CS 8721 Computer Graphics

Ray Tracing

P. Willemsen

University of Minnesota Duluth

April 23, 2013

Outline

Cook Distribution Ray Tracing

Ray tracing is based on sampling light intensity (*radiance*) in the scene.

- Approach up to now has been with 1 sample per pixel
- Not ideal for getting a good evaluation of the radiance integral across the scene

How should we sample

With 1 sample in each pixel center, what happens?

- Case 1 - Sample is fairly representative of the surround

How should we sample

With 1 sample in each pixel center, what happens?

- Case 1 - Sample is fairly representative of the surround

- Case 2 - A couple of choices but which one?

How should we sample

With 1 sample in each pixel center, what happens?

- Case 1 - Sample is fairly representative of the surround

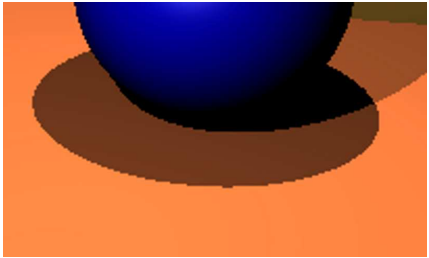
- Case 2 - A couple of choices but which one?

- Case 3 - Now which intensity is the correct choice?

By not sampling well, we introduce potential image artifacts because we did not choose a good representative value for each pixel.

- Aliasing (*jaggies*) - stair step look
- With 1 sample per pixel, sampling is a binary decision.
- Aliasing is a result of sampling too little.
- Also, cannot render shadows produced by area lights well

For example, in the first image to the right, the pixel is either red or black.



Improving Sampling by adding More Samples

Current pixel centered sampling looks like this:

for all pixels $(i, j) \in$ Image **do**
  $c_{ij} =$ computeRayColor($i + 0.5$, $j + 0.5$)
end for

Places the sample in the center of each pixel.

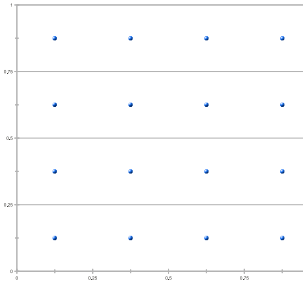
Regular Sampling

We can improve this by increasing the numbers of samples per pixel and aligning them on a grid. So, with $N = \sqrt{rpp}$, and rpp standing for rays per pixel, we get:

for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 0$ to $N - 1$ **do**
    for all $q = 0$ to $N - 1$ **do**
      $c = c +$ computeRayColor($i + (p + 0.5)/N, j + (q + 0.5)/N$)
    end for
  end for
  $c_{ij} = c/N^2$
end for

Regular Sampling

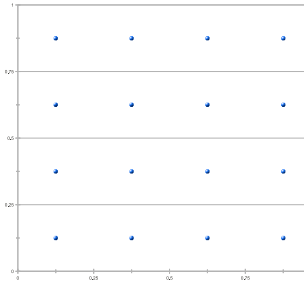
for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 0$ to $N - 1$ **do**
    for all $q = 0$ to $N - 1$ **do**
      $c = c +$ computeRayColor($i + (p + 0.5)/N$,
      $j + (q + 0.5)/N$)
    end for
  end for
  $c_{ij} = c/N^2$
end for



Regular Sampling

Issues?

- Regular artifacts such as moire patterns can show up!
- Moire interference occurs when there is interference between different objects that have high frequency. In this case, the pixels and the sub-pixel sampling.



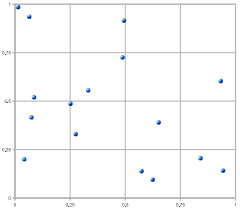
Random Sampling

We can get rid of regular patterns by adding noise to the sampling pattern. In this case, we apply random sampling:

for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 1$ to N^2 **do**
    $c = c +$ computeRayColor($i +$ drand48(), $j +$ drand48())
  end for
  $c_{ij} = c/N^2$
end for

Random Sampling

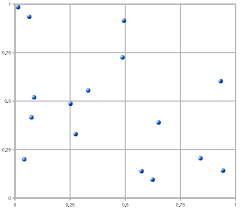
for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 1$ to N^2 **do**
    $c = c +$ computeRayColor($i +$ drand48(), $j +$ drand48())
  end for
  $c_{ij} = c/N^2$
end for



Random Sampling

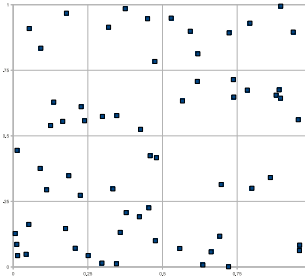
Issues?

- We didn't sample all of the sub pixel regions!



Improving Random Sampling - More Samples

- The only way to improve the pure random sampling is to increase the number of samples!
- May have to increase samples *a lot* to get good coverage!



Jittered or Stratified Sampling

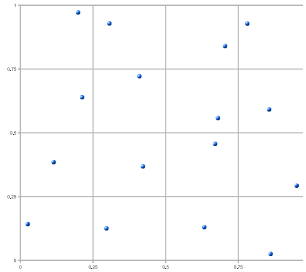
We can get the benefits of the regular grid and random sampling with a hybrid approach:

- Covers all sub pixel regions based on the *rpp*
- Adds noise to help remove regular artifacts

for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 0$ to $N - 1$ **do**
    for all $q = 0$ to $N - 1$ **do**
      $c = c +$ computeRayColor($i + (p+\text{drand48}())/N$, $j + (q+\text{drand48}())/N$)
    end for
  end for
  $c_{ij} = c/N^2$
end for

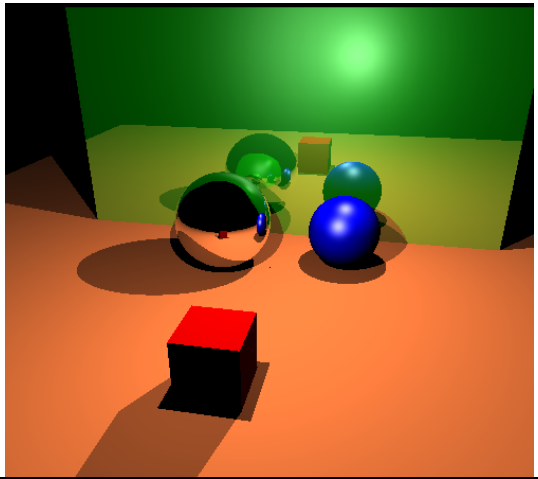
Jittered or Stratified Sampling

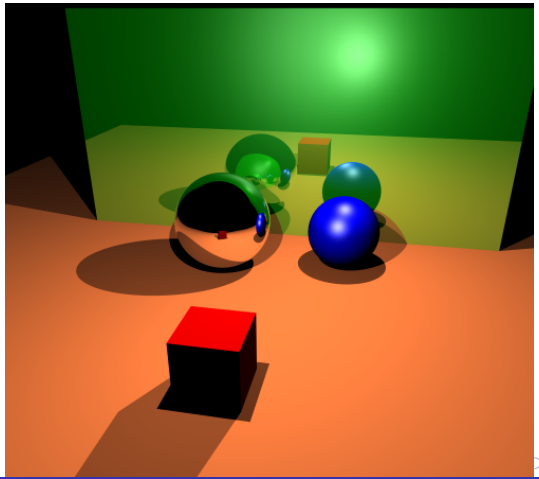
for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  for all $p = 0$ to $N - 1$ **do**
    for all $q = 0$ to $N - 1$ **do**
      $c = c +$ computeRayColor(
      $i + (p +$drand48())$/N$,
      $j + (q +$drand48())$/N$)
    end for
  end for
  $c_{ij} = c/N^2$
end for



Antialiasing - Using Jittered Sampling

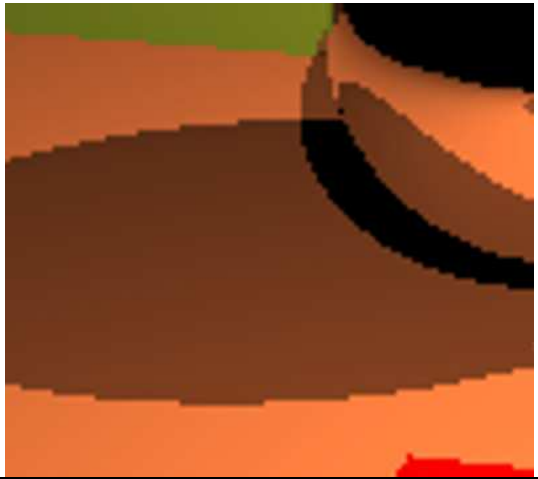
Antialiasing is applied with Jittered (or Stratified) Sampling





Antialiasing - Using Jittered Sampling

Antialiasing is applied with Jittered (or Stratified) Sampling



Applying sampling to Area Lights

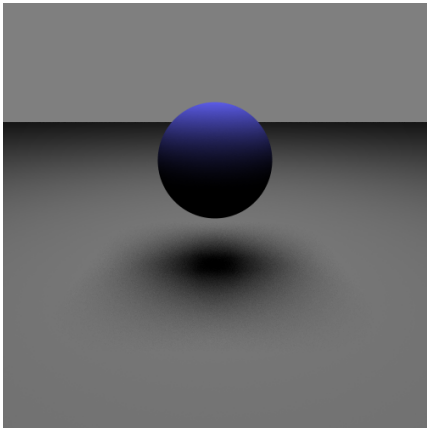
Soft Shadows -

- Area Lights - defined by a center point, a width, a length, and a normal.

Soft Shadows - Area Lights

Tips:

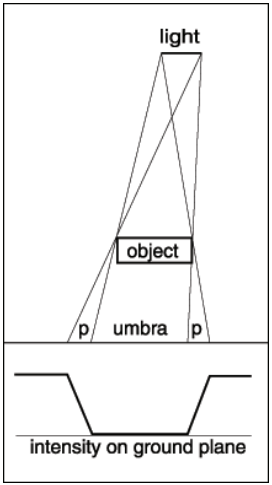
- Apply jittered, stratified sampling to the area of the light
- Each ray from eye samples a different point on the light surface
- Size of area light will increase the softness of the shadow



Soft Shadows - Area Lights

Soft Shadows

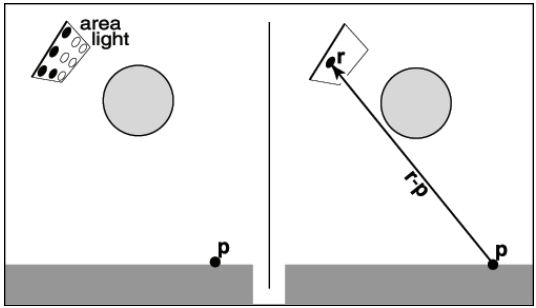
- Composed of dark areas with gradual transition across an unshadowed area.
- *Umbra* - darkest portion of shadow where no light is visible
- *Penumbra* - partially shadowed area in which light has some visibility



Soft Shadows - Area Lights

Using the jittered samples, rays can be sent to different parts of the light:

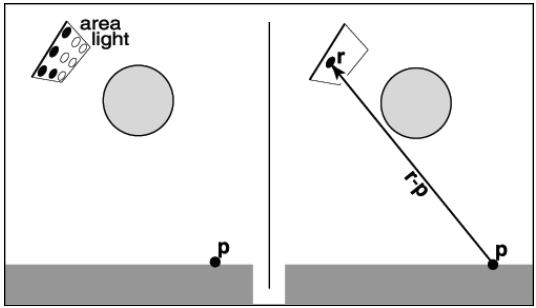
- Each sample samples a single portion of the light
- Values across all the rays sent for each pixel are averaged, resulting in the shadow gradiant
- Tip: Use a jittered, stratified grid to sample the light



Jittered or Stratified Sampling with the Light

Need to be careful when applying the jittered samples to the light!

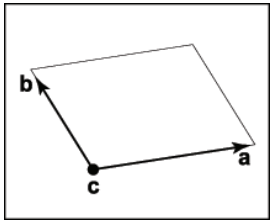
for all pixels $(i, j) \in$ Image **do**
  $c = 0$
  {generate $N = n^2$ jittered 2D points in array $r[]$}
  {generate $N = n^2$ jittered 2D points in array $s[]$}
  {shuffle the points in array $s[]$}
  for all $idx = 0$ to $N - 1$ **do**
    $c = c +$ computeRayColor($i + r[idx].x, j + r[idx].y, s[idx]$)
  end for
  $c_{ij} = c / N$
end for



Soft Shadows - Area Lights

Define a basis for the light!

- Use your basis class to generate a basis for the light.
- Use the light's normal vector as the one input
- $\vec{U}$ and $\vec{V}$ vectors can be used to place the jittered samples
- How?



Simulating Appearance of Rough Objects

Not all shiny objects specularly reflect light pefectly

Goal

Use distributed rays to perturb reflection vectors based on roughness value.

Roughness relates to the glossiness of an ideally specular surface

- Specified in reflective shaders with an optional *roughness* scalar:

  `<roughness>0.3</roughness>`

- Larger values perturb reflection vectors more
- Small values perturb less
- A value of 0 is identical to ideal specular reflection

Simulating Appearance of Rough Objects

Algorithm for glossiness:

if (Has Surface Roughness Specified) **then**

$\vec{r} \leftarrow \vec{ray_{dir}} - 2(\vec{ray_{dir}} \cdot \vec{n}) * \vec{n}$

{Compute Basis for reflection}

$\vec{U}, \vec{V}, \vec{W} \rightarrow$ computeBasisFrom1Vector($\vec{r}$)

{Compute Perturbed Reflection Vector}

float u = -roughness/2.0 + RandomSample.x * roughness;

float v = -roughness/2.0 + RandomSample.y * roughness;

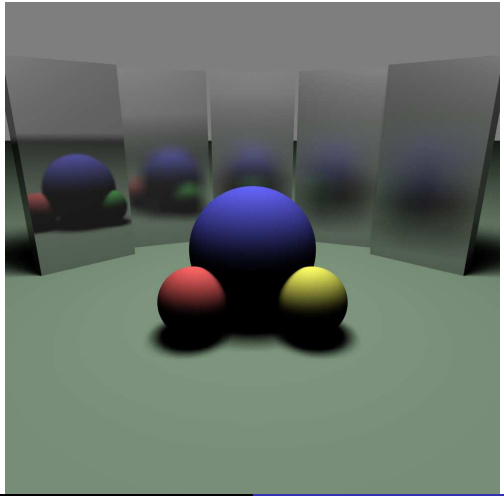
$\vec{r'} \rightarrow \vec{r} + u\vec{U} + v\vec{V}$

{Send our recursive reflection ray, $\vec{r'}$}

end if

Glossiness

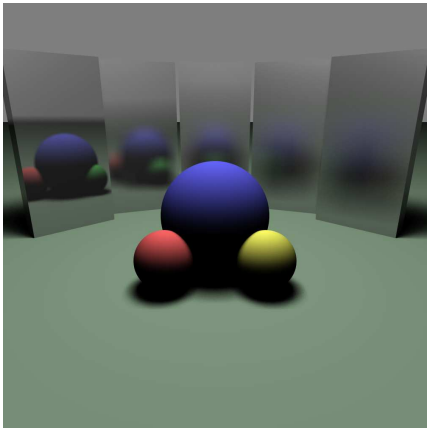
Increasing roughness from left to right:



Glossiness

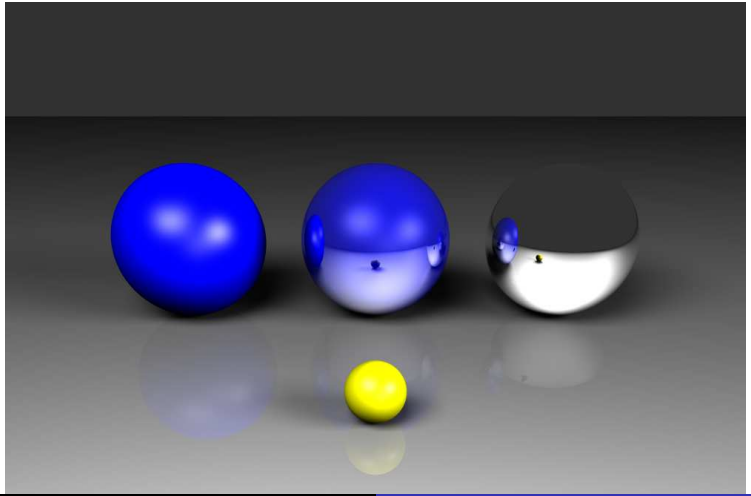
Tips:

- Compute a basis from a single vector - $\vec{r}$
- Use basis, jittered random sample to compute new reflection vector
- Roughness value scales the size of the area to which new reflection vectors can be sampled.

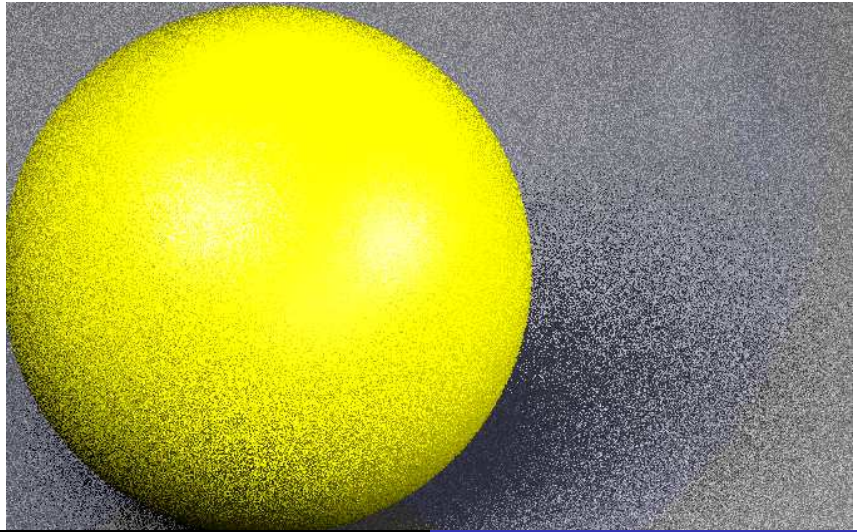


Cook Distribution Ray Tracing

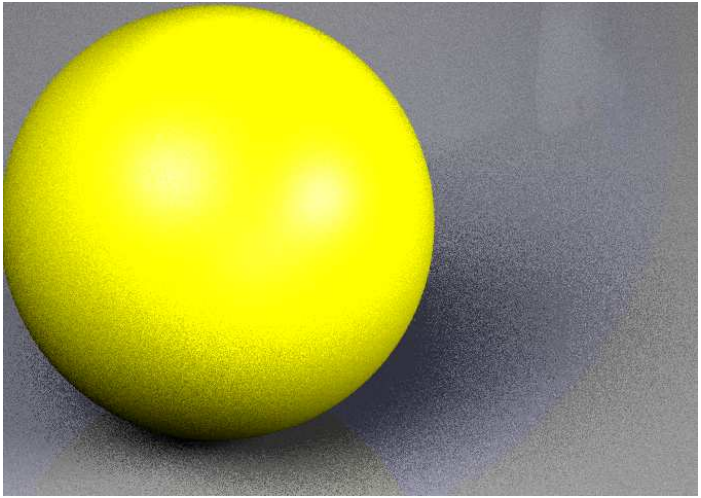
How rays per pixel affects scene quality



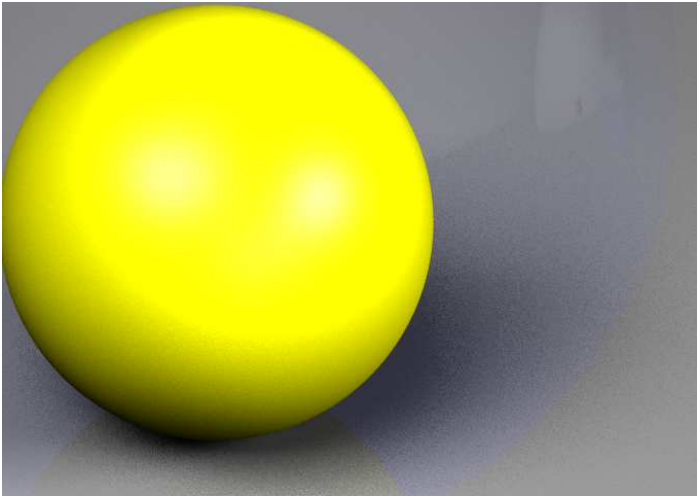
1 RPP



4 RPP



16 RPP



Requirements for PA 2

- Matrix transforms and instanced objects
- Bounding Volume Hierarchies
- Distributed, Cook-style sampling - AA, Soft Shadows, Glossiness
- OBJ Mesh file loading

If you have questions, email Pete or meet in office hours Wed and Thu this week!