SIGGRAPH 2011

State of the Art:

Interactive Global Illumination

Chris Wyman

Department of Computer Science

University of Iowa

What Is Global Illumination?

- Lighting affected by global scene geometry
- Light paths involving more that one bounce

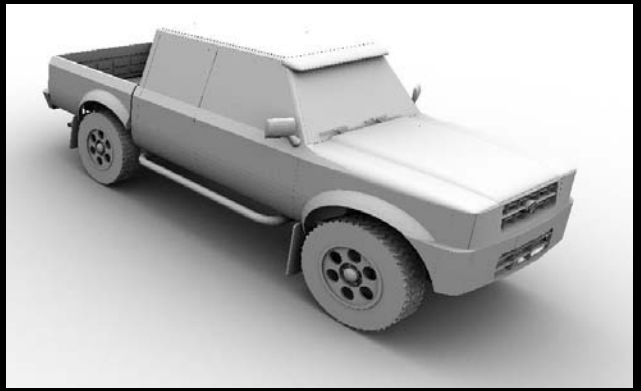
Why Is It Hard?

- Why is GI hard with *programmable shading*?
  - Standard pipe processes prims one by one
    - Difficult to use global info when you don't have it
  - Restricted mostly to precomputed techniques
    - R & D has allowed relatively amazing quality
    - By definition limits interactivity & scene dynamism

Why Is It Hard?

- Why is GI hard, generally?
  - Many surfaces illuminate each other
    - Difficult to minimize bandwidth costs & cache thrashing
    - Accumulate light from many surfaces over hemisphere
  - Key problem: computing visibility
    - Any surface could occlude any other
    - Difficult to identify a minimal set of occluders to test

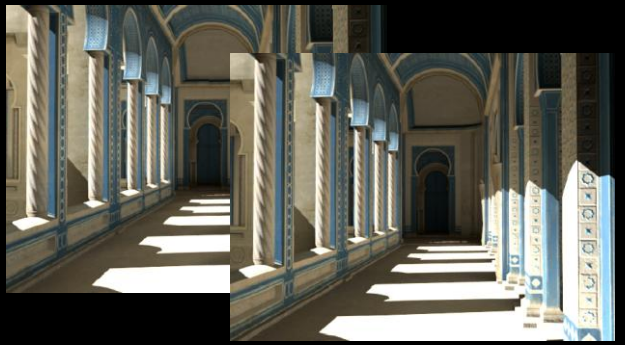
Why Should It Be “Easy”?



- It mostly changes slowly, smoothly
  - Over a wide range of material and light types

Why Should It Be “Easy”?

[Ramanarayanan et al 07]



[Yu et al 09]

- We have a hard time judging visual quality

Ways To Approach Problem?

SIGGRAPH2011

- Precompute approximate lighting
  - Standard programmable shading technique
- Simplify lighting equations
  - Reduce computational complexity
- Simplify geometry or lights
  - Reduce number of computations
- Reuse data to amortize computations
  - Interpolate between “good enough” samples
- Brute force

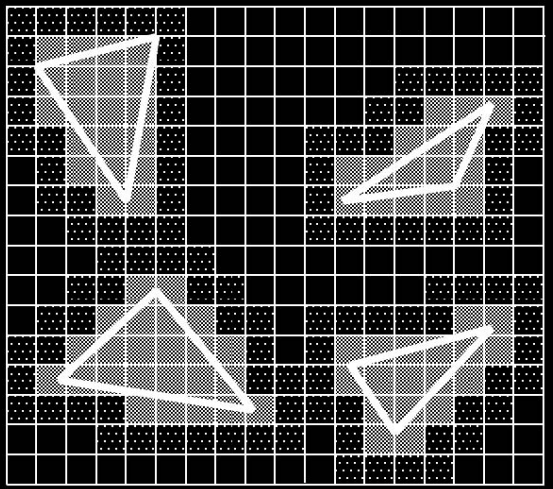
Ways To Approach Problem?

SIGGRAPH2011

- Precompute approximate lighting
  - Widely used; plenty of information available; references in notes
  - Going to *very* briefly review for completeness
- Simplify lighting equations
- Simplify geometry or lights
- Reuse data to amortize computations
- Brute force

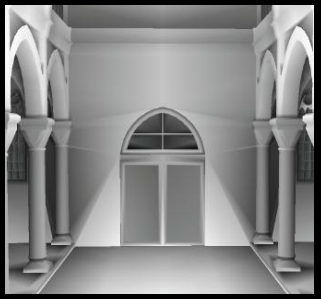
Precomputation Schemes

- Texture atlas
  - Param triangles in texture space
  - Precompute light @ each texel
  - At run time, lookup results
  - Pros: easy, cheap, any type of lighting can be baked
  - Cons: aliasing, static

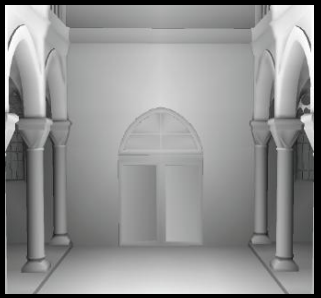


Precomputation Schemes

- Vertex baking
  - Precompute lighting @ each vertex
  - Runtime interpolate from nearby verts
  - Pros: Easy, cheap, less storage
  - Cons: Static, linear interp. issues, limited lighting frequency
  - Recent work smooths gradients using least squares optimization & regularization



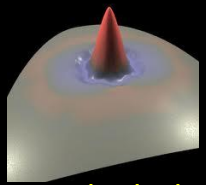
[Kavan et al. 11]



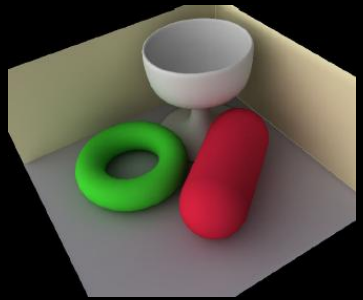
Precomputation Schemes

- Precomputed radiance transfer
  - Precompute and project information
    - Including: illumination, visibility, reflectance
    - To some spherical basis (SH, spherical wavelets)
  - Leverage fast operations in new basis
  - Pros: Allows limited dynamism, useful in many other approaches, many extensions
  - Cons: More storage, typically low frequency

a SH basis



a spherical wavelet



Ways To Approach Problem?

SIGGRAPH2011

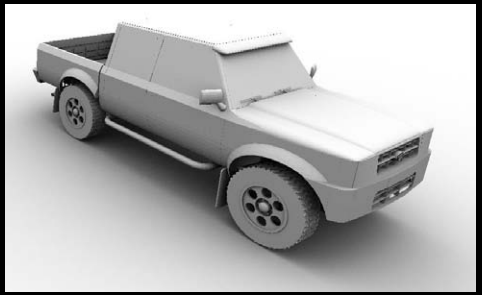
- Precompute approximate lighting
- Simplify lighting equations
  - There are many ways to simplify the rendering equation:

$$L_{out}(\mathbf{x}, \vec{\omega}_{out}) = L_{emit}(\mathbf{x}, \vec{\omega}_{out}) + \int_{\Omega} f_r(\mathbf{x}, \vec{\omega}_{out}, \vec{\omega}_{in}) L_{in}(\mathbf{x}, \vec{\omega}_{in}) \cos\theta_{in} d\vec{\omega}_{in}$$

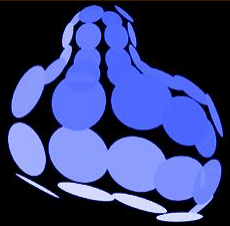
- Simplify geometry or lights
- Reuse data to amortize computations
- Brute force

Simplification Schemes

- Ambient occlusion
  - Assumes incident light uniform, white
  - Just consider visibility
  - Modulate with direct light
    - Or an indirect approx without visibility
  - Pro: Cheaper than GI, surprisingly good, can compute in multiple spaces (screen space, world, object, tangent)
  - Cons: Still not cheap to do well, not suited for high freq. lighting

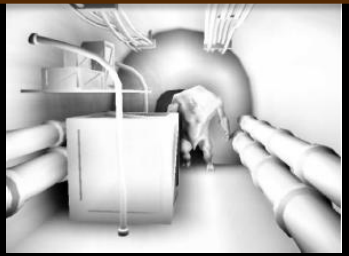


Methods for Ambient Occlusion



Surface discretization
[Bunnell 05]

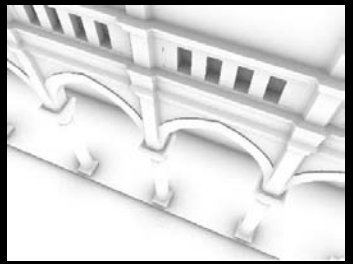
Horizon-Based SSAO
[Bavoil 08]



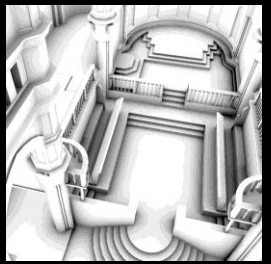
Volumetric AO
[Szirmay-Kalos 09]

Vol. Obscurance
[Loos10]

SSAO
[Mittring 07]



Multi Layer, Multi Res
[Bavoil 09]



Occlusion Volumes
[McGuire 10]

SS Obscurance Est.
[McGuire 11]

Simplification Schemes

- Screen space directional occl.
  - Samples nearby pixels in screen space
  - Considers reflectance in direction
    - Unlike AO, illumination may be non-uniform
  - Add indirect bounce with second pass
  - Pro: Adds direction varying color bleeding over short range
  - Cons: Layer artifacts (as SSAO), still low freq., 30% overhead for subtle effect

[Ritschel et al. 09]

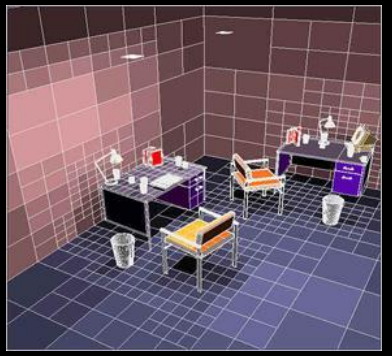
Ways To Approach Problem?

SIGGRAPH2011

- Precompute approximate lighting
- Simplify lighting equations
- Simplify geometry or lights
  - Reduces number of elements we need to query (or compute solutions for)
- Reuse data to amortize computations
- Brute force

Geometry Simplification

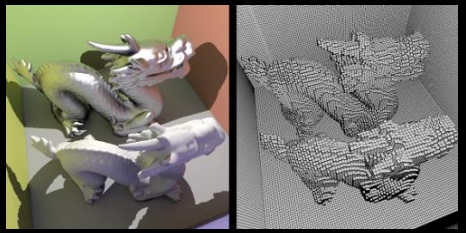
- Radiosity
  - Prototypical geometrical simplification
  - Represent scene as a collection of *patches*
  - Compute patch transfer via form factors
  - Underlies many, many recent interactive techniques
    - Even if not used explicitly
    - Not typically used as proposed in 1980's and 1990's literature



Geometry Simplification

- Voxelize / Rasterize Geometry
  - Avoid using all tiny little polygons
  - Geometric details less important after reflection
  - Can use sparse voxel representations as acceleration structure
    - Some recent voxel representations: [Crassin 09], [Forest 09], [Schwarz 10]
  - Use ray-voxel intersection to query lighting

[Crassin et al. 11]

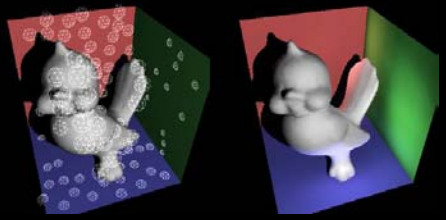


[Thiedemann et al. 11]

Light Simplification

- Instant Radiosity
  - Sample lights as *virtual point lights* (VPLs)
  - Emit photons from VPLs
    - Photon hit points become new VPLs
  - Gather lighting from all VPLs
    - In interactive setting, gather from only a subset of VPLs
  - Can pick VPLs from *reflective shadow map*

[Keller 97]



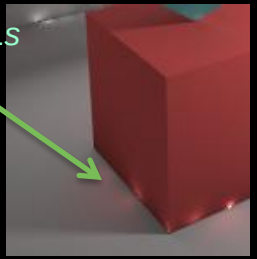
[Dachsbacher 06]

Light Simplification

- VPL based techniques
  - Pros:
    - Very fast when VPLs stored in RSM
    - Give good quality static renderings
  - Cons:
    - Often ignore VPL visibility
    - Restricted VPL radius of influence
    - Illumination singularities near VPLs
    - Temporal coherence issues from singularities

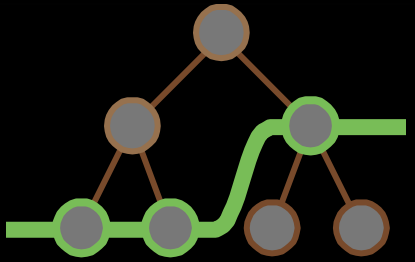
Incrementally refined VPLs
[Laine et al. 07]

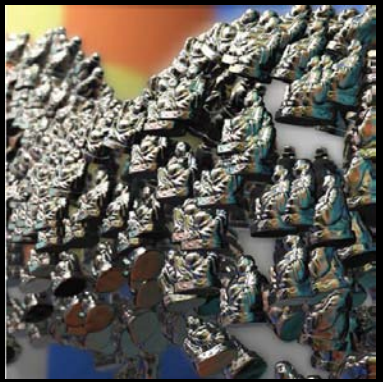
Singularities near VPLs



Light Simplification

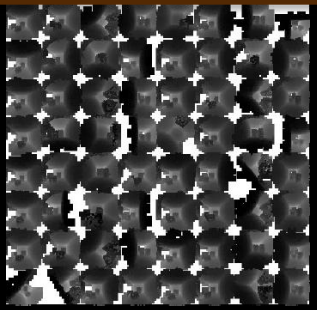
- Improving VPL techniques
  - Light cuts [Walter et al. 05]
    - Use an extremely large set of point light
    - For any point, use important once; cluster others
    - Create a graph of clusters at various levels; pick cut path
  - Geometry cuts (e.g., [Hollander 11])
    - Clustering & cuts, but selecting good LoDs for rendering
  - Require care for good GPU implementation





Light Simplification

- Adding visibility: Imperfect Shadows
  - High quality visibility not needed for indirect
    - Render coarse shadow map for each VPL
    - Render simultaneously (send part of geom to each map)
    - But shadow map bias affects touching surfaces
  - Reduces SM render costs
  - Does not reduce SM lookup costs
  - Could use clustered visibility to reduce lookups



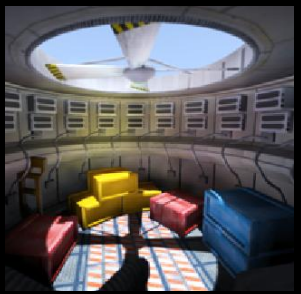
Imperfect Shadow Maps
[Ritschel et al. 08]

Clustered Visibility
[Dong et al. 09]

Light Simplification

- Add de-noising pass
  - E.g., A “final gather” type pass
  - Example: Light Propagation Volumes
    - Sample lots and lots of VPLs
    - Project them into SH basis stored on lattice
    - Interpolate, as in irradiance volumes [Greger 98]
  - LPVs also add propagation steps
    - Allows illumination to diffuse within lattice

Cascaded LPVs
[Kaplanyan10]



Radiance Hints
[Papaioannou 11]

Light Simplification

- Avoid VPL singularities
  - Remove VPL clamp bias
    - Approx. residual in screen space
    - Add in computed residual
    - Repeat as needed or until convergence
  - Removes VPL temporal incoherence
    - Adds a sizable overhead

[Novak et al. 10]

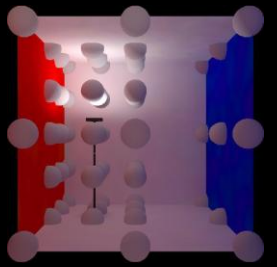
Ways To Approach Problem?

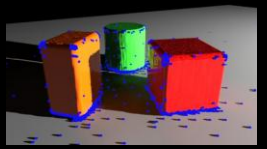
SIGGRAPH2011

- Precompute approximate lighting
- Simplify lighting equations
- Simplify geometry or lights
- Reuse data to amortize computations
  - Avoid redundant computations, interpolate between nearby results
- Brute force

Data Reuse

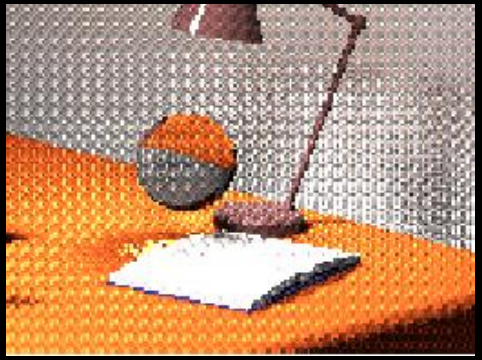
- Many, many such techniques
  - Radiosity (single sample per patch)
  - Irradiance volume (interpolate samples)
  - Render cache (reproject from last frame)
  - (Ir)radiance caching (reuse sample when sufficiently similar)





Data Reuse

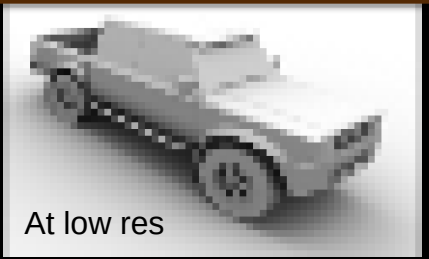
- Interleaved sampling
  - Use different samples in adjacent pixels
  - Use ***all*** samples from nearby pixels
    - Assumes slow change of illumination
  - Used in many spaces for many ray types
    - Screen space
    - Hemisphere (for selecting random rays)



[Wald et al. 02]

Data Reuse

- Render at low resolution
  - Upsample in screen space via interpolation
    - Significantly cheaper, but much worse aliasing
  - Alternatively, geometry aware upsample
    - Commonly done with a bilateral filter
    - Idea: avoid filtering over edges, blur elsewhere



[Yang et al. 08]

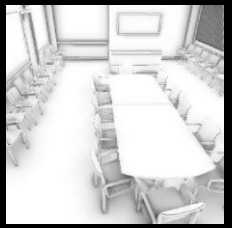
[Bauszat et al. 11]

Data Reuse

- Render at multiple resolutions
  - Upsample in screen space
    - Use higher resolution data when available
    - Can use fixed or adaptive multi-resolution
  - Adaptive techniques
    - Use some metric to identify regions that change fast
    - Sample these regions more finely
    - Significant savings by sampling coarsely when feasible

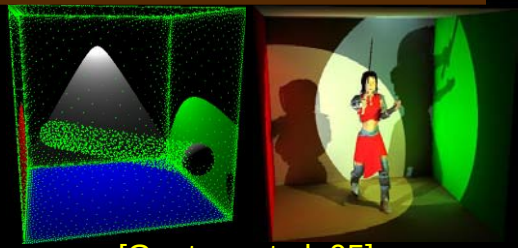
[Nichols et al. 09]


[Nichols 10]


[Hoang 10]

Data Reuse

- Caching Techniques
  - Interactive versions of (ir)radiance caching
    - Leverage smooth changes in lighting
    - Only add new samples near discontinuities
    - Can reuse cache points between frames
    - Enables good quality even with few samples per frame



[Gautron et al. 05]

[Krivanek et al. 05]

Combine Many Methods

- Pick best quality / speed tradeoff
  - Might sample illumination at points
  - Gather on a crude proxy geometry
  - Use coarse sampling of geometry
  - Upsample to render resolution
    - Interpolation in object space
    - Clever mapping techniques to full res geometry
  - Tradeoff may change between generations

Enlighten2
[Martin & Einarsson 10]

Ways To Approach Problem?

- Precompute approximate lighting
- Simplify lighting equations
- Simplify geometry or lights
- Reuse data to amortize computations
- Brute force
  - Sometimes simple algorithms parallelize best!

Ray Trace!

- Modern ray tracers quite fast
  - 100 M+ rays / sec per core on CPU
  - GPU performance faster
  - Simply throw rays at rendering
  - Pro: Simple coding (using the RT)
  - Con: Building acceleration structures
    - Difficult for dynamic scenes

[Djeu et al.07]

[Zhou et al.08]

[Parker et al.10]

Ray Trace!

- Selective ray tracing
  - Only for rays tough to compute other ways
    - Rays through specular interactions
    - Multi-bounce diffuse illumination
  - Traditional rasterization & shadow map
    - To compute initial bounces
  - Can simultaneously use CPU & GPU

[McGuire and Luebke 09]

Micro Rendering

- Rasterize visibility at pixels
  - Represent scene as points
  - Render hemispherical view per sample
    - Computes visibility
    - Computes incident illumination
    - Can consider varying BRDFs
  - Con: Only on the edge of interactivity
    - But can use non-brute force techniques to speed up

[Ritschel et al. 09]

Ways To Approach Problem?

SIGGRAPH2011

- Precompute approximate lighting
- Simplify lighting equations
- Simplify geometry or lights
- Reuse data to amortize computations
- Brute force
- Bonus: Non-diffuse Global Illumination
  - Have mostly focused on diffuse & glossy GI
  - Same types of techniques work for other light interactions

Specular Global Illumination

Multi-Res Sampling
[Wyman and Nichols 09]

Voxel Based Sampling
[Cao et al. 10]

Propagation
through voxels
[Ihrke et al. 07]

Simplified sampling
along eye rays
[Hu et al. 10]

Shadows in Participating Media

Voxel Based Sampling
[Wyman 11]

Reformulate Lighting Eq.
[Chen et al. 11]

Coarser Sampling
with Upsampling
[Engelhardt 10]

Adaptive Sampling
(of shadow map)
[Billeter et al. 10]

Summary

- Lots of work on interactive GI
- Challenge:
  - Access geometry & light from all over scene… efficiently
  - Smoothly sampling rendering equation
- Approaches:
  - Simplify equations, geometry & lights; amortization
  - Easier to do this in a flexible pipeline

Questions?

SIGGRAPH 2011