



Rendering: Monte Carlo Integration II

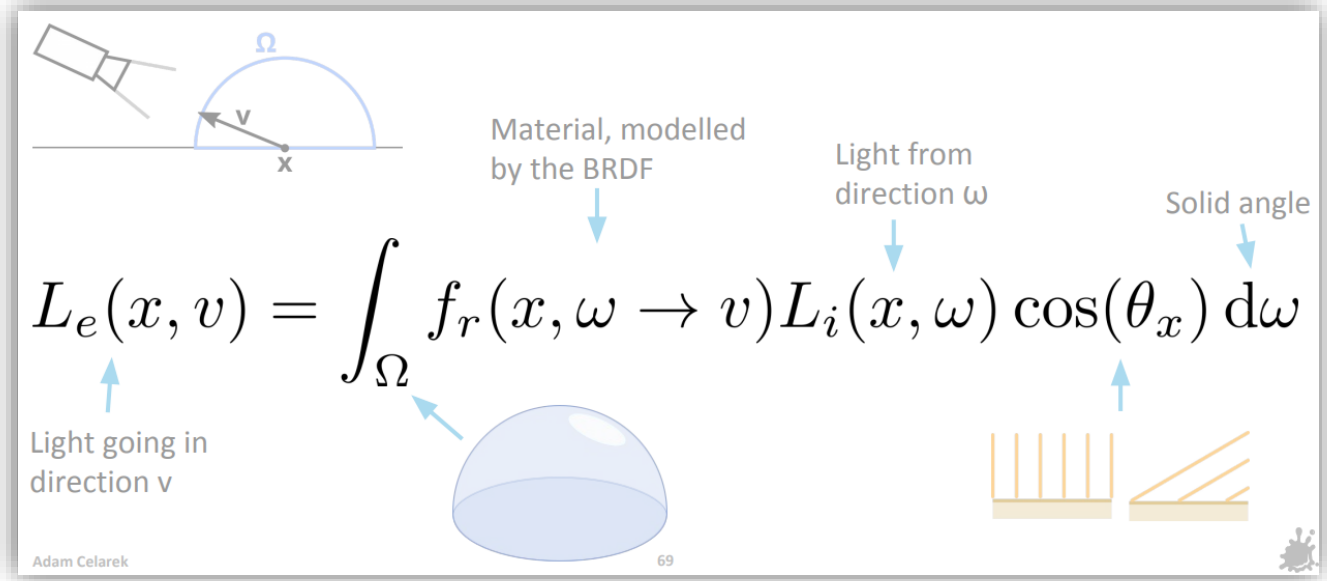
Bernhard Kerbl

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



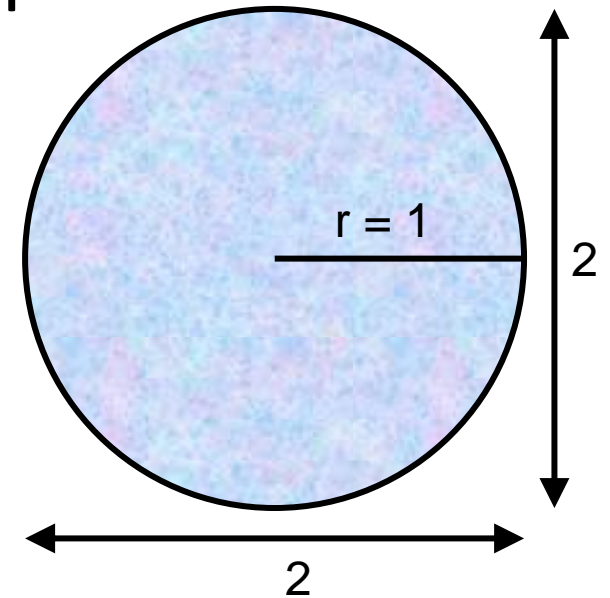
- Integrating the cosine-weighted radiance $L_i(x, \omega)$ at a point x
- Integral of the light function over the hemisphere, w.r.t. direction/solid angle at ω
- Let's find a solution!
 - How **do** we integrate over the hemisphere?
 - How do we do it **smartly**?


$$L_e(x, v) = \int_{\Omega} f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x) d\omega$$

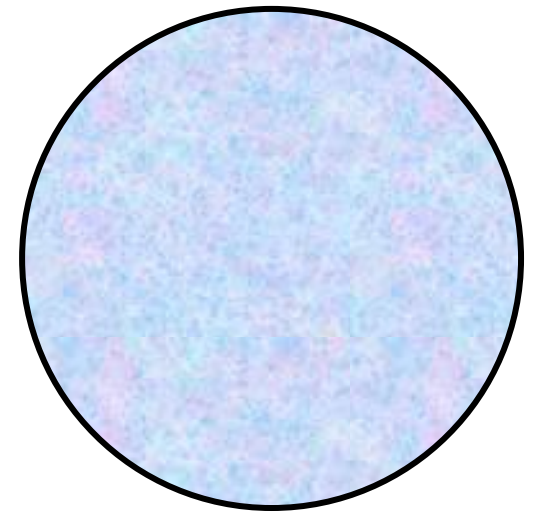
Adam Celarek 69



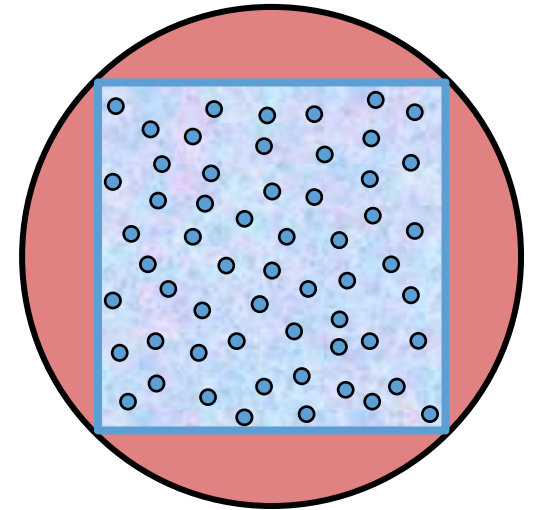
- Imagine we have a disk-shaped surface with radius $r = 1$ that registers incoming light (color) from directional light sources
- As an exercise, we want to approximate the total incoming light over the disk's surface **area**
- We integrate over an area of size π
- We will use the Monte Carlo integral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



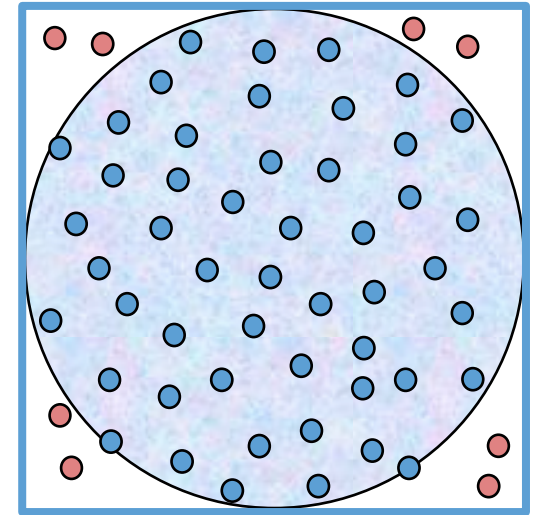
- If we can manage to uniformly sample the disk, then we can compute the Monte Carlo integral as a simple average $\times \pi$
- By drawing uniform samples in x and y , we cannot cover the area precisely
- Inscribed square: information lost
- Circumscribed square: unnecessary samples



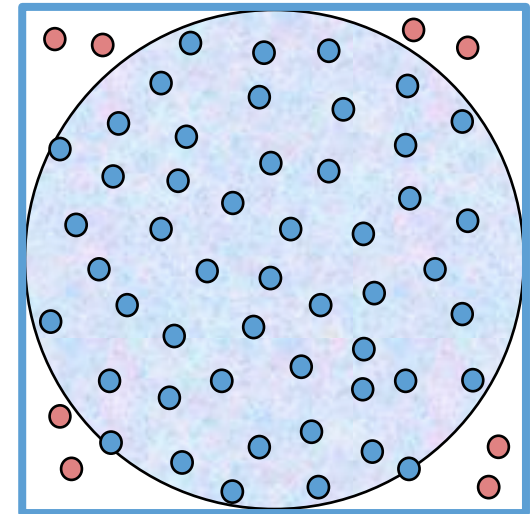
- If we can manage to uniformly sample the disk, then we can compute the Monte Carlo integral as a simple average
- By drawing uniform samples in x and y , we cannot cover the area precisely
- Inscribed square: **information lost**
- Circumscribed square: unnecessary samples



- If we 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 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
 - This is actually somewhat ok!



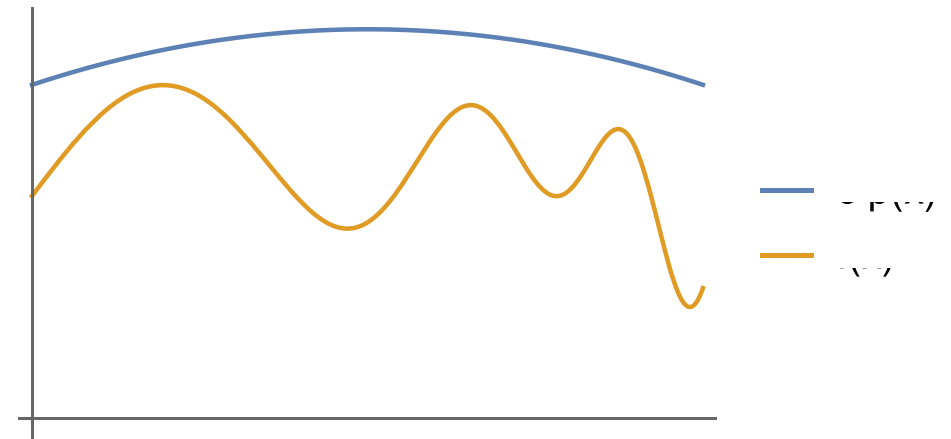
- Requires a PDF $p_X(x)$ and a constant c such that $f(x) < cp_X(x)$
- Draw ξ_i and X_i from their respective distributions. If the point $(X_i, \xi_i cp_X(X_i))$ lies under $f(x)$, then the sample is accepted

loop forever:

sample X from p_X 's distribution

if $\xi_i \cdot cp_X(X_i) < f(X_i)$ then

return X_i



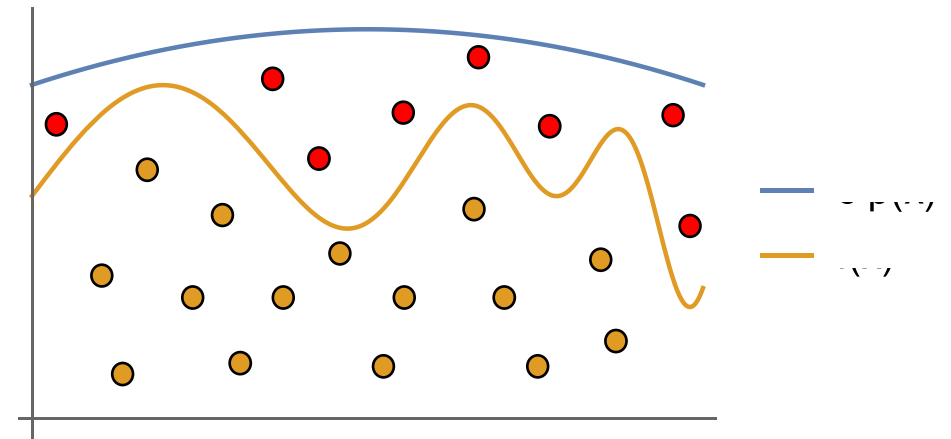
- Requires a PDF $p_X(x)$ and a constant c such that $f(x) < cp_X(x)$
- Draw ξ_i and X_i from their respective distributions. If the point $(X_i, \xi_i cp_X(X_i))$ lies under $f(x)$, then the sample is accepted

loop forever:

sample X from p_X 's distribution

if $\xi_i \cdot cp_X(X_i) < f(X_i)$ then

return X_i



- Unit disk: $f(x, y) = \begin{cases} 1 & \text{if } (\sqrt{x^2 + y^2} \leq 1) \\ 0 & \text{otherwise} \end{cases}$, $p(x, y) = \frac{1}{4}$, $c = 4$



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



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

- Convert to cartesian coordinates

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

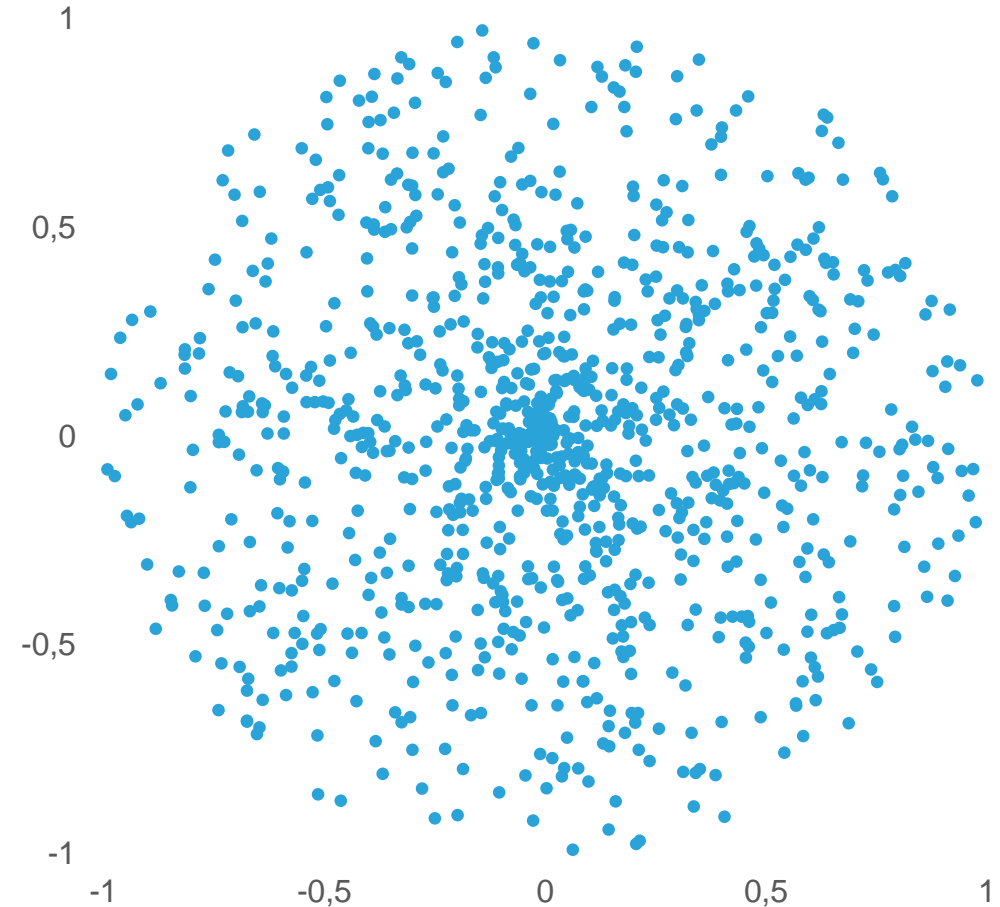
    std::uniform_real_distribution<double> uniform_dist(0.0, 1.0);

    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);
    }
}
```



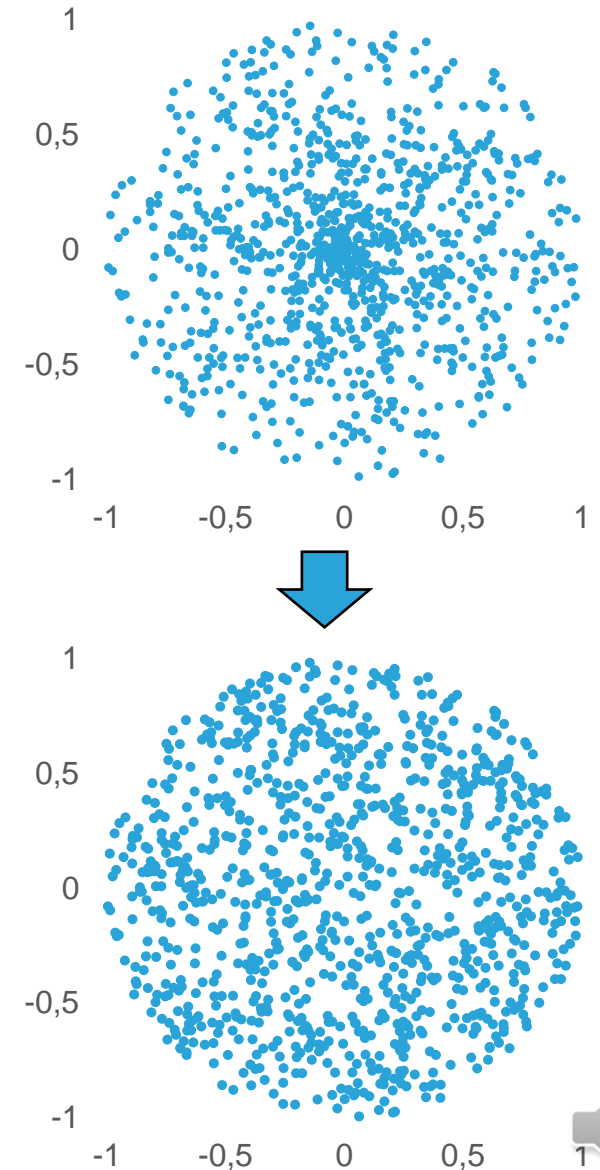
- We successfully sampled the unit disk in the proper range
- However, the distribution is not uniform with respect to the area
- Samples clump together at center
- Averaging those samples will give us a skewed result for the integral!



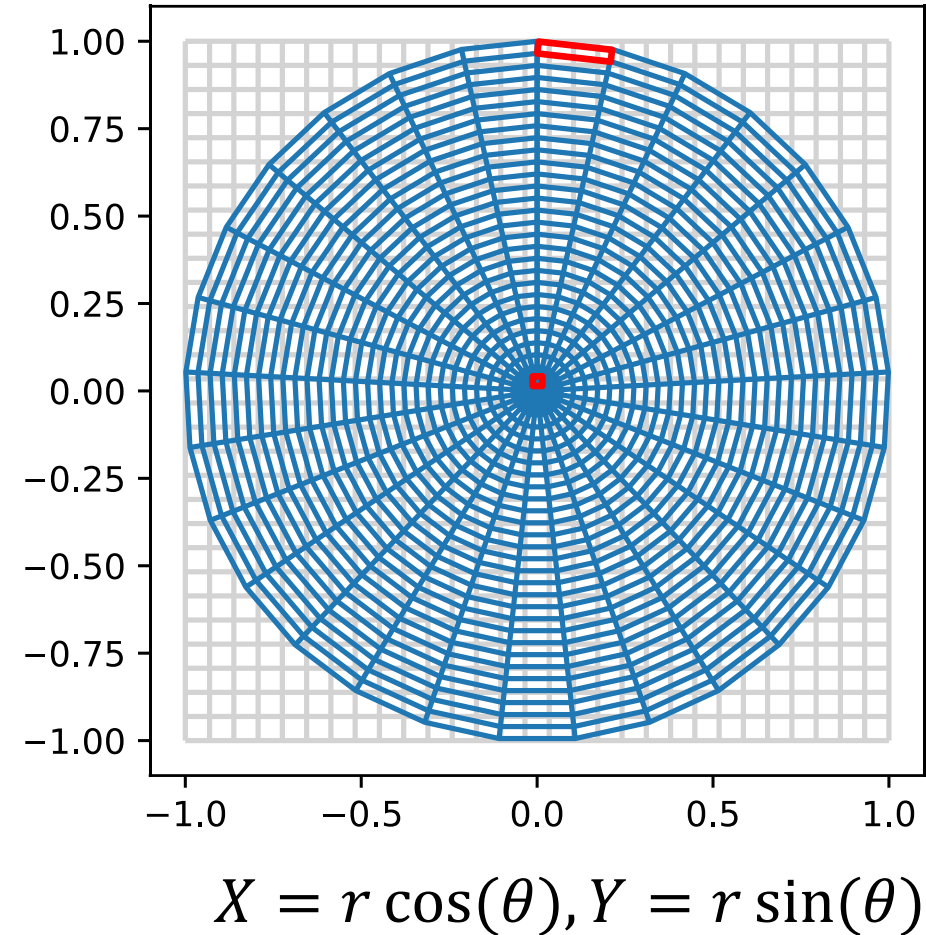
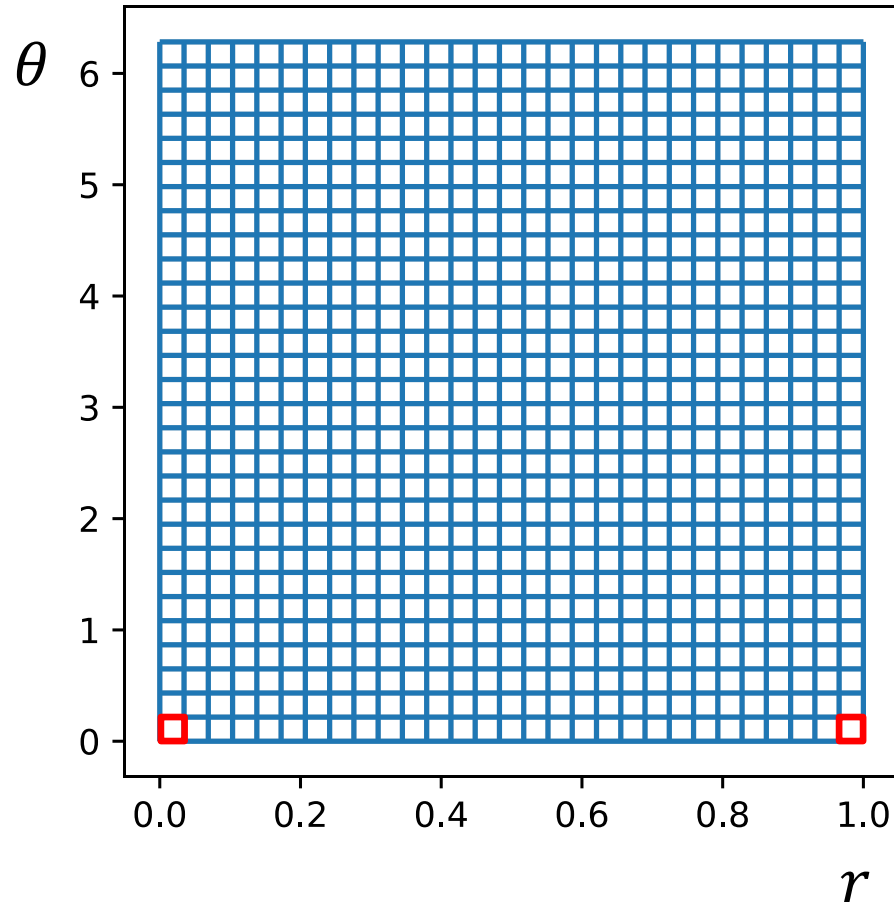
- The area of a disk is proportional to r^2 , times a constant factor π
- If we see the disk as concentric rings of width Δr , the j inner rings up to radius $r_j = j\Delta r$ should contain $\left(\frac{r_j}{r}\right)^2 N$ out of N total samples
- Conversely, the i^{th} sample should lie in the ring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



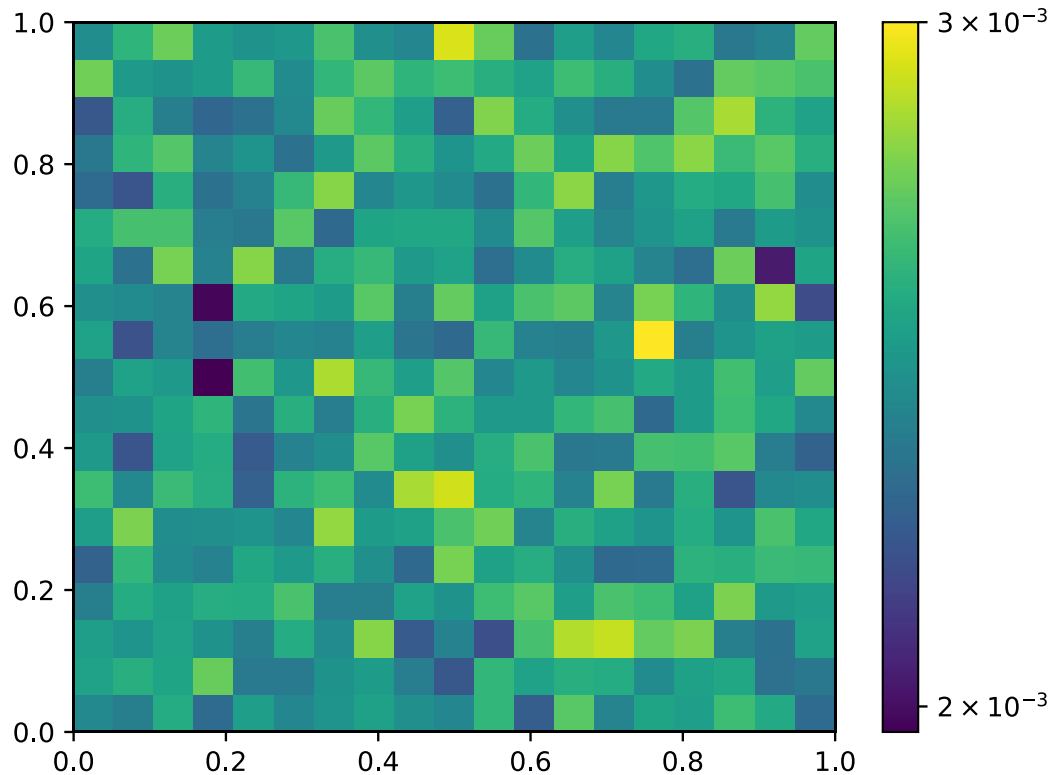
- It works, and it is not even a bad way to arrive at the correct solution
- However, for more complex scenarios, we might struggle to find the solution so easily
- With the tools we introduced earlier, we can formalize this process for arbitrary setups



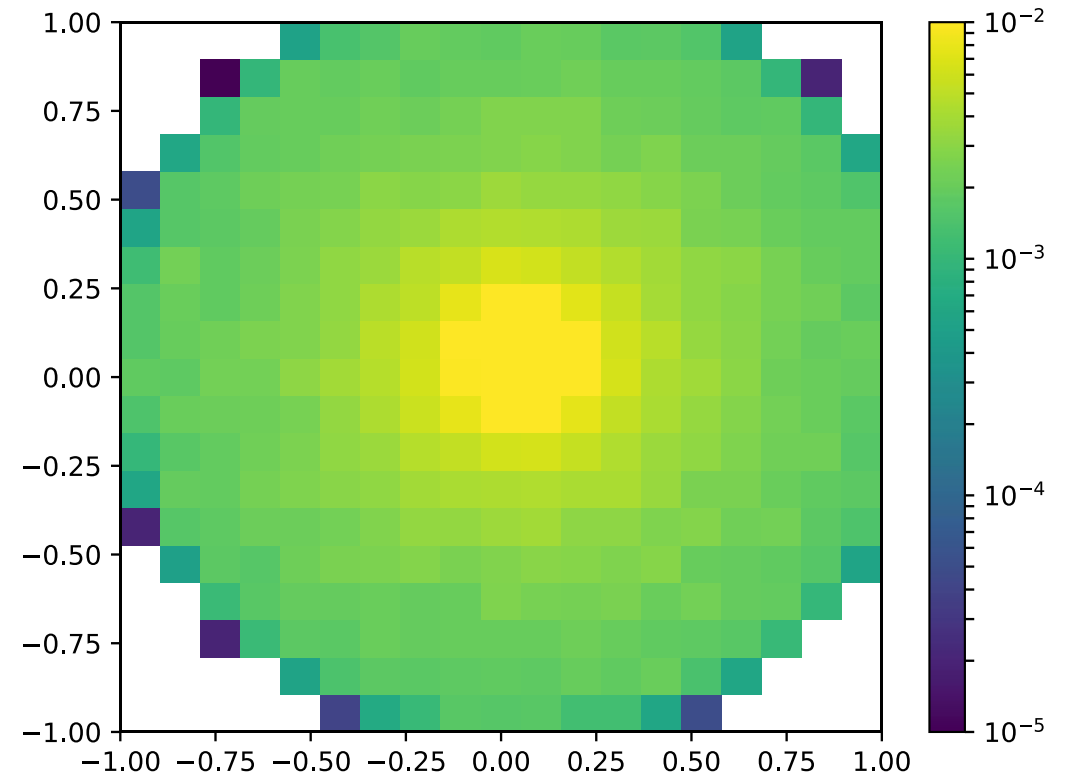
- Let's transform a regular grid from polar to cartesian coordinates



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



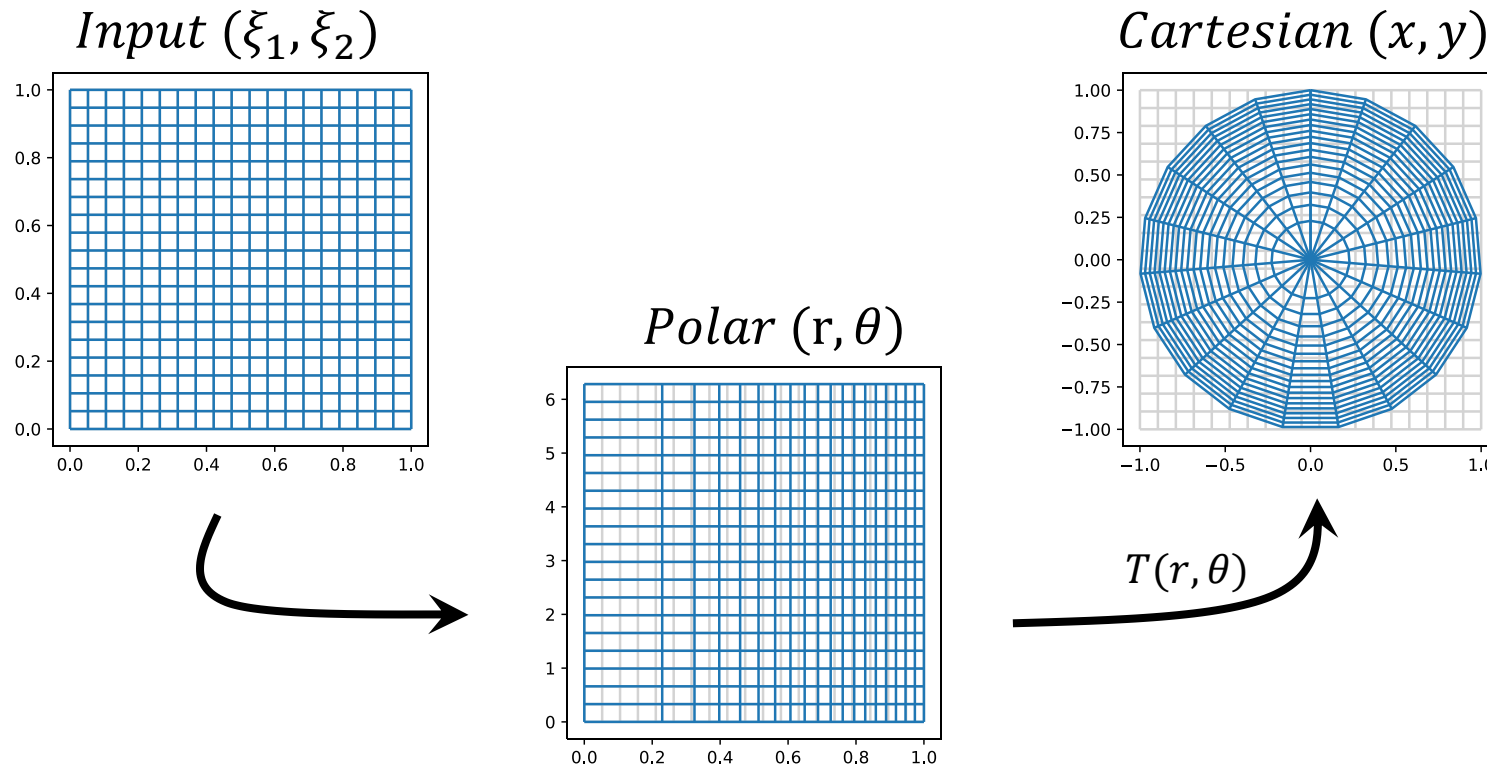
ξ_1, ξ_2



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



- 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



- Assume a random variable A and a bijective transformation T that yields another variable $B = T(A)$
- Bijectivity dictates 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 $P_B(T(a)) = P_A(a)$



- 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:

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

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



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

the change in sampling density at point a if we transform a by T
or,

the inverse change in the volume of an infinitesimal hypercube at point a if we transform a by T



- 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



- 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 $T(\vec{a})$



- 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

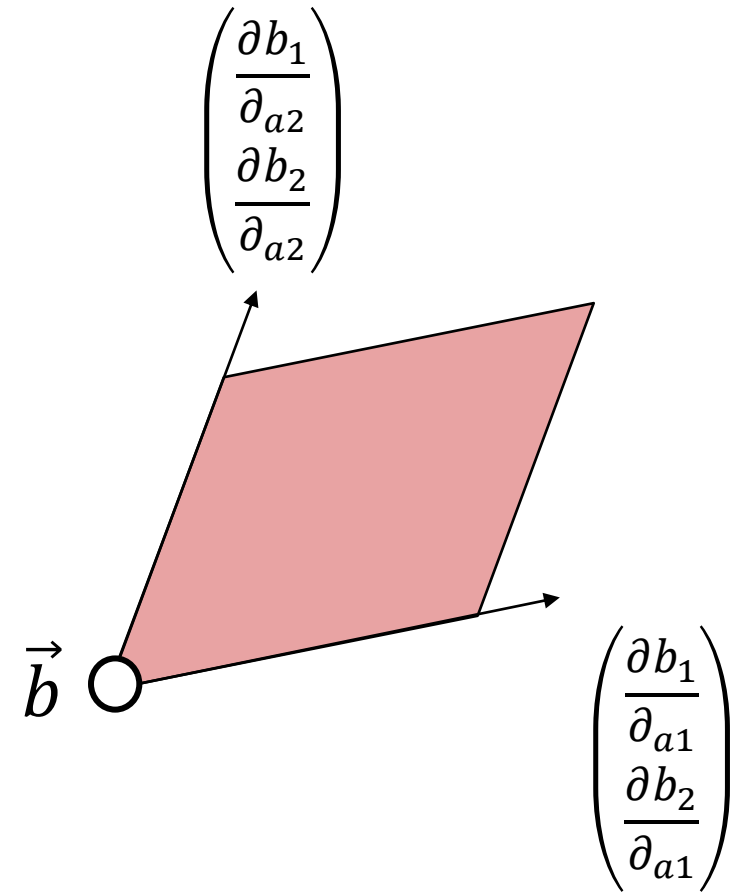
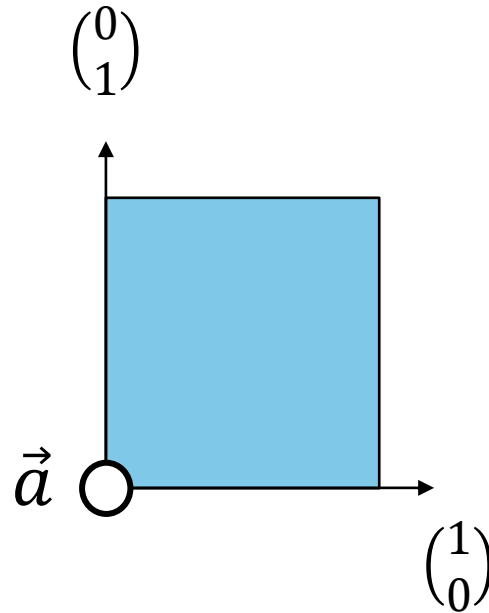
$$J_T(\vec{a}) = \begin{pmatrix} \frac{\partial b_1}{\partial a_1} & \dots & \frac{\partial b_1}{\partial a_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_M}{\partial a_1} & \dots & \frac{\partial b_M}{\partial a_N} \end{pmatrix}$$

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



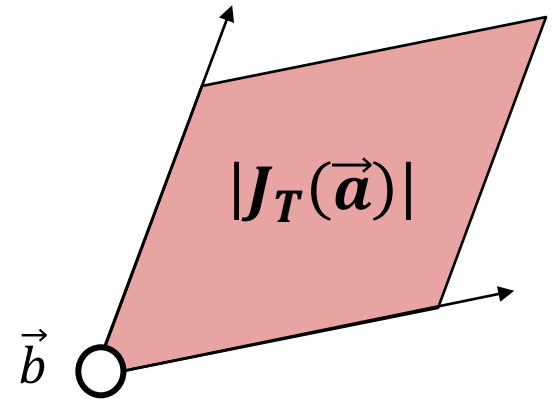
- Change of edges of an infinitesimal hypercube with extent (1,1)

$$J_T(\vec{a}) = \begin{pmatrix} \frac{\partial b_1}{\partial a_1} & \dots & \frac{\partial b_1}{\partial a_N} \\ \frac{\partial b_2}{\partial a_1} & \dots & \frac{\partial b_2}{\partial a_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_N}{\partial a_1} & \dots & \frac{\partial b_N}{\partial a_N} \end{pmatrix}$$



- The columns of a square matrix M 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 M

- The **determinant** $|M|$ of M computes the volume of a parallelepiped spanned by these vectors^[3]



- $|J_T|$, called *the Jacobian of T*, gives the volume change at \vec{a} by T



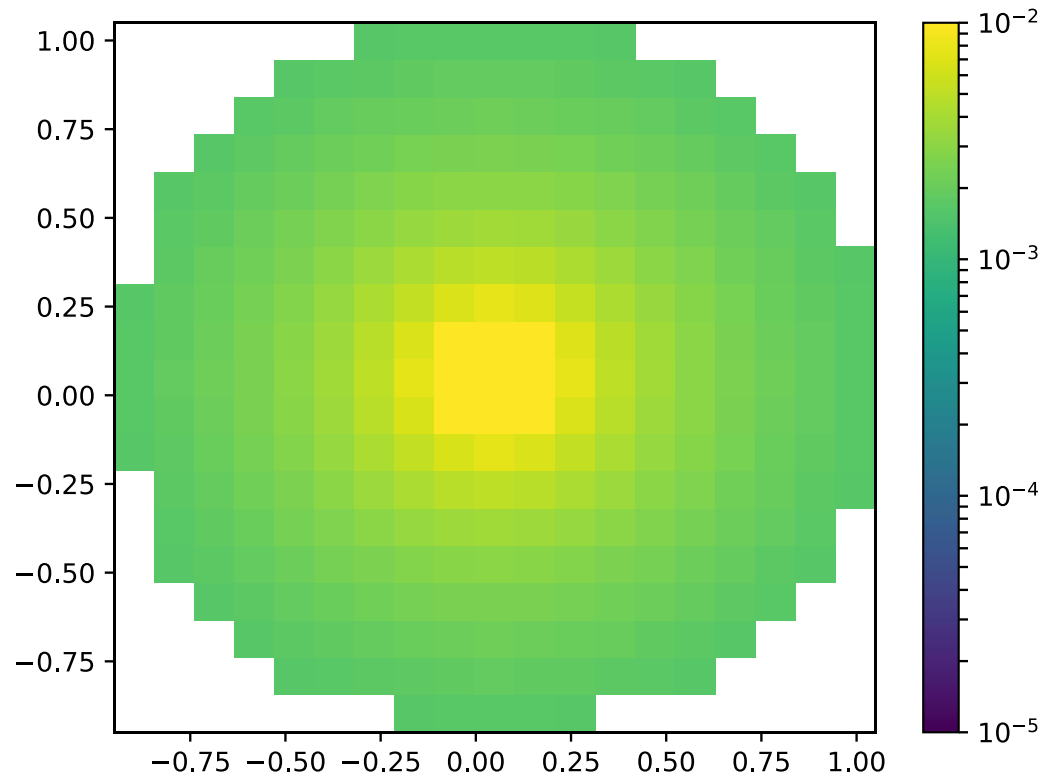
- Let's try polar coordinates again: $\begin{pmatrix} x \\ y \end{pmatrix} = T \begin{pmatrix} r \\ \theta \end{pmatrix} = \begin{pmatrix} r \sin \theta \\ r \cos \theta \end{pmatrix}$

- $\left| \frac{\partial T \begin{pmatrix} r \\ \theta \end{pmatrix}}{\partial \begin{pmatrix} r \\ \theta \end{pmatrix}} \right| = |J_T| = \left| \begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{pmatrix} \right| = \left| \begin{pmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{pmatrix} \right| = r$

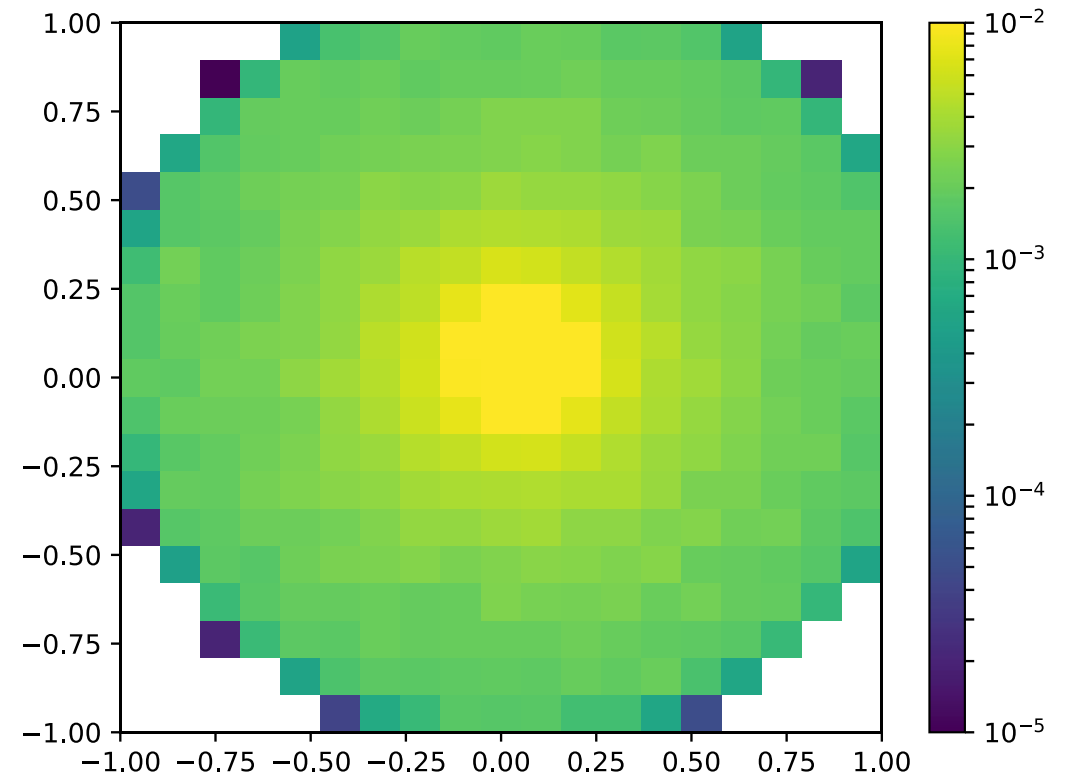
- $p(x, y) = \frac{p(r, \theta)}{r}$, or $p(r, \theta) = r p(x, y)$:
a uniform density in (x, y) must be proportional to r in (r, θ)



Compare PDFs After Transformation



Using $p(r, \theta) = \frac{1}{2\pi}$ and $p(x, y) = \frac{p(r, \theta)}{r}$



Measured



■ Python code, for the interested

```
num_bins = 20

xlist = np.linspace(-1.0, 1.0, num_bins)
ylist = np.linspace(-1.0, 1.0, num_bins)
bin_volume = 4 / (num_bins*num_bins)

X, Y = np.meshgrid(xlist, ylist)
X += 1/num_bins
Y += 1/num_bins

r = np.sqrt(X*X+Y*Y)
uniform_dist = 1/(2 * np.pi)
pdf_transform = 1/r

Z = uniform_dist * pdf_transform * bin_volume
Z[r>1] = 0
fig,ax = plt.subplots(1,1)
cp = plt.pcolor(X, Y, Z,cmap='viridis',norm=matplotlib.colors.LogNorm())
fig.colorbar(cp) # Add a colorbar to a plot
plt.show()
```



- 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



- 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 $p(y|x) = \frac{p(x,y)}{p_X(x)}$



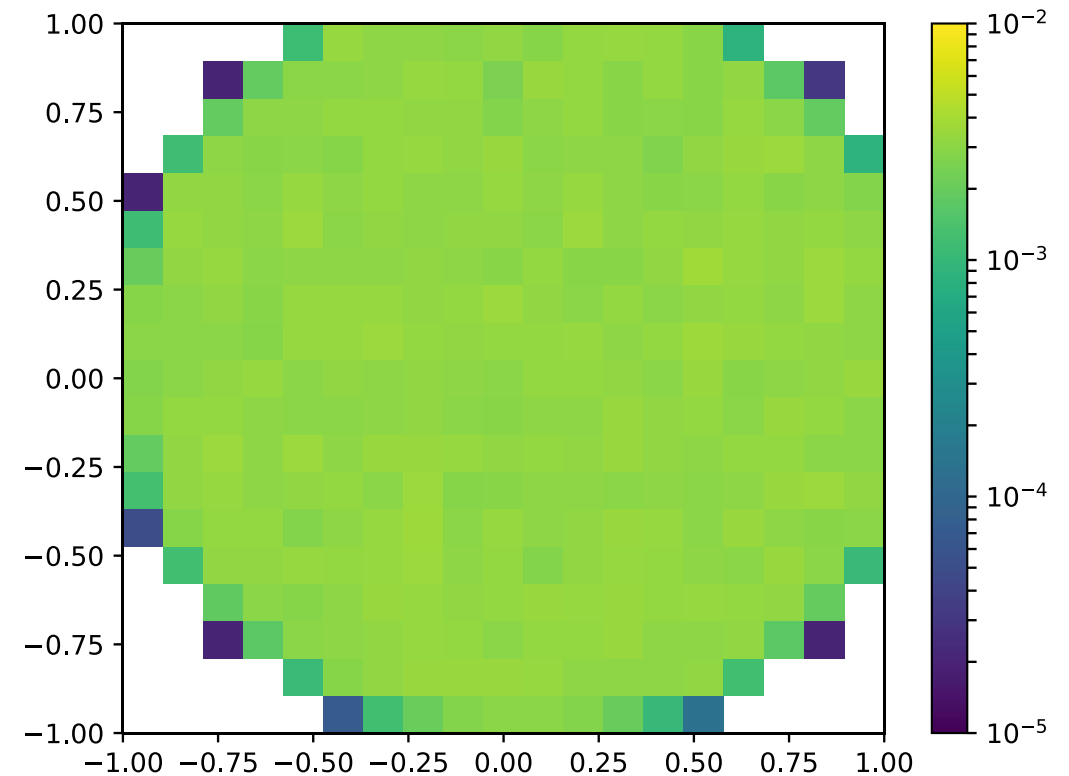
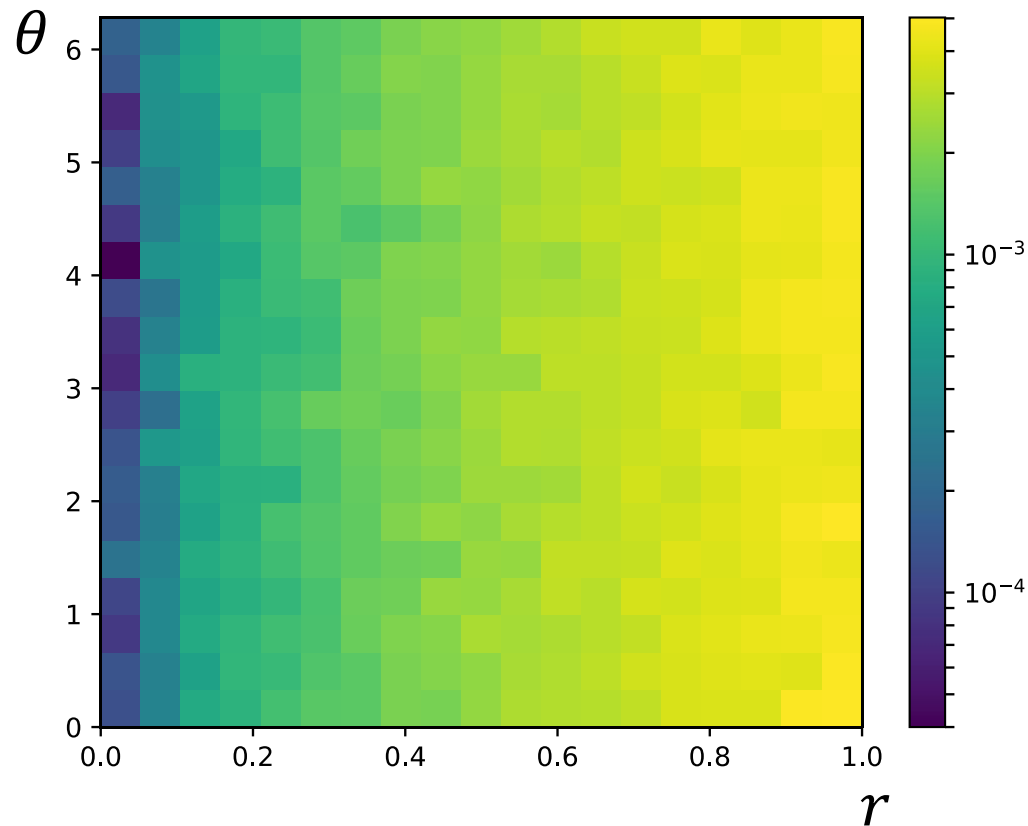
- What to do for multiple variables, e.g. X, Y and Z ?
 - Find first marginal density $p_X(x) = \int_{a_Z}^{b_Z} \int_{a_Y}^{b_Y} p(x, y, z) dy dz$
 - Find first conditional density $p_X(y, z|x) = \frac{p(x, y, z)}{p_X(x)}$
 - Find second marginal density $p_Y(y|x) = \int_{a_Z}^{b_Z} p(x, y, z) dz$
 - 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



- The size of the sampling domain in cartesian coordinates is π
- Since we want uniform sampling and sample probabilities must integrate to 1, the PDF in cartesian coordinates is $p(x, y) = \frac{1}{\pi}$
- We know 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 $p(\theta|r) = \frac{p(r, \theta)}{p_R(r)} = \frac{1}{2\pi}$



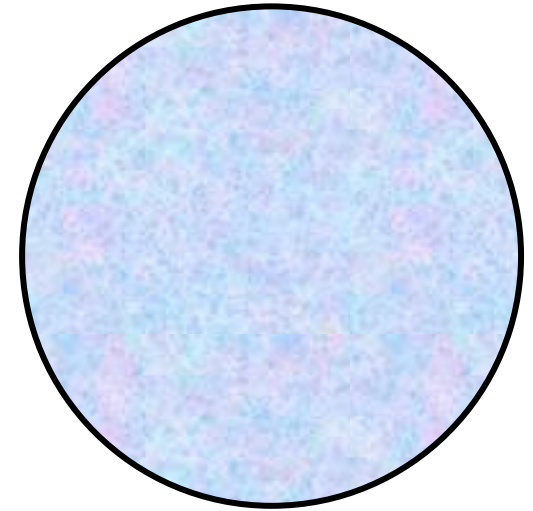
- If we draw samples for our r, θ with the above PDFs, we get uniform distribution in (x, y) after applying transformation T



- Integrate marginal and conditional PDFs and invert: we get the same solution as before:

- $r = P_R^{-1}(\xi_1) = \sqrt{\xi_1}$

- $\theta = P_{\Theta}^{-1}(\xi_2) = 2\pi\xi_2$



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

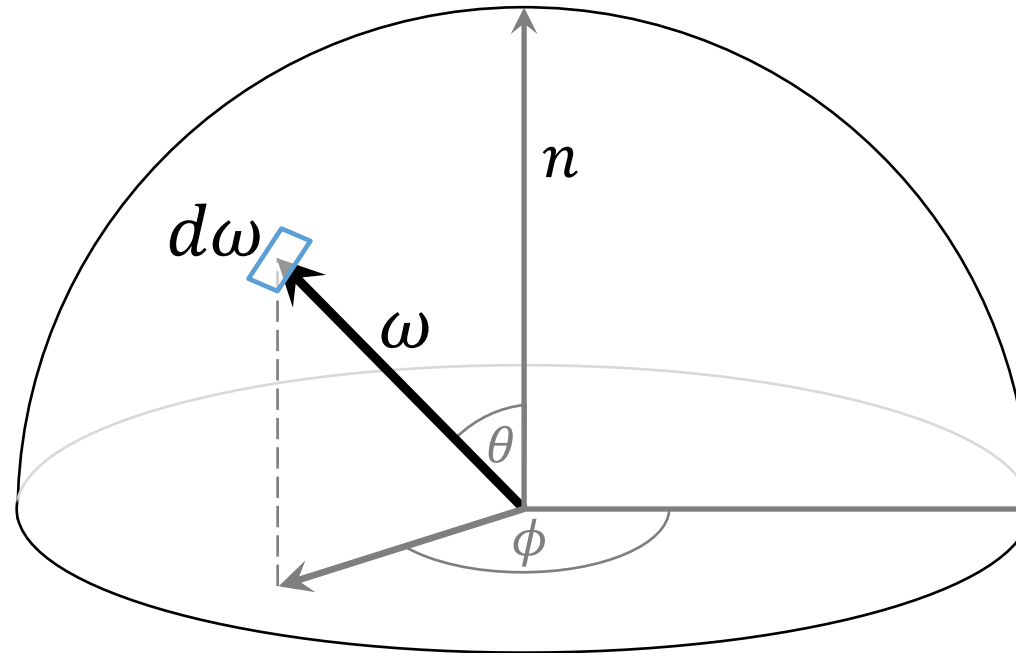
- Final integral: $RGB_{total} = \frac{\pi}{N} \sum_{i=1}^N RGB(R_i \sin \Theta_i, R_i \cos \Theta_i)$



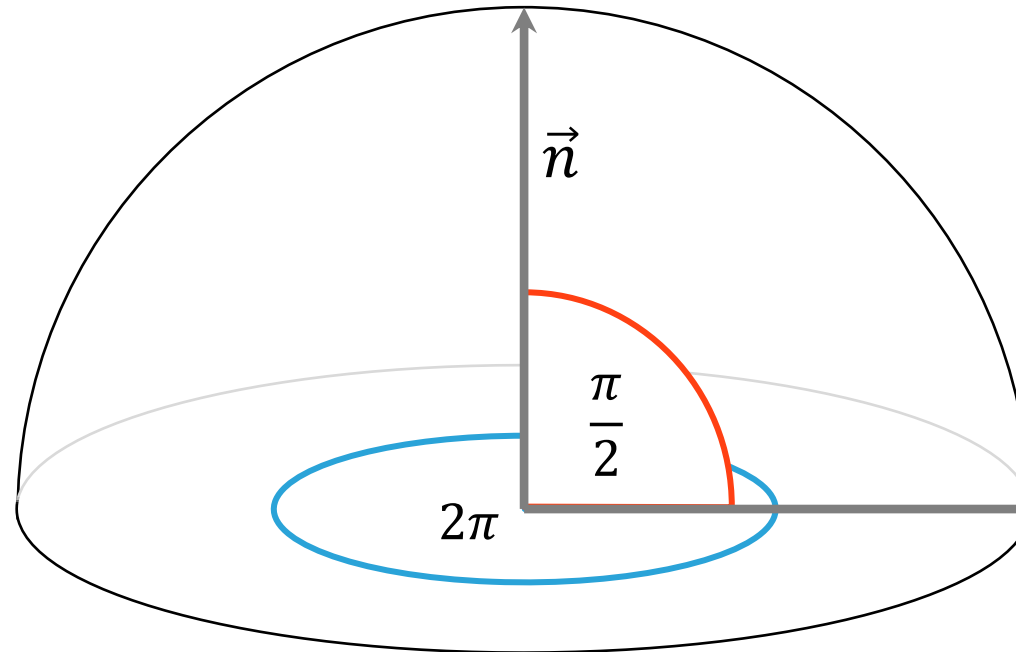
- This took as a while, but we have seen all the formal procedures
- We only need to switch from integrating planar area to points ω on hemisphere surface (i.e., vectors (x, y, z) with length 1)
- Use spherical coordinates and bijective T from (r, θ, ϕ) to (x, y, z) :
$$\begin{aligned}x &= r \sin \theta \cos \phi \\y &= r \sin \theta \sin \phi \\z &= r \cos \theta\end{aligned}$$



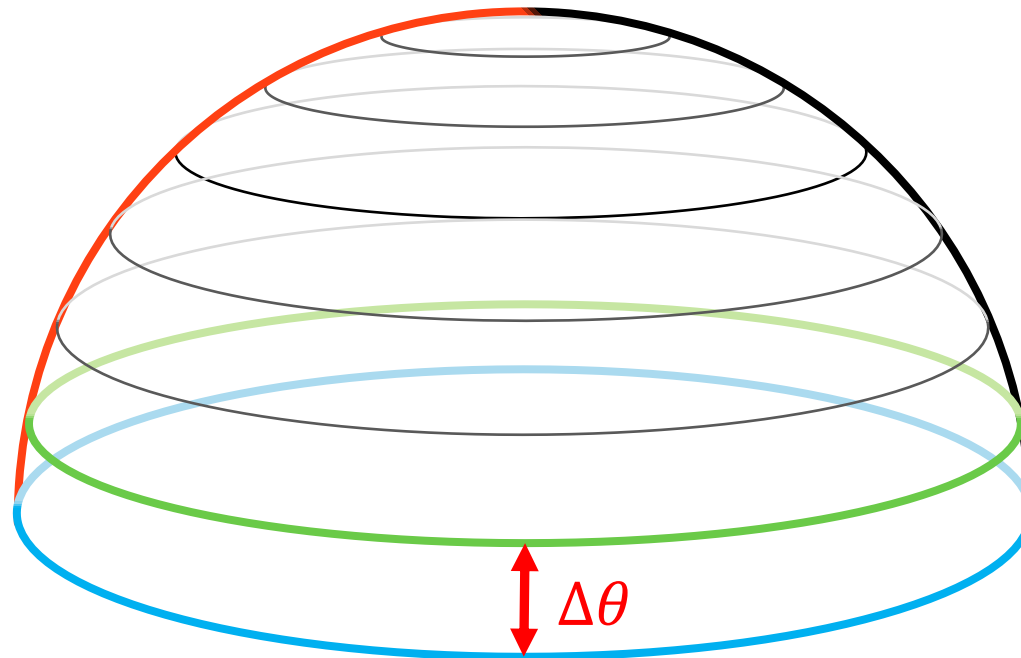
- Each direction ω represents an infinitesimal surface area portion $d\omega$
- How do we integrate a function $f(\omega)$ with differential $d\omega$?
- Integration over points on hemisphere surface ω , w.r.t. (θ, ϕ)



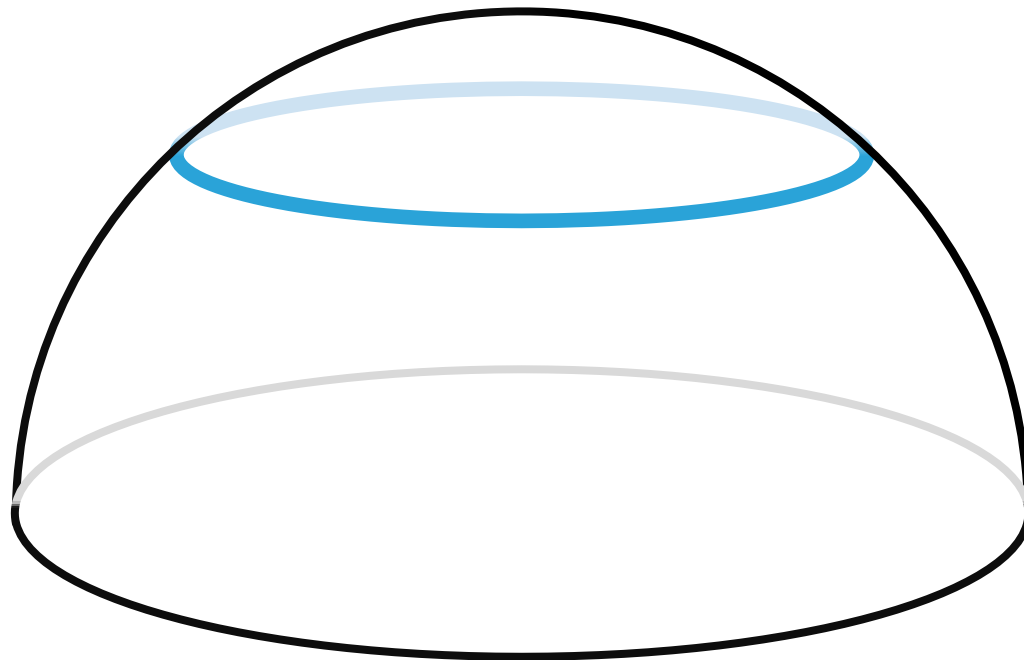
- We assume a planar surface with an upright facing normal \vec{n}
- We use the integral intervals $\theta \in \left[0, \frac{\pi}{2}\right), \phi \in [0, 2\pi)$
- I.e., a **curve** from perpendicular to parallel for θ , a **ring** for ϕ



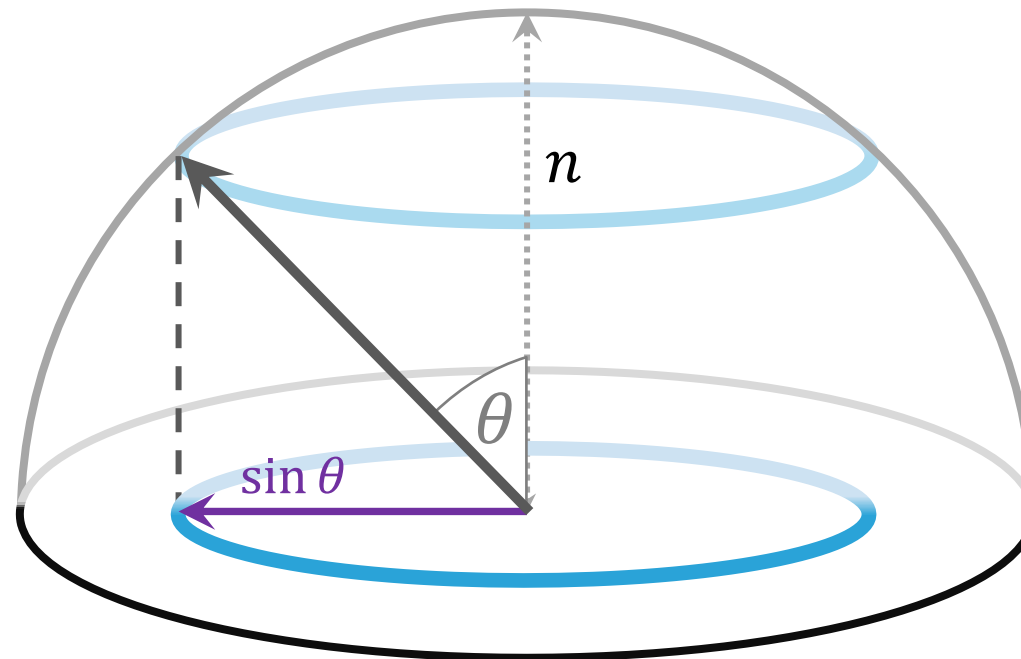
- We can split the surface along θ into ribbons of width $\Delta\theta \rightarrow d\theta$
- The **upper** edge of the ribbon is slightly shorter than the **lower**
- If we keep adding more and more ribbons, this difference vanishes



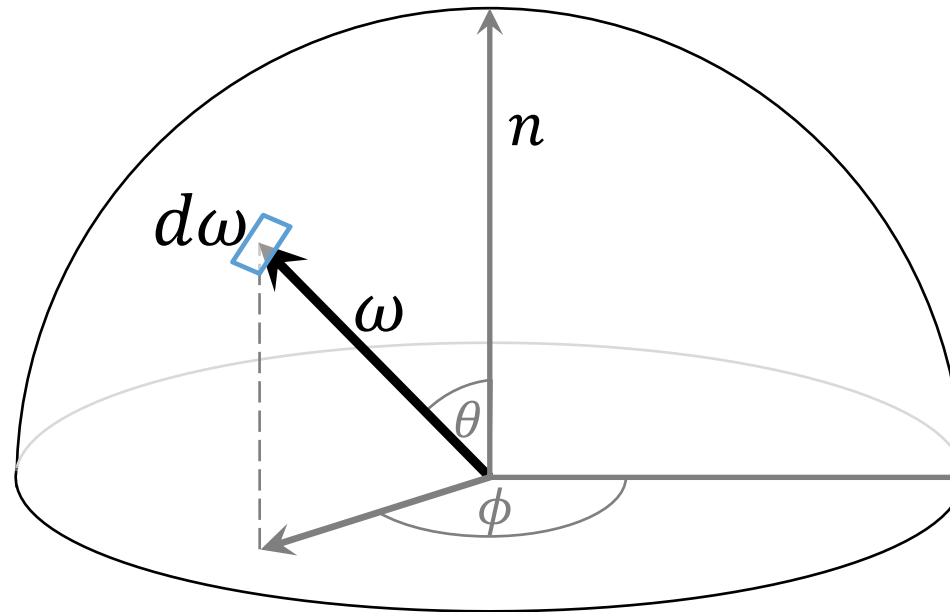
- As a ribbon's width goes to $d\theta$, its area becomes its length times $d\theta$
- We can find this length by projecting the ribbon to the ground
- Using trigonometry, we find the length of a ribbon is $2\pi \sin \theta$



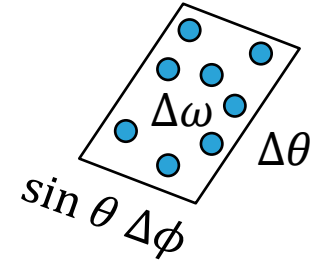
- As a ribbon's width goes to $d\theta$, its area becomes its length times $d\theta$
- We can find this length by projecting the ribbon to the ground
- Using trigonometry, we find the length of a ribbon is $2\pi \sin \theta$



- 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: $\int_{\Omega} f(\omega) d\omega = \int_0^{\frac{\pi}{2}} \int_0^{2\pi} f(\omega) \sin \theta d\phi d\theta$



- 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: $d\omega = \sin\theta d\phi d\theta$

- $\omega \rightarrow (\theta, \phi)$ is bijective, we have $p(\theta, \phi) d\theta d\phi = p(\omega) d\omega$ and:

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



- 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) :
$$\begin{aligned}x &= r \sin \theta \cos \phi \\y &= r \sin \theta \sin \phi \\z &= r \cos \theta\end{aligned}$$
- The Jacobian of the transformation T gives $|J_T| = r^2 \sin \theta$
- Hence, we have $p(r, \theta, \phi) = r^2 \sin \theta p(x, y, z) = 1 \sin \theta p(\omega)$



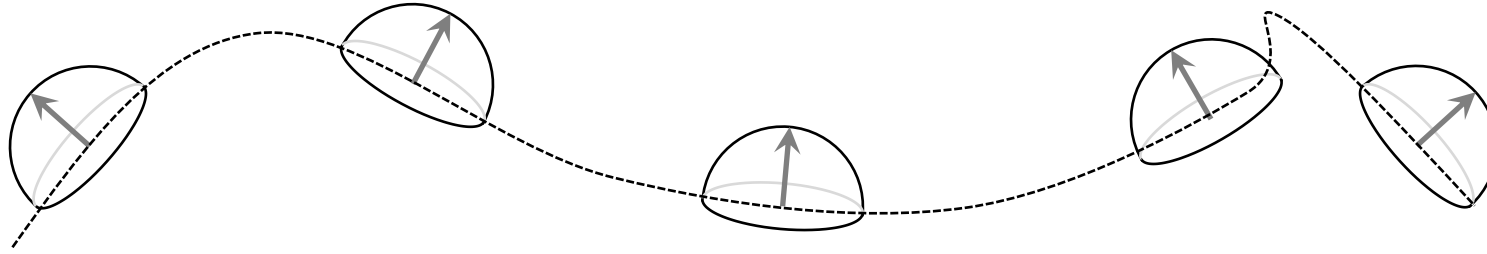
- The domain, i.e., the unit hemisphere surface area, is 2π .
Uniformly sampling the domain over ω implies $p(\omega) = \frac{1}{2\pi}$
- Hence, since $p(\theta, \phi) = r^2 \sin \theta p(\omega)$, we want $p(\theta, \phi) = \frac{\sin \theta}{2\pi}$
- Marginal density $p_{\Theta}(\theta)$: $\int_0^{2\pi}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



- 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
- Done!



- The orientation of sampled hemispheres depends on surface normal




- Use the tangent, bitangent and normal vectors of the intersection
- $\omega_{world} = x \cdot tangent + y \cdot bitangent + z \cdot normal$
- Or, if available, use *ToWorld* transformation methods



- **Diffuse** lighting based on last lecture's insights with constant f_r
- What do we use for L_i ?
- Full rendering equation: next time
- This time: ambient occlusion, direct lighting

Apply

(from the physics chapter)





Material, modelled by the BRDF

Light from direction ω

Solid angle

$$L_e(x, v) = \int_{\Omega} f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x) d\omega$$

Light going in direction v

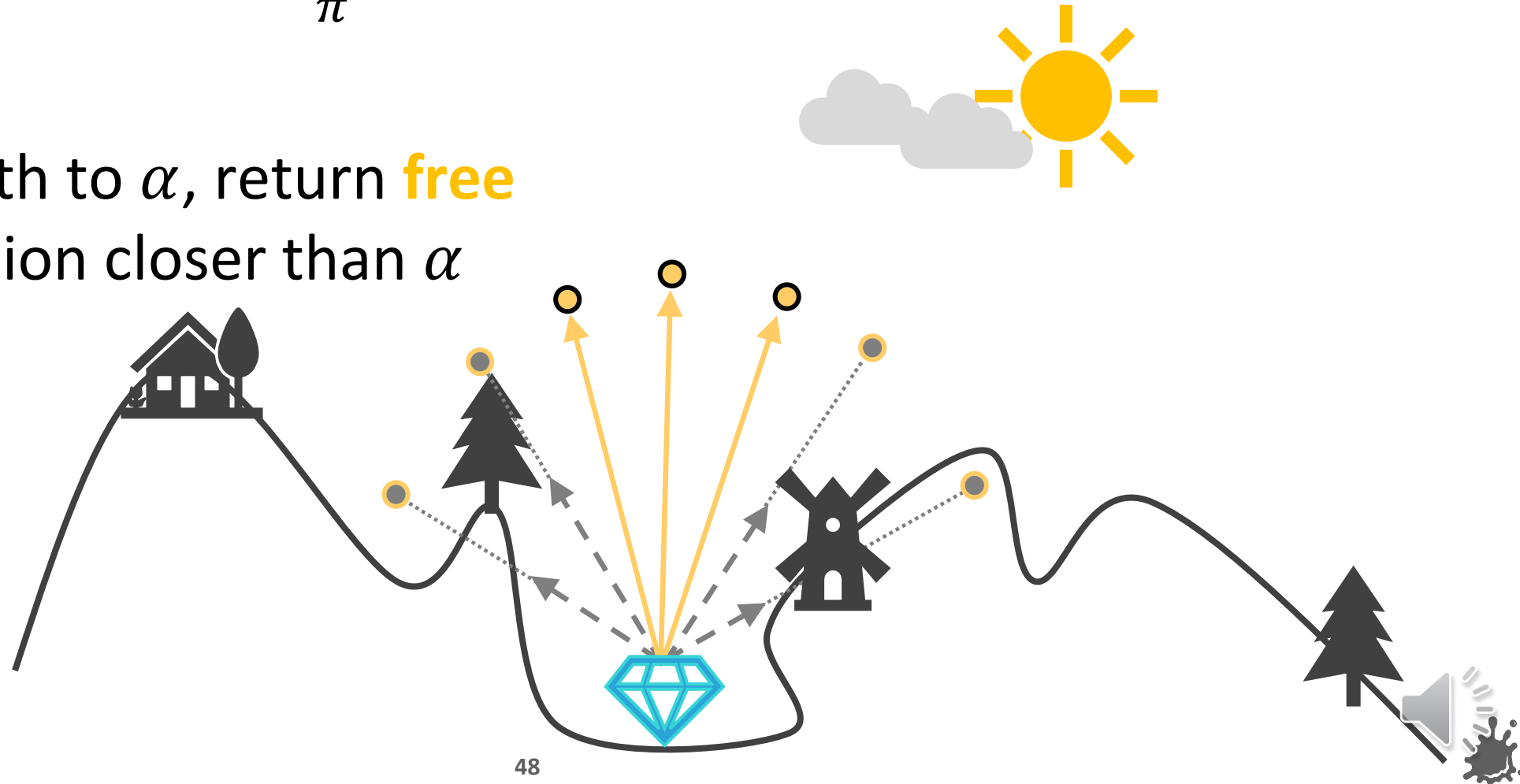


Adam Celarek

69



- Consider all unblocked directions around x as indirect light sources, integrate $V(x, x + \alpha\omega) \frac{\cos \theta}{\pi}$ over directions ω around the normal
- Limit ray length to α , return **free** if no intersection closer than α

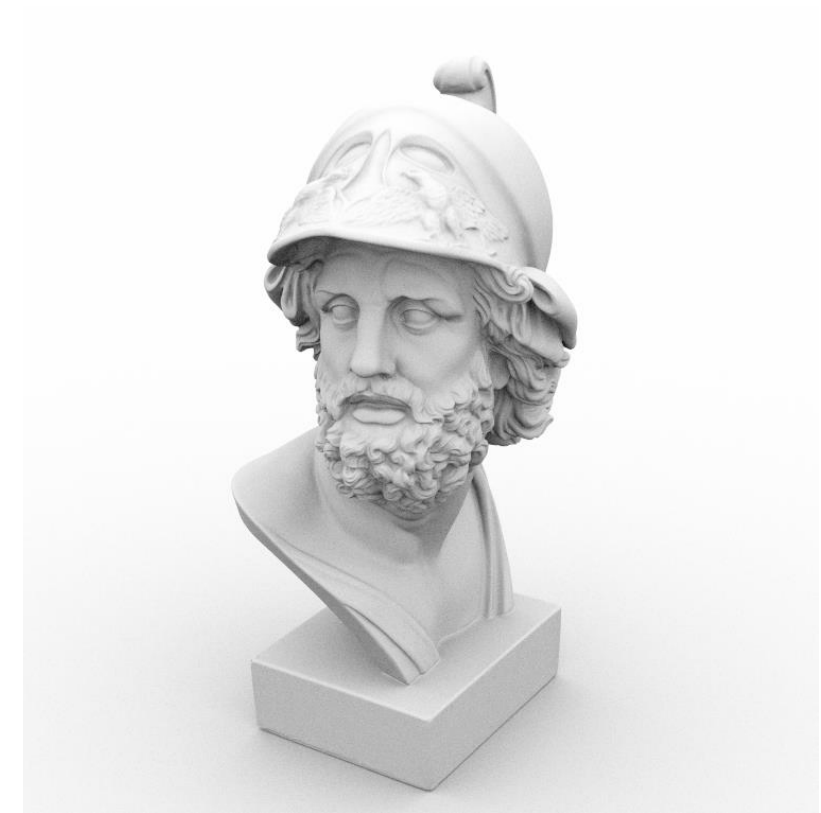


- Fine geometric details on objects are accentuated by the absence of ambient light due to the shadows cast by close-by geometry

$$L_e(x) = \int_{\Omega} V(x, x + \alpha\omega) \frac{\cos(\theta)}{\pi} d\omega$$

- Integrate over directions ω on the unit hemisphere defined by point x , normal n

- $V(x, x + \alpha\omega) = \begin{cases} 1 & \text{if } x \rightarrow (x + \alpha\omega) \text{ free} \\ 0 & \text{if } x \rightarrow (x + \alpha\omega) \text{ blocked} \end{cases}$

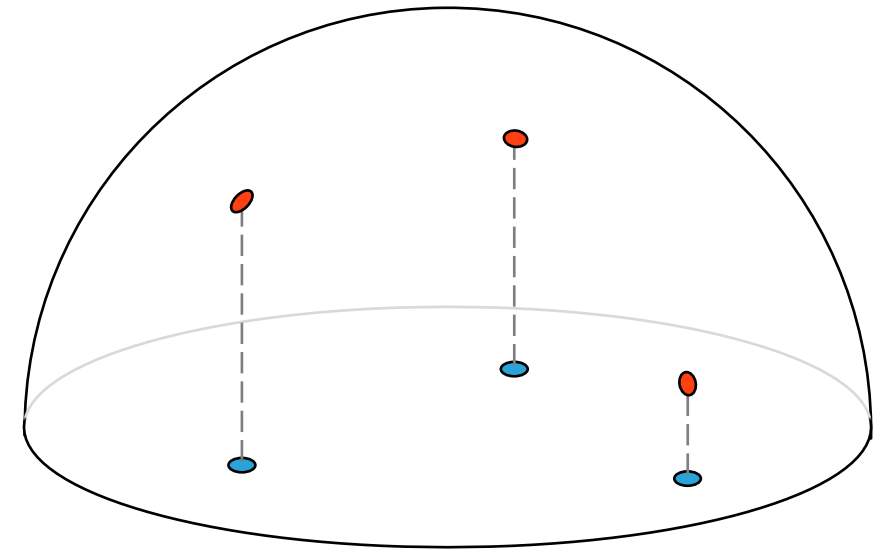


- Recap: variance is low if the sampling function mimics the signal
- We use $f_r = \frac{1}{\pi}$ for ambient occlusion, therefore the contribution of signal samples varies mostly with $\cos \theta$
- It would be best to apply importance sampling: use a sampling strategy for ω , such that $p(\omega) \propto \cos \theta$
- We have gone through all the necessary steps.
Try to solve this formally with the inversion method as an exercise!



- Malley's method: uniformly pick (x, y) samples on the **unit disk**
- Project them to the hemisphere **surface** $\left(z = \sqrt{1 - x^2 - y^2}\right)$

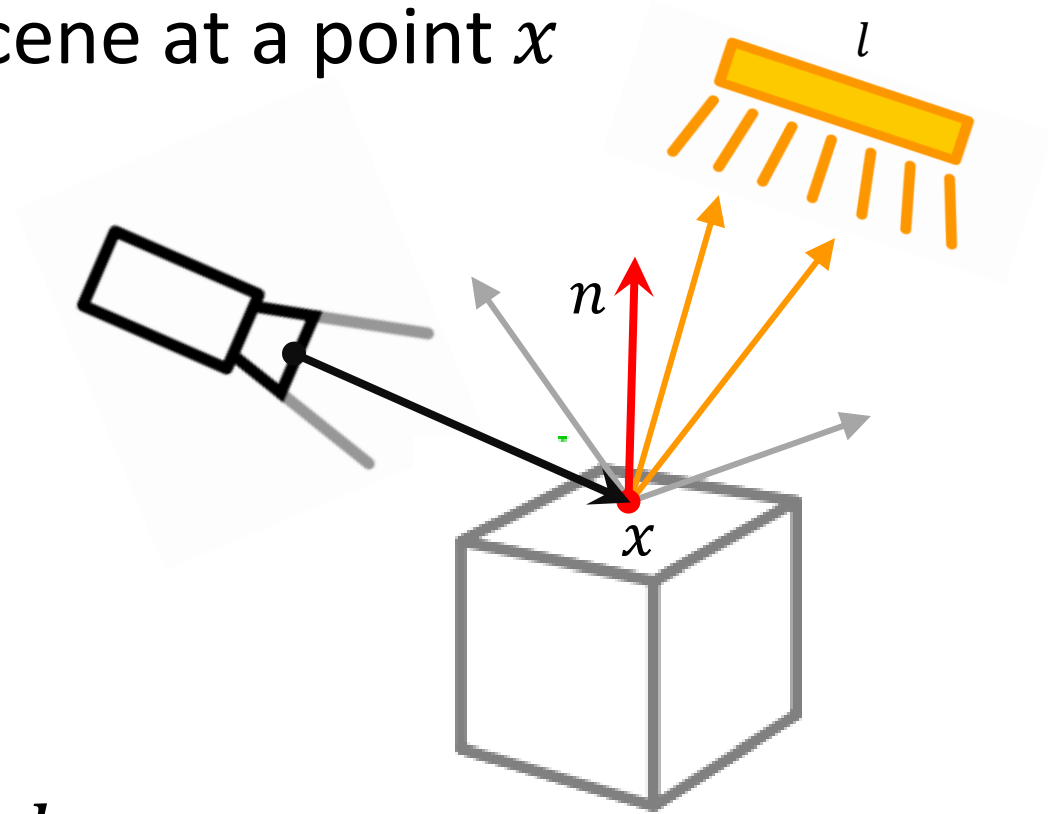
- Done! Your samples are now distributed with $p(\omega) \propto \cos \theta$



- *Why* does this work? Try to come up with your own proof!



- We can use Monte Carlo integration to compute the direct illumination from light sources in the scene at a point x
- Naïve version: sample unit hemisphere uniformly, hoping to hit light sources
- Check **closest** hit for each direction ω
- Use
$$L_i(x, \omega) = \begin{cases} L_e^{[l]} & \text{if hit a light } l \\ 0 & \text{otherwise} \end{cases}$$



- A special kind of importance sampling: integrate over light sources!

- Use $V(x, y) = \begin{cases} 1 & \text{if path from } x \text{ to } y \text{ is unblocked} \\ 0 & \text{otherwise} \end{cases}$

- Pick y on light, e.g. uniformly

- $\frac{\cos \theta_y dA_y}{r^2}$: change in volume of infinitesimal 2D hypercube at y projected onto x 's hemisphere

Apply

Soft shadows (usable for rendering)

light intensity at position y on the surface

visibility (new, ray tracing)

emitter $\cos(\theta)$

receiver $\cos(\theta)$

distance

$$L_e^{[l]}(x) = \int_{S_l} f_r(x, y \rightarrow v) L_e^{[l]}(y) V(x, y) \cos(\theta_x) \frac{\cos(\theta_y)}{r^2} dA_y$$

Light going in direction v

Material, modelled by the BRDF

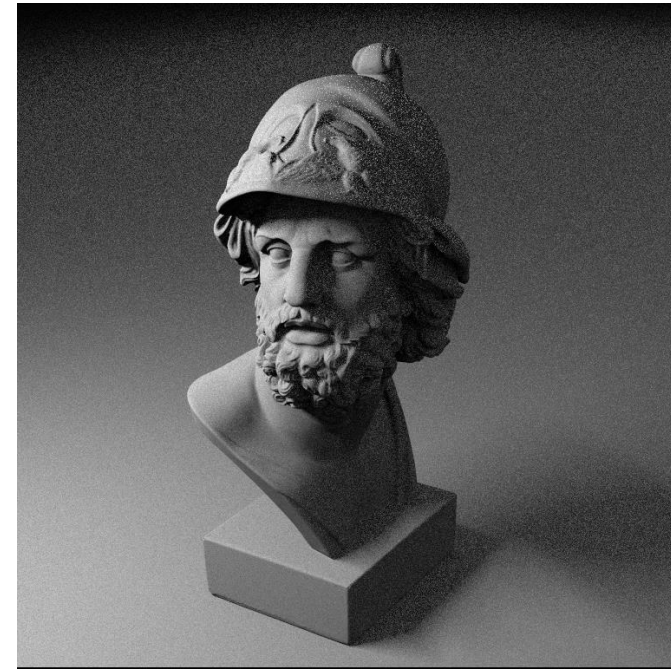
Adam Celarek

71

- For inclusion of simple material color and BRDF, use $f_r = \frac{\rho}{\pi}$
- Works extremely well if the light occupies only a small solid angle



100 samples per pixel, hemisphere sampling



100 samples per pixel, light source sampling





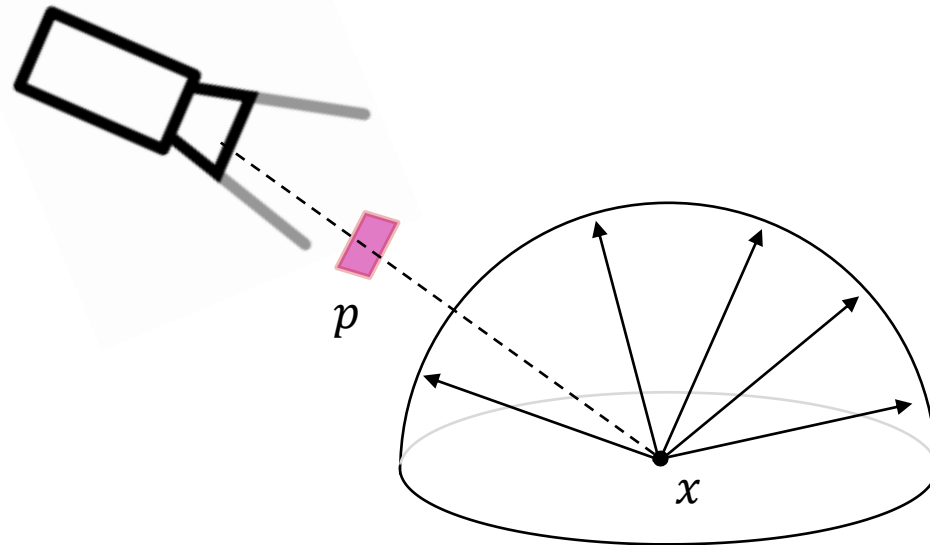


- But where does the actual integration step happen?
- In the basic case, directly in the main sampling loop for each pixel!
 - Static scene: Samples through pixels p always hit the same point x
 - Once x has been hit, the sampling of its hemisphere follows PDF
 - Return sampled values (colors), weighted by the corresponding PDF
- Use N samples for p , sum color values weighted by PDF, average:

$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{L_i(x, \omega)}{p(\omega)}$$



- We achieve integration around x with multiple samples through p
 - A bit wasteful, but is a general, valid solution
 - We will see in a second why this is convenient
- Weight returned values by PDF, sum up and divide by N



Monte Carlo Integration as Loop Over Pixel Samples

Given: camera, pixel p, scene, pdf

rgb = {0,0,0}

for i in [0, N) do

ray = rayThroughPixel(camera, p)

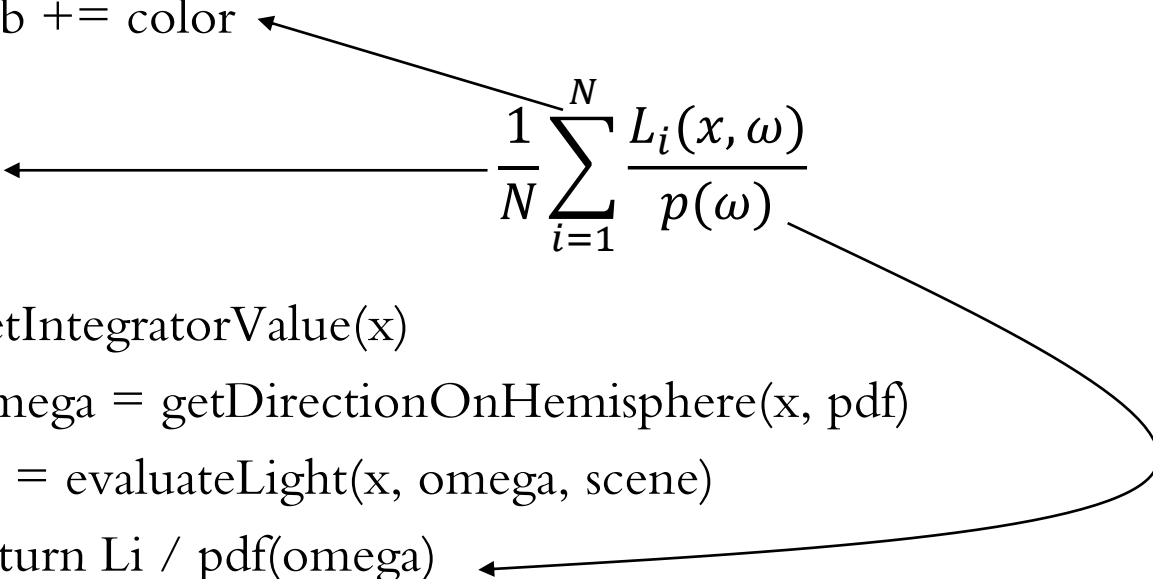
x = findIntersection(ray, scene)

color = getIntegratorValue(x)

rgb += color

end for

rgb /= N

$$\frac{1}{N} \sum_{i=1}^N \frac{L_i(x, \omega)}{p(\omega)}$$


function getIntegratorValue(x)

omega = getDirectionOnHemisphere(x, pdf)

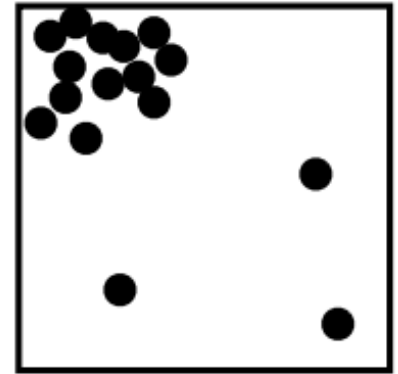
Li = evaluateLight(x, omega, scene)

return Li / pdf(omega)

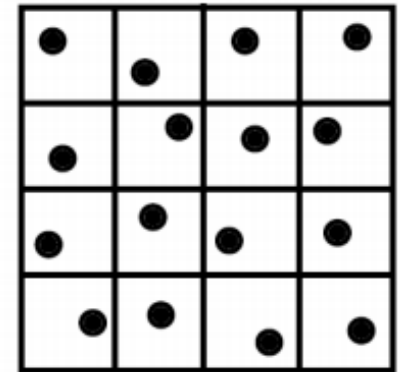


- With random uniform sampling, we can get unlucky

- E.g. all samples clump in a corner
- If we don't know anything of the integrand, we want a relatively uniform sampling
 - Not regular, though, because of alias patterns!



- Subdivide domain into non-overlapping regions (e.g. a regular grid). Each region is called a *stratum*



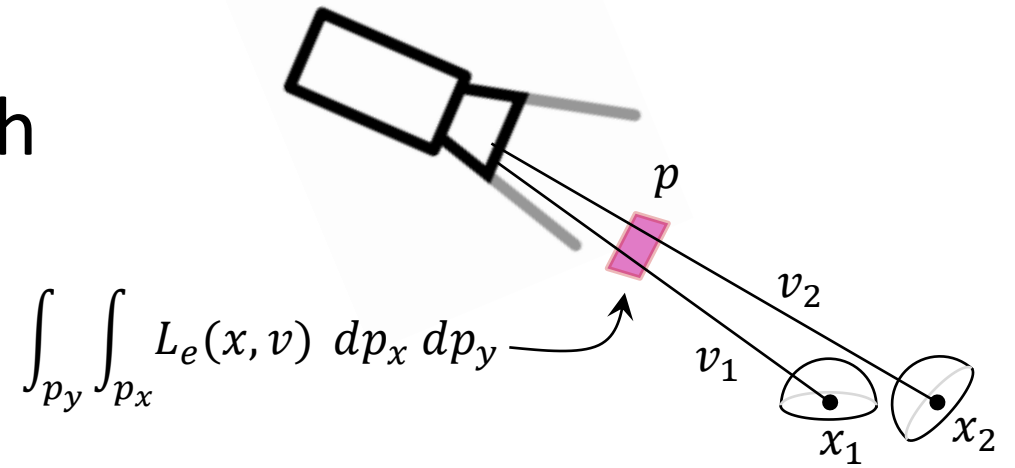
- Pick new random sample in stratum with lowest number of samples



- In the first lecture, we used supersampling to fight off aliasing
- Pixels are another instance where we use Monte Carlo integration
- Choosing samples within pixels is an instance of stratified sampling
- Uniform 2D distribution, average over a pixel rectangle: *box filter*
 - We will see more advanced methods for filtering in future lectures
 - If we didn't use at least one sample per pixel, we would leave holes



- Samples are randomly jittered in each stratum
- Ergo, we **don't** hit the same x with each pixel sample (p_x, p_y) inside pixel p
- We just add two random variables!
- Instead of integrating over a hemisphere, we are integrating over all surface points visible to a pixel **and** their respective hemispheres
- The box filter over pixel color samples implements uniform integral



The Box Filter for a Pixel (Monte Carlo Integration)

Given: camera, pixel coordinates (pixel_x, pixel_y), scene

```
rgb = {0,0,0}
```

```
for i in [0, N) do
```

```
    px = pixel_x + uniform_random_sample() //  $\xi_i = P_\xi(\xi_i) = P_\xi^{-1}(\xi_i)$ 
```

```
    py = pixel_y + uniform_random_sample()
```

```
    ray = rayThroughPixelPos(camera, px, py)
```

```
    x = findIntersection(ray, scene)
```

```
    rgb += getIntegratorValue(x)
```

```
end for
```

```
rgb /= N
```

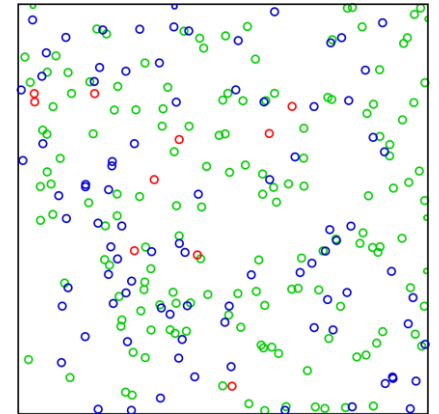
Nothing else necessary: offsets in x and y are uniformly distributed, they are independent, the domain size of each pixel is $1 \times 1 = 1$ and so we don't have to change the integration at all.



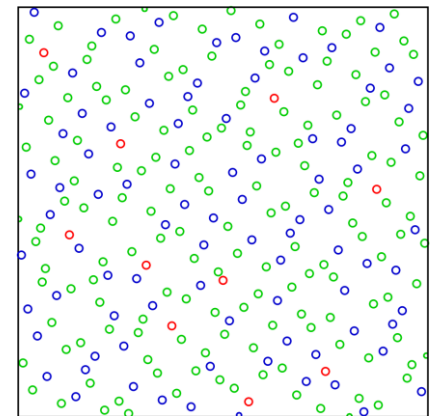
- Stratified sampling is a special case of low-discrepancy sampling^[4]
- Replace the built-in RNG with a sample generation algorithm that sacrifices randomness for good spatial distribution
- Great for:
 - Faster convergence (reduction of noise)
 - GPUs (they don't love random numbers)
 - Transparent and portable sampling behavior



- For n -D, pick n different, co-prime bases (e.g. 2,3)
- Based on *radical inverse*: an integer a can be written with base b and $d \in [0, b - 1]$ as $a = \sum_{i=1}^m d_i(a) b^{i-1}$
- For $b = 2, d \in [0,1]$: this is binary representation of integers: $13 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3$
- Radical inverse with m digits: $\Phi(a) = \sum_{i=1}^m \frac{d_i(a)}{b^i}$



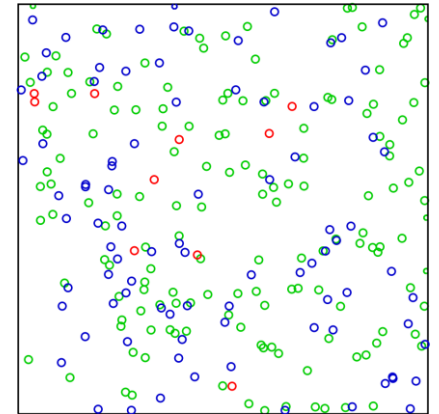
Default RNG



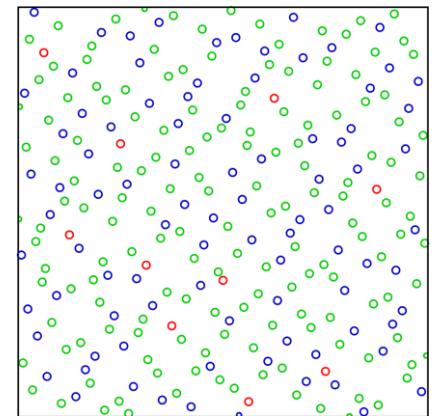
2,3 Halton Sequence



- Start at any integer value for each of the n variables
 - Need data structure to represent $a = \sum_{i=1}^m d_i(a)b^{i-1}$
 - Can be written for arbitrary b with some bit fiddling
- For each new n -D sample, increment all n integers
- Compute their radical inverse with their proper base
- Using same sequence for all pixels \rightarrow patterns ☹️



Default RNG

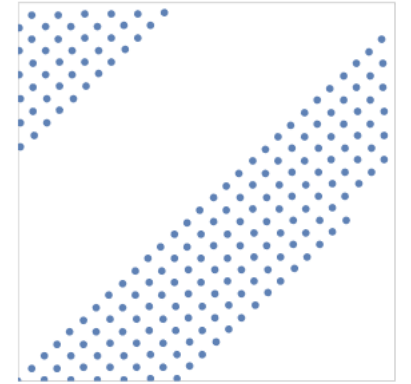


2,3 Halton Sequence

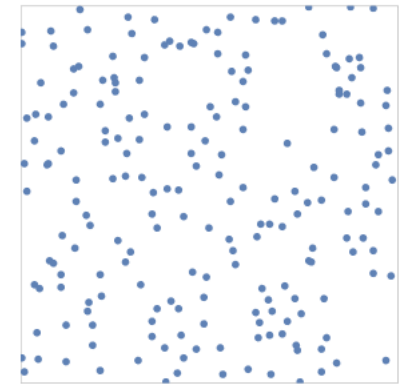
[Jheald](#), Wikipedia, Halton Sequence – “Own work”



- Large primes will behave similarly → patterns 😞
- Repetitive patterns should always be avoided!
- Option 1: use different starting values for different instances (e.g. for each pixel)
- Option 2: instead of incrementing d_i , cycle through random permutations of $[0, b - 1]$ **in each instance**
 - E.g.: $b = 3, \{0, 2, 1\}$ instead of $\{0, 1, 2\}$



29,31 Halton Sequence



29,31 Halton Scrambled



- Get comfortable with all approaches to integration and sampling
 - The mental image of “area under the curve” eventually collapses (infinite-dimensional integral coming up next!)
 - Transformations between sample domains may be non-trivial
 - Once you grasp the underlying concepts, applying the math is easy
- We have seen simpler explanations for the most important parts
 - Uniformly sampling a hemisphere
 - Cosine-weighted sampling of a hemisphere



- 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, *Van der Corput Sequence*, https://en.wikipedia.org/wiki/Van_der_Corput_sequence

