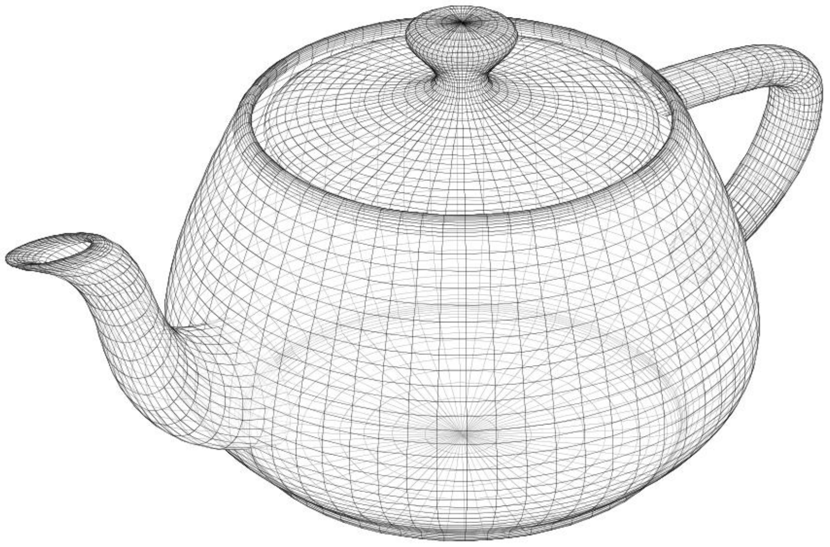


Ambient Occlusion

Production Computer Graphics

Eric Shaffer

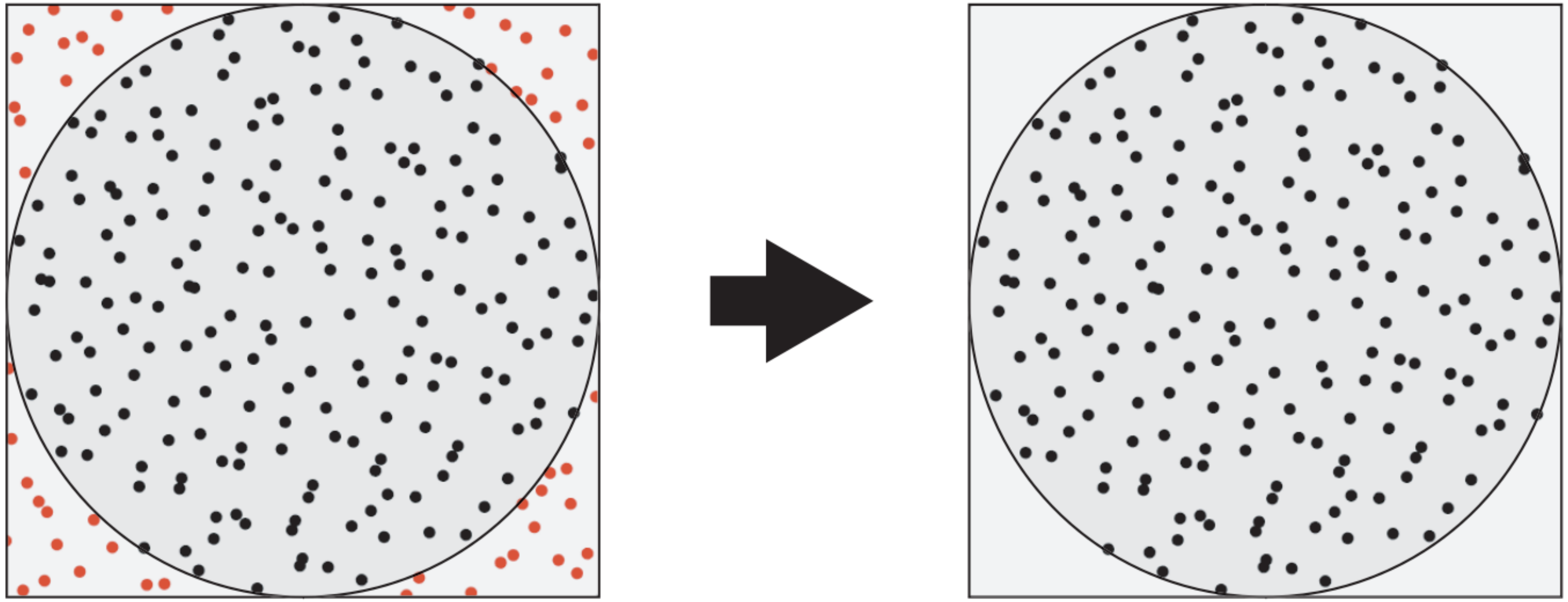
ILLINOIS

Sampling from a Disk

Needed for

- Simulating depth of field effect (camera lens is circular)
- Disk lights

- We know how to sample a unit square. Can we use that?
- Rejection Sampling:



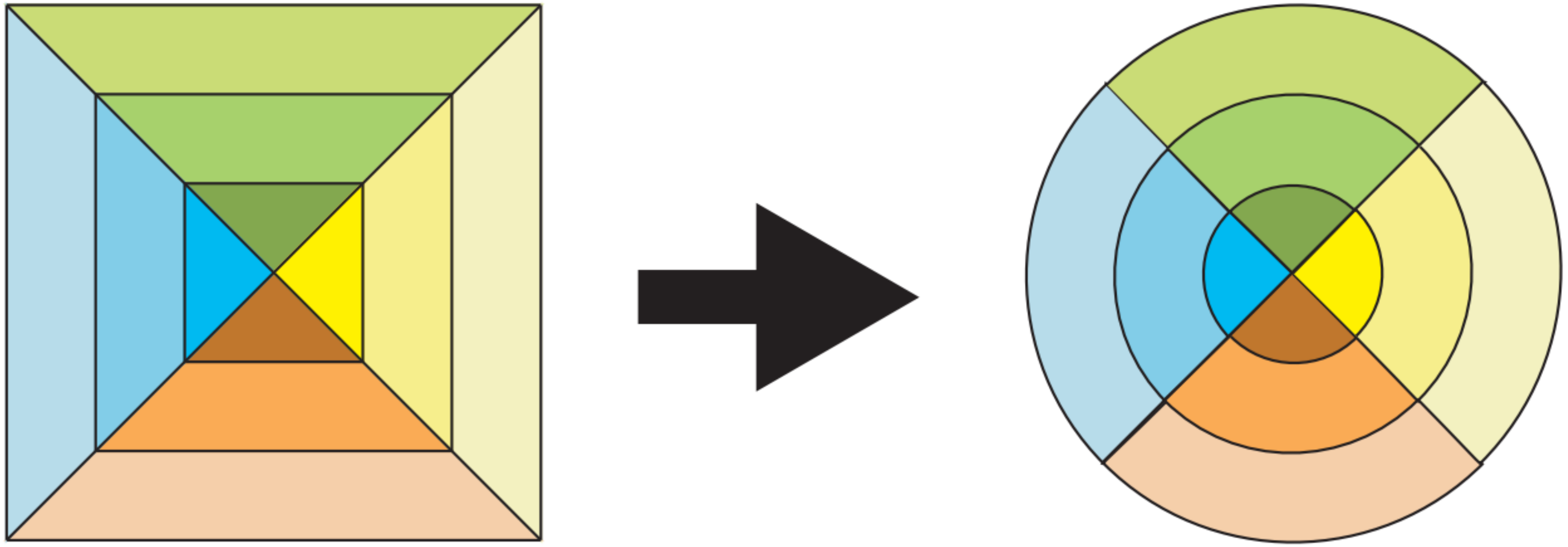
Concentric Mapping

Preserves all samples

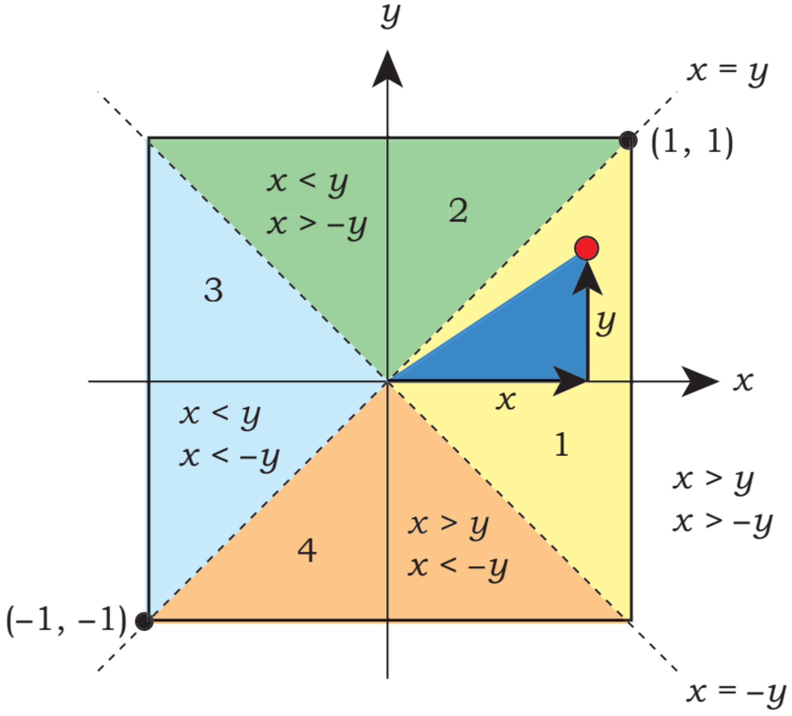
Samples close on square also close on disk

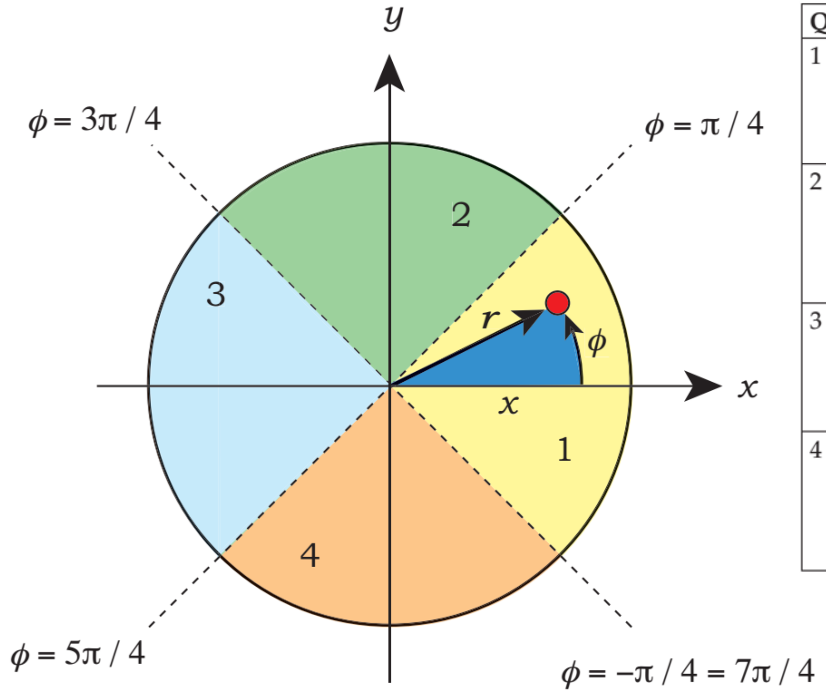
Each concentric band in square mapped to concentric band in circle

Limits distortion



Concentric Mapping





Quarter	x and y Inequalities	Angular Range	Map Equations
1	$x > y$ $x > -y$	$\theta \in [-\pi/4, \pi/4]$	$r = x$ $\phi = \frac{\pi}{4}\frac{y}{x}$
2	$x < y$ $x > -y$	$\theta \in [\pi/4, 3\pi/4]$	$r = y$ $\phi = \frac{\pi}{4}\left(2 - \frac{x}{y}\right)$
3	$x < y$ $x < -y$	$\theta \in [3\pi/4, 5\pi/4]$	$r = -x$ $\phi = \frac{\pi}{4}\left(4 + \frac{y}{x}\right)$
4	$x > y$ $x < -y$	$\theta \in [5\pi/4, 7\pi/4]$	$r = -y$ $\phi = \frac{\pi}{4}\left(6 - \frac{x}{y}\right)$

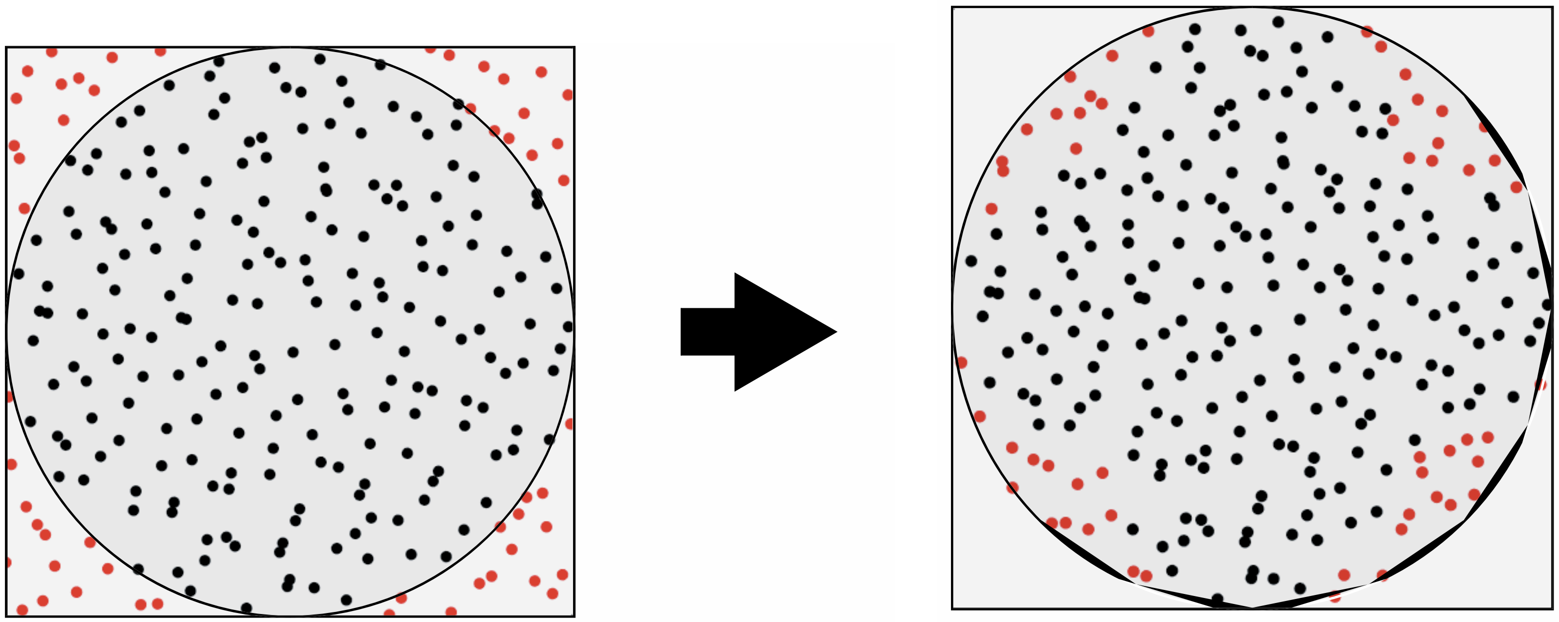
Map x,y to polar values

r mapped from coordinate along axis through the quadrant

Polar angle derived from quotient in [-1,1]

ILLINOIS

Results



ILLINOIS

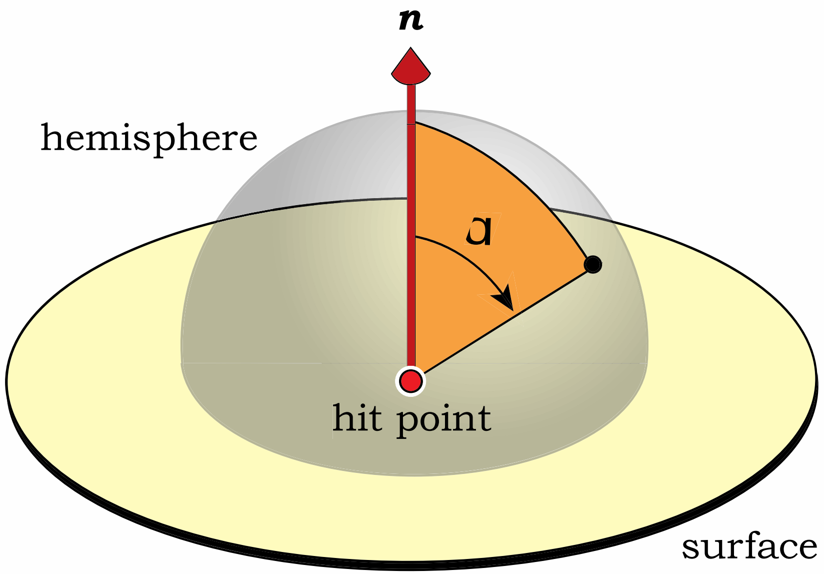
Code

```
void
Sampler::map_samples_to_unit_disk(void) {

    int size = samples.size();
    float r, phi;                              // polar coordinates
    Point2D sp;                                // sample point on unit
    disk

    disk_samples.reserve(size);

    for (int j = 0; j < size; j++) {
        // map sample point to [-1, 1]   [-1,1]

        sp.x = 2.0 * samples[j].x - 1.0;
        sp.y = 2.0 * samples[j].y - 1.0;

        if (sp.x > -sp.y) {                    // sectors 1 and 2
            if (sp.x > sp.y) {                 // sector 1
                r = sp.x;
                phi = sp.y / sp.x;
            }
            else {                             // sector 2
                r = sp.y;
                phi = 2 - sp.x / sp.y;
            }
        }
        else {                                 // sectors 3 and 4
            if (sp.x < sp.y) {                 // sector 3
                r = -sp.x;
                phi = 4 + sp.y / sp.x;
            }
            else {                             // sector 4
                r = -sp.y;
                if (sp.y != 0.0)               // avoid division by zero
                at origin
                    phi = 6 - sp.x / sp.y;
                else
                    phi = 0.0;
            }
        }

        phi *= pi / 4.0;

        disk_samples[j].x = r * cos(phi);
        disk_samples[j].y = r * sin(phi);
    }
}
```

Sampling on a Unit Hemisphere

When a ray hits an object we will

- Center a hemisphere on a hit point
- We will shoot shadow rays, reflected rays, transmission rays
- These rays can be used to simulate
  - ambient occlusion
  - glossy reflection/transmission



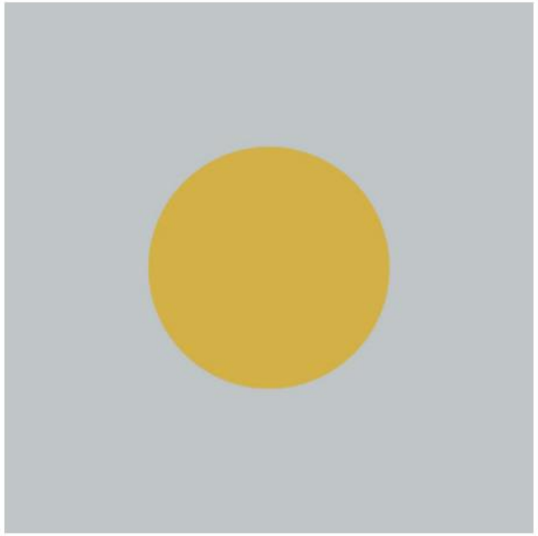
Ambient Occlusion

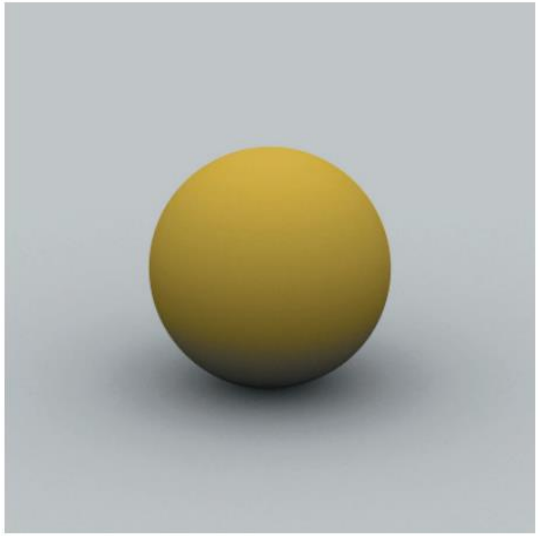
Ambient illumination

- Constant level of illumination throughout a scene
- Surface that only has ambient illumination appears as a constant color

Occlusion

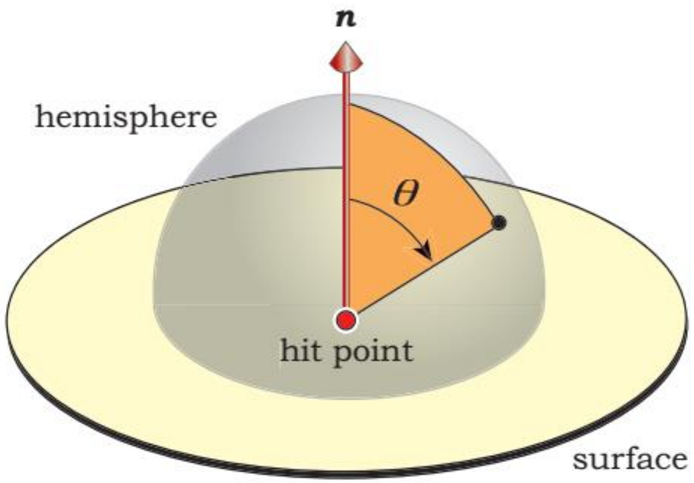
- Let hemisphere H enclose a surface point p
- Amount of illumination at p depends on amount of H unblocked
  - include a visibility term in the model

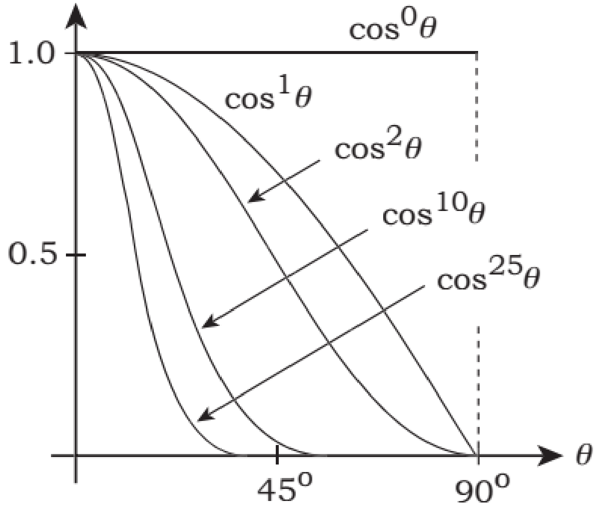




Ambient Occlusion

Cosine Distributions

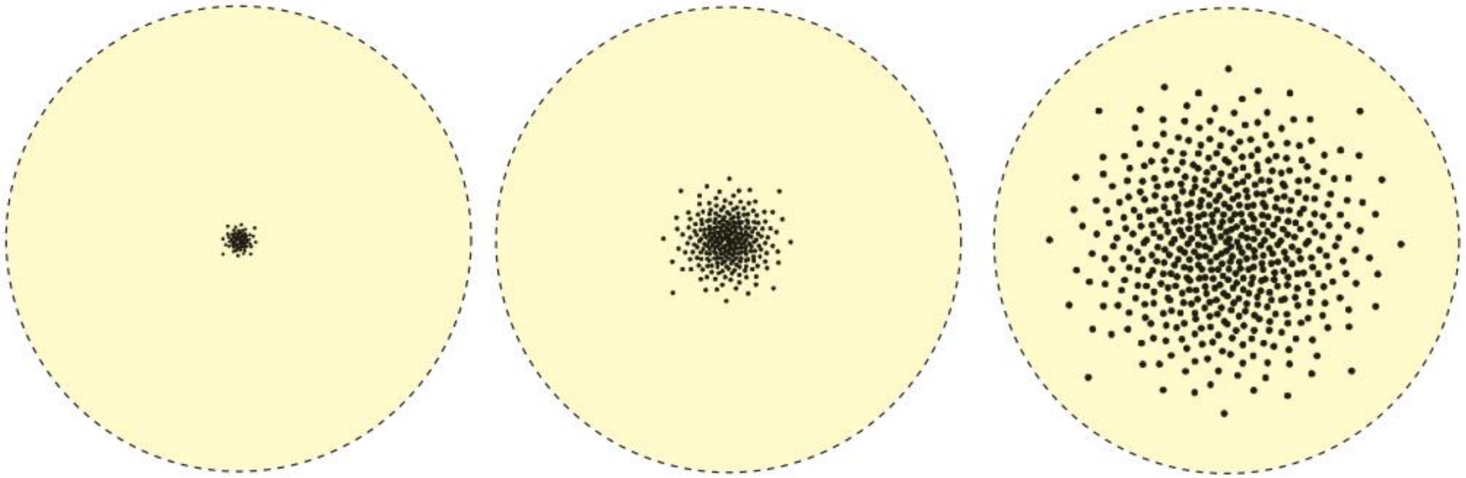




Want to sample hemisphere using a cosine distribution

Sampling distribution varies with θ

$$d = \cos^t \theta, t \in [0, \infty)$$



t=1000 | t=100 | t=10

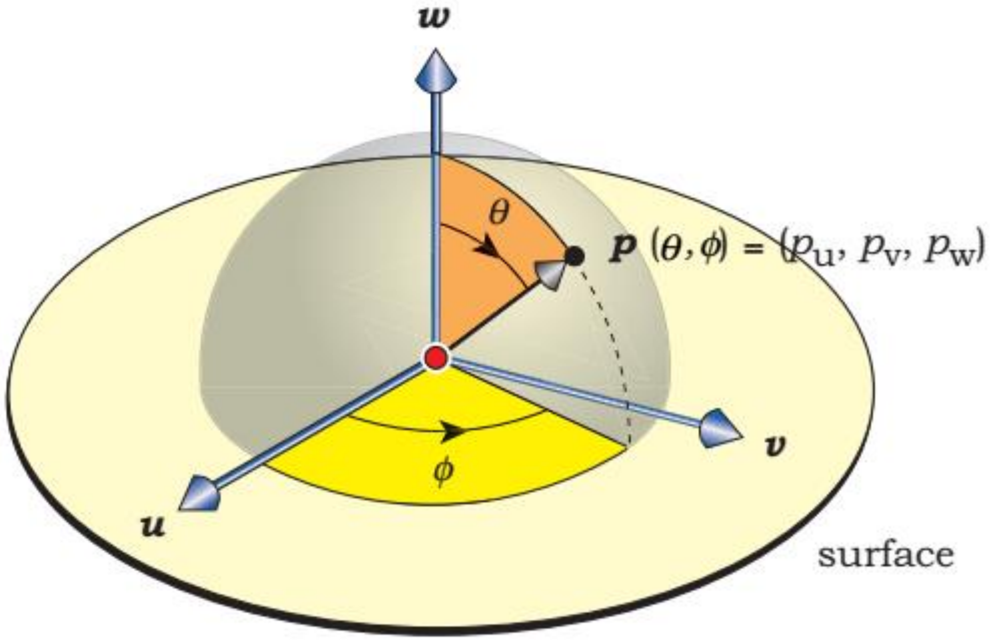
Mapping

Map (r_1, r_2) on unit square to (ϕ,θ)

- ϕ is azimuth angle
- Θ is polar angle

$$f = 2pr_1$$

$$q = \cos^{-1}[(1 - r_2)^{1/(e+1)}]$$



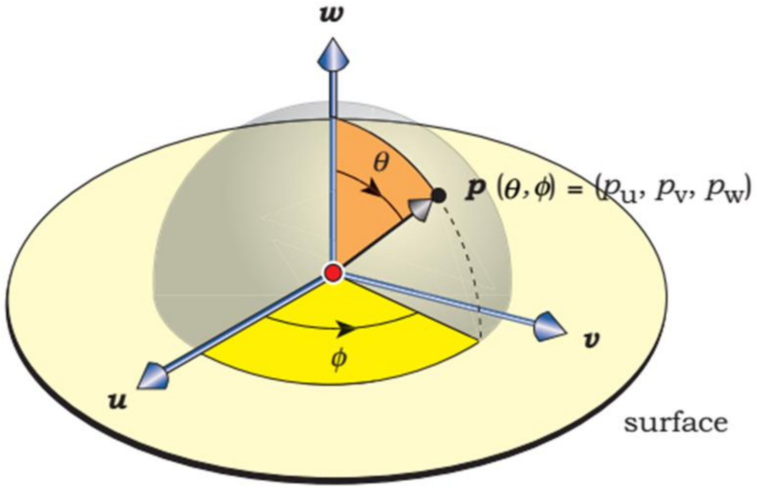
Converting to Cartesian Coordinates

Can define a local orthonormal basis on the surface

Convert sample points to (u,v,w) coordinates

$$
\begin{aligned}
w &= n, \\
v &= w \times up / \|w \times up\|, \\
u &= v \times w.
\end{aligned}
$$

What special case do we need to handle?

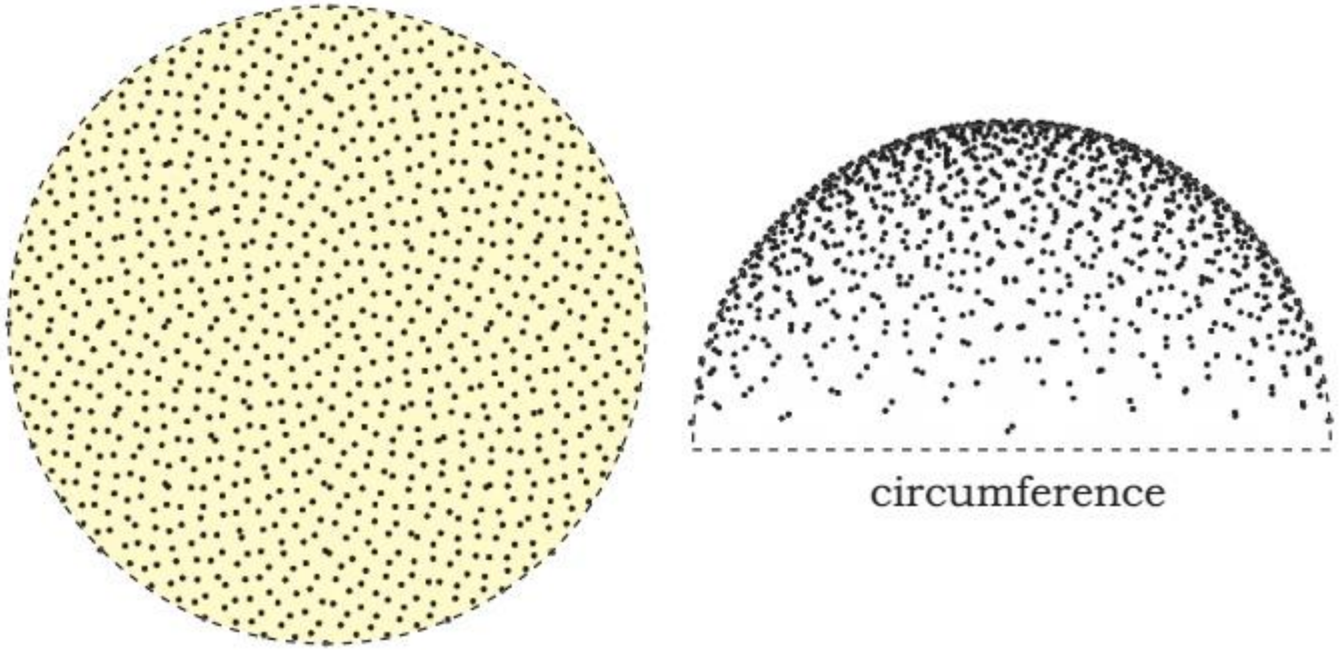


$$p = \sin q \cos f u + \sin q \sin f v + \cos q w$$

Code

```
void
Sampler::map_samples_to_hemisphere(const float e) {
	int size = samples.size();
	hemisphere_samples.reserve(num_samples * num_sets);

	for (int j = 0; j < size; j++) {
		float cos_phi = cos(2.0 * PI * samples[j].x);
		float sin_phi = sin(2.0 * PI * samples[j].x);
		float cos_theta = pow((1.0 - samples[j].y), 1.0 / (e + 1.0));
		float sin_theta = sqrt (1.0 - cos_theta * cos_theta);
		float pu = sin_theta * cos_phi;
		float pv = sin_theta * sin_phi;
		float pw = cos_theta;

		hemisphere_samples.push_back(Point3D(pu, pv, pw));
	}
}
```

Cosine Power Distribution of Hammersley Samples

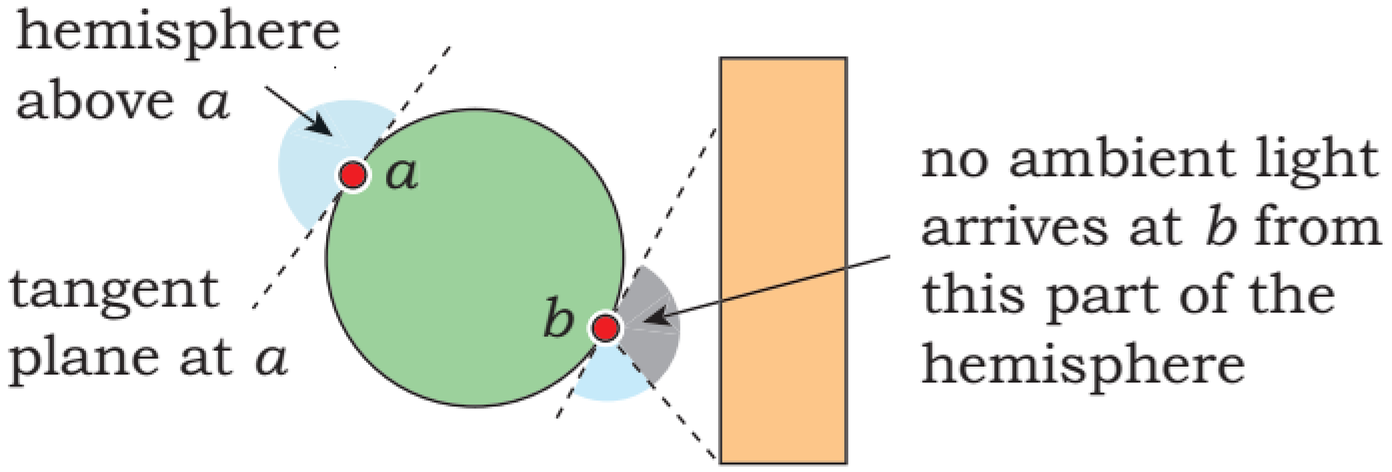


1024 samples, t=1

Ambient Occlusion

We make ambient illumination non-constant

We base the illumination how much of hemisphere is not blocked



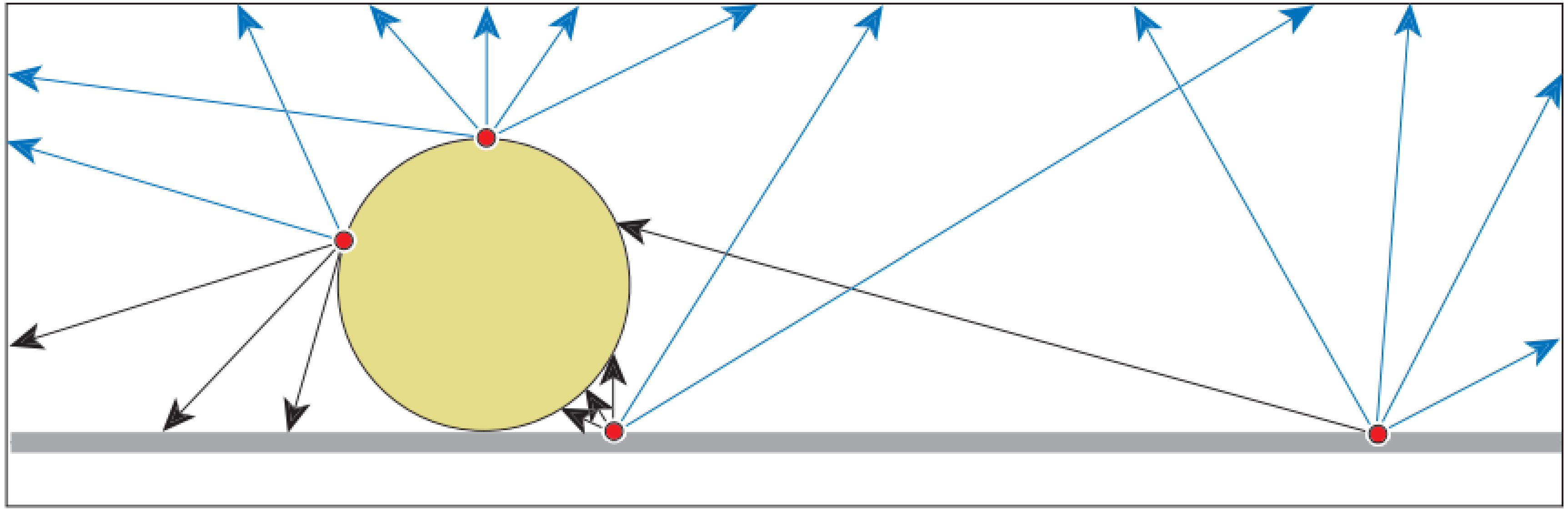
Computing Amibient Occlusion

$L_i(\boldsymbol{p}, \boldsymbol{\omega}_i) = V(\boldsymbol{p}, \boldsymbol{\omega}_i)\, l_s \boldsymbol{c}_l,$

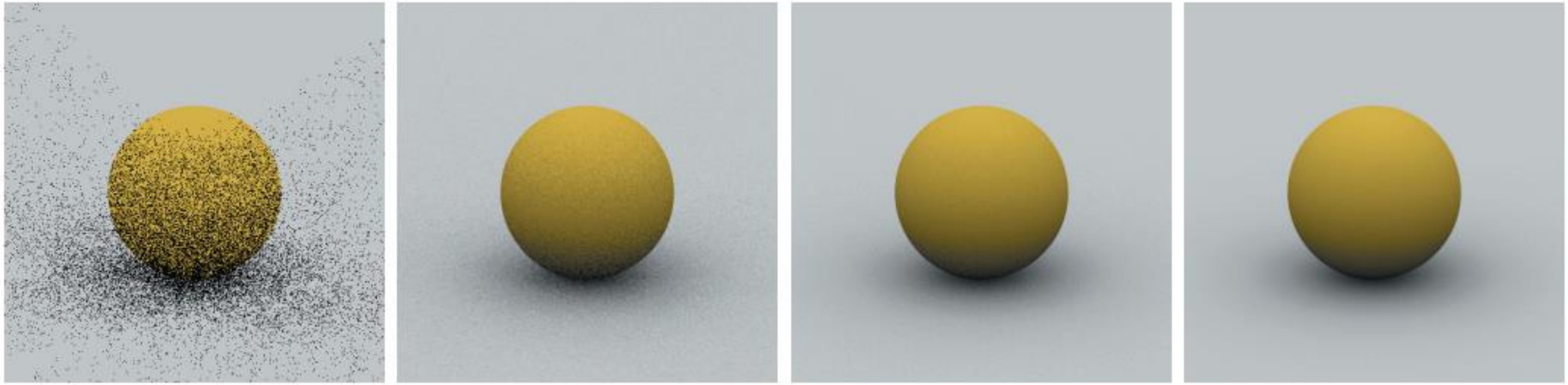
$$V(\boldsymbol{p}, \omega_i) = \begin{cases} 1 & \text{if the direction } \boldsymbol{\omega}_i \text{ at } \boldsymbol{p} \text{ is not blocked,} \\ 0 & \text{if the direction } \boldsymbol{\omega}_i \text{ at } \boldsymbol{p} \text{ is blocked.} \end{cases}$$

$$L_o(\boldsymbol{p}, \boldsymbol{\omega}_o) = (k_a \boldsymbol{c}_d/\pi) * (l_s \boldsymbol{c}_l) \int_{2\pi^+} V(\boldsymbol{p}, \omega_i) \cos\theta_i \, d\omega_i$$

l_s scaling factor for ambient light
$\boldsymbol{c}_l$ rgb color for ambient light
k_a ambient reflection coeffiecient
$\boldsymbol{c}_d$ rgb color for diffuse reflection



Amibient Light Only Example

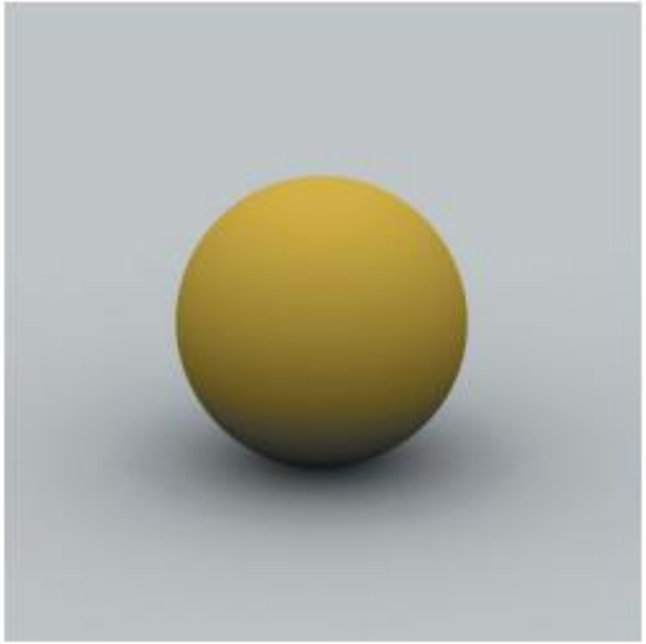


N=1 spp | 16 spp | 64 spp | 256 spp

Here….we trace N paths from pixel
Each hit results in just one shadow ray being shot using hemisphere sampling
Result at pixel is averaged over all samples

Ambient and Direct Example

Ambient only



Ambient+ Direct 100 samples per pixel