In **Whitted Ray Tracing** we computed lighting very crudely
- Phong + specular global lighting

In **Distributed Ray Tracing** we want to compute the lighting as accurately as possible
- Use the formalism of Radiometry
- **Compute irradiance at each pixel** (by integrating all the incoming light)
- Since integrals are can not be done analytically, we will employ **numeric approximations**
Benefits of Distribution Ray Tracing

- Better global diffuse lighting
  - Color bleeding
  - Bouncing highlights
- Extended light sources
- Anti-aliasing
- Motion blur
- Depth of field
- Subsurface scattering
Radiance at a Point

- Recall that radiance (shading) at a surface point is given by
  \[ L(\vec{p}, \vec{d}_e) = \int_{\Omega} \rho(\vec{d}_e, \vec{d}_i) L(\vec{p}, -\vec{d}_i)(\vec{n} \cdot \vec{d}_i) \, d\omega \]

- If we parameterize directions in spherical coordinates and assume small differential solid angle, we get
  \[ L(\vec{p}, \vec{d}_e) = \int_{\phi=0}^{2\pi} \int_{\theta=0}^{\pi} \rho(\vec{d}_e, \vec{d}_i(\phi, \theta)) L(\vec{p}, -\vec{d}_i(\phi, \theta))(\vec{n} \cdot \vec{d}_i(\phi, \theta)) \sin \theta \, d\theta \, d\phi \]
Radiance at a Point

- Recall that radiance (shading) at a surface point is given by
  \[
  L(\mathbf{p}, \mathbf{d}_e) = \int \rho(\mathbf{d}_e, \mathbf{d}_i) L(\mathbf{p}, -\mathbf{d}_i) (\mathbf{n} \cdot \mathbf{d}_i) \, d\mathbf{\omega}
  \]

- If we parameterize directions in spherical coordinates and assume small differential solid angle, we get
  \[
  L(\mathbf{p}, \mathbf{d}_e) = \int_{\phi \in [0, 2\pi]} \int_{\theta \in [0, 2\pi]} \rho(\mathbf{d}_e, \mathbf{d}_i(\phi, \theta)) L(\mathbf{p}, -\mathbf{d}_i(\phi, \theta)) (\mathbf{n} \cdot \mathbf{d}_i(\phi, \theta)) \sin \theta \, d\phi \, d\theta
  \]
  Integral is over all incoming direction (hemisphere)
Irradiance at a Pixel

To compute the color of the pixel, we need to compute total light energy (flux) passing through the pixel (rectangle) (i.e. we need to compute the total irradiance at a pixel)

\[ \Phi_{i,j} = \int_{\alpha_{\text{min}} \leq \alpha \leq \alpha_{\text{max}}} \int_{\beta_{\text{min}} \leq \beta \leq \beta_{\text{max}}} H(\alpha, \beta) \, d\alpha \, d\beta \]

Integrals is over the extent of the pixel.
Numerical Integration (1D Case)

- **Remember**: integral is an area under the curve
- We can approximate any integral numerically as follows

\[
\int_{0}^{D} f(x) \, dx \approx \sum_{i=1}^{N} d_i f(x_i)
\]
Numerical Integration (1D Case)

- **Remember:** integral is an area under the curve
- We can approximate any integral numerically as follows

\[
\int_{0}^{D} f(x) \, dx \approx \sum_{i=1}^{N} \frac{D}{N} f(x_i)
\]
Numerical Integration (1D Case)

- **Problem:** what if we are really unlucky and our signal has the same structure as sampling?

\[
\int_{0}^{D} f(x) \, dx \approx \sum_{i=1}^{N} \frac{D}{N} f(x_i)
\]
Monte Carlo Integration

- **Idea:** randomize points $x_i$ to avoid structured noise (e.g. due to periodic texture)

- Draw $N$ random samples $x_i$ independently from uniform distribution $Q(x) = \mathbb{U}[0,D]$ (i.e. $Q(x) = 1/D$ is the uniform probability density function)

- Then approximation to the integral becomes
  \[
  \frac{1}{N} \sum w_i f(x_i) \approx \int f(x) \, dx, \text{ for } w_i = \frac{1}{Q(x_i)}
  \]

- We can also use other $Q$’s for efficiency !!! (a.k.a. importance sampling)
Monte Carlo Integration

Then approximation to the integral becomes

\[ \frac{1}{N} \sum w_i f(x_i) \approx \int f(x) \, dx, \text{ for } w_i = \frac{1}{Q(x_i)} \]

- We can also use other Q’s for efficiency !!! (a.k.a. importance sampling)
Stratified Sampling

- **Idea:** combination of uniform sampling plus random jitter
- Break domain into $T$ intervals of widths $d_t$ and $N_t$ samples in interval $t$

Integral approximated using the following:

$$\sum_{t=1}^{T} \frac{1}{N_t} \sum_{j=1}^{N_t} d_t f(x_{t,j})$$
Stratified Sampling

- If intervals are uniform $d_t = D/T$ and there are same number of samples in each interval $N_t = N/T$ then this approximation reduces to:

$$\sum_{t=1}^{T} \sum_{j=1}^{N_t} \frac{D}{N} f(x_{t,j})$$

- The interval size and the # of samples can vary !!!

- Integral approximated using the following:

$$\sum_{t=1}^{T} \frac{1}{N_t} \sum_{j=1}^{N_t} d_t f(x_{t,j})$$
Back to Distribution Ray Tracing

- Based on one of the approximate integration approaches we need to compute
- Let’s try uniform sampling

\[
L(\vec{p}, \vec{d}_e) = \int_{\phi \in [0, 2\pi]} \int_{\theta \in [0, 2\pi]} \rho(\vec{d}_e, \vec{d}_i(\phi, \theta)) L(\vec{p}, -\vec{d}_i(\phi, \theta)) (\vec{n} \cdot \vec{d}_i(\phi, \theta)) \sin \theta \, d\theta \, d\phi
\]

\[
\approx \sum_{m=1}^{M} \sum_{n=1}^{N} \rho(\vec{d}_e, \vec{d}_i(\phi_m, \theta_n)) L(\vec{p}, -\vec{d}_i(\phi_m, \theta_n)) (\vec{n} \cdot \vec{d}_i(\phi_m, \theta_n)) \sin \theta \Delta \theta \Delta \phi
\]

where

\[
\theta_n = \left( n - \frac{1}{2} \right) \Delta \theta
\]

\[
\phi_m = \left( m - \frac{1}{2} \right) \Delta \phi
\]

\[
\Delta \theta = \frac{\pi}{2M}
\]

\[
\Delta \phi = \frac{2\pi}{N}
\]

midpoint of the interval (sample point)  
Interval width
Importance Sampling in Distribution Ray Tracing

- **Problem:** Uniform sampling is too expensive (e.g. 100 samples/hemisphere with depth of ray recursion of 4 => $100^4 = 10^8$ samples per pixel ... with $10^5$ pixels => $10^{15}$ samples)

- **Solution:** Sample more densely (using importance sampling) where we know that effects will be most significant
  - Direction toward point or extended light source are significant
  - Specular and off-axis specular are significant
  - Texture/lightness gradients are significant
  - Sample less with greater depth of recursion
Importance Sampling

- **Idea:** randomize points $x_i$ to avoid structured noise (e.g. due to periodic texture)

\[
\frac{1}{N} \sum w_i f(x_i) \approx \int f(x) dx \quad \text{for} \quad w_i = \frac{1}{Q(x_i)}
\]
Shadows in Ray Tracing

- Recall, we shoot a ray towards a light source and see if it is intercepted.

\[ \vec{c}_k = -\vec{d}_{i,j} \]

Images from the slides by Durand and Cutler
Anti-aliasing in Distribution Ray Tracer

- Lets shoot multiple rays from the same point and attenuate the color based on how many rays are intercepted.

\[ \vec{c}_k = -\vec{d}_{i,j} \]

Same works for anti-aliasing of Textures!!!
Anti-aliasing by Deterministic Integration

- **Idea:** Use multiple rays for every pixel

- **Algorithm**
  - Subdivide pixel \((i,j)\) into squares
  - Cast ray through square centers
  - Average the obtained light

- Susceptible to structured noise, repeating textures
Anti-aliasing by Monte Carlo Integration

- **Idea:** Use multiple rays for every pixel

- **Algorithm**
  - Randomly sample point inside the pixel \((i, j)\)
  - Cast ray through square centers
  - Average the obtained light

- **Does not** suffer from structured noise, repeating textures
How many rays do you need?

1 ray/light  
10 ray/light  
20 ray/light  
50 ray/light

Let's shoot multiple rays from the same point and attenuate the color based on how many rays are intercepted.

\[ \mathbf{c}_k = -\mathbf{d}_{i,j} \]

Images from the slides by Durand and Cutler.
Antialiasing – Supersampling

Images from the slides by Durand and Cutler
Specular Reflections

- Recall, we had to shoot a ray in a perfect specular reflection direction (with respect to the camera) and get the radiance at the resulting hit point.

\[ \vec{c}_k = -\vec{d}_{i,j} \]

\[ \vec{r}_k = 2(\vec{s}_k \cdot \vec{n}_k)\vec{n}_k - \vec{s}_k \]

\[ \vec{m}_s = 2(\vec{c}_k \cdot \vec{n}_k)\vec{n}_k - \vec{c}_k \]
Specular Reflections with DRT

- Same, but shoot multiple rays

\[ \vec{c}_k = -\vec{d}_{i,j} \]

\[ \vec{r}_k = 2(\vec{s}_k \cdot \vec{n}_k)\vec{n}_k - \vec{s}_k \]

Spread is dictated by BRDF

Perfect Reflections (Metal)

Perfect Reflections (glossy polished surface)
So far with our Ray Tracers we only considered **pinhole camera model** (no lens)
- or alternatively, lens, but tiny aperture
Depth of Field

- So far with our Ray Tracers we only considered pinhole camera model (no lens)
  - or alternatively, lens, but tiny aperture
- What happens if we put a lens into our “camera”
  - or increase the aperture
- Remember the thin lens equation?

\[
\frac{1}{f} = \frac{1}{z_0} + \frac{1}{z_1}
\]
Depth of Field

- So far with our Ray Tracers we only considered pinhole camera model (no lens)
  - or alternatively, lens, but tiny aperture
- What happens if we put a lens into our “camera”
  - or increase the aperture
- Remember the thin lens equation?

\[
\frac{1}{f} = \frac{1}{z_0} + \frac{1}{z_1}
\]
Changing the focal-length in DRT

Increasing focal length

50m
1m
0.5m
0.25m*

*=dist in focus
f-3.5

50m
1m
0.5m*
0.25m

*=dist in focus
f-3.5

50m
1m
0.5m
0.25m

*=dist in focus
f-3.5

50m*
1m
0.5m
0.25m

*=dist in focus
f-3.5

220x400 pixels
144 samples per pixel
~4.5 minutes to render

z_0
z_1

optical axis
Changing the aperture in DRT

decreasing aperture

*—dicated in focus  
f-2.8  

*—dicated in focus  
f-5.6  

*—dicated in focus  
f-11.0  

*—dicated in focus  
f-22.0  

optical axis

220x400 pixels  
144 samples per pixel  
~4.5 minutes to render
Depth of Field
Depth of Field
Depth of Field
Depth of Field
Depth of Field
Camera Shutter

- We ignored the fact that **it takes time to form the image**
  - We ignored this for radiometry

- During that time the shutter is open and light is collected
  - We need to **integrate temporally**, not only spatially

\[ \int \int \int H(\alpha, \beta, t) \, d\alpha \, d\beta \, dt \]
Motion Blur
Motion Blur

Long Exposure Photography
Motion Blur (long exposures)

Golden Gate Bridge
30 sec. exposure @ f4

Bodie State Park
30 min. exposure @ f4
Motion Blur (short exposures)

Doc Edgerton, 1936
Sub-surface Scattering
Sub-surface Scattering

Bidirectional Surface Scattering Reflectance Distribution Function

Rendering with BRDF

Rendering with RSSRDF

H. W. Jensen
Bidirectional Surface Scattering Reflectance Distribution Function

[Images taken from Wikipedia]
Semi-Transparencies

Image form http://www.graphics.cornell.edu/online/tutorial/raytrace/
Caustics

- Hard to do in Distribution Ray Tracing
  - Why?
Caustics

- Hard to do in Distribution Ray Tracing
  - Why?

Hard to come up with a good importance function for sampling,
Hence, VERY VERY slow
Caustics

- Often done using bi-directional ray tracing (a.k.a. photon mapping)
  - Shoot light rays from light sources
  - Accumulate the amount of light (radiance) at each surface
  - Shoot rays through image plane pixels to “look-up” the radiance (and integrate irradiance over the area of the pixel)
Photon Mapping

- Simulates **individual photons**
  - In DTR we were simulating radiance (flux)
- Photons are emitted from light sources
- Photons bounce off of specular surfaces
- Photons are **deposited on diffuse surfaces**
  - Held in a 3-D spatial data structure
  - Surfaces need not be parameterized
- Photons **collected by ray tracing** from eye
Photons

- A photon is a particle of light that carries flux, which is encoded as follows:
  - magnitude (in Watts) and color of the flux it carries, stored as an RGB triple
  - location of the photon (on a diffuse surface)
  - the incident direction (used to compute irradiance)

- **Example** (point light source, photons emitted uniformly)
  - Power of source (in Watts) distributed evenly among photons
  - Flux of each photon equal to source power divided by total # of photons
  - 60W light bulb would sending 100 photons, will result in 0.6 W per photon
How does this actually work?

Special data structures are required to do fast look-up (KD-trees)
Photon Mapping Results

Radiance estimate using 50 photons

Radiance estimate using 500 photons