SIGGRAPH2008

Irradiance Caching Algorithm

Greg Ward
Dolby Canada

Spatial Coherence of Indirect

- In nearly all cases, indirect illumination changes slowly over surfaces
- Cost of indirect sampling may therefore be reduced through interpolation

Radiosity techniques work for the same reason -- and have trouble with point light sources because direct illumination does *not* change slowly over surfaces.

Two-bounce *Radiance* Rendering

Radiance is a physically-based renderer that has been around for over 20 years, and the irradiance cache was in the first public release in 1989.

Indirect Irradiance (two bounces)

Shown here is the indirect (RGB) irradiance, which changes slowly over flat surface areas, and more rapidly over curved regions. Notice the saturation of red near the chairs, though their final color is not shown. Depicted is only the light arriving at surfaces after one or more bounces.

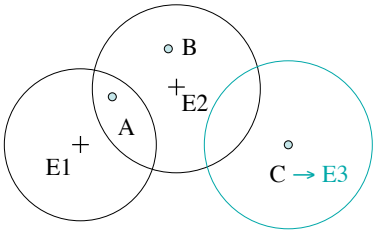
Irradiance Caching Idea (1)

- If interpolating indirect irradiance works, why not precompute it at selected points?
- Better still, why not compute it on an as-needed basis?

I didn't coin the term "irradiance caching" -- that was someone else's good idea. I just called it "lazy evaluation," being too lazy myself to think up a good name.

Irradiance Caching Idea (2)

- Lazy evaluation scheme
- Point A interpolates
- Point B extrapolates
- Point C calculates

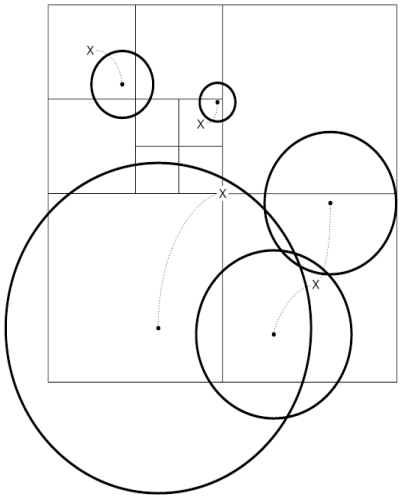


Q: How do we find nearby values?

The E's are evaluation points. E1 and E2 shown here are previous evaluations, where E3 is a new evaluation triggered at query position C.

Octree Cache Lookup

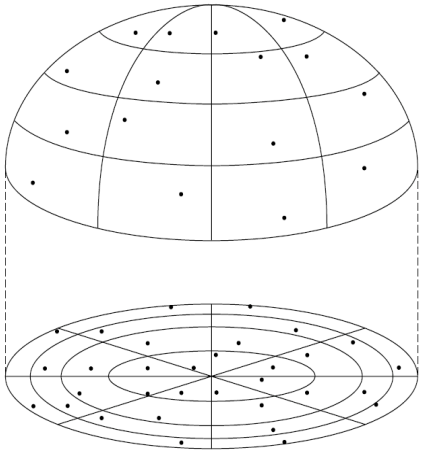
- Each node contains list of indirect irrad. values with valid radii less than width
- Lookup proceeds down octree until all values within valid radius are found



Using an octree that is independent of surfaces avoids placing restrictions on the geometric representation. It even works for volume data, sort of…

Indirect Irradiance Calculation

- Stratified Monte Carlo sampling over hemisphere
- Breakup into altitude and azimuth simplifies gradient computation (later)



Stratified sampling reduces noise in the results at no extra cost, without adding bias. Divisions are placed to maintain constant projected areas on unit circle (Nusselt analog).

Irradiance Computation

$$E_{ind} = \iint L_{ind}(\theta_i,\phi_i)\cos\theta_i \sin\theta_i d\theta_i d\phi_i$$

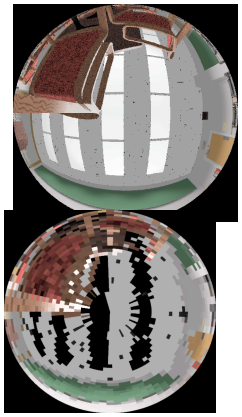
$$E = \left(\frac{\pi}{M \cdot N}\right) \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} L_{j,k}$$

$L_{j,k}$ is the indirect radiance in the direction $(\theta_j,\phi_k) = \left(\text{asin}\sqrt{\frac{j+X_j}{M}}, 2\pi\frac{k+Y_k}{N}\right)$

Integral equation may be converted to equal-weighted sum of selected “measures” using standard Monte Carlo integration techniques (inverting cosine projection). X_j and Y_k are uniformly distributed random variables between 0 and 1. To compute indirect irradiance, we count as zero any samples that intersect a light source.

Indirect Irradiance Example

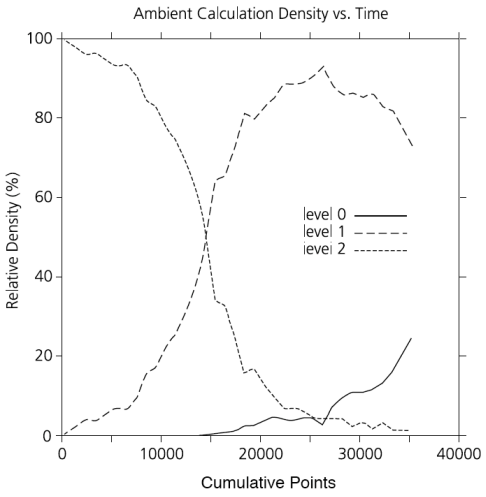
- Send hemisphere ray samples
- Do not count direct sources
- Deeper levels use fewer samples
- Main *Radiance* controls:
  - **-ab** number of bounces
  - **-ad**, **-as** samples per hemisphere
  - **-aa** interpolation accuracy
  - **-ar** spatial resolution



The top image shows a hemispherical equal-contribution projection from the floor near one of the chairs in the conference room model. The bottom image shows the actual shape and number of samples sent for a particular indirect irradiance computation. Note that the sources are black, and the sampling areas follow altitude and azimuth divisions. Actual sample placement within each area is random.

Irradiance Cache Growth

- At deeper levels in the ray tree, growth is limited by reuse of values
- Average of upper levels may be used to estimate constant "ambient" term



In a multi-bounce calculation, the deepest level is filled first, as the initial sample starts a hemisphere calculation, which begets another at the next level, and so on. The plot shows a three-bounce calculation, which starts by filling in the "level 2" cache, then spends time on the "level 1" cache before finally reaching the "level 0" cache -- the final indirect irradiance.

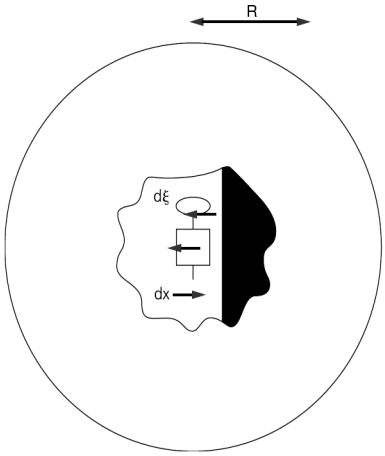
Irradiance Value Spacing & Weights

- Nearby geometry affects irradiance constancy (I.e., gradient)
- Surface curvature also influences gradient
- How can we quantify these to decide how closely to space our values?
- Can the same criteria be used to determine weights for interpolation?

We have an intuition about the behavior of indirect irradiance -- we need to translate this into an algorithm that is robust and reliable.

Split Sphere Approximation

- Target worst case
- Half light, half dark sphere
- Point at center, looking at break
- Q: How does irradiance change?



The “split sphere” is not technically a “worst case” -- it’s just a “pretty bad case.” It is better to qualify this as an assumption. I.e., we assume our actual environment will behave no worse than the split sphere. If it does, our technique won’t necessarily break, but it won’t be as accurate as we hoped, either.

Split Sphere Gradients

$$\varepsilon \le \left| \frac{\partial E}{\partial x}(x - x_0) + \frac{\partial E}{\partial \xi}(\xi - \xi_0) \right|$$

$$\varepsilon \le \frac{4}{\pi}\frac{E}{R}\left|x - x_0\right| + E\left|\xi - \xi_0\right|$$

$$\varepsilon(\vec{P}) \le \frac{4}{\pi} E \frac{\|\vec{P} - \vec{P}_0\|}{R_0} + E\sqrt{2 - 2\vec{N}(\vec{P}) \cdot \vec{N}(\vec{P}_0)}$$

R_0 = average distance to surfaces at $\vec{P}_0$

The first two inequalities bound the split sphere partial derivatives. The final equation uses these bounds to derive a “pretty bad case” error estimate for movement and rotation based on this first-order analysis. The calculation of R0 uses a harmonic mean to neighboring surfaces as measured by our ray samples. A harmonic mean is preferred since the radius appears in the denominator of our equation. Obviously, zero ray lengths are forbidden.

Generalized Error Estimate

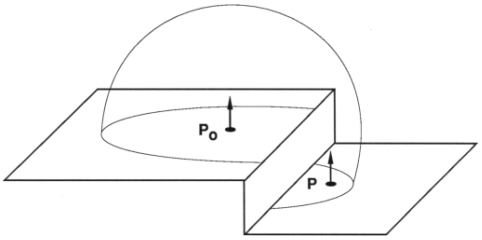
$$E(\vec{P}) = \frac{\sum_{i \in s} w_i(\vec{P}) E_i(\vec{P})}{\sum_{i \in s} w_i(\vec{P})}$$

$$w_i(\vec{P}) = \frac{1}{\frac{\|\vec{P} - \vec{P}_i\|}{R_i} + \sqrt{1 - \vec{N}(\vec{P}) \cdot \vec{N}(\vec{P}_i)}}$$

$$S = \{ i : w_i(\vec{P}) > 1/a \}$$

Ignoring the constants from our previous equation, we can derive a weighting function based on the split sphere approximation. Since our weights correlate to 1/error, applying them in a weighted average distributes error uniformly between the interpolated irradiance values. We hope to maintain final accuracy by restricting our set of values to ones whose estimated error is below some user tolerance, “a”.

Additional Constraint



- Point $\boldsymbol{P}$ is within the valid radius of $\boldsymbol{P}_o$, but is occluded by nearby geometry
- Assume surface of sphere to decide whether point is "behind" reference using distance vs. normals

$$\left(\vec{P} - \vec{P}_i\right) \cdot \left[\vec{N}(\vec{P}) + \vec{N}(\vec{P}_i)\right] / 2 \geq 0$$

One special case we need to consider occurs when an irradiance value *thinks* it is valid over a larger region than it actually is. This case is depicted in the figure, where we are thinking about using an irradiance value at P0 at the new query position P. We can avoid this problem condition by adding a "behind test" to our criteria, which via the weighting function already incorporates a curvature test and a relative distance test.

Cabin model showing single-bounce calculation.

Indirect irradiance record placement shown as red spheres. Note how record spacing is large on flat surfaces far from neighboring geometry, then bunches up on outside curves and (especially) inside corners.

Irradiance Cache Data Structure

```
struct indirect_irradiance_value {
        float  pos[3];          /* position in space */
        float  dir[3];          /* normal direction */
        int    lvl;             /* recursion level of parent ray */
        float  weight;          /* weight of parent ray */
        float  rad;             /* validity radius */
        COLOR  val;             /* computed ambient value */
        float  gpos[3];         /* gradient wrt. position */
        float  gdir[3];         /* gradient wrt. direction */
};
```

We discuss the gradient vectors next…

The ray level and weight are used to determine when to truncate the ray tree. The position and normal vectors are used in the weight/threshold computation, together with the “validity radius.”

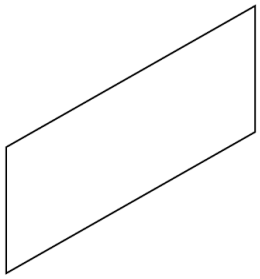
Irradiance Gradients

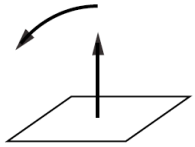
- From hemisphere sampling, we can also compute change w.r.t. position and direction
  - Gradient information comes essentially free
- Equivalent to higher-order interpolation method, i.e., cubic vs. linear

Initially, Paul Heckbert and I were thinking a gradient calculation could inform better value spacing. A year went by before we realized that it was better to apply them directly during interpolation to reduce discontinuities. This also avoids bias issues.

Rotational Gradient

- As view rotates, surface sees more (or less) of bright object
- Estimate rotational change based on hemisphere samples

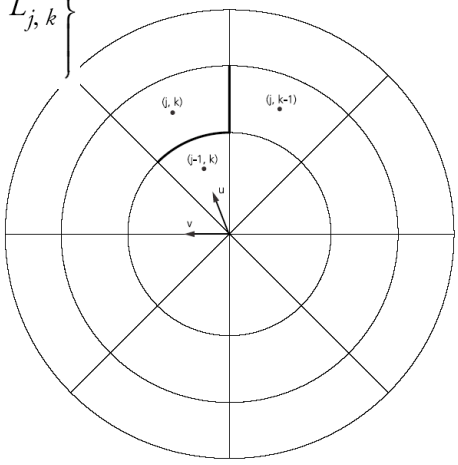




Caveat: we cannot know what will appear over the horizon.

Rotational Gradient Formula

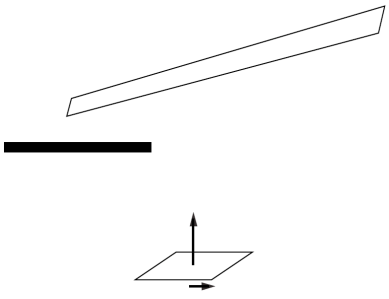
$$\vec{\nabla}_r E = \frac{\pi}{M \cdot N} \sum_{k=0}^{N-1} \left\{ \hat{\mathbf{v}}_k \sum_{j=0}^{M-1} -\tan\theta_j \cdot L_{j,k} \right\}$$



The rotational gradient is zero at the zenith because the derivative of the cosine is zero, hence small rotations don't affect contribution for samples looking straight up.

Translational Gradient

- Translation of surface element exposes (or hides) bright occluded objects and shifts boundaries
- Estimate changes using hemisphere



The distance to our sampled geometry is important, and we have to assume that disocclusions look like what we already see of the background object. Hence, the boundary moves as the nearer edge moves rather than the further surface.

Translational Gradient Formula

$$\vec{\nabla}_t E = \sum_{k=0}^{N-1} \left\{ \hat{u}_k \frac{2\pi}{N} \sum_{j=1}^{M-1} \frac{\sin\theta_{j\_} \cdot \cos^2\theta_{j\_}}{Min(r_{j,k}, r_{j-1,k})} \cdot (L_{j,k} - L_{j-1,k}) \; + \; \hat{v}_{k\_} \sum_{j=0}^{M-1} \frac{\sin\theta_{j_+} - \sin\theta_{j\_}}{Min(r_{j,k}, r_{j,k-1})} \cdot (L_{j,k} - L_{j,k-1}) \right\}$$

where:

$\hat{u}_k$ is the unit vector in the ϕ_k direction

$\hat{v}_{k\_}$ is the unit vector in the $\phi_{k\_} + \frac{\pi}{2}$ direction

$\theta_{j\_}$ is the polar angle at the previous boundary, $\sin^{-1}\sqrt{\frac{j}{M}}$

θ_{j_+} is the polar angle at the next boundary, $\sin^{-1}\sqrt{\frac{j+1}{M}}$

$\phi_{k\_}$ is the azimuthal angle at the previous boundary, $2\pi\frac{k}{N}$

$r_{j,k}$ is the intersection distance for cell (j,k)

Refer to the vector diagram on the rotational gradient slide for variable definitions. Here, the '+' and '-' suffixes refer to the vectors (not drawn) corresponding to the leading and trailing edges of each cell, respectively. The Min(a,b) function is used to determine the closer of two neighboring surface samples.

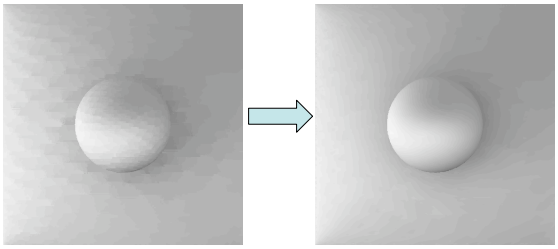
Gradient Interpolation

$$E(\vec{P}) = \frac{\sum_{S} w_i(\vec{P}) \left[E_i + (\hat{n}_i \times \hat{n}) \cdot \vec{\nabla}_r E_i + (\vec{P} - \vec{P}_i) \cdot \vec{\nabla}_t E_i \right]}{\sum_{S} w_i(\vec{P})}$$

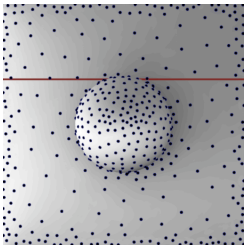
- Weights $w_i(\boldsymbol{P})$ same as before
- Essentially modifies E_i's used for interpolation
- Gradient also used to cap valid radii

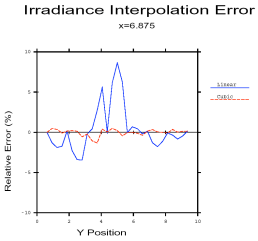
Once we have our rotational and translational gradients corresponding to each irradiance value, we can apply them in a first-order, weighted interpolation as shown.

Irradiance Gradient Results



(no overture)

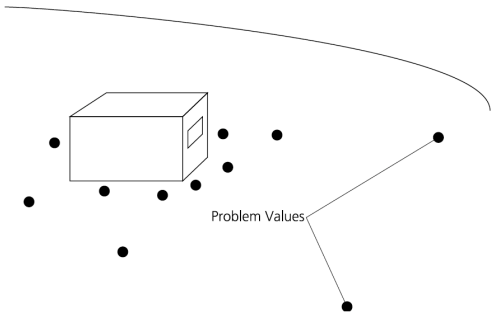




The effect of applying the irradiance gradient technique is highlighted in this extrapolated calculation, where lazy evaluation is shown in its "full glory." Despite discontinuities introduced by adding values to the cache during scanline rendering, our first-order extrapolation makes these artifacts all but disappear. Value placement is shown together with the comparative error when applying gradients to interpolation (I.e., adding an overture calculation).

Irradiance Cache Limitations

- Cached values over very different scales
  - May cause light leaks if value spacing not limited properly
- Also, “hairy” geometry: forests, grass, etc.



Jaroslav has a solution called “neighbor clamping” for this problem. Hairy geometry is best dealt with by switching to a noisy MCPT method for busy topologies.

Sources of Bias in Irradiance Cache

- Super-sampling of hemisphere
  - Naïve adaptive sampling approach is biased
- Truncation bias from limited bounces
  - Caching overrides Russian roulette in *Radiance*
- Placing limits on value spacing
  - Degrades fine-scale features
- Assumed average scene reflectance
  - Undermines accuracy in white-walled enclosure

To eliminate bias, don't use adaptive super-sampling, add Russian roulette to final bounce via path tracing, eliminate minimum value spacing (expensive) and use actual surface reflectances (undermines sharing). Decreasing number of hemisphere samples (by reflectance) and increasing sample spacing (by reflectance^-0.5) distributes errors evenly between levels and speeds convergence.