Assassin’s Creed 4: Black Flag

Lighting, Weather and Atmospheric Effects

Bart Wronski, 3D Programmer, Ubisoft Montreal
Next-gen consoles launch title!
...but done for previous generation as well
Current-gen was dominating install-base and we cared a lot about those players
Constraints on assets duplication and extra artist work
Next-gen improvements had to be procedural
Presentation overview

- Lighting and Global Illumination
  - Improvements over Assassin’s Creed 3
  - Global illumination solution
  - Ambient Occlusion: temporal-supersampled SSAO
  - Multi-resolution ambient occlusion

- Atmospheric and weather effects
  - Rain ripples and wet materials
  - Screenspace reflections
  - GPU simulated rain
  - Volumetric fog
LIGHTING
Assassin’s Creed 3 lighting - PS3 screenshot
Lighting improvements

- Key problems
  - Overall flatness
  - Lack of sense of position and directionality
  - Loss of normal mapping information

- Requirements
  - Prototyped multiple real-time GI solutions, none of them looked good enough
  - Partially baked solution
  - Vast open world with sparse dense areas
  - Dynamic weather / time of day
  - Work on current gen (~1ms / < 1MB for GPU)
  - Small impact on art pipelines
Far Cry 3 solution: Deferred Radiance Transfer Volumes

<table>
<thead>
<tr>
<th>Offline</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRT based - compute radiance transfer</td>
<td>Compute irradiance on the CPU</td>
</tr>
<tr>
<td>Store in 2nd order SH in light probes for 4 basis vectors</td>
<td>Store it in volume texture</td>
</tr>
<tr>
<td></td>
<td>Perform deferred lighting in half res on GPU</td>
</tr>
</tbody>
</table>
Far Cry 3 GI - Deferred Radiance Transfer Volumes

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robust and fully dynamic</td>
<td>Low spatial resolution (4m)</td>
</tr>
<tr>
<td>Compatible with PS3&amp;X360</td>
<td>Low contrast and temporal variance</td>
</tr>
<tr>
<td>Cheap on GPU (0.7ms on PS3!)</td>
<td>Low occlusion precision</td>
</tr>
<tr>
<td>Production ready</td>
<td>Half resolution edge artifacts</td>
</tr>
<tr>
<td>Volumetric</td>
<td>Generation/baking time</td>
</tr>
<tr>
<td>Low memory and CPU cost</td>
<td></td>
</tr>
</tbody>
</table>
New solution - background

- One key light
- Weather has no influence on light direction
- Small amount of local lights
Deferred Normalized Irradiance Probes
Normalized irradiance

- Surrounding environment lit with neutral color
- Integrate in one direction like standard irradiance
- Denormalize = multiply by current light color

\[ E_i = \int_{\Omega_i} L_i \cos \theta_i \, d\Omega_i \]

Deferred Normalized Irradiance Probes

- On GPU bake sunlight bounce **irradiance**
- Store irradiance at 8 different hours
- Compute 2D VRAM textures (many lightprobes)

- De-normalize and blend irradiances
- Blend out bounced lighting with height
- Combine with indirect sky lighting and AO
Implementation - tool side

- Place probes with navmesh
Data storage

- 8 key-framed values a day
- 8-bit RGB normalized irradiance
- 4 basis vectors (FC3 basis)
- Uniform grid $2\text{m} \times 2\text{m}$
- Only one layer
  - 2.5D world layout / structure
Implementation - tool side

- Place probes with navmesh
- For each probe, generate multiple cubemaps
For each cube map, compute irradiance in 4 directions, packed in 3 RGBA
Do that for all times of day
Implementation - tool side

- Each Texture contains this data for 16x16 probes
Implementation - tool side

Optimizations

- Limit cubemap camera far plane (10-30m should be enough)
- Re-use G-buffer between different day times
- Reuse shadowmaps between multiple probes
- Use only one, big in size and resolution cascade per sector
- Special, simpler path for scene traversal code
- Do the convolution and keep all the data on the GPU until you need to save final textures!
Deferred Normalized Irradiance Probes

**Offline**
- On GPU bake sunlight bounce irradiance
- Store irradiance at 8 different hours
- Compute 2D VRAM textures (many lightprobes)

**Runtime**
- De-normalize and blend irradiances
- Blend out bounced lighting with height
- Combine with indirect sky lighting and AO
Implementation - runtime

- Generate on GPU denormalized data in 3 textures
- Blit sectors surrounding camera: select slice / apply sun color
Implementation - runtime
## Benchmarks and summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU performance cost</strong></td>
<td>1.2ms fullscreen pass - PS3</td>
</tr>
<tr>
<td><strong>Memory cost (probe data)</strong></td>
<td>600kb (VRAM only)</td>
</tr>
<tr>
<td><strong>Memory cost (render targets)</strong></td>
<td>56kb</td>
</tr>
<tr>
<td><strong>CPU cost</strong></td>
<td>0.6ms (amortized)</td>
</tr>
<tr>
<td><strong>Num probes in Havana bruteforce</strong></td>
<td>~110 000</td>
</tr>
<tr>
<td><strong>Num probes in Havana trimmed</strong></td>
<td>~30 000</td>
</tr>
<tr>
<td><strong>Full baking time for Havana</strong></td>
<td>8 minutes (nVidia GTX 680, one machine)</td>
</tr>
</tbody>
</table>
Limitations and future improvements

Lack of side bounce
Ground color bleeding
Basis not normalized
Lack of energy conservation 😞
Future work

- Change basis to more accurate one
- Increase probe density in X/Y/Z
- Use real HDR irradiance with sky lighting
- Multiple bounces
- Update closest probes in the runtime (cheap, GPU-only code!)
Ambient Occlusion
Scalable Ambient Obscurance

- Extensions to algorithm by McGuire et al
- Great performance - ~1.6ms full res on consoles with filtering and applying
- Large radius ~1.5m, takes normal mapping into account
- Our extension - **temporal supersampling**
- Different than existing approaches by changing sampling pattern every frame
- Rotate AO samples spiral in 3 distinct patterns every frame (120 degrees)
- Effectively the triple number of samples!
- Blend and accumulate frames, reject on depth difference
SSAO sample number increase by using temporal supersampling
No temporal
Temporal
No temporal supersampling / smoothing
Ambient occlusion

- SSAO is not enough to represent sky lighting occlusion...
- Even multi-res approaches won’t catch all occlusion
- Idea - separate ambient occlusion into multiple frequency bands
- Every frequency band calculated with a different algorithm!
Multi-resolution ambient occlusion
Example by Inigo Quilez

Source: http://www.iquilezles.org/www/articles/multiresaocc/multiresaocc.htm
World Ambient Occlusion

- Needed large scale sky occlusion for direct sky lighting from buildings and trees
- World Ambient Occlusion
- Technique developed for Assassin’s Creed 3
- See Jean-Francois St-Amour talk from GDC 2013!
World Ambient Occlusion

**OFFLINE**

- Render scene depth from top
- 7x7 Gaussian blur on the depth

**AT RUNTIME:**

- Sample world AO to get an estimated occluder height
- Sample offset by XY normal
- Apply AO to sky ambient lighting **ONLY**
- Baked sun bounce irradiance already contains this frequency of information!
Atmospheric and weather effects
Atmospheric effects

- Caribbean tropical climate extremely unpredictable
- Goes from dusty and dry to showers and storms within minutes
- Already had coherent and robust weather system from AC3
- Need for new cool, next-gen procedural atmospheric effects!
GPU Procedural Rain
Rain ripples
No ripples
Ripples
Rain ripples

CS: Spawn and evolve rain ripples

PS: Draw rain ripples into signed heighfield texture

PS: Calculate heightfield derivatives

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>float2</td>
</tr>
<tr>
<td>Life</td>
<td>float</td>
</tr>
<tr>
<td>Max life</td>
<td>float</td>
</tr>
<tr>
<td>Strength</td>
<td>float</td>
</tr>
<tr>
<td>Max radius</td>
<td>float</td>
</tr>
</tbody>
</table>

Updated rain ripple structure

256x256 signed R8 texture

256x256 signed R8G8 texture
Drawing rain ripples - geometry multiplication using geometry shaders
Rain ripples

- Single pass to apply it on screen and perturb existing normals
- Wrap texture around in world space
- World AO is sky occlusion...
- ... so use it for rain occlusion = no additional runtime cost
- Rain ripples update and texture generation cost ~0.2ms
- Perturbing normals can be a separate pass (~0.4ms) or combined with lighting (pipelined well and “free”!)
Wet materials

- Surface wetness stored in G-buffer
- “Baked” for wet areas or modified dynamically by weather
- Use it during lighting pass to update gloss and albedo
- Increase the gloss, darken the albedo
- Same technique used in Assassin’s Creed 3
- ..but enhanced using screenspace reflections!
No screen-space reflections
Screen-space reflections
GPU Procedural Rain
Rain drops
Procedural rain drops

- Fully GPU-driven – compute and geometry shaders
- Simulate 3x3 grid of rain clusters around the camera
  - Avoids “popping” of new rain drops and guarantees uniform distribution
- Render only visible clusters (CPU culling)
Procedural rain simulation

- Multiple factors taken into account
  - Random rain drop mass and size
  - Wind and gravity
  - Rain-map for simple sky occlusion
    - Top-down close range 128x128 “shadowmap”
Rain map
Procedural rain simulation

- Rain map used together with WorldAO for rain occlusion
  - (different range and precision)
- Screen-space collisions with depth buffer
- Spawning new particles on collision
  - Simulating bounced rain drops
GPU Procedural Rain update scheme

Frame N

CS: Spawn point sprites

CS: Update/simulate point sprites

VS/GS/PS: Expand point sprites to particles and draw

read/write structured buffers

Frame N+1

CS: Spawn point sprites

CS: Update/simulate point sprites

VS/GS/PS: Expand point sprites to particles and draw
Rain performance

- CS Particle update cost negligible
- Possible to implement complex update logic
- Some features ("true" random()) are tricky
- Move more particle systems to the GPU
- Didn’t really need to optimize any of CS shaders
- Geometry Shaders were huge performance bottleneck 😞

<table>
<thead>
<tr>
<th>CS: Update rain drops (up to 320k particles)</th>
<th>&lt;0.1ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS: Screenspace collision</td>
<td>0.2ms</td>
</tr>
<tr>
<td>CS: Update bounced drops</td>
<td>&lt;0.05ms</td>
</tr>
<tr>
<td>GS/VS/PS: Draw rain drops</td>
<td>0.4-4.0ms</td>
</tr>
</tbody>
</table>
Geometry shader optimizations

- Minimize memory processed and generated by GS
  - Minimize number of generated vertices
  - Minimize input/output vertex size
  - Implement GPU frustum/occlusion culling in GS
- ...but if you don’t have to, just don’t use them!
- Change for indexed draw call with manual instancing / fetching
- Better performance, less shader stages
Volumetric Fog
Volumetric fog
Volumetric fog

- Works great for fog, mist and haze in humid climate
- Perfectly complements rain storm effect (procedural animation)
- Possible to simulate all atmospheric scattering phenomena
- Light shafts and multiple light sources
- 1.1ms total on consoles!
- ...for implementation details see my GDC 2014 talk!
<table>
<thead>
<tr>
<th>Credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexandre Lahaise</td>
</tr>
<tr>
<td>Benjamin Goldstein</td>
</tr>
<tr>
<td>Benjamin Rouveyrol</td>
</tr>
<tr>
<td>Benoit Miller</td>
</tr>
<tr>
<td>John Huelin</td>
</tr>
<tr>
<td>Lionel Berenguier</td>
</tr>
<tr>
<td>Luc Poirier</td>
</tr>
</tbody>
</table>
Special thanks

- Benjamin Rouveyrol for materials for GI slides
- Rest of GI Team: Benjamin Rouveyrol, John Huelin and Mickael Gilabert
- Wei Xiang for initial implementation of rain
- Assassin’s Creed 4 technical art directors: Danny Oros, Guillaume Lefebvre, Philippe Ringuette-Angrignon, Philippe Trarieux, Sebastien Larrue
Questions?  
Contact me!

- Email: bartlomiej.wronski@ubisoft.com
- Twitter: @BartWronsk
- Slides on my blog
- www.bartwronski.com
References

• Gilabert and Stefanov “Deferred Radiance Transfer Volumes - Global Illumination in Far Cry 3”, GDC 2012
• Mitchell, “Shading in Valve’s Source Engine”, SIGGRAPH 2006
• Sloan et al, “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments”, SIGGRAPH 2002
• Quilez, post about Multi-resolution ambient occlusion http://www.iquilezles.org/www/articles/multiresaocc/multiresaocc.htm
• St-Amour, “Rendering Assassin’s Creed III”, GDC 2013
• Hoffman, “Rendering Outdoor Light Scattering in Real Time”, GDC 2002
• Myers, “Variance Shadow Mapping”, NVIDIA Corporation
• Annen et al, “Exponential Shadow Maps”, Hasselt University
• Andersson and Bavoil, “Stable SSAO in Battlefield 3 with Selective Temporal Filtering”
• Smedberg and Wright, “Rendering Techniques in GEARS OF WAR 2”
• Bilodeau, “Vertex Shader Tricks”
• Wronski, “Assassin’s Creed 4 - road to next-gen graphics”, GDC 2014