- In this case, the CDFs for the two variables fulfill $P_B(T(a)) = P_A(a)$

Assume we have a random variable A and a bijective transformation T whose input is A and whose output is B, which itself is another random variable.

If the transformation is bijective, this implies that the output of T must be either monotonically increasing or decreasing with a.

This in turn means that we can find a unique output value for every input value, and vice versa.

Consequently, we find that the output of the CDF of the transformed random variable for a transformed value T(a) is equal to the PDF of the original variable A for value a.

Computing the PDF after a Transformation

- If $b = T(a)$ and b increases with a, we have: $\frac{dP_B(b)}{da} = \frac{dP_A(a)}{da}$
- If b decreases with a (e.g., $b = -a$), we have: $-\frac{dP_B(b)}{da} = \frac{dP_A(a)}{da}$

- Since p_B is the non-negative derivative of P_B, we can rewrite as:

$$p_B(b)\left|\frac{db}{da}\right| = p_A(a), \qquad Using: \frac{dP_X(x)}{dy} = \frac{p_X(x)\,dx}{dy}$$

$$p_B(b) = \left|\frac{db}{da}\right|^{-1} p_A(a)$$

So that leaves two possibilities for the variable b. It can either increase with a, or decrease with a, for instance if b is just negative a.

In both cases, we have an equality for the change in the CDF of B over a change in a and the change in the CDF of A over a, just the sign in front differs.

The term on the right is just the derivative of the CDF for A, and we know the derivative of the CDF is the PDF.

For the term on the left, we can use that the change in the CDF is equal to the PDF times dx, so we get pB of b times db over da equals pa of a.

We take the absolute value for db over da, because the differentials could be going in different directions if for

instance b is negative a.

But since we want to find only the amount by which the PDF changes, and PDFs must always be positive, so we take the absolute value of db over da.

Taking the reciprocal and moving the term to the right, we get a formula for expressing the PDF of the transformed random variable as a change of to the A’s PDF!

Computing the PDF after a Transformation

- Let's interpret $p_B(b) = \left|\frac{db}{da}\right|^{-1} p_A(a)$
- It is the probability density of A at a, multiplied by $\left|\frac{db}{da}\right|^{-1}$
- $\left|\frac{db}{da}\right|^{-1}$ has two intuitive interpretations:

  the change in sample density at point a if we transform a by T
  or,
  the reciprocal change in volume (space) for a volume element (hypercube) at point a if we transform A by transformation T

Let's try to find an intuitive interpretation for this equality:

The PDF of b is that PDF of a times db over da inverted. Now, based on what we have done before, we can interpret this term in two ways.

We can either see it as the change in sampling density at a if we transform by T
or,
the inverse change in the volume of an infinitesimal hypercube at location a if we transform a by T

Try to make some sense of the latter interpretation, because we will return to it in a minute

Multidimensional Transformations

- If we try to apply the above to the unit disk, we fail at $x = r \sin\theta$
- We can't evaluate $\left|\frac{dx}{dr}\right|^{-1}$: the transformation that produces one target variable is dependent on both input variables and vice-versa
- We cannot compute the change in the PDF between individual variables, we must take them all into account simultaneously
- It's matrix time!

Ok great,so we try to apply this method to the unit disk but we find that our tools don't work here because the transformation is not a function of just one variable

It's a function of TWO variables.

So in order to quantify the change in the PDF in this multidimensional setup, we must find a solution that takes all involved variables into account simultaneously.

And we can do this by moving from individual values to handling matrices of random variables.

Multidimensional Transformations

- We write the set of N values from a **multidimensional** variable $\vec{A}$ as a vector $\vec{a}$ and the N outputs of transformation T as a vector $\vec{b}$:

$$\vec{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix}, \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} = \begin{pmatrix} T_1(\vec{a}) \\ \vdots \\ T_N(\vec{a}) \end{pmatrix} = T(\vec{a})$$

- Instead of quantifying the change in volume incurred by $T(a)$, $\left|\frac{dT(a)}{da}\right|$, our goal is now to quantify the change incurred by $T(\vec{a})$

We can collect all of our individual, single output variabes into one multidimensional random variable. The output from such a variable is a vector. We can do this for the original variable A and the transformed variable b.

If we do this, then we can just take equality from before and replace the PDF of A with the joint PDF of the multidimensional variable A arrow and the same goes for b.

But for the actual change in the PDF, we have to look at the term that before was the inverse of db over da, or d T(a) over da.

We now have to make sure that this term incorporates the full change of the joint PDF when we transform one multi-dimensional variable to another.

The Jacobian Matrix

- For a transformation $\vec{b} = T(\vec{a})$, we can define the Jacobian matrix that contains all b_j, a_i combinations of partial differentials

$$J_T(\vec{a}) = \begin{pmatrix} \frac{\partial b_1}{\partial a_1} & \cdots & \frac{\partial b_1}{\partial a_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_M}{\partial a_1} & \cdots & \frac{\partial b_M}{\partial a_N} \end{pmatrix}$$

- If we consider $\vec{A}$'s domain as a space with N axes, $J_T(\vec{a})$ gives the change of the edges of a volume element from $\vec{a}$ to $\vec{b} = T(\vec{a})$

This is where we have introduce, or recap if you already know it, the Jacobian matrix.

If we have a transformation from vector a to vector b, the Jacobian matrix contains all combinations of partial differentials for the individual values in both vectors.

Therefore the Jacobian contains the values of all possible partial derivatives that we can form with vectors and b.

So if we take the first row and first column, the entry indicates how the first value of vector b changes with the first value in vector a.

In the second row, first column, we see how the second value of vector b changes with the first value in vector a, and so on and so forth.

Basically the Jacobian matrix in our application

encodes for a given location vector a, how each transformed variable changes with each untransformed variable.

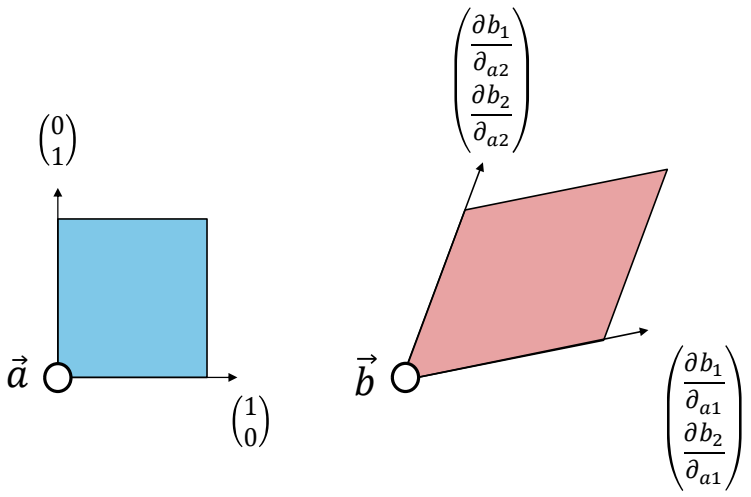
The first column indicates how all transformed variables would be influenced by infinitesimal changes in the first untransformed variable.

The second column indicates how all transformed variables would be influenced by infinitesimal changes in the second untransformed variable, and so on and so on.

The Jacobian Matrix, Visualized

- Change in edges of a volume element (infinitesimal hypercube) at $\vec{a}$

$$J_T(\vec{a}) = \begin{pmatrix} \frac{\partial b_1}{\partial a_1} & \cdots & \frac{\partial b_1}{\partial a_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_N}{\partial a_1} & \cdots & \frac{\partial b_N}{\partial a_N} \end{pmatrix}$$

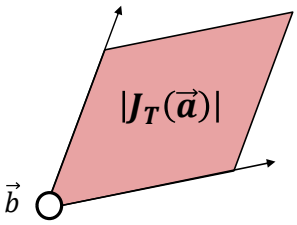


If we again take the help of our imaginary infinitesimal hypercubes, we could say that each column of this Jacobian matrix shows us how the edges of an axis-aligned hypercube in our input domain change during the transformation T.

And this is interesting because it means, for any given position vector a, we can see the Jacobian matrix itself as a transformation matrix that is valid only for an infinitesimally small region at position vector a.

The Jacobian

- The columns of a square matrix can be interpreted as the natural base vectors of a space $\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}, \begin{pmatrix}0\\1\\\vdots\\0\end{pmatrix}$ if they were transformed by it
- The **determinant** $|\,.\,|$ of a matrix yields the volume of a parallelepiped spanned by these vectors[3]



- $|J_T|$, *the Jacobian of* T, gives the change in volume at $\vec{a}$ due to T

The final piece that we need is the determinant. If we are comfortable with seeing the Jacobian matrix as a transformation matrix, then we can quickly quantify the change in volume that generates. The columns of a square transformation matrix can be seen as the basis vectors of the transformed space. With this interpretation, the determinant of a matrix computes the volume of the parallelepiped that is formed by these basis vectors.

So we are going to replace the term of change in range from earlier with the determinant of the the Jacobian matrix of, or the Jacobian of T, for short.

This gives us the change in volume of an infinitesimal hypercube at a multidimensional location if we transform it with transformation T. This is exactly what we have been looking for.

Now if this made your head spin a little, that is totally fine. Feel free to revisit this part of the lecture, because it is definitely a tough concept to grasp.

You don’t need to remember these steps to apply the techniques,

But we wanted to show you the underlying idea and some reasoning for why it makes sense to use the reciprocal of the Jacobian to quantify the change in the PDF, rather than just give you the final formula.

Computing the PDF of a Transformation

- Let's try polar coordinates again: $\begin{pmatrix} x \\ y \end{pmatrix} = T\begin{pmatrix} r \\ \theta \end{pmatrix} = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix}$
- $\left|\frac{\partial T\begin{pmatrix} r \\ \theta \end{pmatrix}}{\partial\begin{pmatrix} r \\ \theta \end{pmatrix}}\right| = |J_T| = \left|\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{pmatrix}\right| = \left|\begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix}\right| = \mathrm{r}$
- $p(x,y) = \frac{p(r,\theta)}{r}$, or $p(r,\theta) = r\,p(x,y)$, which tells us: the change in probability density from (r,θ) to (x,y) is **inverse proportional to r**

Returning to the unit disk, we try measure the change in the PDF caused by the transformation, this time with multidimensional random variables.

We can easily find the Jacobian matrix of T by computing all partial derivatives, and if we take the determinant of the resulting matrix, we get radius r

So we have equality, p(x,y) is equal p(r, theta) over r, or p(r,theta) is equal to r times p(x, y).

What this tells us, is, a uniform density in cartesian coordinates x,y must be proportional to r in polar coordinates r, theta

Sampling Joint PDFs Correctly

- For independent variables, the joint PDF $p(x, y, \dots)$ is $p_X(x)p_Y(y) \dots$
- In general, this is an assumption that we should not rely on
- Furthermore, after a transformation, only the joint PDF is known
- The proper way to sample multiple variables X, Y is to compute
  - the *marginal density function* $p_X(x)$ of one
  - the *conditional density function* $p_Y(y|x)$ of the other

The last challenge we need to face is the correct sampling of a joint PDF. Until now, we only saw distributions from independent variables, we knew their individual PDFs, so we could simply invert their CDFs und sample them separately. But this is an assumption that we should not make in general, and there is a proper way to do it that only requires a little extra effort.

In the case of two variables and given their joint PDF, we can do this as follows: we first compute the marginal density function of one of the variables, and then the conditional density function of the other.

We will use these, marginal and conditional density functions, instead of the PDFs for individual variables, because that is only guaranteed to work if they are independent.

But the steps we need to perform on them are the

same as before: integrate them, invert the indefinite integrals, and use them for sampling.

If we do this, it basically equates to a procedure where we first sample one of the random variables and then sample the other, because the second variable's sampling might depend on what was returned by the first, hence "conditional".

Marginal and Conditional Density Function

- Assume we have obtained the joint PDF $p(x, y)$ of variables X, Y with ranges $[a_X, b_X)$ and $[a_Y, b_Y)$
- In a 2D domain with X, Y we can think of $p_X(x)$ as the average density of $p(x, y)$ at a given x over all possible values y
- We can obtain the *marginal density function* for one of them by *integrating out* all the others, e.g.: $p_X(x) = \int_{a_Y}^{b_Y} p(x, y)\, dy$
- We can then find $p(y|x) = \frac{p(x,y)}{p_X(x)}$

How do we get these marginal and conditional density functions? We will look at the case for 2 variables first.

You might have heard about marginal probability already. If you have multiple variables that define your outcomes, the marginal density function of a variable is the probability density function for that one variable alone, we don't care about the values of the other.

So for example, if out of all possible combinations of x and y we have 50% chance of seeing x=1, then the marginal density for x based on p(x,y) will be 0.5 for x = 1.

You can also see a variable X's marginal pdf as the function tells you the average density at a given value x over the entire range of the other variable.

There is an easy way to get a closed-form solution for the marginal pdf of a variable, and that is by “integrating out”.

We do this by integrating the joint PDF over the full range of the other variable.

The rules of probability tell us how we can get the conditional density as well, that is, the probability of seeing a certain value y for a given value x.

Once we have the marginal density function, computing the conditional density function is easy, we just divide the joint PDF by it.

Adding More Variables

- What to do for multiple variables, e.g. X, Y and Z?
  - Find first marginal density $p_X(x) = \int_{aZ}^{bZ} \int_{aY}^{bY} p(x,y,z)\, dy\, dz$
  - Find first conditional density $p_X(y,z|x) = \frac{p(x,y,z)}{p_X(x)}$
  - Find second marginal density $p_Y(y|x) = \int_{aZ}^{bZ} p(x,y,z)\, dz$
  - Find second conditional density $p_X(z|x,y) = \frac{p(y,z|x)}{p_Y(y|x)}$
  - Integrate + invert first marginal, first and second conditional densities
  - Sample each of them
  - Extend ad libitum to even more variables

For the sake of completeness, we also include for you here the procedure to extend this to more variables. We won't be using this anytime soon, but in case you were wondering how this can be scaled, you can simply follow this method here.

Sampling the Unit Disk: The Formal Solution

- We know the proportionality constant is π (area of sampled disk)
- Since we want uniform sampling and sample probabilities should integrate to 1, the target PDF in cartesian coordinates is $p(x, y) = \frac{1}{\pi}$
- $|J_T|$ told us that $p(r, \theta) = r\, p(x, y)$, so we want $p(r, \theta) = \frac{r}{\pi}$
- $p_R(r) = \int_0^{2\pi} p(r, \theta)\, d\theta = 2r$ and $p(\theta|r) = \frac{p(r,\theta)}{p_R(r)} = \frac{1}{2\pi}$

Ok finally! Let's apply all that we have learned and find a formal solution for uniformly sampling the disk. Now that we know all the steps, this will be really quick.

We know that the sampling domain has a total area of pi, and we know that the PDF must integrate to one over the sampling domain, so we know that the density for any 2D point should be 1 over pi.

We also know from before that, if we want a certain PDF in cartesian coordinates, we must multiply by the r to get the corresponding PDF in polar coordinates, so we should the PDF r over pi in polar coordinates for a uniform PDF in cartesian coordinates.

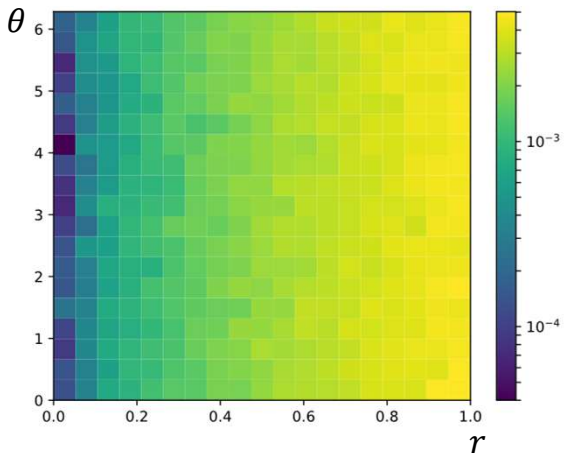
Finally, we compute the marginal density function for r

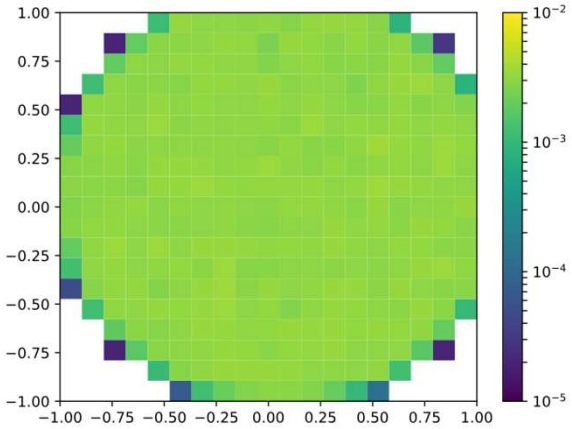
by integrating the joint density function over the full range of theta, and then the conditional density for theta given r.

And from that, we already know what to do: integrate the PDFs to get CDFs, invert them, and sample them to samples for our custom distributions. Let’s quickly check if it worked.

Sampling the Unit Disk: The Formal Solution

- If we create samples in polar coordinates for these PDFs, we will get the uniform distribution in (x, y) after applying transformation T

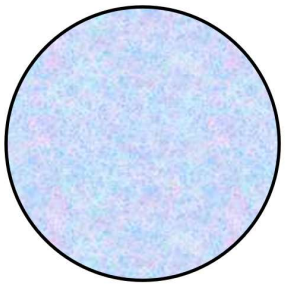




And yes, in fact we can confirm that this sampling method in polar coordinates gives us a uniform distribution in cartesian coordinates!

Sampling the Unit Disk: The Formal Solution

- Integrate marginal and conditional PDFs and invert—we get the same solution as before:
  - $r = \mathrm{P}_R^{-1}(\xi_1) = \sqrt{\xi_1}$
  - $\theta = P_\Theta^{-1}(\xi_2) = 2\pi\xi_2$



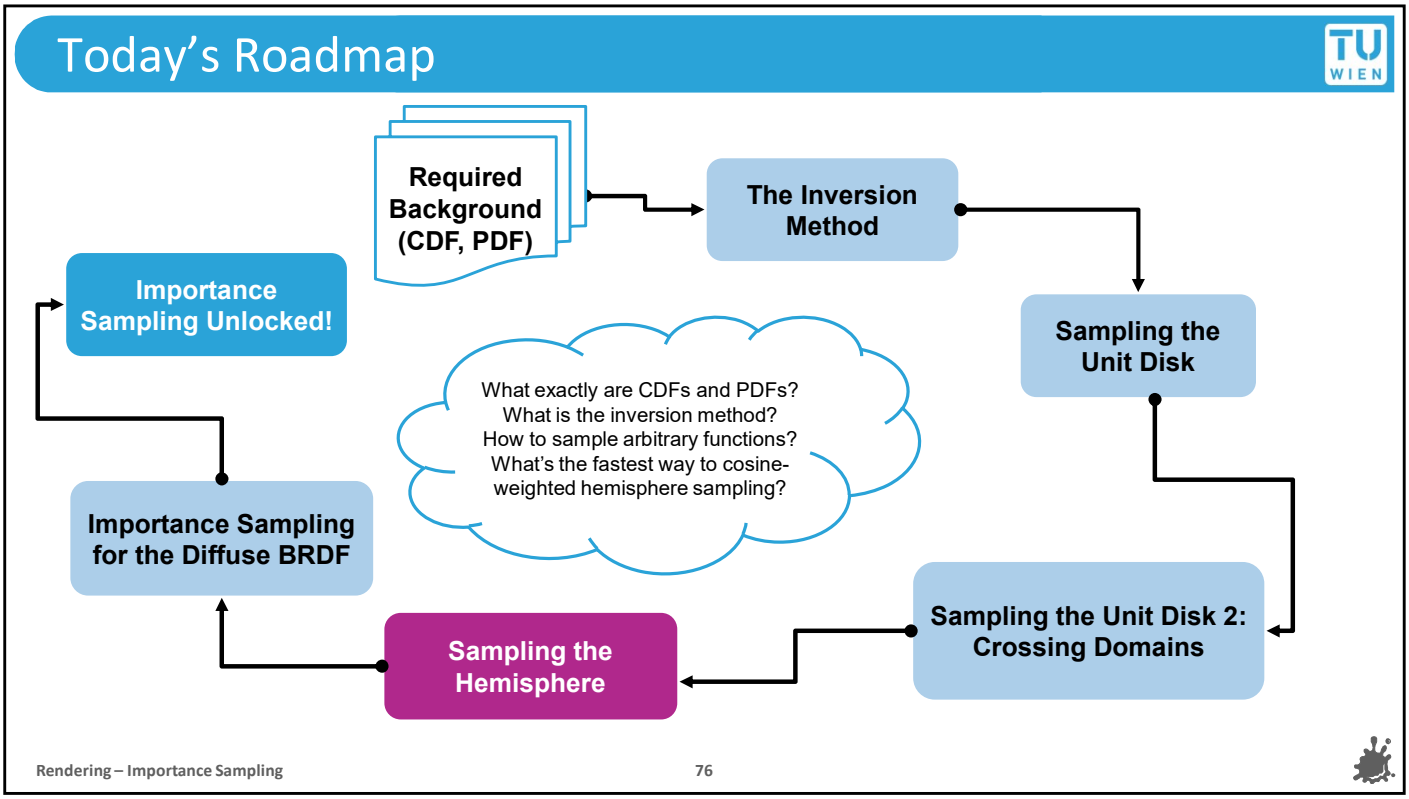
- $p(\theta|r)$ is constant: no matter what radius we are looking at, all positions on a ring of that radius (angle) should be equally likely

- Final integral: $RGB_{total} = \frac{\pi}{N}\sum_{i=1}^{N} RGB(R_i \sin\Theta_i, R_i \cos\Theta_i)$

So it is no surprise, the solution is exactly the same as before, so we can confirm that it is correct.

Something that we can confirm now: even though we computed the conditional density of theta, it didn't actually change much. We are still sampling uniformly from 0 to 2pi, as we did when we first tried sampling the disk. But in more complex setups, this may not be the case, so it's best to stick with the procedure and compute the conditional density whenever you try to enforce a particular sampling after a transformation.

The final integral over the disk is just a formality, because the samples are uniformly distributed over the area and we are integrating over the area, we can use the simpler Monte Carlo integration, which is a simple average of the sampled RGB colors, times PI.



Ok, that was quite a ride. But the good thing is, this was the final mathematical tool that we needed to learn about. From here on out, it's basically just repeating and applying things we already know when we get to our next important stop, sampling the hemisphere.

Moving on to the Hemisphere

- This took as a while, but we have seen all the formal procedures
- We only need to switch from integrating planar area to points ω on hemisphere surface (i.e., vectors (x, y, z) with length 1)
- Use spherical coordinates and bijective T from (r, θ, ϕ) to (x, y, z):

$$x = r \sin\theta \cos\phi$$
$$y = r \sin\theta \sin\phi$$
$$z = r \cos\theta$$

Now that we have mastered the unit disk, let's move on to the unit hemisphere. This is what we wanted all along, a way to integrate functions over the hemisphere surrounding a point in the scene to find out the total amount of incoming light.

So instead of integrating over a planar area, we will now be integrating over the surface area of the unit hemisphere, or direction omega in 3d of length 1.

But sampling in 3D with the restriction that vectors all must be of length 1 would be kind of awkward.

We would have to throw away all the samples that don't land exactly on the surface of the sphere.

So instead, we are gonna use a space where the surface of the sphere is an easy shape to get, and for that we use spherical coordinates, which are defined by a radius r and two angles theta and phi.

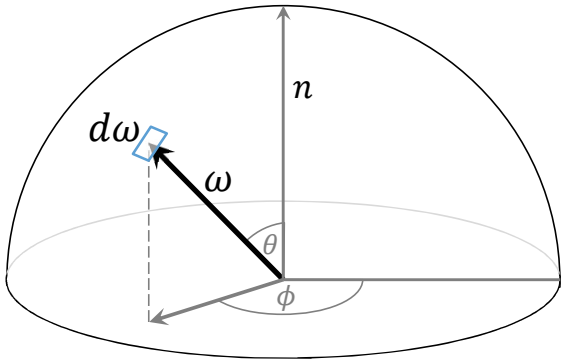
Like with the unit disk, we have a transformation from one sample space to the other.

The bijective transformation from r, theta and phi to x,y, and z is something that we can easily find in a textbook, so we will use it to transform spherical samples into vectors.

Again, we will look for a way to find a sampling strategy with geometric reasoning before looking at the formal method.

Deriving Integration Over Hemisphere

- Each direction ω represents an infinitesimal surface area portion $d\omega$
- How do we integrate a function $f(\omega)$ with differential $d\omega$?
- Integration over points on hemisphere surface ω, w.r.t. (θ, ϕ)



We can define an infinitesimally small piece of surface area on the unit hemisphere for each direction vector omega.

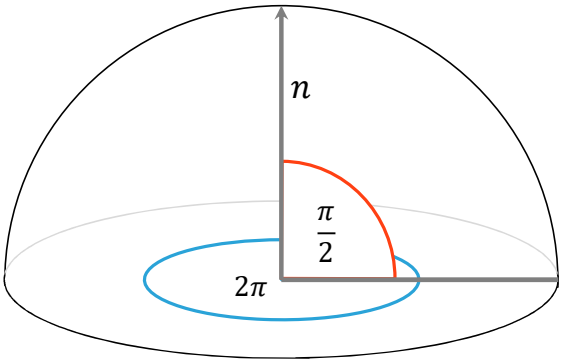
Each direction can be uniquely identified by spherical coordinates r, theta and phi. On the unit hemisphere, r = always 1, so we can ignore it for now

So let's assume we want a solution for the integral over the surface of the hemisphere, how can we write this as an integral over theta and phi?

Again, if you want to, take 5 minutes and try to come up with a solution if you care to. Remember that instead of a planar surface, we are now integrating over the curved surface of the hemisphere.

Deriving Integration Over Hemisphere

- We assume a planar surface with an upright facing normal n
- We use the integral intervals $\theta \in \left[0, \frac{\pi}{2}\right), \phi \in [0, 2\pi)$
- I.e., a curve from perpendicular to parallel for θ, a ring for ϕ



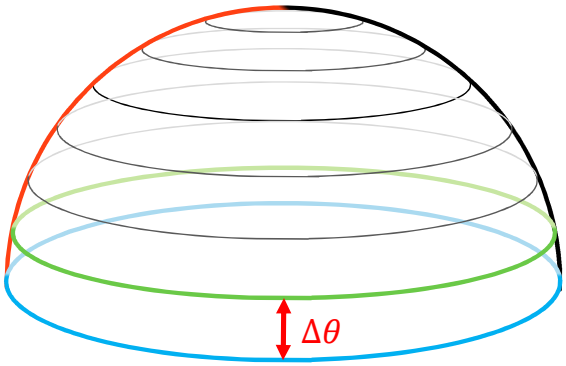
First let's define our setup. We have a planar surface that the point in the scene and the hemisphere rests on.

The surface has a normal, and the hemisphere is oriented around it. So the range of theta, from normal to parallel, covers exactly pi over 2 radians

Since phi covers a full circle, its range is from 0 to 2pi.

Deriving Integration Over Hemisphere

- We can split the surface along θ into ribbons of width $\Delta\theta \rightarrow d\theta$
- The upper edge of the ribbon is slightly shorter than the lower
- If we keep adding more and more ribbons, this difference vanishes



Again, we will try to disassemble the curved surface of the hemisphere into smaller parts that we can easily analyze, and then put them back together again.

Let's assume we split the surface of the hemisphere along theta into ribbons of width delta theta.

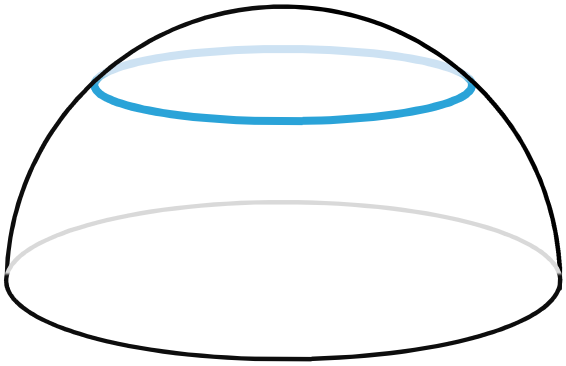
We keep splitting the ribbons, and therefore delta theta gets smaller and smaller.

In the beginning,, the upper edge of each ribbon will be slightly shorter than the lower one, but the more we split, the smaller this difference becomes.

If we split the ribbons infinitely often and delta theta goes towards an infinitesimally small number, this difference goes away completely

Deriving Integration Over Hemisphere

- As a ribbon's width goes to $d\theta$, its area becomes its length times $d\theta$
- We can find this length by projecting the ribbon to the ground
- Using trigonometry, we find the length of a ribbon is $2\pi\sin\theta$

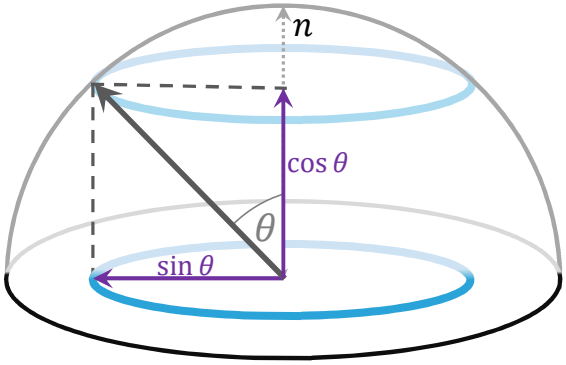


So after infinitely many splits, the area of each ribbon is equal to its length around the surface, times an infinitesimal delta theta.

We can find the length of each ribbon by projecting it down to the ground.

Deriving Integration Over Hemisphere

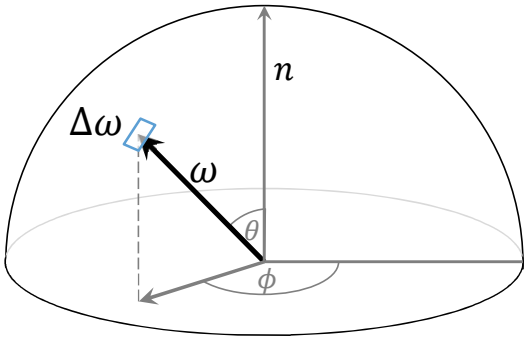
- As a ribbon's width goes to $d\theta$, its area becomes its length times $d\theta$
- We can find this length by projecting the ribbon to the ground
- Using basic trigonometry, we find the length of a ribbon is $2\pi\sin\theta$



Using some basic trigonometry rules, we ca then find the length of the ribbon as the circumference of a circle with radius of sine theta.

Deriving Integration Over Hemisphere

- The length of a ribbon spans the entire interval $\phi \in [0, 2\pi)$
- Convert the length to an integral over $d\phi$: $2\pi \sin\theta = \int_0^{2\pi} \sin\theta \, d\phi$
- The final integral: $\int_\Omega f(\omega) \, d\omega = \int_0^{\frac{\pi}{2}} \int_0^{2\pi} f(\omega) \sin\theta \, d\phi \, d\theta$



The length of the ribbon covers the entire domain of phi, all the way around the hemisphere's normal.

Also we don't have a reason to expect that the surface area of the hemisphere changes between different angles of phi, so we can assume that the integral over phi is uniform.

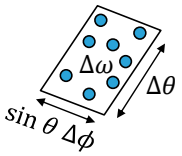
So if we want to write the found length of the ribbon as the integral over phi, we can simple make it range from 0 to 2pi.

That cancels out the 2pi in the length of the ribbon length, so all that we are left with is sin phi.

And with that, we already have the final integral over the hemisphere w.r.t. theta and phi.

Deriving PDF for Hemisphere Sampling

- Integral of $f(\omega)$ over area $\Delta\omega = \int_{\Delta\omega} f(\omega)\, d\omega$



- Integral of $f(\omega)$ w.r.t. $(d\theta, d\phi) = \int_{\Delta\theta}\int_{\Delta\phi} f(\omega)\sin\theta\, d\phi\, d\theta$
- Integration domain and $f(\omega)$ are identical, thus: $d\omega = \sin\theta\, d\phi\, d\theta$
- $\omega \leftrightarrow (\theta, \phi)$ is bijective, we have $p(\theta, \phi)\, \mathrm{d}\theta\, d\phi = p(\omega)\, d\omega$ and:

$$p(\theta, \phi) = \sin\theta\, p(\omega)$$

Now we have derived the integral over the hemisphere for theta and phi, but actually it is also valid for any arbitrary small region on the surface of the hemisphere.

Let's look at an area on hemisphere surface, delta omega that we want to integrate over. We compare an integral over directions with the integral w.r.t. theta and phi that we just derived.

Since we can always transform theta and phi into an omega, it is ok to integrate a function depending on omega in both versions. Now if we compare the two, we see that integration domain and f of omega are the same. The result of both integrals must also be the same, so that leaves us with the conclusion that d omega is equal to sine of theta d phi d theta

We already know that if the transformation of a variable is bijective, the result of that transformation is again a variable and their CDFs must be equal. That means that their PDFs multiplied by their differentials must also be equal. So we can combine the equalities on this slide to get the relative difference between PDFs in spherical coordinates and over directions omega.

Concretely, we find that if we want a particular PDF for our directions on the unit hemisphere, we must multiply by sine of theta to get the corresponding PDF for sampling in spherical coordinates.

You can see that this is not as intuitive as perhaps our solution for the unit disk was. So it might have taken you quite a while to arrive at this on your own.

Luckily, we have our formal way of doing it, and this is now much shorter.

Deriving PDF for Hemisphere Sampling, the Formal Way

- Target distribution in ω, which is (x, y, z) with $\sqrt{x^2 + y^2 + z^2} = 1$
- The transformation T from (r, θ, ϕ) to (x, y, z):

$$x = r \sin\theta \cos\phi$$
$$y = r \sin\theta \sin\phi$$
$$z = r \cos\theta$$

- The Jacobian of the transformation T gives $|J_T| = r^2 \sin\theta$
- $r = 1$, so we have $p(1, \theta, \phi) = \sin\theta \; p(x, y, z) = \sin\theta \; p(\omega)$

We use the transformation from spherical coordinates to points on the unit hemisphere. We have

X = r times sine of theta sine of phi

Y = r times sine of theta cosine of phi

Z = r times cosine of theta

If we compute the Jacobian of this transformation, we get the factor r^2 times sine of theta

So we know that if we want a particular PDF for the distribution of points on the hemisphere, we must multiply this distribution by r squared times sine of theta for sampling in spherical coordinates.

Notice that on the unit hemisphere, r is 1, so it has no effect and we can ignore it moving forward.

And that's it.

Uniformly Sampling the Unit Hemisphere

- The domain, i.e., the unit hemisphere surface area, is 2π.
  Uniformly sampling the domain over ω implies $p(\omega) = \frac{1}{2\pi}$
- Hence, since $p(1, \theta, \phi) = \sin\theta \; p(\omega)$, we want $p(\theta, \phi) = \frac{\sin\theta}{2\pi}$
- Marginal density $p_\Theta(\theta)$: $\int_0^{2\pi} p(\theta, \phi) \, d\phi = \sin\theta$
- Conditional density $p(\phi|\theta)$: $\frac{p(\theta,\phi)}{p_\Theta(\theta)} = \frac{1}{2\pi}$

The rest is now strictly by the book. We first find the volume of the domain, and we know the surface area of a unit hemisphere is 2pi.

So that means a uniform distribution over points on the hemisphere would need constant density 1 over 2 pi

We just saw how to convert a PDF for points on the hemisphere into a PDF for sampling in spherical coordinates, so we convert the PDF by multiplying with sine of theta.

Then we find the marginal density for theta by integrating out phi.

And we find the conditional density of phi given theta.

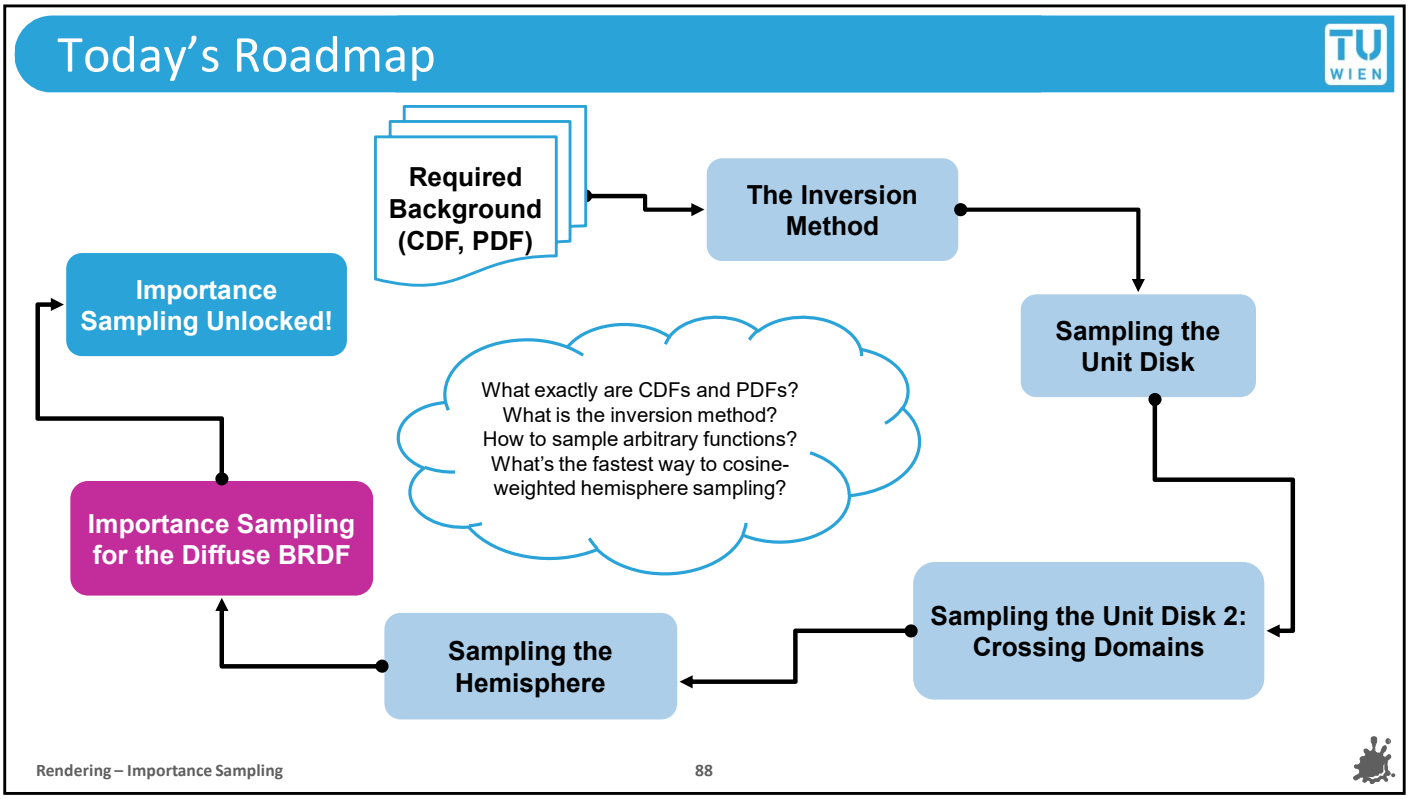
Uniformly Sampling the Unit Hemisphere – Complete

- Antiderivative of $p_\Theta(\theta)$: $\int \sin\theta \, d\theta = 1 - \cos\theta$ (added constant 1)
- Antiderivative of $p(\phi|\theta)$: $\int \frac{1}{2\pi} \, d\phi = \frac{\phi}{2\pi}$
- Invert them to get $\theta = \cos^{-1}\xi_1$ (cos is symmetric), $\phi = 2\pi\xi_2$
- Apply transformation T on (θ, ϕ) to obtain uniformly distributed ω
- Finally done!

With these two solutions for marginal and conditional density, we find their antiderivatives. To get a valid CDF for theta, we use a little trick, we add 1. Remember that this is allowed for antiderivatives.

We can then invert these CDFs and get a sampling strategy for uniformly sampling the surface of the hemisphere with canonical random variables as inputs.

And because we will be using cartesian coordinates for our rendering routine, so basically XYZ directions for light and view rays, we transform them from spherical coordinates to points on the unit hemisphere in XY and Z.



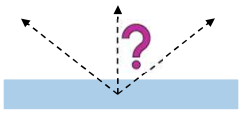
Hopefully you saw, how much quicker the formal way for finding a uniform sampling strategy on the hemisphere was, compared to deriving it by hand and with intuition. This is what today's agenda was all about, getting comfortable with a method that allows you to generate samples with custom distributions in a non-trivial domain. And in the future, if you ever go beyond the hemisphere, you will be able to apply this technique by following the steps, without having to derive it for every individual case. But for now, let's use them what we came here for: importance sampling for path tracing.

Importance Sampling the Diffuse BRDF

- Let's look once more at the reflected light in the rendering equation

$$\int_{\Omega} \underbrace{f_r(x, \omega \rightarrow v) L(x \leftarrow \omega) \cos(\theta_\omega)}_{f(x)} \, d\omega$$

- When we bounce at a point x, we already know quite a bit:
  - If we use a diffuse BRDF, then $f(x, \omega \rightarrow v)$ is a constant factor $\frac{\rho}{\pi}$
  - We can predict the cosine term—it depends on our choice of ω
  - The tricky part, the big unknown, is the $L(x \leftarrow \omega)$
  - Which directions will indirect light come from?



Like we did so many times already, let's look at the reflection part of the rendering equation.

Once again, we will restrict ourselves to diffuse materials for now.

That way, when we are at a hit point and generate a sample for integrating the function for reflected light f(x) from all directions of the hemisphere, we know exactly what terms we are dealing with. We have th BRDF, which is a simple constant factor for the diffuse materials we looked at so far.

We have the cosine which is easy enough to predict, because it depends on our omega, but the tricky part is the incoming light itself, because as we know, that is an infinitely recursive function. And that is what makes it so difficult, because this infinite recursive function is dependent on every single shape and every single material in our scene, and how they all are arranged

relative to each other… clearly we can‘t make an exact guess what f(x) looks like.

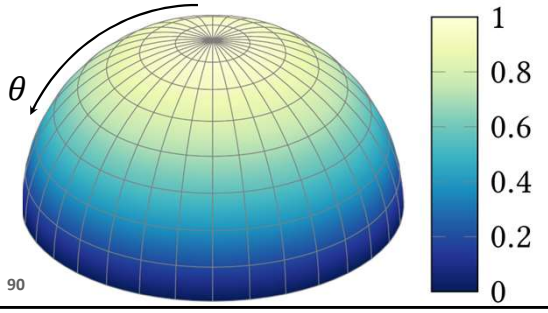
Importance Sampling the Diffuse BRDF

- If we don‘t know anything about L, let‘s just assume a constant k

$$\int_{\Omega} \frac{\rho}{\pi} k \cos(\theta_\omega)\, d\omega$$

- $\frac{\rho}{\pi}$ and k are constant, so clearly, $\frac{\rho}{\pi} k \cos(\theta_\omega) \propto \cos(\theta_\omega)$

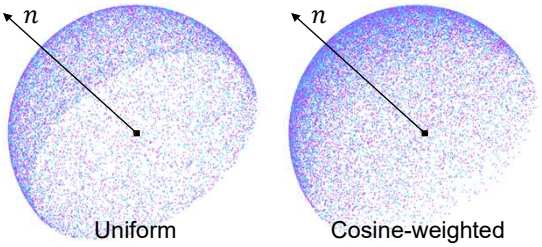
- With these assumptions, the integrand function is governed entirely by the term $\boldsymbol{cos(\theta)}$!



But if we have no idea what it looks like, that tells us something. It tells us that, without additional information, light has an equal chance to come from all directions! So let‘s just assume a constant factor k that represents uniform incoming light from all directions. With that setting, where we assume constant incoming light and a diffuse BRDF, the shape of the resulting integrand is only dependent on the cosine of theta!

Importance Sampling the Diffuse BRDF

- We know that the ideal distribution $p(x)$ for importance sampling a function $f(x)$ is the one that minimizes variance, i.e., $\propto f(x)$ itself
- With the assumption of constant light from all directions, our integrand $f(x)$ was simplified to something proportional to $\cos(\theta)$
- Idea: Importance-sample hemispheres around hit points with diffuse materials with distribution $p(\omega) \propto \cos(\theta_\omega)$



Uniform hemisphere sampling

Cosine-weighted hemisphere sampling

This brings us actually to the primary method of importance sampling the hemisphere for diffuse materials. For importance sampling, we want our distribution to mimic the shape of the actual function we are integrating, and if that shape is only influenced by the cosine, then what we need is a sampling distribution that follows the cosine of theta over the hemisphere. The cosine is, of course, highest when the input is zero, which in our case means, more samples would be distributed close to the apex of the hemisphere, or in other words, they would focus around the surface normal. On the bottom right, we see two different sample distributions on the hemisphere, one with uniform distribution, the other with cosine. Note how with cosine-weighted sampling, they are much denser around normal, and how they fade out towards the edge of the hemisphere!

Cosine-Weighted Hemisphere Sampling?

- In the first half, we saw how you can apply the inversion method for sampling arbitrary distributions
- In the second half, we were all about making sure that we can reach our target distribution when we move from one domain to another
- Cosine-weighted hemisphere sampling is a combination of the two
- We have gone through all the necessary steps.
  Try to solve this formally with the inversion method as an exercise!

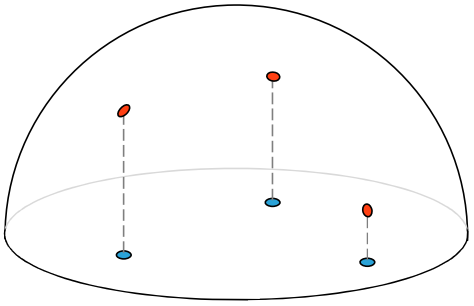
In the first half of the lecture, we learned about the inversion method to make non-uniform distributions in simple domains. In the second half, we learned about how we can achieve a desired distribution in a non-trivial target domain. Now it's time to combine the two.

We will actually not derive the exact steps. We think that if you paid close attention and were eager to learn something new, you should have everything you need to do this yourself, although you may want to go back and look at all the previous steps a little longer before trying this in practice. But all that is required here is, create a PDF candidate that is based on the cosine of omega, and then make sure that you can achieve this distribution on the surface of the hemisphere when you actually draw your random inputs in spherical coordinates.

Now we actually promised you a shortcut to importance sampling for diffuse materials, and we haven’t forgotten about it. We believe that doing this yourself might give you a significant heureka moment and help you understand one of the more challenging aspects of rendering and sampling.

Cosine-Weighted Hemisphere Sampling!

- Malley's method: uniformly pick (x, y) samples on the unit disk
- Project them to the hemisphere surface $\left(z = \sqrt{1 - x^2 - y^2}\right)$
- $p(\omega) = \frac{\cos}{\pi}$
- Done! Your samples are now distributed with $p(\omega) \propto \cos\theta$!
  (Why? And why does this work? Try to find your own proof!)



But if you don't want to spend energy on that, there is a quick solution, called Malley's method.

Malley's method gives us exactly what we want: cosine-weighted samples on the unit hemisphere, and it's extremely easy.

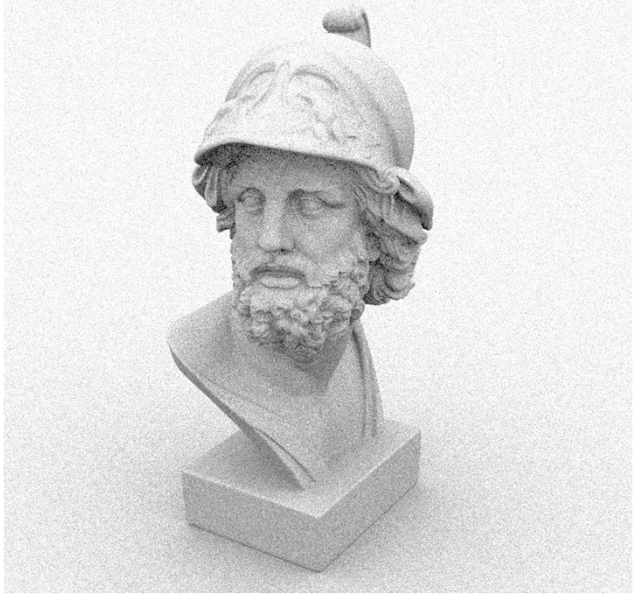
All we have to do is draw uniform samples on the unit disk (we know how to do that already) and then project them to the surface of the hemisphere.

Done! The resulting sample distribution over the surface will be weighted with the cosine of theta.

Why does this work? Again, we encourage you to invest some time in solving this yourself as an exercise.

Importance Sampling the Diffuse BRDF

AO, 64 samples, uniform hemisphere sampling

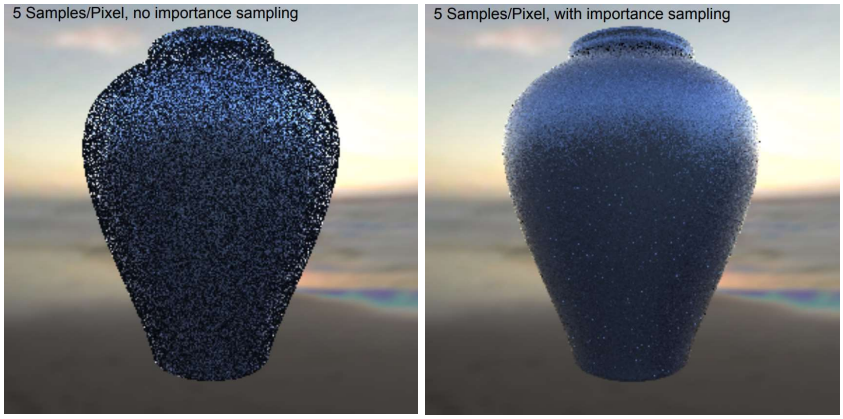


AO, 64 samples, diffuse BRDF importance sampling

And the results do pay off. Here we see two renderings of the Ajax bust, rendered with the same number of samples. However, the one on the left used uniform hemisphere sampling, the one on the right used importance sampling. Clearly, we have cleaned up a lot of noise by changing only a few lines of code.

More Importance Sampling

- The impact will be much greater when we add non-diffuse materials
- BRDF functions can be rather complex…
- …but can often be nicely approximated



- You will want to sample with distributions more complex than $\cos\theta$

And later on, we will see that the effect is not only much more pronounced in other materials, it is actually REQUIRED for some of them to function properly.

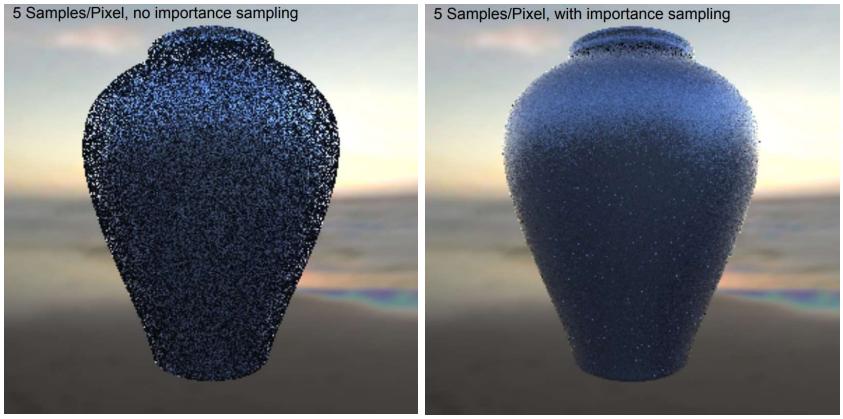
For some of the more complex materials, the importance sampling strategy will not be as straightforward as the one for diffuse materials, and sometimes we won‘t be able to make exact distributions for given BRDFs at all. But in most cases, you will still be able to find good approximations, and if you do that, you will be glad that you have the inversion method at your disposal that we discussed today.

More Importance Sampling

- Consider the modified Beckmann distribution for microfacet BRDFs

- $D(\theta, \phi) = \frac{e^{\frac{-\tan^2 \theta}{\alpha^2}}}{\pi \alpha^2 \cos^3 \theta}$

- Yes, seriously!



- Good luck with intuitive reasoning! Challenging, but doable task with basic trigonometric identities and the inversion method!

Take for instance the modified Beckmann distribution, which is commonly used for simulating specular reflection in microfacet models, which we will discuss later in the lecture.

Here, we have given you the distribution function parameterized by theta and phi. Yes, that's the real formula to compute the probability density of a single sample, and you can imagine that making samples with this distribution is not something that you can easily come up with by using your intuition alone. However, it is very possible to use the inversion method and a few helpful trigonometric identities to come up with the sample generation method yourself with a pen and a piece of paper. For those who are looking for a challenge, this is the task for you.

Importance Sampling the Full Rendering Equation

- As you can imagine, this is a much more complex task
- In fact, an enormous amount of research in rendering is actively pursuing better and better ways to make this happen
- Other sophisticated methods, like multiple importance sampling (MIS), can be of great help here!
- We will hear more about MIS in upcoming lectures…

So how far can importance sampling go? And will we someday learn about a method to actually importance sample the FULL rendering equation?

This is actually an incredibly important topic because, as you can imagine, the better your sampling strategy is, the more viable path tracing becomes, the faster we get nice-looking images with a low number of samples. So accurately importance sampling the renderin equation is kindof the holy grail of path tracing, and an enormous amount of research is invested in this. There are many methods use heuristics and statistics that can get us a lot closer to an optimal sampling behavior, like multiple importance sampling, but there is still a lot of room for improvement. Next time, we will actually hear about multiple importance sampling and see how it will clean up our renderings even more.

Importance Sampling Summary

- If we do Monte Carlo integration of $f(x)$, it's best to use a sample distribution $p(x)$ that closely mimics $f(x)$
- For a desired $p(x) \propto f(x)$, we can use the **inversion method** to get the methods for generating samples and probability densities
- If you cannot turn $f(x)$ into a valid PDF, try to find a close match
- When we transform samples between domains, we have to make sure they have the desired distribution in the target domain!

So to summarize, beyond the mathematical knowledge, here is what you should take with you after today's lecture on importance sampling:

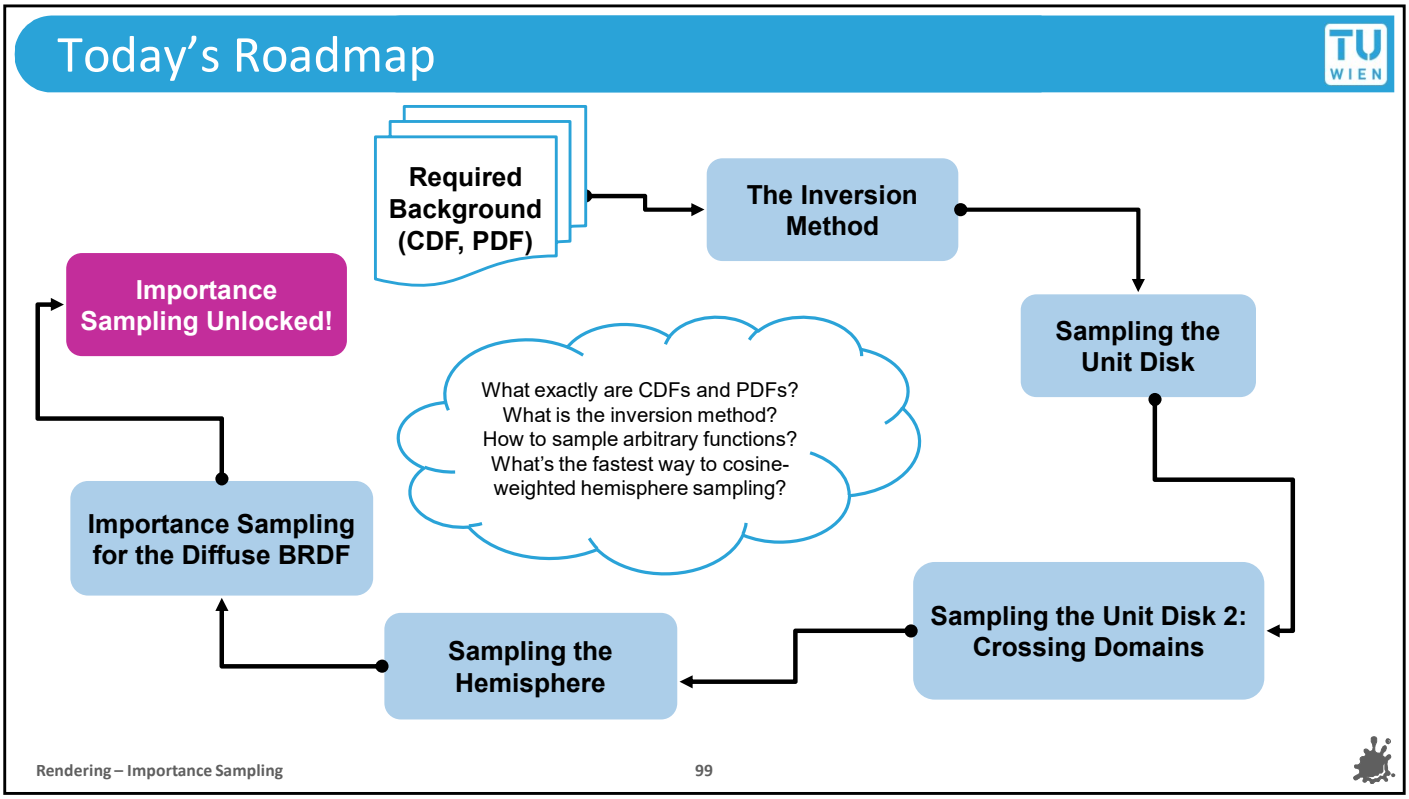
Whenever we do Monte Carlo integration, ist best to find distributions for generating samples that correspond exactly to the target function f(x).

If we have an f(x) that we want our PDF to mimic, we can use the inversion method to come up with methods for generating samples and probability densities with the corresponding distribution, that can often be easily put into code right away.

However, if we have an idea about a good distribution shape but cannot turn it into a PDF, a good idea is to

find as close an approximation as possible, and often this is the only way because usually might not even know the exact target function.

Finally, if we have to integrate over non-trivial domains, we can use bijective transformations and draw our samples in one domain and transform them to another. But when we do that, we have to account for the fact that transformations can shift the distributions of our samples and account for that when we generate the samples.



If you came this far without giving up, you have successfully unlocked importance sampling, and we recommend that you try apply your new-learned skills to figure out cosine-weighted hemisphere sampling and perhaps even implement it in your path tracer from before. But if it all seems a bit overwhelming right now, we encourage you to revisit the earlier sections or check out the links on the next slide to get a little extra input.

References and Further Reading

- Slide set based mostly on chapter 13 of *Physically Based Rendering: From Theory to Implementation*
- [1] Steven Strogatz, *Infinite Powers: How Calculus Reveals the Secrets of the Universe*
- [2] Video, *Why "probability of 0" does not mean "impossible" | Probabilities of probabilities, part 2*: https://www.youtube.com/watch?v=ZA4JkHKZM50
- [3] Video, *The determinant | Essence of linear algebra, chapter 6*: https://www.youtube.com/watch?v=Ip3X9LOh2dk
- [4] SIGGRAPH 2012 Course: *Advanced (Quasi-) Monte Carlo Methods for Image Synthesis*, https://sites.google.com/site/qmcrendering/
- [5] Wikipedia, *Volume Element*, https://en.wikipedia.org/wiki/Volume_element

As always, here are a few helpful references that might give you some extra information, especially if you lost track during the math-heavy part, these might help you out there so that you feel more comfortable with the steps we discussed.

We hope you had a good time and picked up something useful from today's lecture.

We will end it here, thank you sticking around until the end and we hope to also see you next time.