Ambient Occlusion Fields and Decals in Infamous 2

Nathan Reed
Rendering Programmer, Sucker Punch Productions

GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9

Background: Infamous 2

- PS3 exclusive
- Open-world, urban environment
- Deferred-shading renderer
- Supports per-vertex baked AO, and SSAO

AO – large or small scale?

- Baked AO is great, but…
  - Per-vertex needs tessellation for fine detail
  - Lightmaps need a lot of memory for fine detail
  - Can’t move things around at runtime
- Best for large-scale, static objects

AO – large or small scale?

- SSAO is great, but…
  - Limited radius in screen space
  - Missing data due to screen edges, occlusion
  - Inconsistent from one camera position to another
- Best for very fine details

Our hybrid approach

- Can complement baked AO and SSAO
- Medium-scale, partly static
- Work in world space: precompute AO from an object onto the space around it, store in a texture.

Our hybrid approach

- Precompute based on source geometry only, not target. Can be moved in real-time.
- Apply like a light in deferred shading: evaluate AO per pixel, within region of effect.
- Two variants: AO Fields & AO Decals

Ambient Occlusion Fields

AO Fields

- Similar to previously reported techniques
  - Kontkanen and Laine, “Ambient Occlusion Fields”, SIGGRAPH ’05
  - Malmer et al. “Fast Precomputed Ambient Occlusion for Proximity Shadows”, Journal of Graphics Tools, vol. 12 no. 2 (2007)
  - Hill, “Rendering with Conviction”, GDC ’10

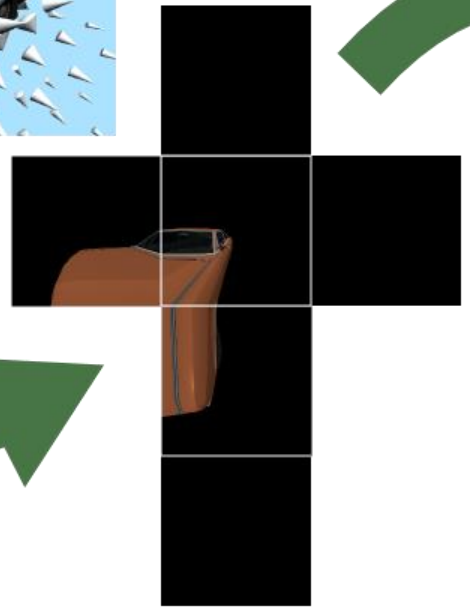
AO Fields: Precomputing

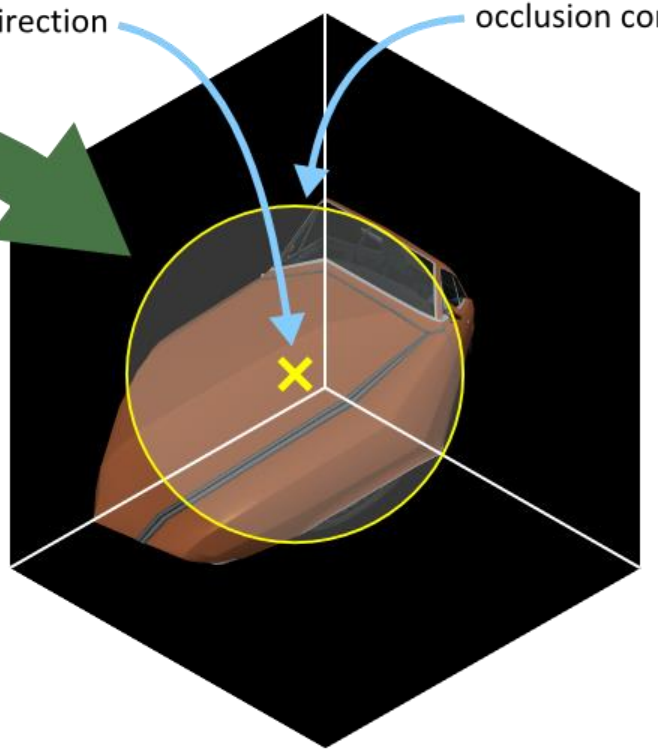
- Put a volume texture around the source object
- Each voxel is an occlusion cone:
  - RGB = average direction toward occluder
  - A = width, as fraction of hemisphere occluded

AO Fields: Precomputing

- Iterate over volume texture voxels
  - Render geometry into a 32×32 cubemap centered on each voxel
  - Read-back and compute average direction of drawn pixels (weighted by solid angle)
  - Compute occluded fraction of hemisphere around that direction

AO Fields: Precomputing





AO Fields: Applying

- Draw the bounding box; pixel shader retrieves world pos and normal of shaded point
  - Just like a light in deferred shading – same tricks & optimizations apply
- Sample texture, decode occlusion vector and width
  - Transform world pos to field local space
  - Transform occlusion vector back to world space

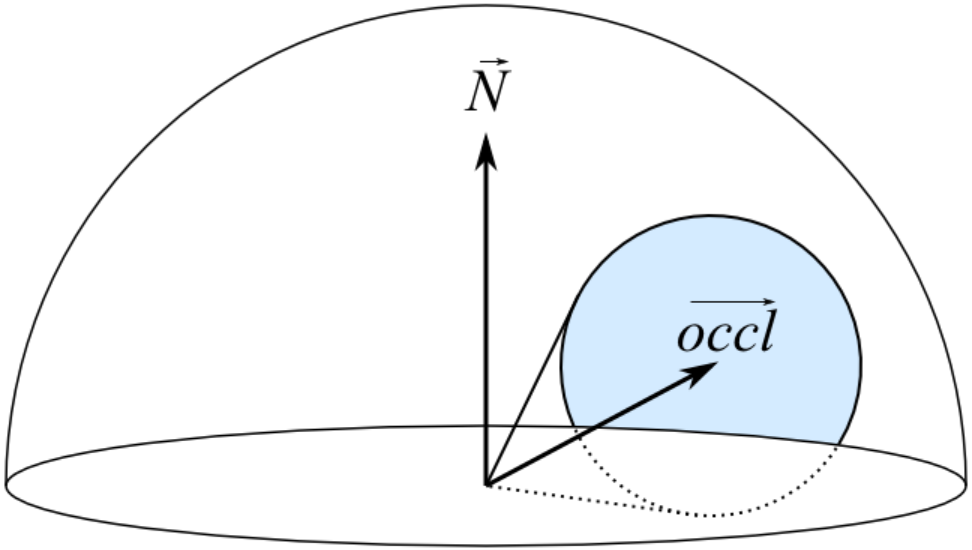
AO Fields: Applying

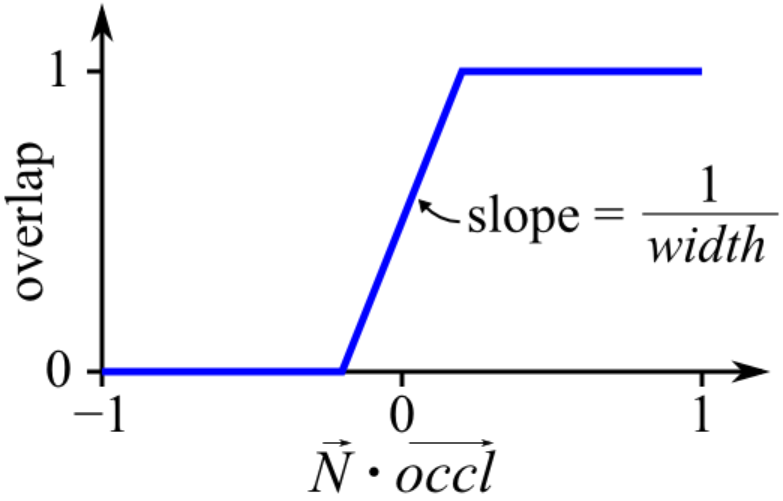
- Estimate occlusion using equation:

$$AO = 1 - strength \times width \times \text{saturate}\left(\frac{\vec{N} \cdot \overrightarrow{occl}}{2 \times width} + 0.5 \right)$$

- Strength is an artist-settable parameter per object; controls how dark the AO gets

AO Fields: Applying

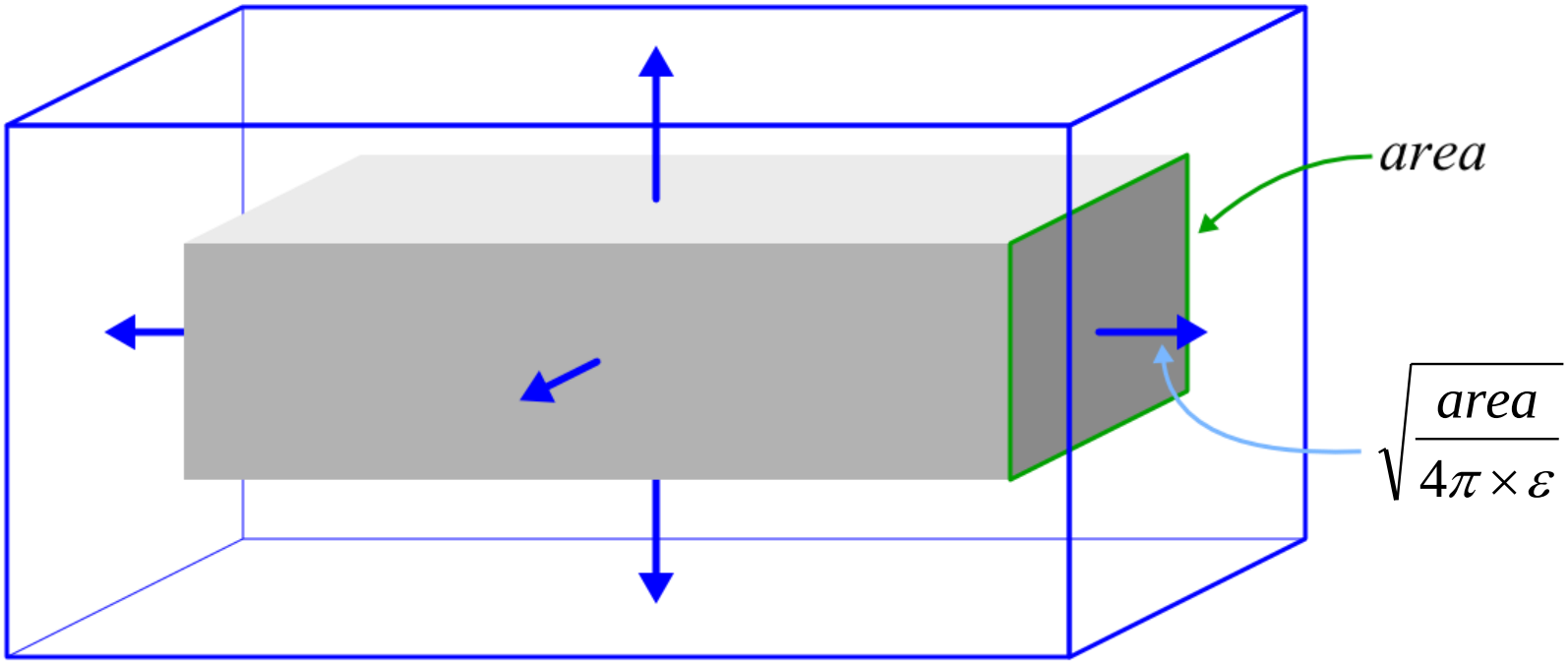




AO Fields: Applying

- Blend result into G-buffer's AO channel using multiplicative blending
  - No special treatment for double-blending – in our use cases, not really an issue

AO Fields: Bounding Box Size



AO Fields: Bounding Box Size

- From Malmer paper:

$$extend = \sqrt{\frac{area}{4\pi \times \varepsilon}}$$

- Epsilon is desired error. We used 0.25.

AO Fields: Texture Details

- Texture size: chosen by artist, typically 8–16 voxels along each axis
  - Car: 32×16×8 (= 16 KB)
  - Park bench: 16×8×8 (= 4 KB)
  - Trash can: 8×8×8 (= 2 KB)
- Format: 8-bit RGBA, no DXT
  - Density so low, DXT artifacts look really bad
  - No mipmaps necessary

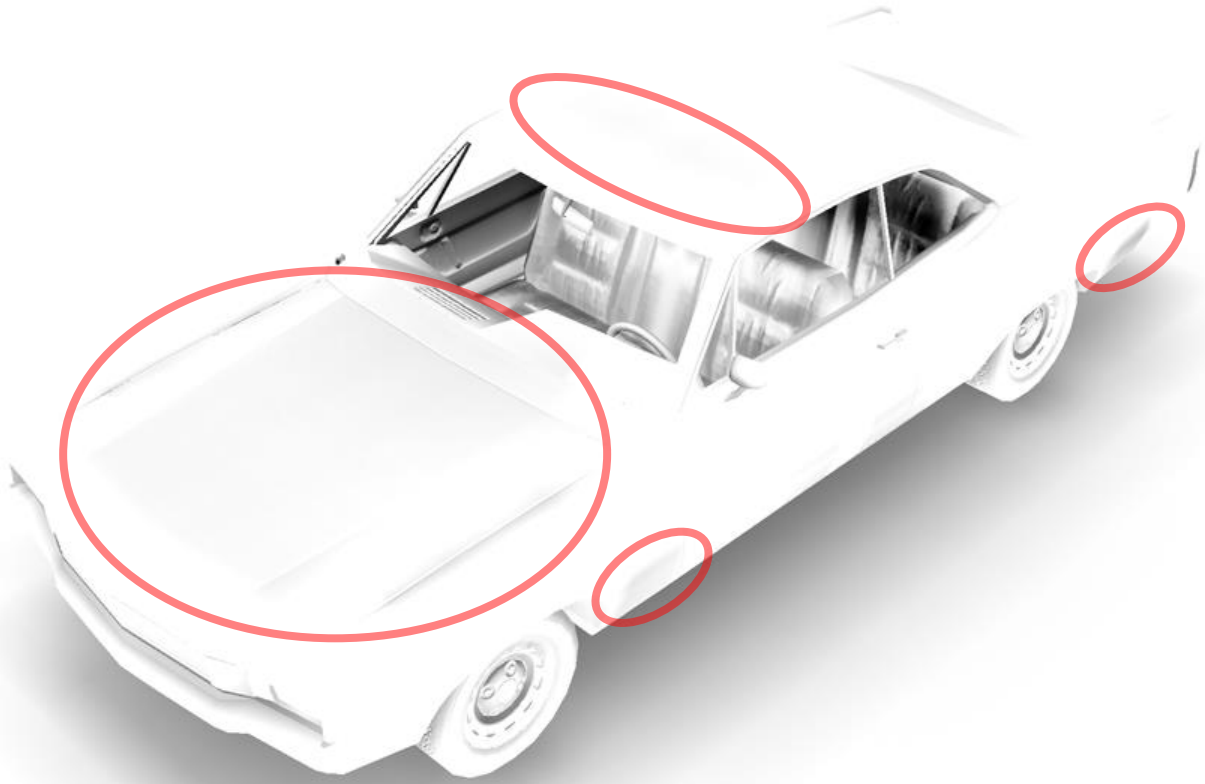
AO Fields: Visible Boundary

- Remap alpha (width) values at build time
  - Find max alpha among all edge voxels
  - Scale-bias all voxels to make that value zero:

$$alpha := \text{saturate}\left(\frac{alpha - alphaMaxEdge}{1 - alphaMaxEdge}\right)$$

Before

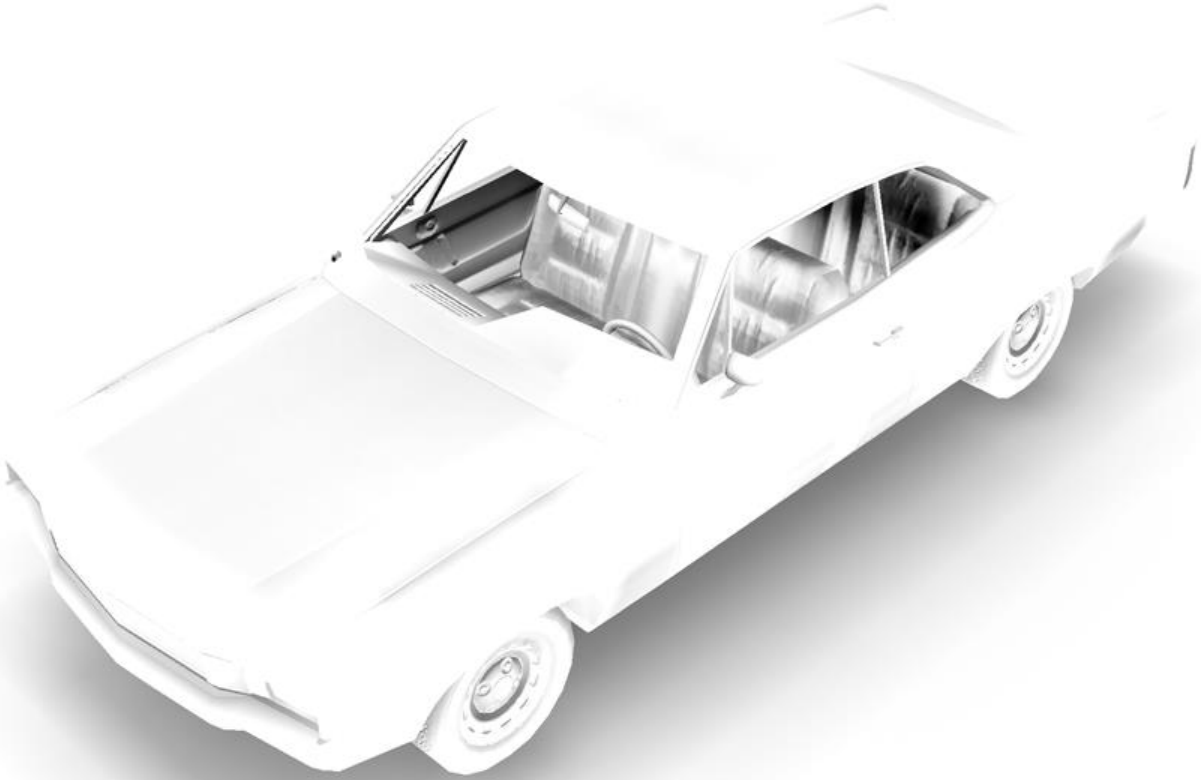
After



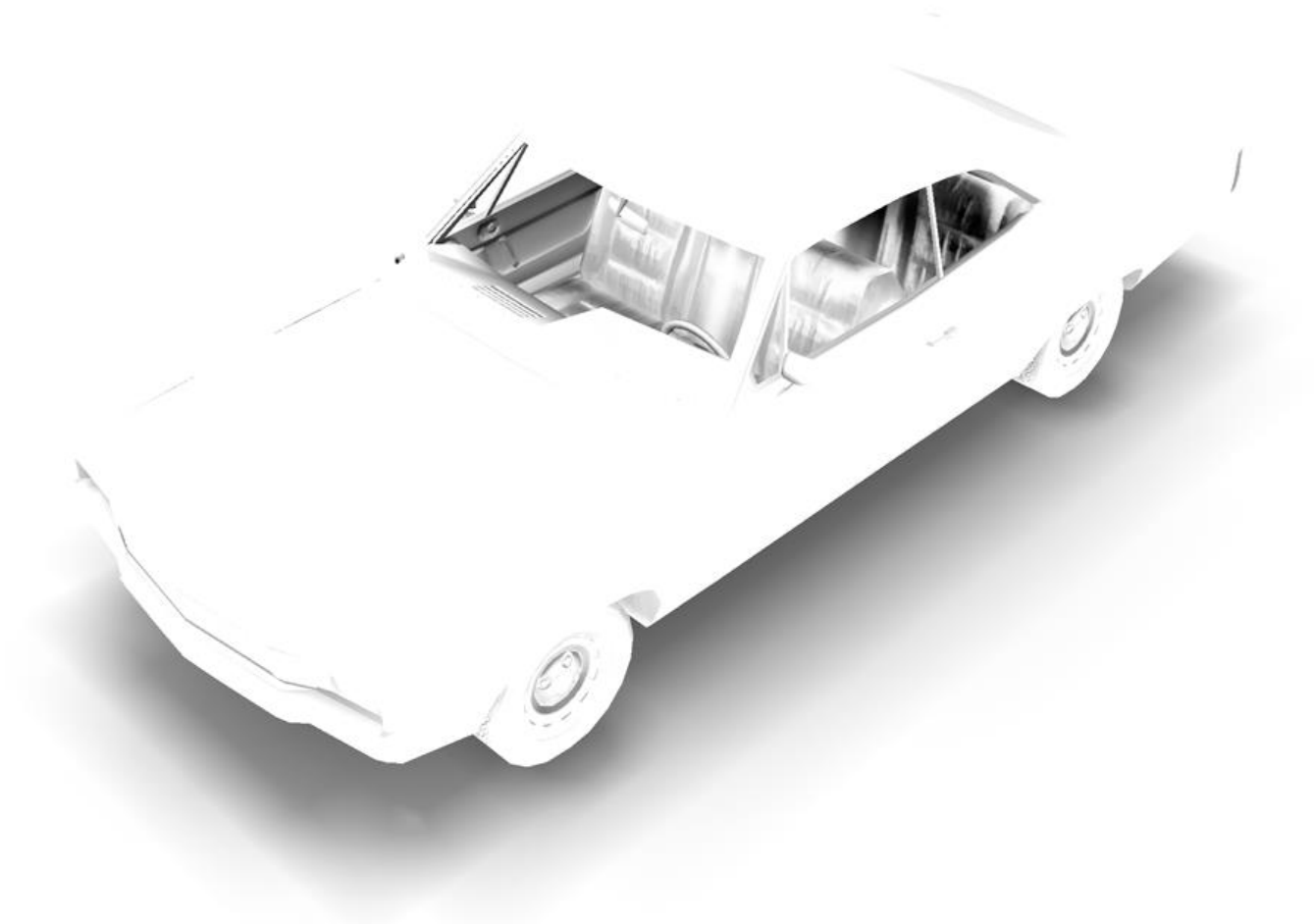
AO Fields: Incorrect Self-Occlusion

- Ideally: detect interior voxels and fix up
  - But identifying interior voxels is tricky
- Bias sample point away from target surface
  - In pixel shader, offset sample pos along normal
  - Bias length: half a voxel (along its shortest axis)

Before



After



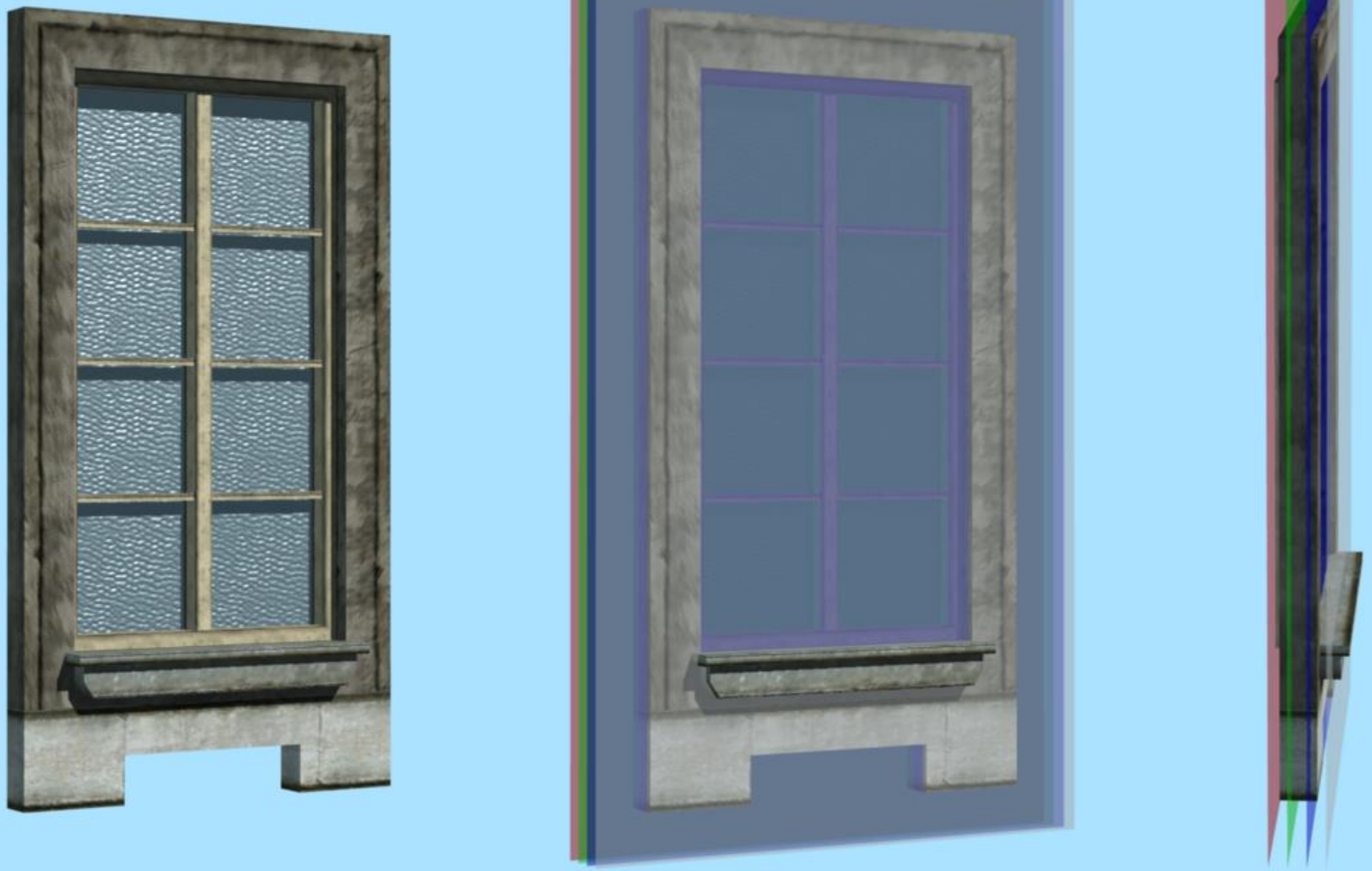
Ambient Occlusion Decals

AO Decals

- Planar version of AO Field
- Use cases: thin objects embedded in or projecting from a flat surface (wall or floor)
  - Window and door frames, air conditioners, electric meters, chimneys, manhole covers

AO Decals: Precomputing

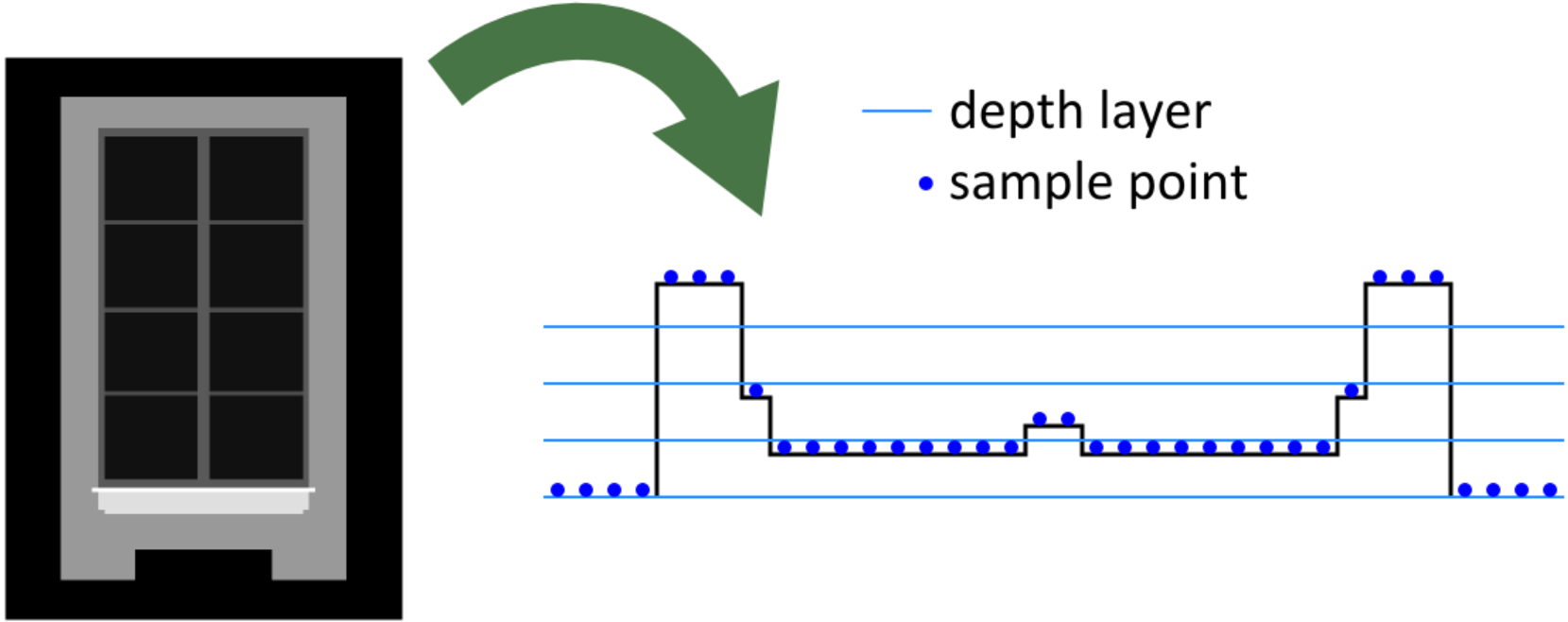
- Store a 2D texture, oriented parallel to the wall/floor
- Four depth slices stored in RGBA channels
  - No directional information stored; just occlusion fraction for hemisphere away from wall



AO Decals: Precomputing

- Render heightmap of source geometry
  - Parallel projection looking at wall/floor from front
  - Draw geom in grayscale, black at back of depth range to white at front
- Iterate over texels, take an AO sample just above heightmap at each texel
  - Trying to make sure we capture AO at the surface well, since that's where it will be evaluated

AO Decals: Precomputing



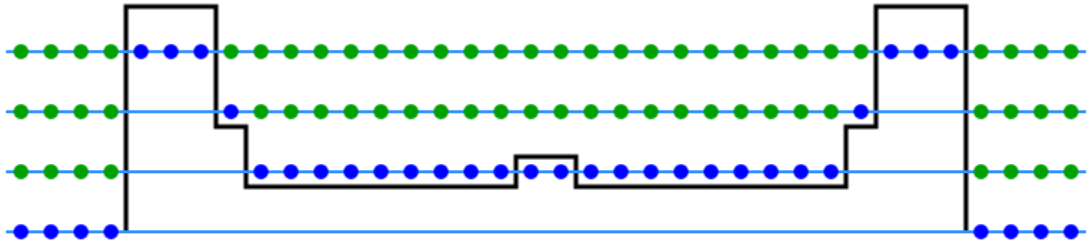
AO Decals: Precomputing

- Assign sample to nearest depth slice
  - Depth slice positions are depthRange * i / 4.0 (i = 0, 1, 2, 3)
  - Front of depth range (i = 4) always 0 occlusion
- Take additional samples above heightmap, to top of depth range

AO Decals: Precomputing

depth layer

- heightfield sample
- additional sample



AO Decals: Applying

- Same as for AO Fields, adjusted to work on depth slices in 2D texture
- No direction, so equation is just:

$$AO = 1 - strength \times occlusion$$

AO Decals: Applying

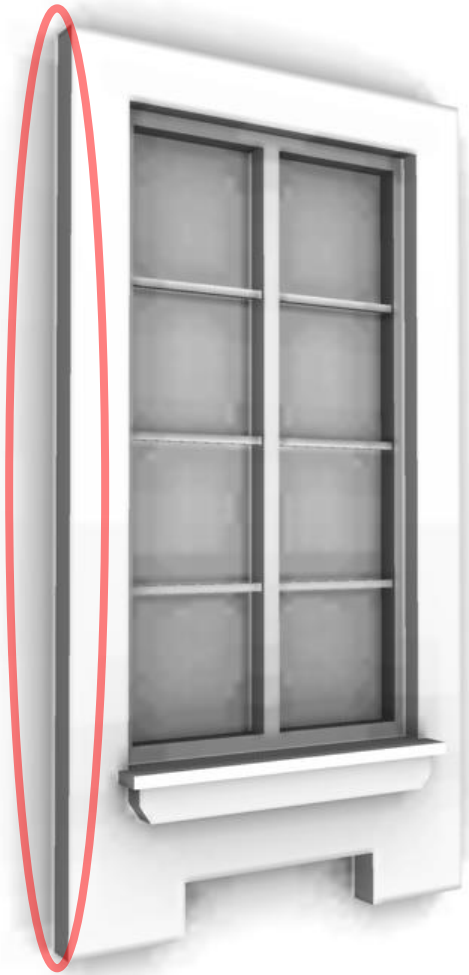
- Trick for linearly filtering samples packed into RGBA channels:

```
half4 deltas = half4(rgba.yzw, 0) - rgba;
half4 weights = saturate(depth*4 - half4(0,1,2,3));
half occlusion = rgba.x + dot(deltas, weights);
```

  - `rgba` is sample from decal texture
  - `depth` goes from 0 at back to 1 at front of depth range

AO Decals: Details

- Bounding box size: same formula as for AO Fields
  - Used 0.7 epsilon instead of 0.25 (smaller boxes)
- Texture size: 64–128 texels on each axis
- Format: DXT5
  - Introduces noise, but in practice not noticable when combined with color/normal maps etc.
  - 4–16 KB per texture



AO Decals: Halos Around Height Changes

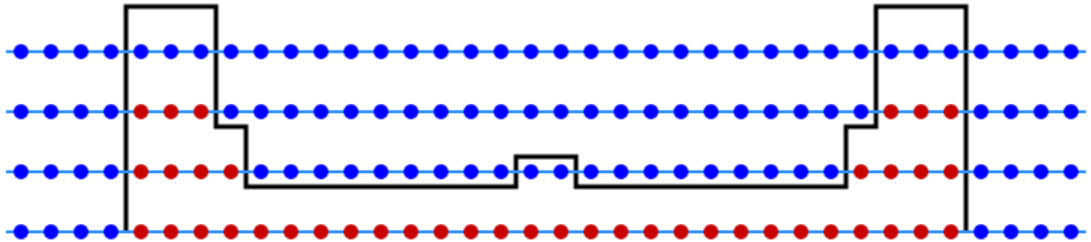
- Solution:
  - During precompute, mark samples underneath the heightmap as invalid
  - Run a “dilation” step to propagate valid samples into adjacent invalid ones

AO Decals: Artifacts

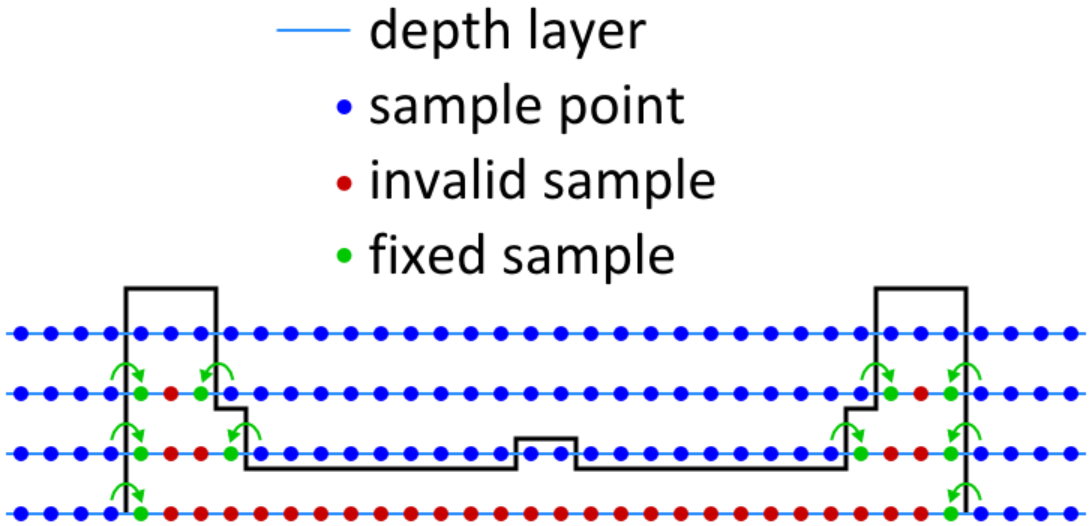
depth layer

sample point

invalid sample



AO Decals: Artifacts

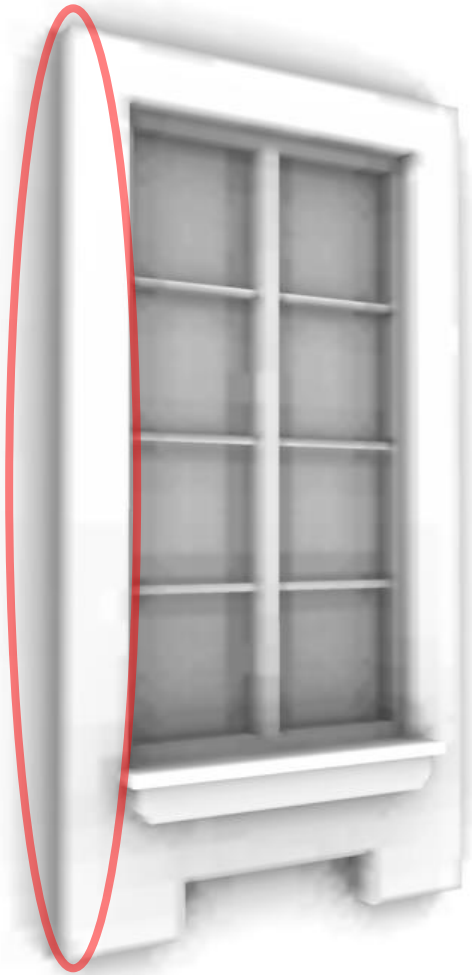


Before

After

Before

After



AO Decals: Edges Too Soft

- Solution: bake wall-occlusion term onto vertices

$$AO_{Wall} = 1 - strength \times \left(\vec{N} \cdot (-\vec{D}) \times 0.5 + 0.5 \right)$$

  - Unit vector *D* is the direction the decal faces
  - Multiply this into any other per-vertex AO on the source geometry

Before

After

Before

After

Infamous 2 – Fields/Decals Memory Use

- 116 assets with AO fields or decals applied
  - Heavy reuse: 9604 instances of those assets throughout the game world
- 569 KB total texture data
  - Average 4.9 KB per asset
  - Not all loaded at once (streaming open-world game)

Infamous 2 – Fields/Decals Performance

- Pixel-bound
- Typical frame draws 20–100 fields & decals
- Takes 0.3–1.0 ms on PS3
- Up to 2.3 ms in bad cases
  - Lots of fields in view, field covers the whole screen, etc.

Future Enhancements

- Faster offline renderer – precompute is slow
  - AO Field: 512–4096 samples each
  - AO Decals: 16K–64K samples each
- Handle undersampling better for AO Fields
  - Current solution can introduce additional artifacts
- Try it on characters
  - A field on each major bone

Wrap-up

- AO Fields & Decals fill in the gap between baked AO and SSAO
  - Medium-scale occlusion
- More interesting & dynamic ambient lighting

That's all, folks!

- Slides & videos at: http://reedbeta.com/gdc/
- Contact me: nathanr@suckerpunch.com