Advanced Graphics

Radiosity

Part 2
Progressive Refinement Radiosity

New equation (for one iteration):

\[ \Delta B_j = \rho_j B_i F_{ij} \frac{A_i}{A_j} \]

for \( j = 1, N \)

\( B_i \) is the unshot light

The initial \( B_i \) is \( E_i \)

\[ B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{ij} \]

\[ F_{ij} A_i = F_{ji} A_j \]
Substructuring

Group neighboring shooting patches
  – Less important for observed quality
  – Can render fewer hemicubes

Terminology switch
  – Receiving patches are called *elements*
  – Shooting patches are still called *patches*
Adaptive Subdivision

Subdivide patches and elements to increase quality
  – Only subdivide patches/elements which benefit

Patches
  – Subdivide large patches which shoot a lot of light

Elements
  – Subdivide at large radiosity gradients

 Depends on the light
  – Subdivision must be done on-the-fly
GPU Implementation

Store total and unshot light in textures
  – One texel is one element

Repeat:
  – Find patch with most unshot light (compute shader)
  – Render hemicube for that patch
  – Execute a compute shader for each element
    • Check if the element is visible in the hemicube
    • Calculate the light shot towards the element
The hemicube
Progressive refinement radiosity
Non-diffuse reflectance
Incremental radiosity
Hierarchical radiosity
Non-diffuse reflectance

We ignored (glossy) specular reflections so far
- How can we incorporate it?

Modify radiosity equations
- Include BRDF in the form factor
  • BRDF (and thus form factor) depends on the outgoing light direction
- Need to store $B_i$ for every outgoing light direction

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{ij} \quad F_{ij} = \int_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} \text{HID} \, dA_j$$
Directional Storage

Directional information adds a new dimension

Store the directional distribution of light in $B_i$

– Can discretize directions and explicitly store light in a direction

– Can encode the distribution using spherical harmonics
Directional Storage

Accurate directional storage requires a lot of memory
  – In practice, only suited for low frequency glossy specular

Perfect specular reflections (mirror-like)
  – Use distributed ray tracing instead
The hemicube
Progressive refinement radiosity
Non-diffuse reflectance
Incremental radiosity
Hierarchical radiosity
Comparison

– Radiosity uses a discretized surface and caching
– Path Tracing is continuous and stochastic

Radiosity is view *independent*
– As opposed to path tracing

We can re-use the global illumination each frame
– And even progressively refine it
  • Amortize cost over many frames
– Until... an object or light source moves
Incremental Radiosity

When something moves, update the radiosity solution

– As opposed to recalculating everything again

Shoot *negative* light to undo lighting
Incremental Radiosity

When a light source changes intensity
   – Shoot (negative) light

When a surface property changes (e.g. albedo)
   – Shoot (negative) light

What if a patch moves?
   – The geometric relations change
Incremental Radiosity

Basic steps

– Remove the patch at the old position
  • And undo all lighting contributions
– Insert the patch at the new position
  • Create new lighting contributions

– Iterate using progressive refinement
  • Propagates the lighting changes through the environment
  • Lighting changes are stored as unshot light
Remove Patch

Undo 3 types of contributions

– The patch received light on itself
  • Clear patch lighting

– The patch sent light to other patches
  • Send negative light

– The patch occluded light from other patches
  • ...?
Remove Patch

Render from all shooting patches to find occlusions

– Shoot positive light through the silhouette
Insert Patch

Inverse 3 types of contributions

– Receive light
  • Shoot from all other patches

– Send light to other patches
  • Shoot from the new patch

– Occlude light from other patches
  • Shoot negative light through the new silhouette
Incremental Radiosity

From all shooting patches
- Render only the changed area
  1. Shoot negative light in the **before** scene
  2. Propagate negative light on the moving object
  3. Shoot positive light in the **after** scene
  4. Propagate all light changes

From the shooting patch’s perspective:
The hemicube
Progressive refinement radiosity
Non-diffuse reflectance
Incremental radiosity
Hierarchical radiosity
Efficiency

All patches can interact with all other patches
  – $O(n^2)$ complexity

Patches must be small to prevent form factor error
  – Depends on the distance to other patches
  – Subdivide near patches
Efficiency

The subdivided patches are unnecessarily small for patches further away.

- Good size ratio
- Bottom patch is unnecessarily small
Patch Hierarchy

Each patch is a hierarchy

- We can use a larger or smaller patches
  - Depending on the distance between the patches
Patch Hierarchy

Higher levels are used for more distant patches

The black patches visible in the image are the lowest hierarchy level
Push-Pull

Radiosity is updated at different levels
  – Propagate the updates through the hierarchy

Push new radiosity down, pull new radiosity up
  – Area weighted
Create *links* between patches

– At the correct level in the hierarchy
– Only if the patches can see each other
– Store form factor in the link

Radiosity is transported using the links
Overview

– Subdivide patches
– Create links
– Repeat:
  • Transport radiosity through links
  • Push-pull radiosity through patch hierarchy
Normal radiosity has $O(n^2)$ “links”
  – Where $n$ is the number of patches

Hierarchical radiosity has only $O(n)$ links
  – Scales very well
  – Constant in complexity depends on the accepted error
BF Refinement

\[ B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{ij} \]

Assumed error depends on the distance
  – Also depends on the actual light transported

Take lighting into account when subdividing and creating links
  – Hierarchy creation interleaved with light propagation
Antiradiance

Do no visibility testing at all

– What about shadow?
  • Handle implicitly

– When receiving light, send antiradiance out from the back
  • This “shadow” radiance reverts the incorrect lighting
Antiradiance

Causes double shadowing

– Received *antir*adiance has to be sent out as normal radiance from the back
Antiradiance

Shadow is directional

- Need to store directional lighting information
- Create bins for directions (normally 128-512 bins)
Antiradiance

Still poor directional accuracy
  – Only use for indirect lighting
  – Use shadow mapping for direct light

Suitable for GPU acceleration
  – Lack of visibility test is very efficient
Antiradiance

Real-time antiradiance (without link creation and only indirect light)
Antiradiance

Real-time antiradiance (with link creation)
Overview

Radiosity

- The hemicube
  - Progressive refinement
    - Incremental radiosity
    - Non-diffuse radiosity

- Hierarchical radiosity
  - Antiradiance
Next time

Global illumination approximations

– Various techniques
– Cannot create *perfect* global illumination
– Various other limitations
– Real-time performance
  • Used in games