Shadows & Area Lights

Steve Rotenberg

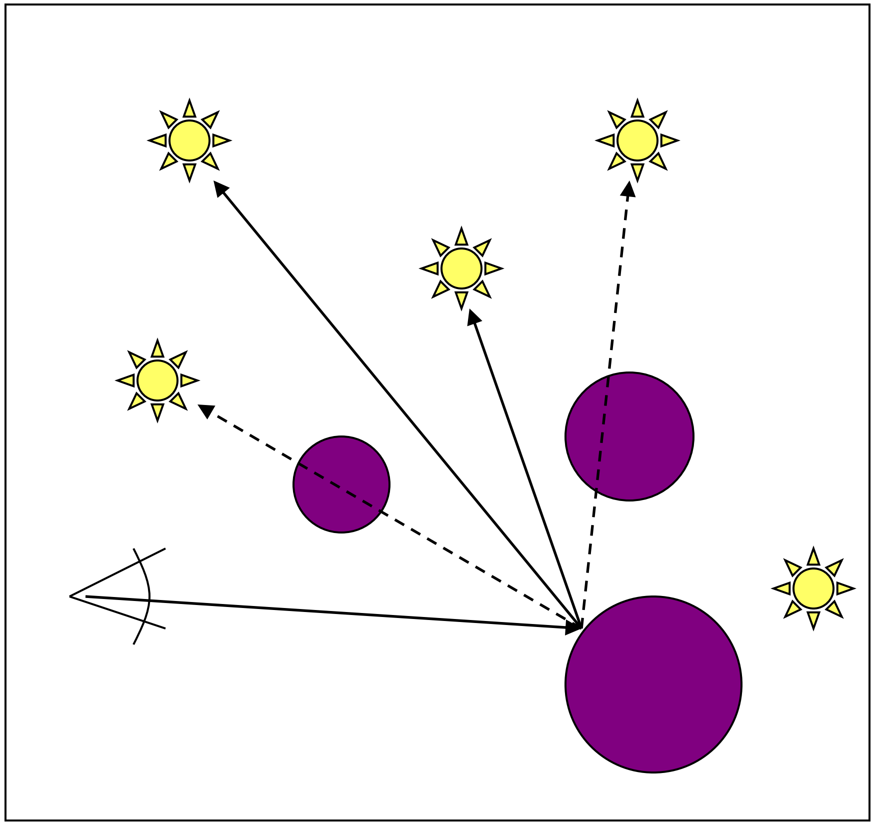
CSE168: Rendering Algorithms

UCSD, Spring 2014

Shadows

- Adding shadows to the ray tracer we have so far is very straightforward
- Instead of just adding the contribution from each light, we first want to check if the light in question is blocked by some other surface
- To do this, we can shoot a *shadow ray* towards the light source
- We want to make sure we don't intersect with things past the light source, so we set the Intersection::HitDistance to be the distance to the light source before we call Scene::Intersect(ray,hit)
- The Light::Illuminate() routine sets a Vector3 &toLight, which is the ray direction to the light source that can be used to generate a shadow ray
- Light::Illuminate() also sets the Vector3 &ltPos, which is the position of the light source. We can use this to compute the initial HitDistance value
- Light::Illuminate() also returns the intensity of the light at the position, so don't bother testing for shadows if the intensity is 0 or if the angle between the hit.Normal and toLight vector is greater than 90 degrees (this happens if hit.Normal.Dot(toLight)<0)

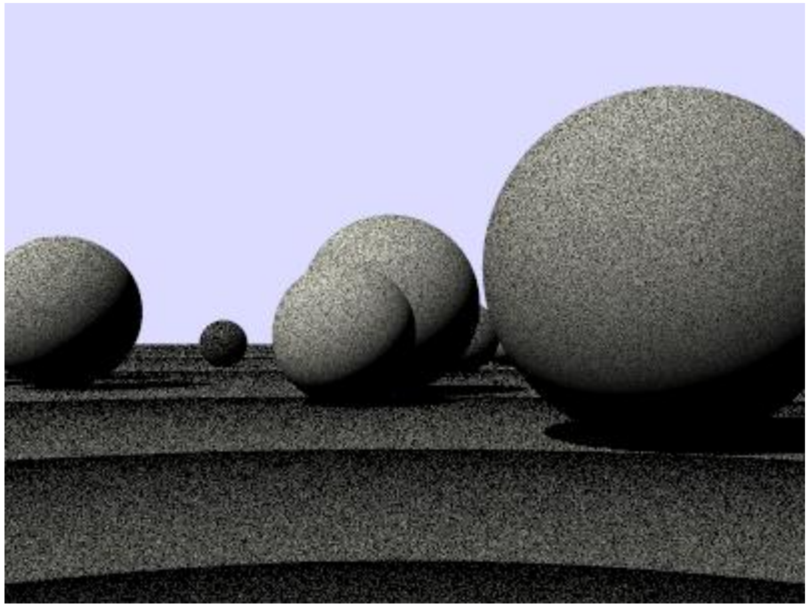
Shadow Rays

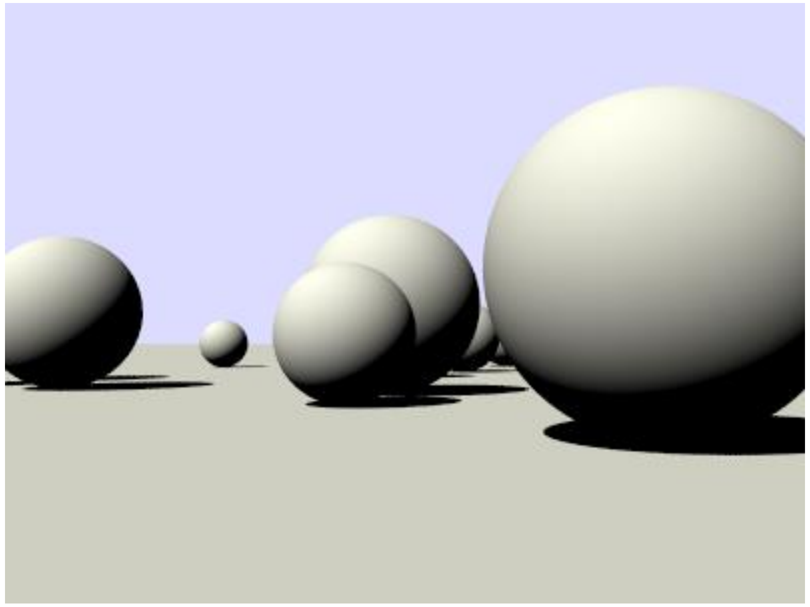


Shadow Ray Offset

- Note that due to floating point precision limitations the shadow ray could actually intersect the original surface itself. This could happen if the shadow ray origin actually ends up being slightly below the actual surface (due to round-off)
- Historically, this was fixed by pushing the ray origin slightly above the surface: origin = position + 0.0001*normal
- A more modern approach to fixing this is to use the original position and just modify the ray intersection routines to reject any intersection where $t<0.0001$ (or some other small epsilon value)
- Also, the HitDistance to the light should be shortened by a similar small distance to avoid hitting the light surface (if there is one)

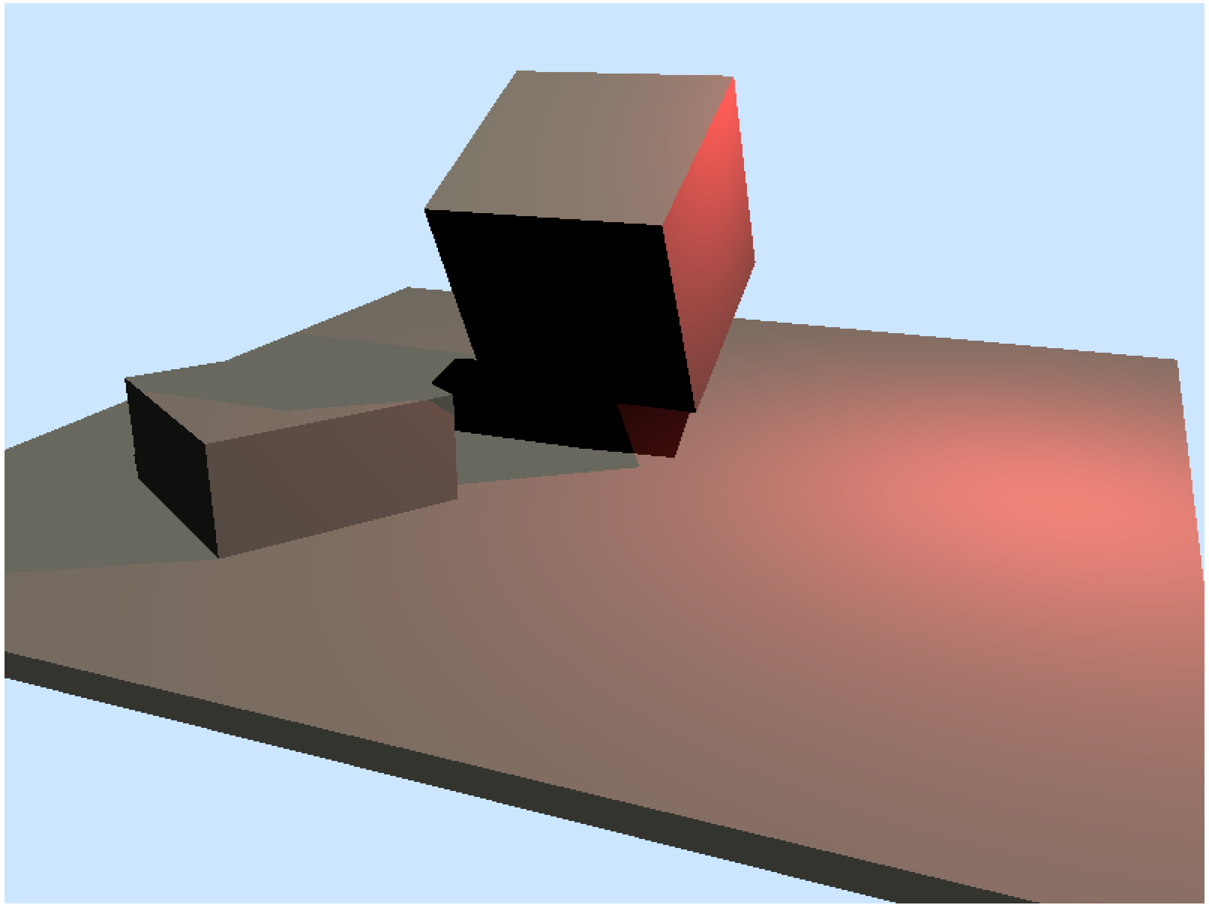
Shadow Ray Offset





Shadows

- This is project 1 with shadows added:



Area Lights

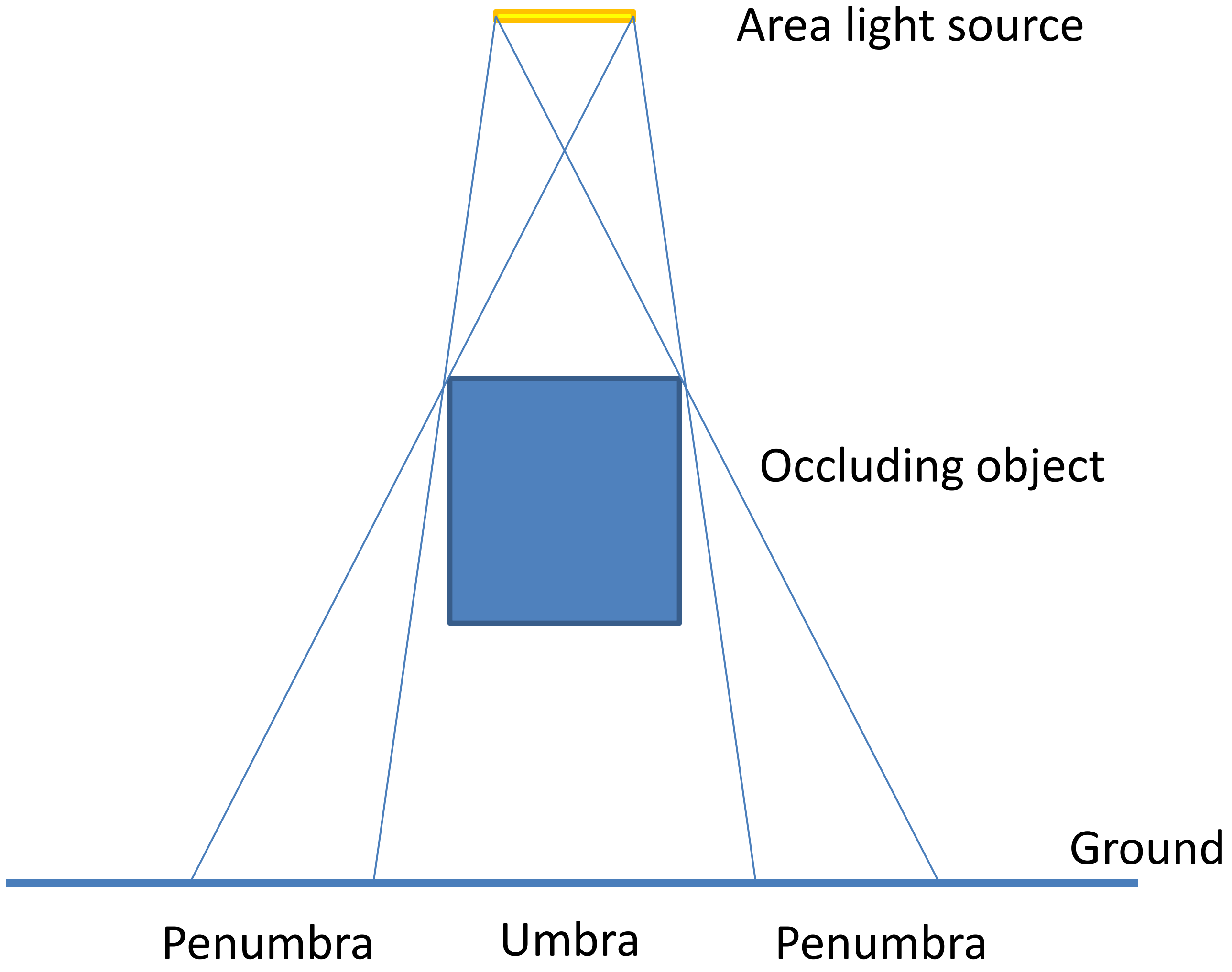
Soft Shadows

- Both point lights and directional lights generate very sharp shadows
- In the real world, we are used to seeing softer edges on shadows
- This is because real lights are not points (or perfectly unidirectional)
- Real lights have some finite area
- In computer graphics, we call these *area lights*
- Larger area lights cast very soft shadows, while smaller area lights cast sharper edged shadows

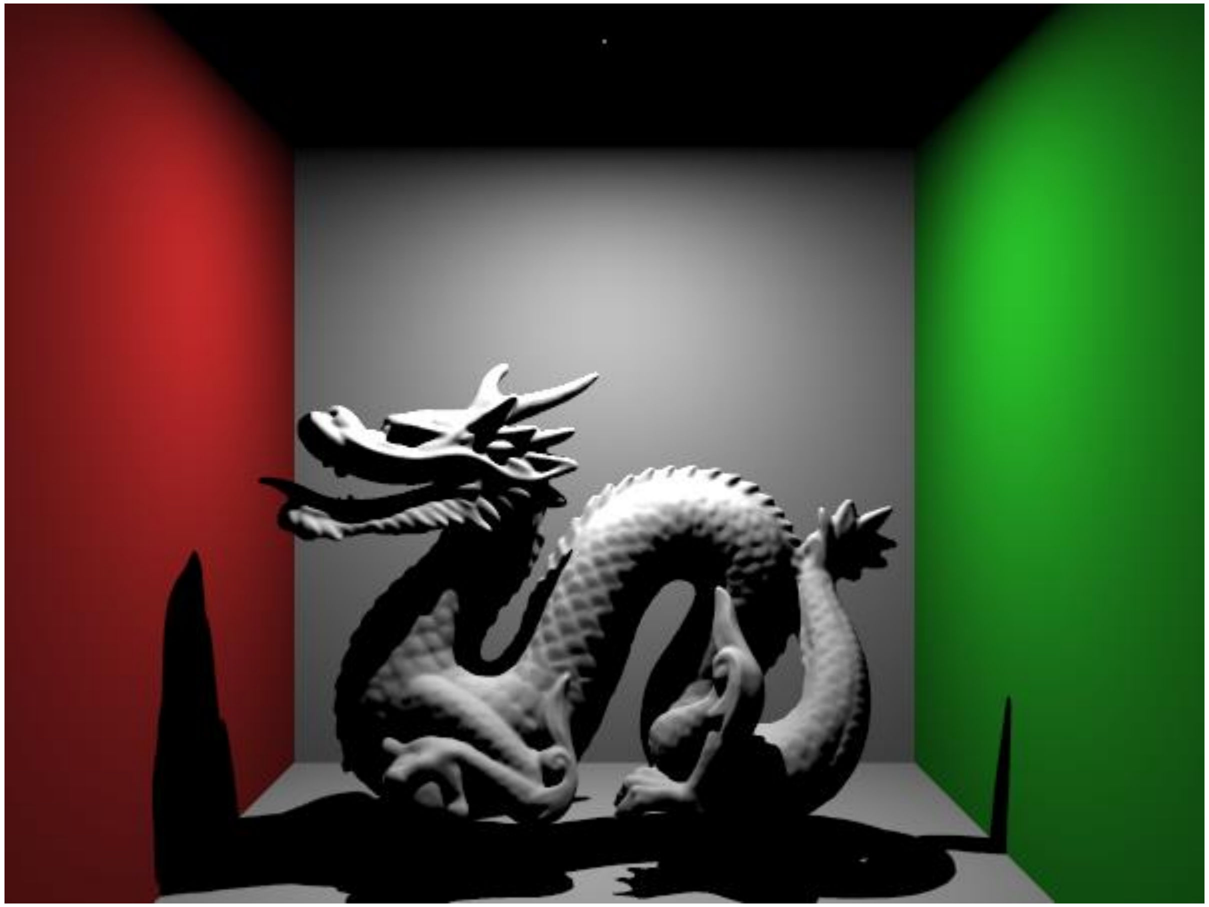
Umbra & Penumbra

- The fully shadowed area is called the *umbra*
- The partially shadowed area on the edge is called the *penumbra*
- During a total eclipse of the sun, you are in the umbra cast by the moon
- During a partial eclipse, you are in the penumbra

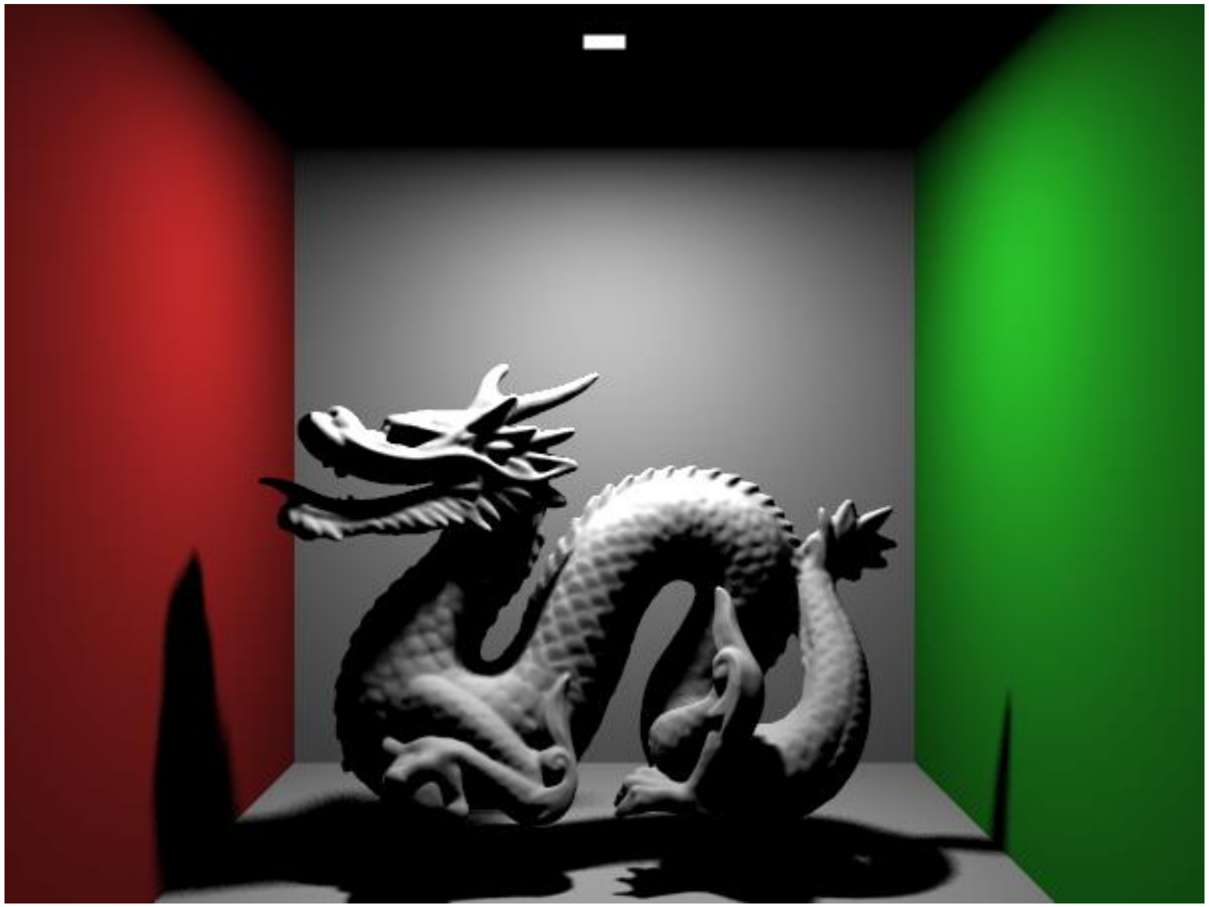
Soft Shadows



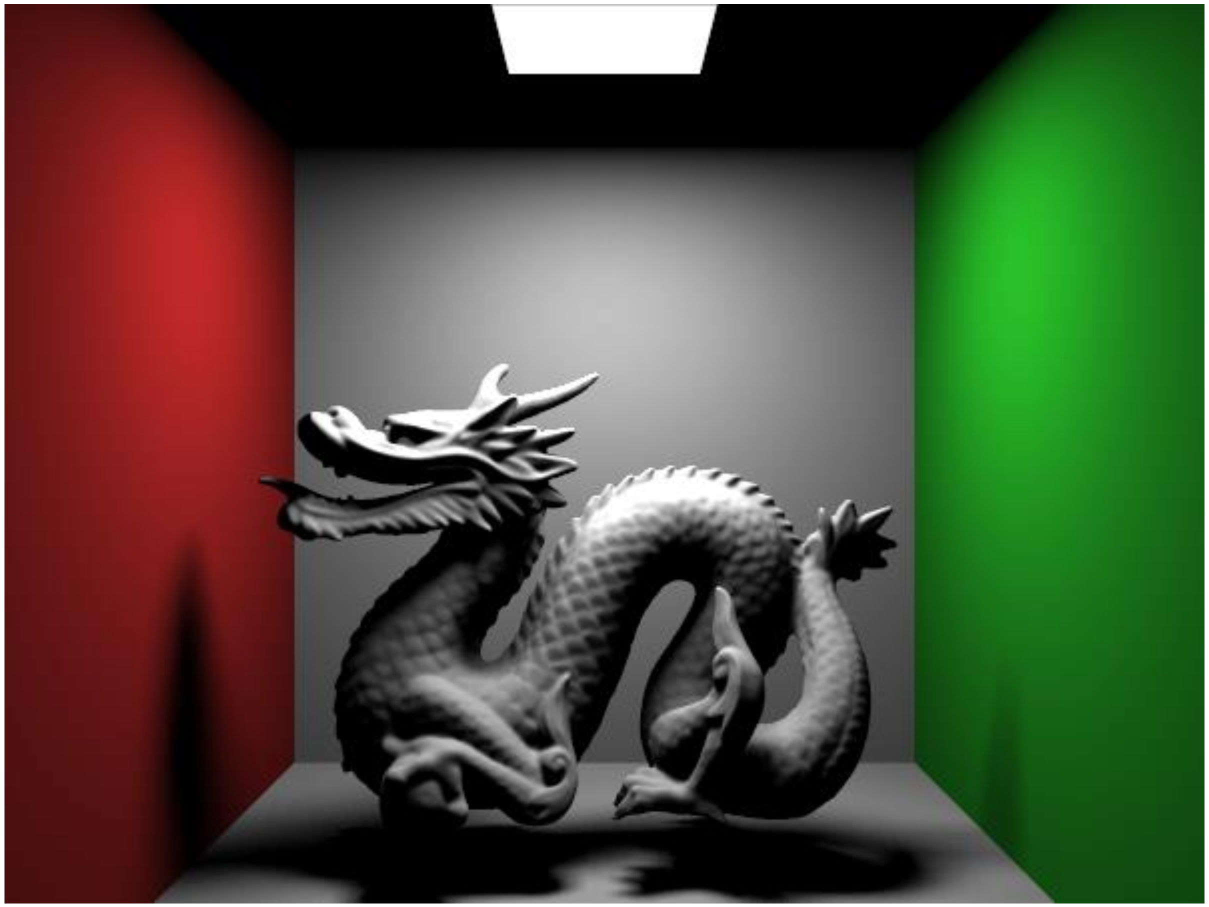
Very Small Area Light



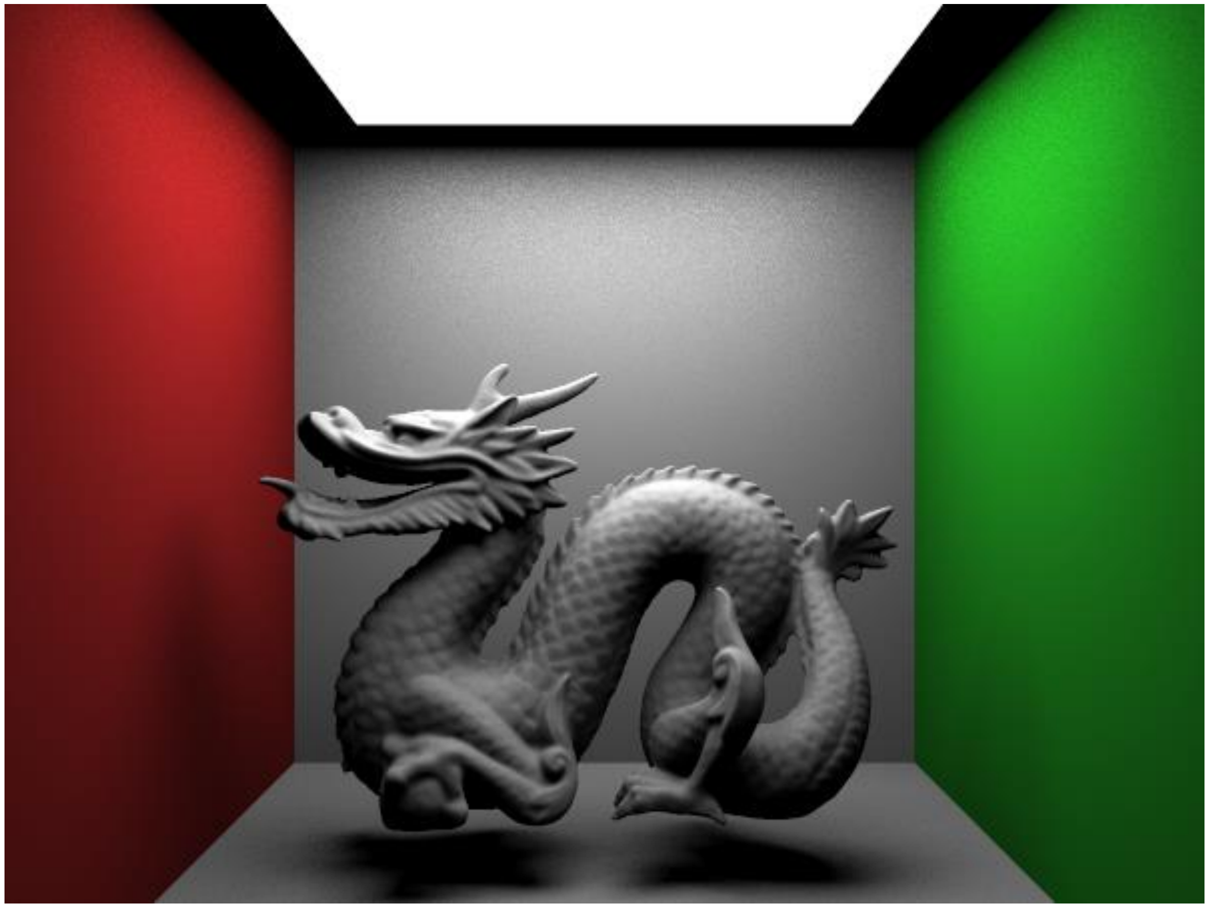
Small Area Light



Medium Area Light



Large Area Light



Ray Tracing Soft Shadows

- In order to render with soft shadows, we can modify our shadow casting routine
- For non-area lights, we test a single shadow ray from the surface point to the light source
- For area lights, we need to test a bunch of rays from the surface point to points scattered across the area of the light, and then average the results

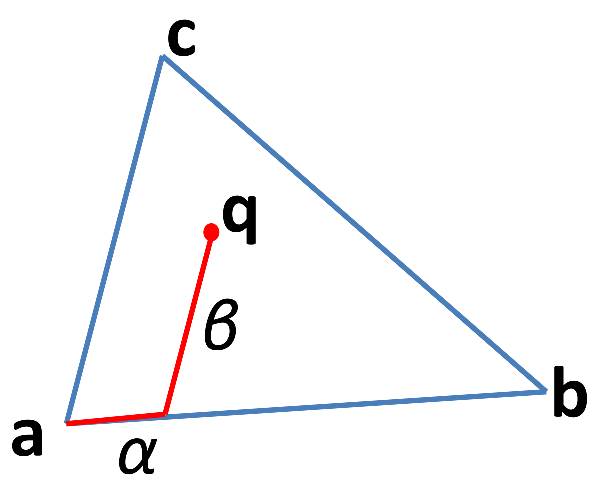
Random Sampling

- In order to generate a bunch of rays to the area light, we need to select several random points on the light
- Assuming that our area lights are modeled as triangles, this would mean that we have to generate a random point on a triangle

Barycentric Coordinates

- Recall our discussion about barycentric coordinates α and β of a triangle, where:

$$\mathbf{q} = \mathbf{a} + \alpha(\mathbf{b}-\mathbf{a}) + \beta(\mathbf{c}-\mathbf{a})$$



$0 < \alpha < 1$

$0 < \beta < 1$

$\alpha+\beta < 1$

Triangle Sampling

- We need to generate random barycentric coordinates α and β
- Let’s say we have a random number generator that generates random numbers equally spaced from 0…1
- As we are sampling a two-dimensional area, we will need two random numbers
- However, we can’t just pick random values for α and β

Triangle Sampling

u = random number from 0…1

v = random number from 0…1

α=sqrt(u)*v

β=1-sqrt(u)

- We will take a closer look at this and some similar examples in a later lecture

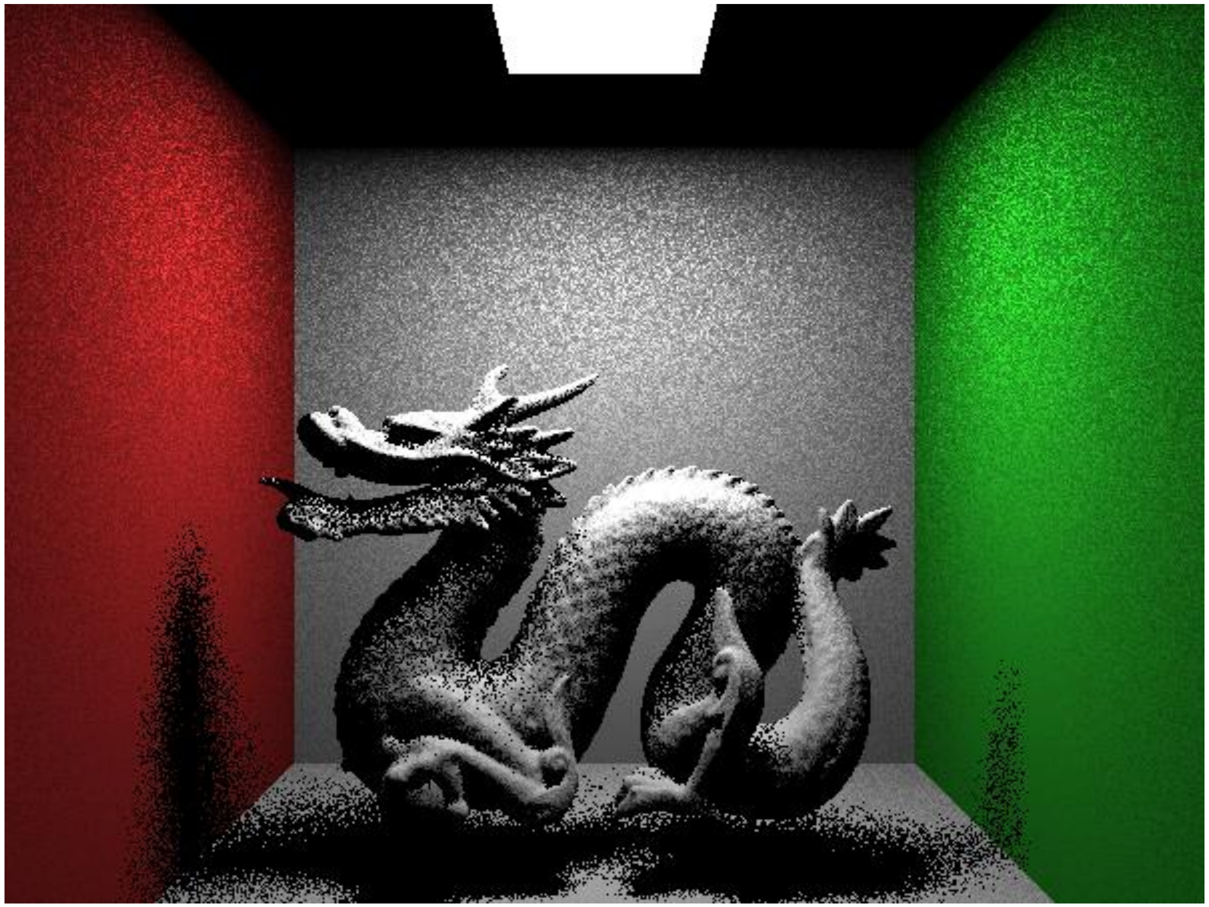
Triangle Sampling

- Let’s say that we are going to shoot 25 shadow rays towards our area light
- We choose 25 random points on the triangle and assume that each of them accounts for 1/25 of the area of the light
- If the light source intensity is defined in terms of intensity/meter^2 so we’ll need to know the total area of the triangle to compute the intensity of one sample
- Also, as we are assuming that the area light is a flat surface, the intensity is going to decrease as the normal of the light turns away from us
- It will decrease by the cosine of the angle between the ray and the light’s normal
- It will also decrease by the inverse square law

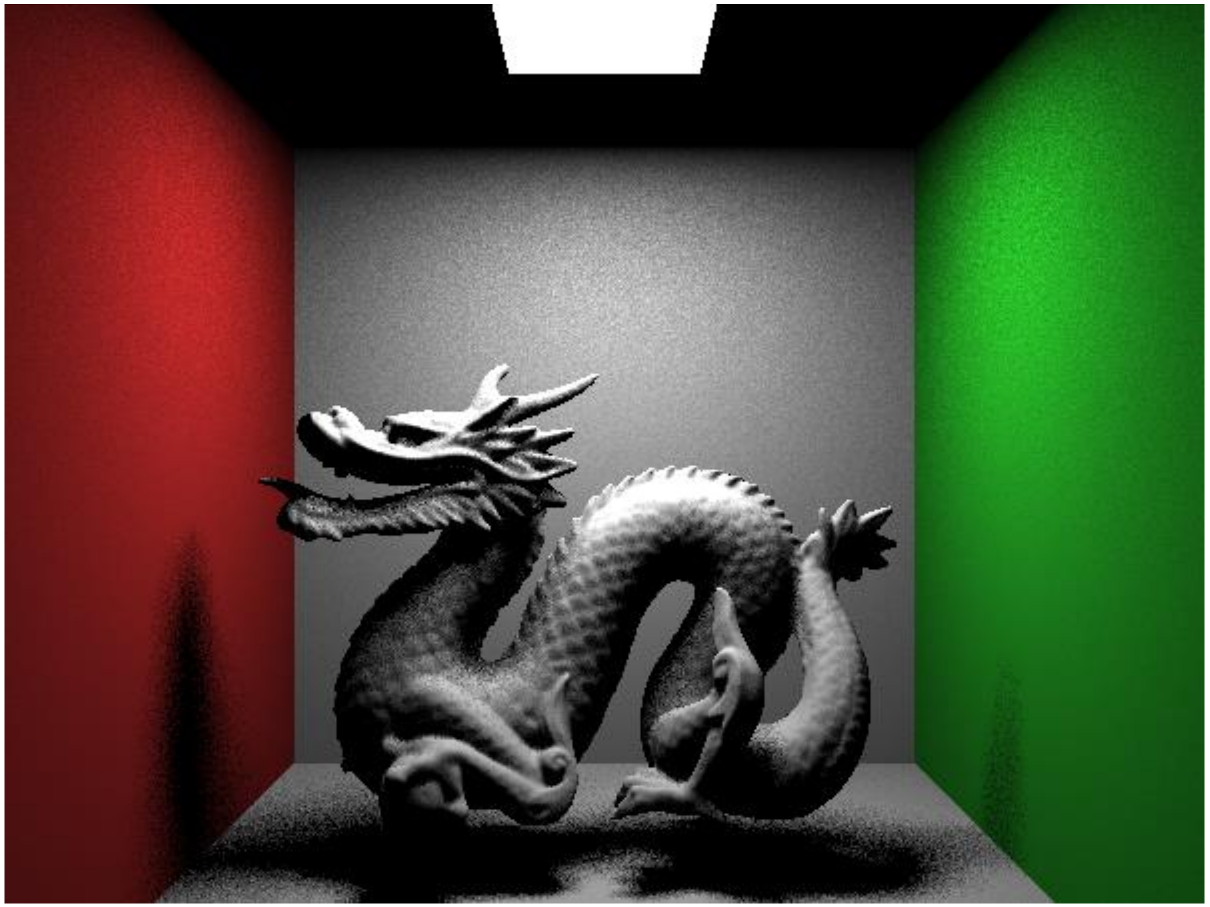
How Many Samples?

- How many shadow rays should we shoot towards our area light?
- This is difficult but important question
- Too few samples will generate a noisy result
- Too many samples will take too long

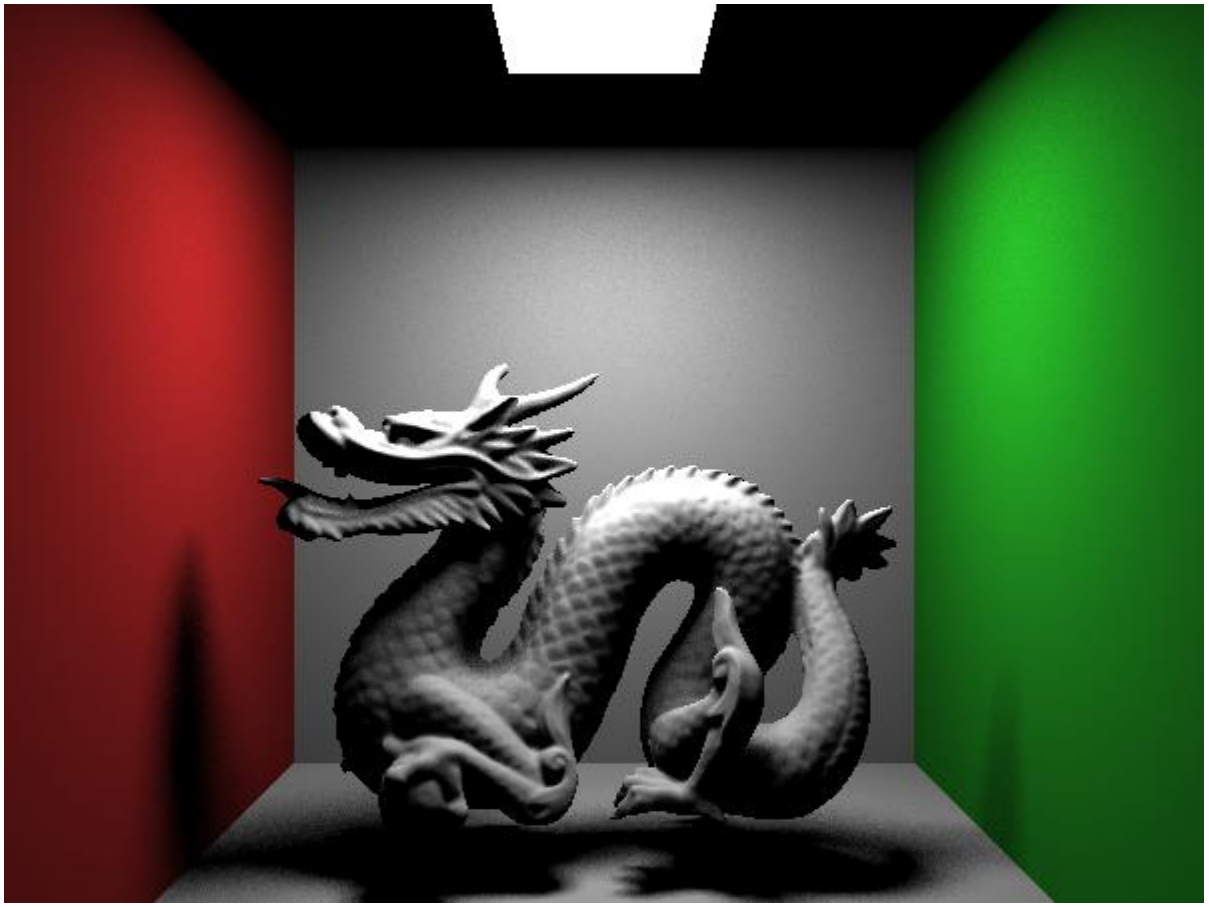
1 Shadow Ray / Pixel



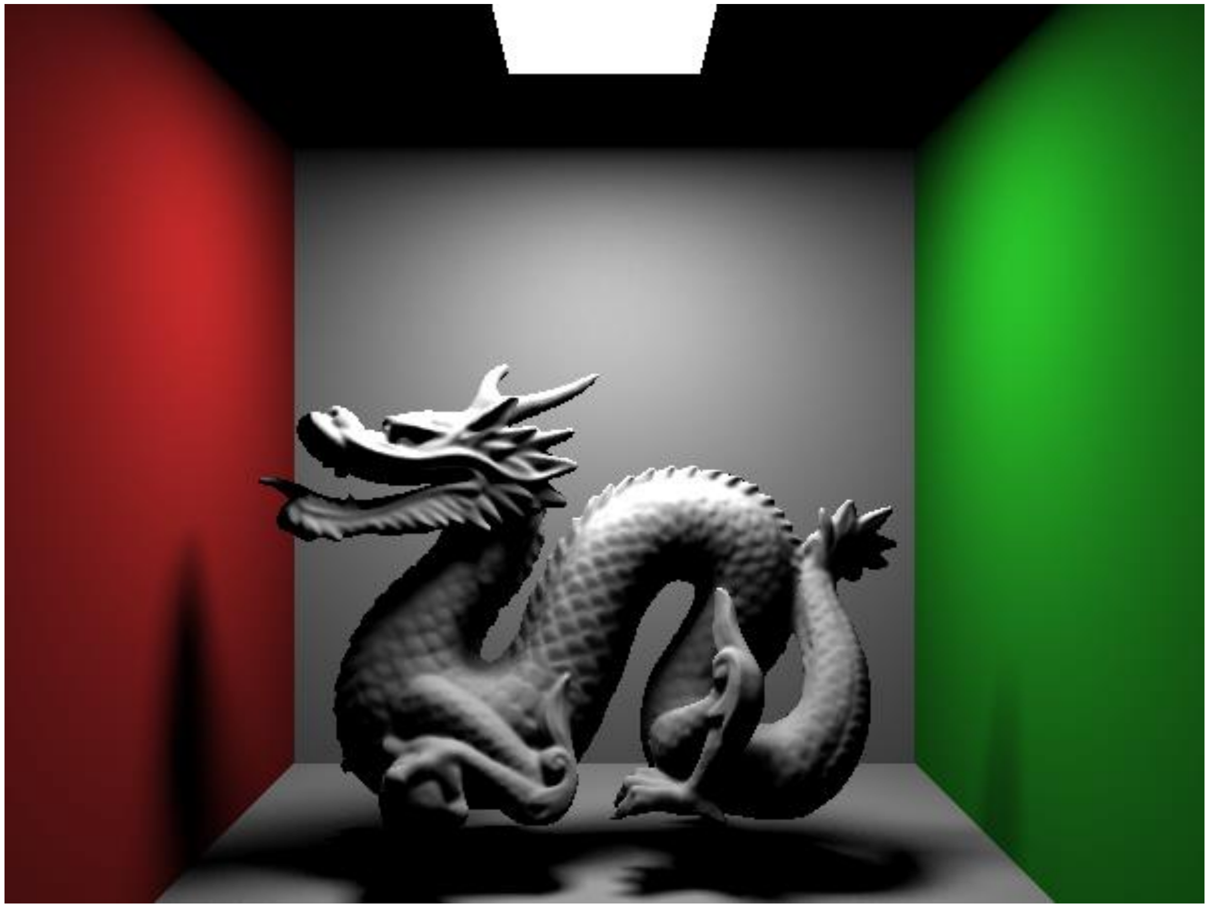
10 Shadow Rays / Pixel



100 Shadow Rays / Pixel



1000 Shadow Rays / Pixel

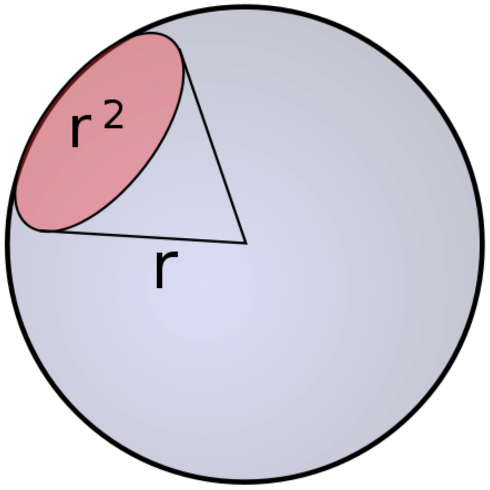


How Many Samples?

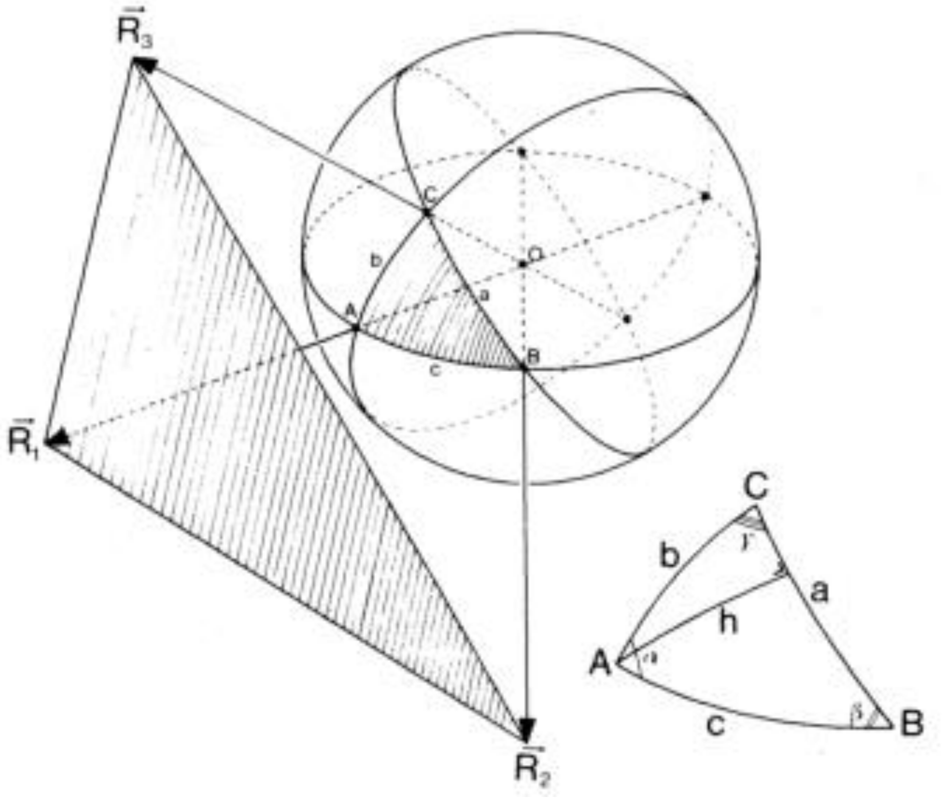
- Unfortunately, we can see that it may require quite a large number of samples to generate a clean image
- A reasonable heuristic would be to generate a number of samples proportional to the size of the light from the point of view of the sample
- We could also scale that by the intensity of the light, so darker lights would use fewer samples (because the variance/noise will increase with brightness)
- In addition, we could scale this by the albedo of the surface material (for the same reason)
- This gives us a reasonable guideline, but it’s not perfect

Solid Angles

- How do we measure the size of a light source from some particular point of view?
- We can look at the *solid angle* that the area light covers
- An angle represents the proportion of a circle (times 2π), and a solid angle represents the proportion of a sphere (times 4π)
- Solid angles are measured in *steradians*



Solid Angle of a Triangle



Solid Angle of a Triangle

- To compute the solid angle of a triangle made up of points **a**, **b**, and **c** (from the point of view of the origin):

$$\Omega = 2 \cdot \operatorname{atan}\left(\frac{\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})}{abc + (\mathbf{a} \cdot \mathbf{b})c + (\mathbf{a} \cdot \mathbf{c})b + (\mathbf{b} \cdot \mathbf{c})a}\right)$$

where $a=|\mathbf{a}|$, etc.

- Also note that the sign will get flipped if the triangle is facing away from the origin
- This formula comes from “The Solid Angle of a Plane Triangle” by A. Van Oosterom and J. Strackee

Many Light Sources

- Some scenes contain *many* lights
- An example would be a city scene at night
- Can we make some optimizations for situations like this?

Many Lights

- One thing we can do is to assume that each light has a finite range
- For example, we can find the distance where intensity drops below some threshold
- A reasonable threshold is 0.5/255, as this would mean that the light's contribution to the final pixel brightness is below the lowest intensity we can display (assuming a 24-bit display)
- We can also build spatial data structures around the volumes associated with the lights, so we can quickly query which lights affect a particular point
- Another option is to randomly select some of the lights weighted by their intensity at the point of interest