Direct3D 11 Indirect Illumination

Holger Gruen
European ISV Relations AMD
Direct3D 11 Indirect Illumination

Holger Gruen
European ISV Relations AMD
Direct3D 11 Indirect Illumination

Holger Gruen
European ISV Relations AMD
Indirect Illumination

Introduction 1

- Real-time Indirect illumination is an active research topic

- Numerous approaches exist

  Reflective Shadow Maps (RSM) [Dachsbacher/Stammiger05]
  Splatting Indirect Illumination [Dachsbacher/Stammiger2006]
  Multi-Res Splatting of Illumination [Wyman2009]
  Light propagation volumes [Kapalanyan2009]
  Approximating Dynamic Global Illumination in Image Space [Ritschel2009]

- Only a few support indirect shadows

  Imperfect Shadow Maps [Ritschel/Grosch2008]
  Micro-Rendering for Scalable, Parallel Final Gathering (SSDO) [Ritschel2010]
  Cascaded light propagation volumes for real-time indirect illumination
  [Kapalanyan/Dachsbacher2010]

- Most approaches somehow extend to multi-bounce lighting
Indirect Illumination
Introduction 2

This section will cover

An efficient and simple DX9-compliant RSM based implementation for smooth one bounce indirect illumination

- Indirect shadows are ignored here

A Direct3D 11 technique that traces rays to compute indirect shadows

- Part of this technique could generally be used for ray-tracing dynamic scenes
Indirect Illumination w/o Indirect Shadows

1. Draw scene g-buffer
2. Draw Reflective Shadowmap (RSM)
   RSM shows the part of the scene that receives direct light from the light source
3. Draw Indirect Light buffer at ½ res
   RSM texels are used as light sources on g-buffer pixels for indirect lighting
4. Upsample Indirect Light (IL)
5. Draw final image adding IL
Step 1

- G-Buffer needs to allow reconstruction of World/Camera space position, World/Camera space normal, Color/Albedo
- DXGI_FORMAT_R32G32B32A32_FLOAT positions may be required for precise ray queries for indirect shadows
Step 2

- RSM needs to allow reconstruction of World/Camera space position
- World/Camera space normal
- Color/Albedo
- Only draw emitters of indirect light
- DXGI_FORMAT_R32G32B32A32_FLOAT position may be required for ray precise queries for indirect shadows
Step 3

- Render a ½ res IL as a deferred op
- Transform g-buffer pix to RSM space
  -> Light Space -> project to RSM texel space
- Use a kernel of RSM texels as light sources
  RSM texels also called Virtual Point Light (VPL)
- Kernel size depends on
  Desired speed
  Desired look of the effect
  RSM resolution
Computing IL at a G.buf Pixel 1

Transform to RSM space

G-Buffer pixel

RSM texels/VPLs

Sum up contribution of all VPLs in the kernel
Computing IL at a G-buf Pixel 2

\[ D = \frac{P_L - P_p}{|P_L - P_p|} \]

\[ Contribution_{VPL} = \frac{\text{sat}(N_p \cdot D) \cdot \text{sat}(N_L \cdot (-D))}{|P_L - P_p|^2} \cdot Col_{VPL} \cdot Area_{VPL} \]

This term is very similar to terms used in radiosity form factor computations.
Computing IL at a G-buf Pixel 3

A naive solution for smooth IL needs to consider four VPL kernels with centers at t0, t1, t2 and t3.

\( \text{stx} \) : sub RSM texel x position \([0.0, 1.0]\)

\( \text{sty} \) : sub RSM texel y position \([0.0, 1.0]\)
IndirectLight = (1.0f-sty) * ((1.0f-stx) * \[VPL kernel at \(t0\)] + stx * \[VPL kernel at \(t1\)] ) +
(0.0f+sty) * ((1.0f-stx) * \[VPL kernel at \(t2\)] + stx * \[VPL kernel at \(t3\)] )

Evaluation of 4 big VPL kernels is slow 😞
Computing IL at a g-buf pixel 5

SmoothIndirectLight =

\[(1.0f-sty)*(((1.0f-stx)*(B0+B3)+stx*(B2+B5))+B1)+
(0.0f+sty)*(((1.0f-stx)*(B6+B3)+stx*(B8+B5))+B7)+B4\]

\textbf{stx} \text{ : sub RSM texel x position of g-buf pix [0.0, 1.0[}
\textbf{sty} \text{ : sub RSM texel y position of g-buf pix [0.0, 1.0[}

This trick is probably known to some of you already. See backup for a detailed explanation!
Indirect Light Buffer
Step 4

- Indirect Light buffer is ½ res
- Perform a bilateral upsampling step


- Result is a full resolution IL
Step 5

- Combine
  Direct Illumination
  Indirect Illumination
  Shadows (not mentioned)
Scene without IL
Combined Image

~280 FPS on a HD5970 @ 1280x1024 for a 15x15 VPL kernel

Deffered IL pass + bilateral upsampling costs ~2.5 ms
How to add Indirect Shadows

1. Use a CS and the linked lists technique
   Insert blocker geometry of IL into 3D grid of lists – let’s use the triangles of the blocker for now
   ✗ see backup for alternative data structure

2. Look at a kernel of VPLs again

3. Only accumulate light of VPLs that are occluded by blocker tris
   Trace rays through 3d grid to detect occluded VPLs
   Render low res buffer only

4. Subtract blocked indirect light from IL buffer
   Blurred version of low res blocked IL is used
   ✗ Blur is combined bilateral blurring/upsampling
Insert tris into 3D grid of triangle lists

Scene

Rasterize dynamic blockers to 3D grid using a CS and atomics
Insert tris into 3D grid of triangle lists

Rasterize dynamic blockers to 3D grid using a CS and atomics

World space 3D grid of triangle lists around IL blockers laid out in a UAV

eol = End of list (0xffffffff)
3D Grid Demo
Indirect Light Buffer

Emitter of green light

Blocker of green light

Expected indirect shadow
Blocked Indirect Light
Indirect Light Buffer
Subtracting Blocked IL
Final Image
~300 million rays per second for Indirect Shadows

~70 FPS on a HD5970 @ 1280x1024

Ray casting costs ~9 ms
Future directions

- Speed up IL rendering
  - Render IL at even lower res
  - Look into multi-res RSMs
- Speed up ray-tracing
  - Per pixel array of lists for depth buckets (see backup)
  - Other data structures
- Raytrace other primitive types
  - Splats, fuzzy ellipsoids etc.
  - Proxy geometry or bounding volumes of blockers
- Get rid of Interlocked*() ops
  - Just mark grid cells as occupied => >150 fps
  - Lower quality but could work on earlier hardware through scattered splats
Q&A

Holger Gruen  holger.gruen@AMD.com
Nicolas Thibieroz  nicolas.thibieroz@AMD.com

Credits for the basic idea of how to implement PPLL under Direct3D 11 go to Jakub Klarowicz (Techland), Holger Gruen and Nicolas Thibieroz (AMD)
Backup Slides IL
Computing IL at a g-buf pixel 1

- Want to support low res RSMs
- Want to create smooth indirect light
- Goal is bi-linear filtering of four VPL-Kernels
  Otherwise results don’t look smooth
Computing IL at a g-buf pixel 2

\[ stx : \text{sub texel x position } [0.0, 1.0[ \]

\[ sty : \text{sub texel y position } [0.0, 1.0[ \]
Computing IL at a g-buf pixel 3

For smooth IL one needs to consider four VPL kernels with centers at t0, t1, t2 and t3.

**stx**: sub texel x position [0.0, 1.0]
**sty**: sub texel y position [0.0, 1.0]
Computing IL at a g-buf pixel 4

Center at t0

\textbf{stx} : sub texel x position \([0.0, 1.0]\)

\textbf{sty} : sub texel y position \([0.0, 1.0]\)
Computing IL at a g-buf pixel 4

\[ \text{stx : sub texel x position } [0.0, 1.0] \]
\[ \text{sty : sub texel y position } [0.0, 1.0] \]

Center at t1
Computing IL at a g-buf pixel 5

Center at t2

\[ \text{stx} : \text{sub texel x position } [0.0, 1.0[ \]
\[ \text{sty} : \text{sub texel y position } [0.0, 1.0[ \]
Computing IL at a g-buf pixel 6

\[ \text{stx} : \text{sub texel x position } [0.0, 1.0[ \]
\[ \text{sty} : \text{sub texel y position } [0.0, 1.0[ \]
Computing IL at a g-buf pixel 7

\[
\text{IndirectLight} = (1.0f - \text{sty}) \times ((1.0f - \text{stx}) \times \text{VPL kernel at } t0 + \text{stx} \times \text{VPL kernel at } t1) + (0.0f + \text{sty}) \times ((1.0f - \text{stx}) \times \text{VPL kernel at } t2 + \text{stx} \times \text{VPL kernel at } t3)
\]

Evaluation of 4 big VPL kernels is slow 😞

\textbf{stx} : sub texel x position [0.0, 1.0]
\textbf{sty} : sub texel y position [0.0, 1.0]
Computing IL at a g-buf pixel 8

\[ \text{stx} : \text{sub texel x position } [0.0, 1.0[ \]
\[ \text{sty} : \text{sub texel y position } [0.0, 1.0[ \]
Computing IL at a g-buf pixel 9

\[ \text{stx} : \text{sub texel x position} \ [0.0, 1.0[ \]
\[ \text{sty} : \text{sub texel y position} \ [0.0, 1.0[ \]
Computing IL at a g-buf pixel 9

IndirectLight =

\[(1.0f - sty) \times (((1.0f - stx) \times (B0 + B3) + stx \times (B2 + B5)) + B1) +
(0.0f + sty) \times (((1.0f - stx) \times (B6 + B3) + stx \times (B8 + B5)) + B7) + B4\]

Evaluation of 7 small and 1 bigger VPL kernels is fast 😊

\textbf{stx} : sub texel x position [0.0, 1.0[
\textbf{sty} : sub texel y position [0.0, 1.0[
Insert Tris into 2D Map of Lists of Tris

Rasterize blockers of IL from view of light

Light  Scene  2D buffer
Insert Tris into 2D Map of Lists of Tris

Rasterize blockers of IL from view of light using a GS and conservative rasterization

Light  Scene

2D buffer of lists of triangles written to by scattering PS

eol = End of list (0xffffffff)