Towards Cinematic Quality, Anti-aliasing in Quantum Break

Tatu Aalto
Senior Graphics Programmer, Remedy Entertainment

GAME DEVELOPERS CONFERENCE EUROPE COLOGNE, GERMANY · 15–16 AUGUST 2016

I’m Tatu Aalto, Senior Graphics Programmer at Remedy Entertainment. You might know Remedy from the titles like Max Payne, Alan Wake, and Quantum Break. Quantum Break was released earlier this year on xbox one and PC. We have also steam version coming out later this year. In this presentation, I will go over the anti-aliasing methods, used in Quantum Break.

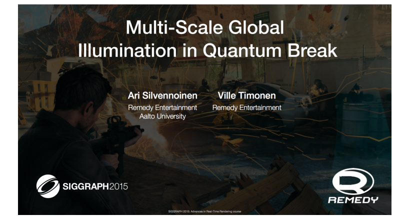
Cinematic Quality

- Quality of lighting
  - Indirect lighting
  - Participating Media
- Pixel quality
  - Geometry aliasing
  - Specular aliasing

This talk is titled with ‘towards cinematic quality’. I’ll start off by quickly explaining what I mean by that. One of the hardest problems when making real-time graphics is to bind different elements on the screen together and make the picture temporally stable. We gave a presentation about our Multi-Scale Global Illumination approach on the 2015 Siggraph.

Cinematic Quality

- Quality of lighting
  - Indirect lighting
  - Participating Media
- **Pixel quality**
  - **Geometry aliasing**
  - **Specular aliasing**



SIGGRAPH 2015

In this presentation, I'm going to focus on pixel quality and more specifically anti-aliasing. We shipped Alan Wake on Xbox360 with 4xMSAA and were happy with the results. Especially alpha to coverage played an important part in Alan Wake, as most of the environments were filled with trees and foliage. In Quantum Break, we were not building as much forest environments, but still wanted to focus on pixel quality rather than the quantity.

Contents

- Introduction to light pre-pass rendering
- Our MSAA pipeline
- Temporal anti-aliasing and upscale

We have light pre-pass rendering engine with clustered light culling. I'll start off by describing what is light pre-pass renderer, what good sides it has, and which parts we found problematic while working on Quantum Break.
After that, I'll present our slightly unconventional solution to MSAA.
And finally, I'll go over temporal Anti-Aliasing and upscaling used in Quantum Break.

Introduction to light pre-pass rendering

Light pre-pass - Overview

- Like full deferred, idea is to separate lighting from geometry rendering.
- Unlike full deferred, geometry is drawn two times.
- First pass writes everything needed for lighting.
- Second pass reads lighting and adds rest of the material.

Geometry Buffer, 3 x 32-bits

<table>
<tr><td colspan="3">Normal</td><td>Smoothness</td></tr>
<tr><td>Translucency</td><td>Specular Intensity</td><td colspan="2">Material ID</td></tr>
<tr><td colspan="4">Depth</td></tr>
</table>

Similar to full deferred style rendering, the core idea is to separate lighting and geometry drawing into two separate steps. This is done in order to reduce complexity of shading. What separates light pre-pass, is that we draw all geometry twice. On the first geometry pass, we write everything needed for the lighting into textures. For us, this includes Normal, Depth, Smoothness, Intensity of specular albedo and Material ID. Based on these properties, we calculate lighting and related screen space techniques. Finally, on the second geometry pass, we combine lighting with the rest the of the material properties.

GAME DEVELOPERS CONFERENCE EUROPE COLOGNE, GERMANY · 15–16 AUGUST 2016

Light pre-pass - Advantages

- Thin geometry buffer for lighting.
- Easy to add material variation on the second geometry pass.
- Often combined with MSAA on the second geometry pass.

Compared to full deferred, there are few advantages to light pre-pass. First of all, you can often keep geometry buffers slightly smaller than with full deferred. Properties that you leave out from the geometry buffer, must be independent of lighting. In Quantum Break, we are only adding diffuse colour for most of the materials.

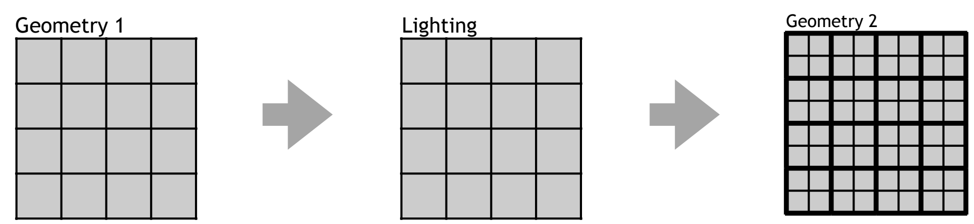
Light pre-pass - Advantages

- Thin geometry buffer for lighting.
- Easy to add material variation on the second geometry pass.
- **Often combined with MSAA on the second geometry pass.**

For us, the main motivation on running the second geometry pass, is to support MSAA. Lets dive a bit deeper into that.

Light pre-pass - MSAA

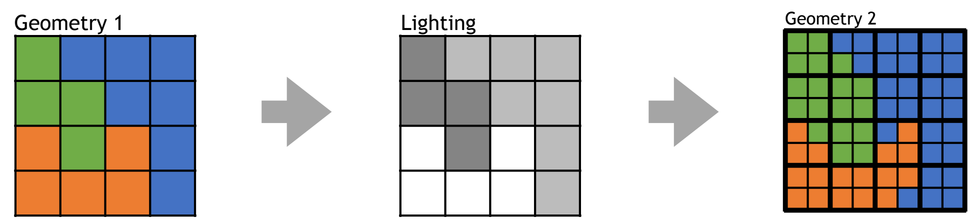
- Second geometry pass decides what lighting sample to use.
- Common way is to compare current geometry to previously drawn geometry buffer and choose the closest match.
- Works very nicely when you have a match. Problems arise when there is no good samples to choose from.



With light pre-pass, MSAA is usually implemented in a such way that you first draw single sampled geometry buffer. Then you calculate lighting on top of that, in the same resolution. And finally, in the second pass, turn on MSAA and sample lighting into what ever sample count you choose. In the diagram here, you can see that the sample count stays the same during the first geometry pass and the lighting, but is increased for the final geometry drawing.

Light pre-pass - MSAA

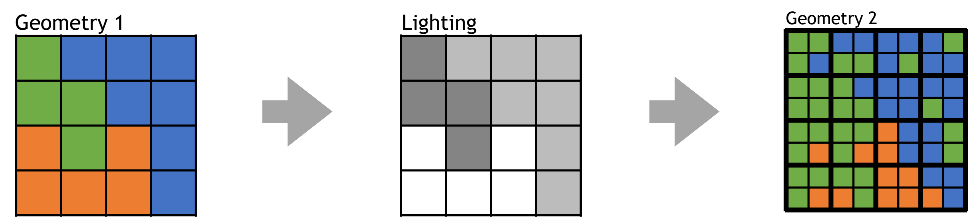
- Second geometry pass decides what lighting sample to use.
- Common way is to compare current geometry to previously drawn geometry buffer and choose the closest match.
- Works very nicely when you have a match. Problems arise when there is no good samples to choose from.



As there is no one-to-one mapping from lighting to second geometry pass, you need to somehow decide which light sample is used at each geometry sample. Decision is often made by calculating geometry properties of the currently drawn sample, and then by comparing it against geometry buffer. This works nicely as long as you happen to have good lighting samples to choose from. What happens if the geometry is a lot more scattered on the second geometry pass?

Light pre-pass - MSAA

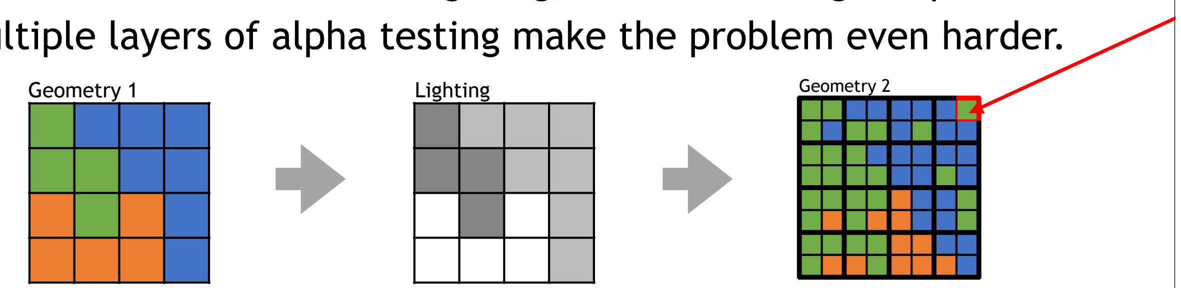
- For 2013 E3 demo we were upgrading our hair and foliage shaders to use checker pattern for alpha test output.
- Checker pattern on alpha test output is hard to control. Either you run on half resolution lighting, or start missing samples.
- Multiple layers of alpha testing make the problem even harder.



The problem is, that by first drawing the geometry buffer single sampled, you are basically counting on your luck on quality of lighting samples. While doing a demo for 2013 E3, we were upgrading our hair and alpha test rendering. Our plan was to dither hair geometry in a such way, that we get consistent lighting for the hair through our normal pipeline. In this diagram, I have more scattered geometry on the right side of the image where the MSAA is used. Left side could still be the result of drawing such geometry without MSAA.

Light pre-pass - MSAA

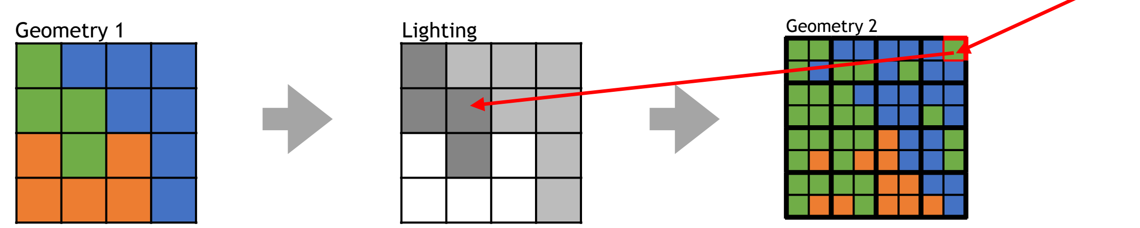
- For 2013 E3 demo we were upgrading our hair and foliage shaders to use checker pattern for alpha test output.
- Checker pattern on alpha test output is hard to control. Either you run on half resolution lighting, or start missing samples.
- Multiple layers of alpha testing make the problem even harder.



If you think of search radius needed for finding a valid lighting sample for the top right corner pixel on the rightmost chart, it is pretty clear that reconstructing the lighting for this sample is going to be expensive.

Light pre-pass - MSAA

- For 2013 E3 demo we were upgrading our hair and foliage shaders to use checker pattern for alpha test output.
- Checker pattern on alpha test output is hard to control. Either you run on half resolution lighting, or start missing samples.
- Multiple layers of alpha testing make the problem even harder.



And because the closest light sample, that's on the same surface, is quite far a way, the lighting quality will not be good. Lets look at the real world example.

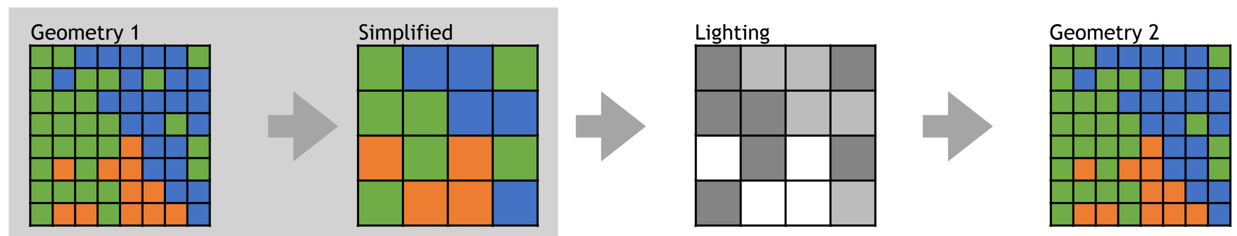
This is work in progress image of the demo scene we were doing for E3. We applied better BRDF, and tuned alpha testing on hair drawing among some other improvements.

This is closeup image of the hair. Let’s turn on debug visualisation, to see what kind of lighting samples we have here.

When looking at the hair in debug mode, you can see the alpha test pattern we were using back then. This shows the sample distribution in geometry buffer after the first geometry pass. In order to produce the final image, we take the lighting from this, but apply it to MSAA samples to get smooth result. It is very hard to tweak the sample pattern into such, that you always find a good sample within a reasonable sized region. Especially if you want the search region to be fixed to four closest pixels. You can imagine that the situation just gets worse, if there is second alpha layer behind the hair.

Light pre-pass - MSAA idea

- Keeping the relevant samples is the key issue to solve.
- Would like to use 4xMSAA geometry buffer, but can’t afford lighting it directly.
- Reduce data with k-means?



It would be awesome to actually run the first geometry pass with MSAA also. That would guarantee good lighting information for all the samples we need. But, we didn’t want to run expensive lighting on top of increased sample count. Our first test was to run k-means on geometry buffer in order to reduce the data. Resulting quality was looking promising, but the performance wasn’t. We needed to do something in between. It needed to be fast enough to be running on console platforms, but still a lot better than just dropping randomly something off.

Our MSAA pipe

Now you are hopefully familiar with light pre-pass rendering and the problem of keeping the relevant geometry samples for the MSAA. Next, I'll go over what we do in Quantum Break.

Geometry Clustering Overview

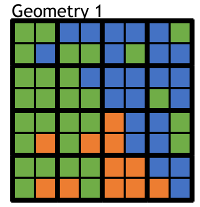
- Render Geometry Buffers to 4xMSAA targets.
- Reduce samples from 4xMSAA to non-MSAA (from 4 samples to 1 sample).
- Run expensive shading on reduced samples.
- Reconstruct back to 4xMSAA.

- More geometry samples (relatively small cost), less samples to light (compute heavy)

The basic idea is to run both geometry passes with MSAA, and reduce the sample count for expensive light shading. Our geometry passes are running on 4xMSAA in 720p resolution. This gives us close to 3.7 million nicely distributed geometry samples to choose from. Lighting is still calculated on non-MSAA 720p, with less than million samples to update. Our approach has some similarities to Aggregate G-Buffer Anti-Aliasing [by Cyril Crassin. et al], but was done to target console performance.

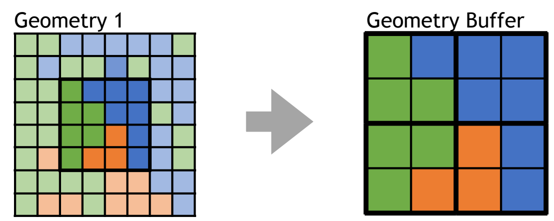
Geometry Clustering

Geometry 1



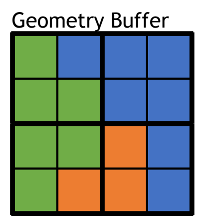
Let's use the complex geometry I showed you earlier as an example. I'll take a small portion of this geometry, so that we have something simple to focus on.

Geometry Clustering



The region I chose, covers 2x2 pixels. Each pixel contains 4 MSAA samples. In total, there is 16 geometry samples selected.

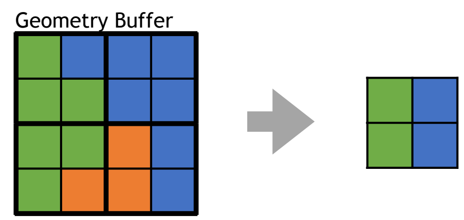
Geometry Clustering



Our reduction runs with input of these 16 samples, and the output is four samples. This gives us four to one ratio on downsampling.

Geometry Clustering

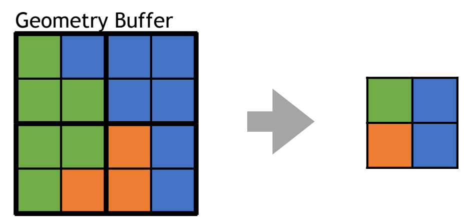
- This is not a good way to choose four samples out of 16.



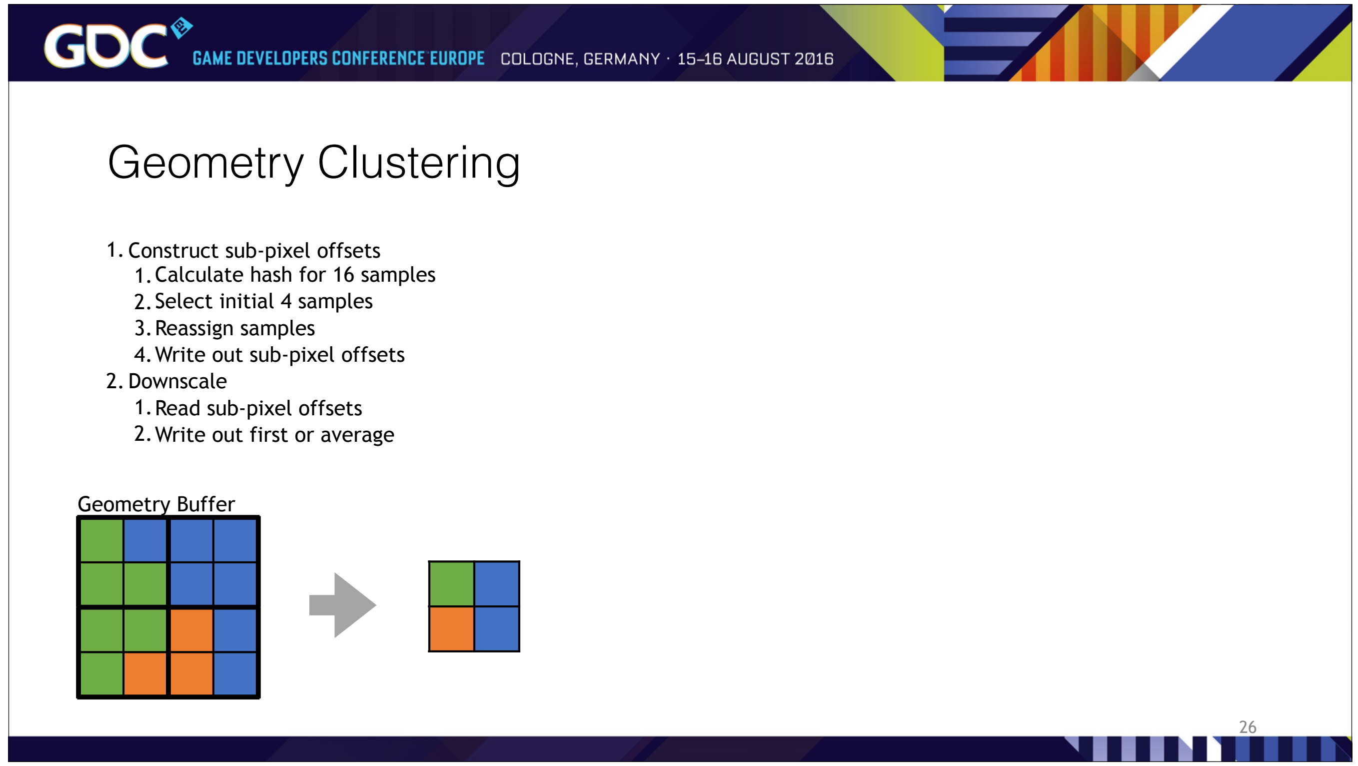
Here is an example of how to not choose the samples. We are missing the orange region completely on output.

Geometry Clustering

- This looks a lot better. We are not missing any region completely.



This is what we want. In here, we have preserved all the different regions of the original geometry buffer. Let’s look at how we get there with reasonable performance.



On the left side, you can see the breakdown of the algorithm, that I'll go over step-by-step.

1.1 Calculate hash

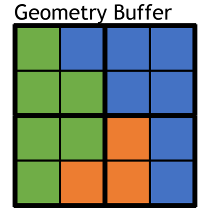
1. Construct sub-pixel offsets
   1. **Calculate hash for 16 samples**
   2. Select initial 4 samples
   3. Reassign samples
   4. Write out sub-pixel offsets
2. Downscale
   1. Read sub-pixel offsets
   2. Write out first or average

- 32-bit value with 16-bit normal, 8-bit depth, 8-bit material ID.

Hash value

Normal																Depth								Material ID							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Geometry Buffer



Hash

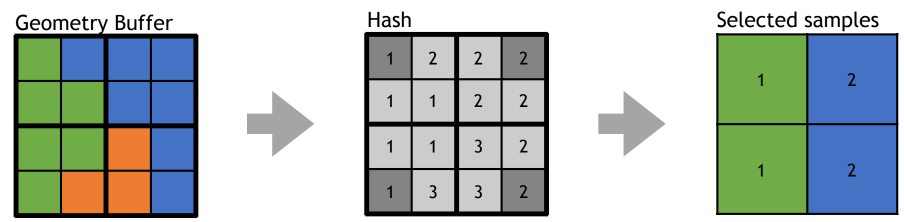
1	2	2	2
1	1	2	2
1	1	3	2
1	3	3	2

We start off by calculating a hash value for each sample in geometry buffer. Here you can see 16 geometry buffer samples that are turned into hash values each by packing normal, depth and Material ID of the geometry into 32-bits.

1.2 Initial selection

1. Construct sub-pixel offsets
    1. Calculate hash for 16 samples
    2. **Select initial 4 samples**
    3. Reassign samples
    4. Write out sub-pixel offsets
2. Downscale
    1. Read sub-pixel offsets
    2. Write out first or average

- We start by selecting four samples on the corners.
- By selecting the corners, we hope to maximise the amount of unique samples.

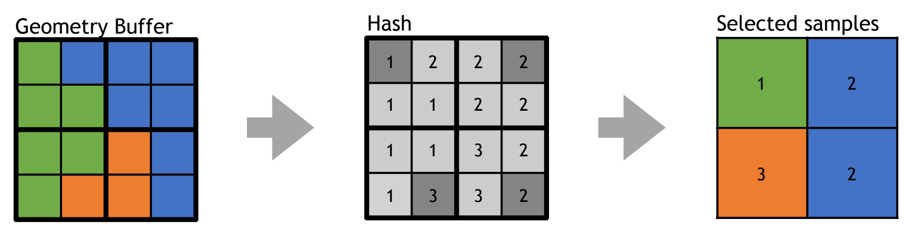


We first pick the corners as our initial guess on good samples to preserve. Selected samples are darkened in the middle chart. On the rightmost chart, you can see that we managed to pick only two of the regions. If we use this one directly, we would lose orange region 3 completely.

1.3 Reassign

1. Construct sub-pixel offsets
   1. Calculate hash for 16 samples
   2. Select initial 4 samples
   3. **Reassign samples**
   4. Write out sub-pixel offsets
2. Downscale
   1. Read sub-pixel offsets
   2. Write out first or average

- For each selected sample, test if duplicate exists. Replace duplicates with a value that is not selected.
- Make sure to avoid unnecessary shuffling. Staying aligned to screen makes sense.

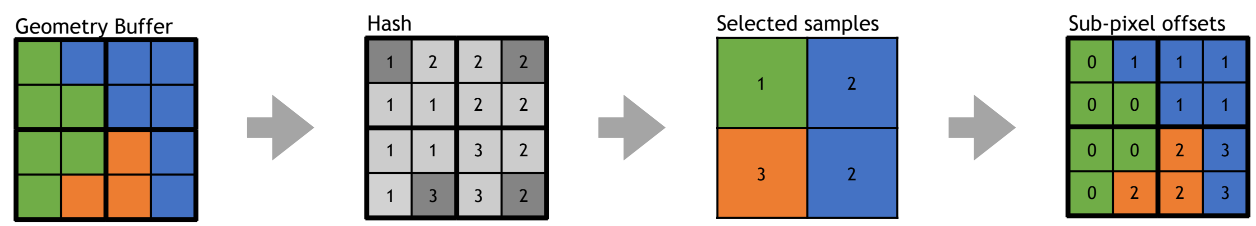


After initial guess, we start doing reassignment rounds. Each round checks if there are duplicate values on the four selected values on the right side. If there are, we try to replace those with something unique from the left side. If all the values on the right side are already unique, there is no point trying to reassign anything. This is slightly similar to k-means, but tuned to work nicely on exactly this problem.

1.4 Write sub-pixel offset

1. Construct sub-pixel offsets
   1. Calculate hash for 16 samples
   2. Select initial 4 samples
   3. Reassign samples
   4. **Write out sub-pixel offsets**
2. Downscale
   1. Read sub-pixel offsets
   2. Write out first or average

- 32-bits per 2x2 pixel region.
- Each MSAA sample has 2-bits, telling sampling location on the final 2x2 output.
- Pixel shader outputs half sized 360p image.



Finally we write out which sample from the reduced buffer should be used for each MSAA sample. These sub-pixel offsets are very important to keep in mind. By storing sub-pixel offset here, we always know what low resolution sample corresponds to each MSAA sample. By keeping this information, we don't need to make guesses by comparing geometry properties of different buffers. Constructing the offsets takes roughly .7ms on xbox one.

White dots on the right side of this image are showing the locations that have sub pixel offset applied. You can see that the offsets are following geometry edges. These locations would be on risk of not having a good lighting samples with traditional light pre-pass MSAA implementation.

GDC GAME DEVELOPERS CONFERENCE EUROPE COLOGNE, GERMANY · 15–16 AUGUST 2016

2.1 Read offsets

1. Construct sub-pixel offsets
   1. Calculate hash for 16 samples
   2. Select initial 4 samples
   3. Reassign samples
   4. Write out sub-pixel offsets
2. Downscale
   1. **Read sub-pixel offsets**
   2. Write out first or average

Sub-pixel offsets

0	1	1	1
0	0	1	1
0	0	2	3
0	2	2	3

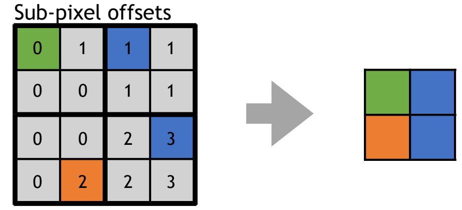
- Each 2x2 tile shares single 32-bit sub-pixel offset.
- Full screen pass running in normal pixel shader. Output is final geometry buffer that will be used for AO, SSR, GI and lighting.

After sub-pixel offsets are written out, we use those to construct actual geometry buffer for the lighting.

2.2 Write geometry buffer

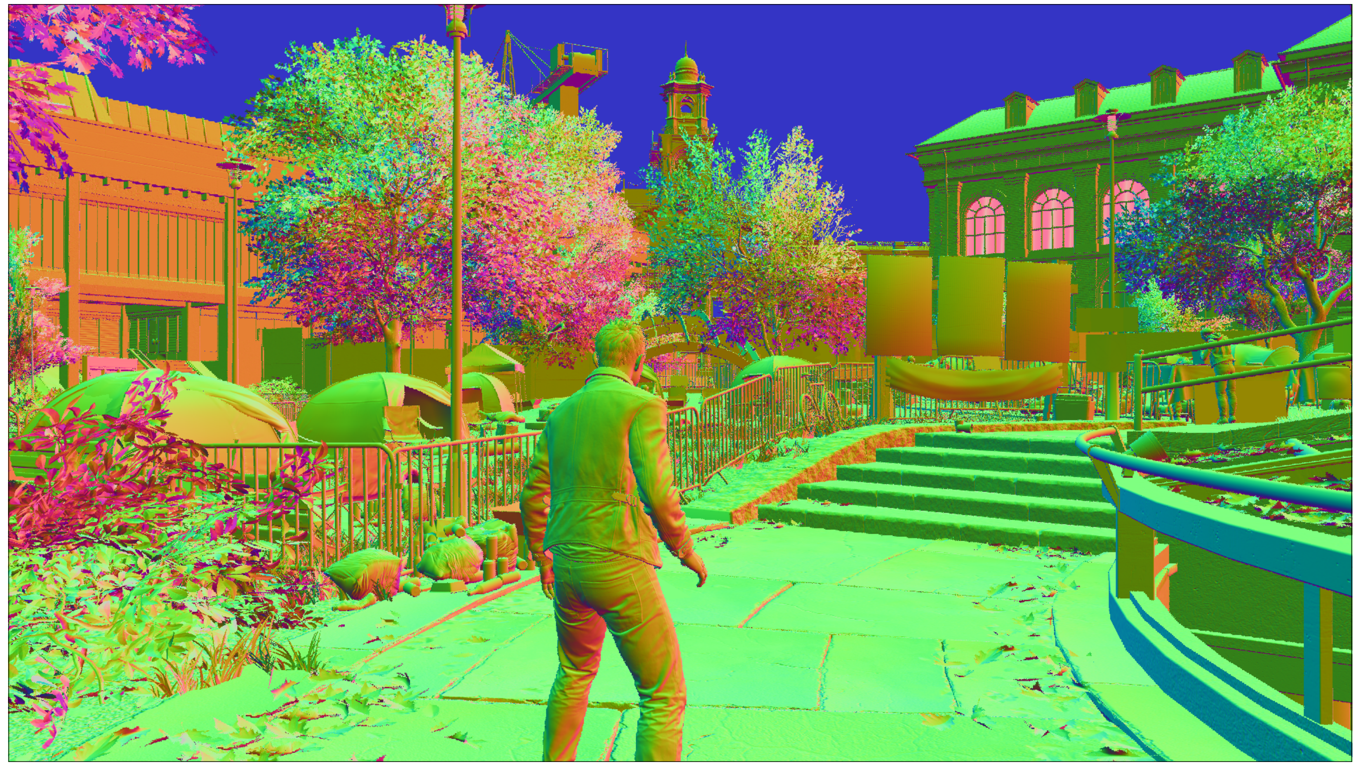
1. Construct sub-pixel offsets
    1. Calculate hash for 16 samples
    2. Select initial 4 samples
    3. Reassign samples
    4. Write out sub-pixel offsets
2. Downscale
    1. Read sub-pixel offsets
    2. **Write out first or average**

Sub-pixel offsets

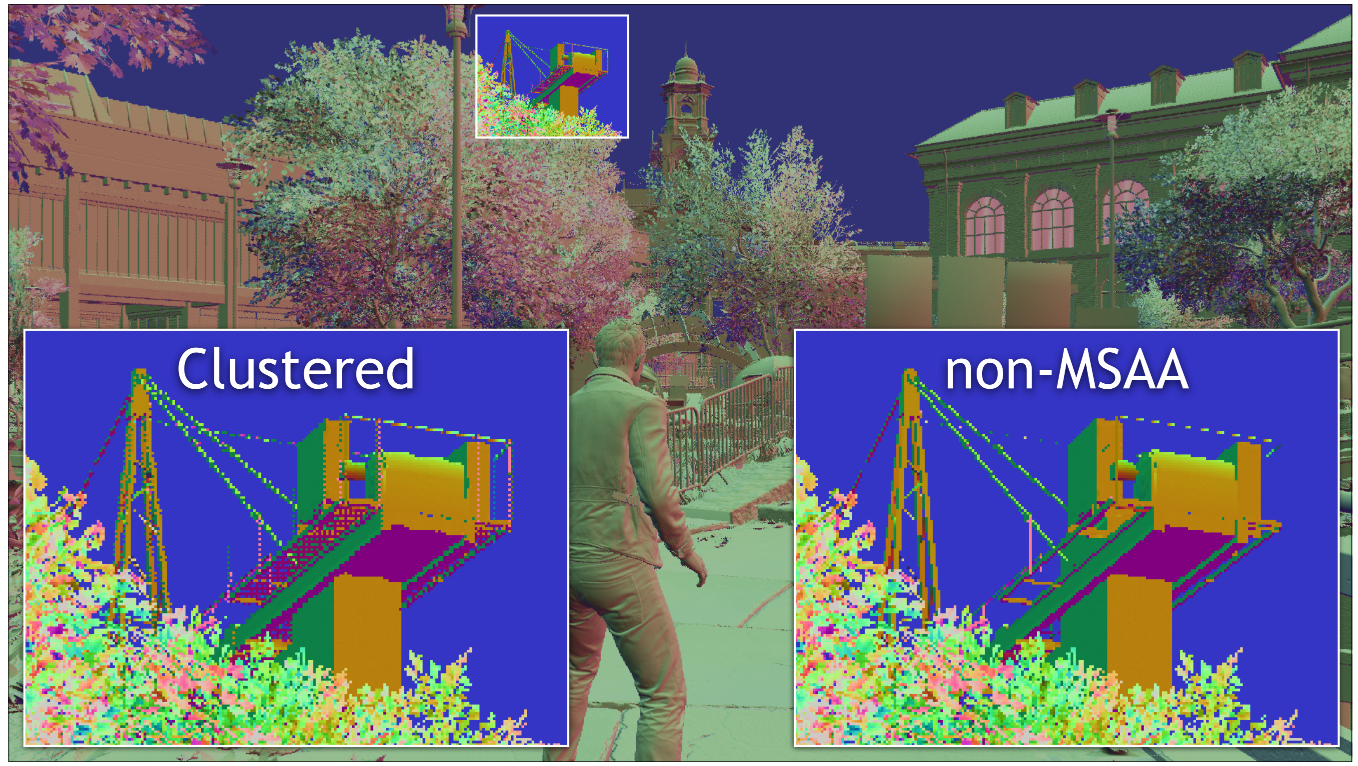


- We output first match. Average gives slightly better results with increased cost.

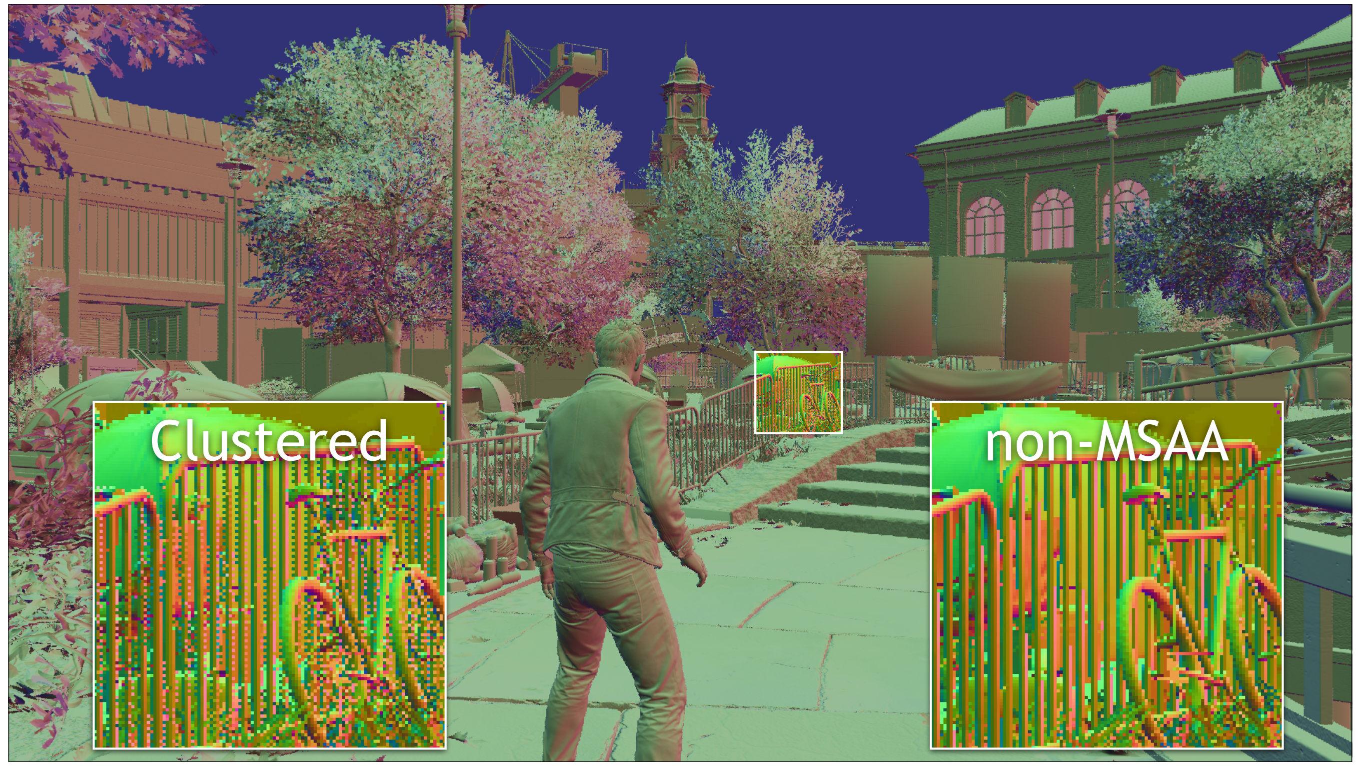
There can be multiple different MSAA samples in the original geometry buffer, that have been grouped into the same slot on the output. We ended up just picking one of the samples and writing it out directly. It would be also possible to take average of all contributing samples, but it comes with increased performance cost. We didn’t find using average important enough. Downscale by using the first hit takes roughly .5ms on xbox one. Total cost of clustering varies a bit based on how many MSAA samples are actually filled.



This is the resulting normal buffer after resampling. It seems traditional, but lets look at it closer.



Clustering MSAA preserves geometry detail, and plays well with alpha testing. If you look at the closeup on the right side, wires have almost disappeared. You would need large search neighbourhood in order to reconstruct anything decent with light samples calculated based on the right side buffer.

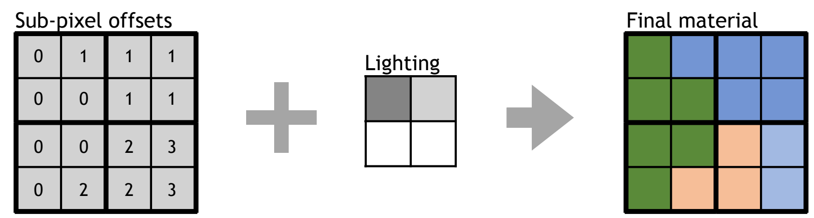


Here is another closeup. Remember that our reconstruction neighbourhood is 2x2 pixels. Every block of that size, must contain the information needed for upscaling afterwards. This is why almost all geometry edges actually end up containing some raster pattern. After these geometry buffers are constructed, we calculate lighting on the 720p resolution.

This is the final lighting at the location. Next, lets look how we scale this back into 4xMSAA resolution with good quality. Remember that our engine is built in light pre-pass style, so we draw scene geometry twice.

Second geometry pass

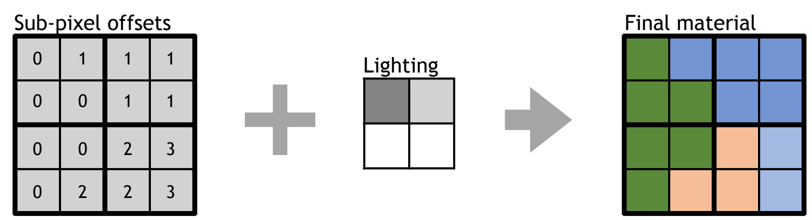
- Sub-pixel offset tells which lighting sample to use for each MSAA sample. Read lighting sample based on 2-bit offsets.
- No need to calculate geometry properties for current sample in order to compare against geometry buffer.



Second geometry pass is drawn using 4xMSAA again. We input lighting, based on the sub-pixel offset for the currently shaded geometry sample. Without sub-pixel offset, we would need to compute geometry properties, by sampling normal maps and other affecting properties that geometry might have. By comparing those values against what is written into geometry buffer, we would need to determine which light sample to uses.

Second geometry pass

- Use sample coverage in the pixel shader (SV_Coverage in HLSL). Using average of covered samples improves quality, but is a lot more expensive. We use firstbitlow and sample once.



In the sample coverage bit mask, all the covered MSAA samples of the pixel are marked as set bits. It would be correct to sample lighting at all the set samples and use the average of the lighting pointed by these offsets. Instead of calculating the average, we find the first set bit, and use the lighting pointed by corresponding sub-pixel offset. Using the average, provides slightly better results, but is a lot more expensive to calculate. Lets see how the end result looks like..

This is the final image with MSAA.

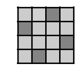
And here, you can see a close up comparison. On the left side, I have closeup with previously presented MSAA clustering. On the right side, I have comparison to non-MSAA version. You can see that the small geometry detail is better preserved with our clustering, and we are still calculating the lighting in non-MSAA 720p resolution. Sub-pixel offsets can be used for upscaling other draw passes also in addition to opaque geometry. In this image, we apply sub-pixel information to non-MSAA transparent targets also. This gives us sub-pixel accurate edges between opaque and transparent geometry. In addition to MSAA, we also use temporal anti-aliasing.

Here, I have the same location magnified with temporal anti-aliasing enabled. In addition to geometry quality, temporal anti-aliasing works with the aliasing that comes from the shading.

Temporal anti-aliasing and upscaling

I will now go over our approach to temporal anti-aliasing and upscaling.

Temporal Anti-Aliasing - Overview

- Four frames with sub pixel camera offset.
- We use rotated grid offsets. 

Our temporal anti-aliasing and upscaling are based on the final image of the current frame and three previous frames that are kept in memory. On each frame, we move camera with sub-pixel offset from rotated grid pattern. In addition to anti-aliasing, we take advantage of previous frames when upscaling the image to display resolution. I'll start of with upscaling, as that is the reason we store multiple frames, instead of using accumulation buffer. Lets start with comparison between our upscale and simple linear upscale from 720p to 1080p.

On the left side, you can see the upscaling used in Quantum Break. For the comparison, the right side uses linear filtering for magnification. As you would expect, linear upscale from single image results in serious blockiness. Our upscale on the left side is actually also taking linear samples, but from four different buffers: current and three previous frames, that have different sub-pixel offset applied. I'll first go over how we prepare three stored frames for upscale, and then I will show how we combine the frames.

Temporal Anti-Aliasing - Transform

- Transform all three previous frames to current projection every frame to keep delta between frames as small as possible.
- Single compute pass shares clamp data.

We start of by transforming all three previous frames to current viewport. In order to keep the transformation delta as small as possible, it is important to update all stored buffers every frame. In the small pictures, you can see the amount of difference between previous frames and the current frame. I'll show larger images, so that it is easier to see the difference.

This is the difference of the current frame and the previous frame, before anything has been transformed. The difference is quite big everywhere, as you would expect, as I was rotating the camera around the player when taking these images. Note for instance that the light poles are duplicated in this image. Now, lets apply the transformation and look at the difference again.

Second lamp poles have disappeared, but now we have quite a large difference where the previous frame didn't contain any good information because of occlusion. In order to make ghosting less visible, we use the colour neighbourhood of the pixel on the most recent frame, to clamp results after transformation. This method has been documented quite extensively, but I'll quickly go over what we do.

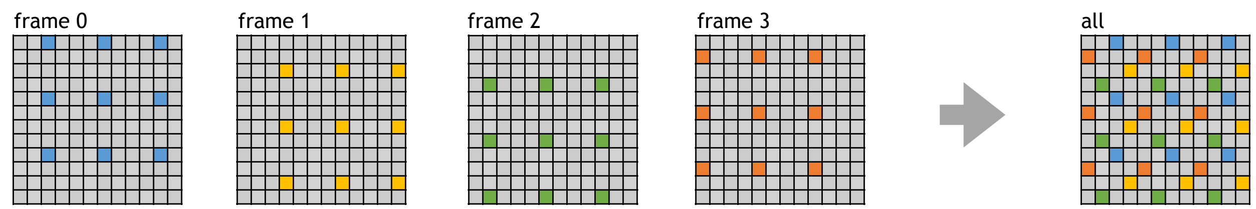
Temporal Anti-Aliasing - Clamp

- Min/max of neighbour pixels on current frame.
- We ended up extending range on low velocity to damp flickering caused by sub pixel offsetting camera. Would like to have something more robust.
- Clamp in xyY and extend only luminance.
- Balance between aliasing and ghosting.

Different methods for clamping look a bit different, but to put it short we are balancing between aliasing and ghosting here. If the input signal has a lot of high frequency noise, you need large bounds to avoid aliasing and flickering caused by per frame camera offset. Extending bounds leads to more ghosting where the re-projection fails. We guide extension bounds with velocity of opaque geometry.

Temporal Anti-Aliasing - Upscale

- Combine results from four frames.
- Previous frames reprojected and clamped already.
- Good location for upscale, as the samples between frames are interleaved. Linear sampling works surprisingly well.



After clamp and transformation, we have four textures that represent current frame with slight sub pixel offset. On the bottom, you can see the visualisation of the sample offsets between four different frames. Right side shows these samples combined into single image. On 720p resolution we have roughly 3.7 million samples. When drawing to 1080p we are in theory super sampling. This is of course not quite true, as the image is rarely completely still, and even if it is, clamping samples to colour neighbourhood puts limit to amount of detail that can come though. Still, doing the upscale from multiple frames has a huge advantage when compared to traditional single image upscale.

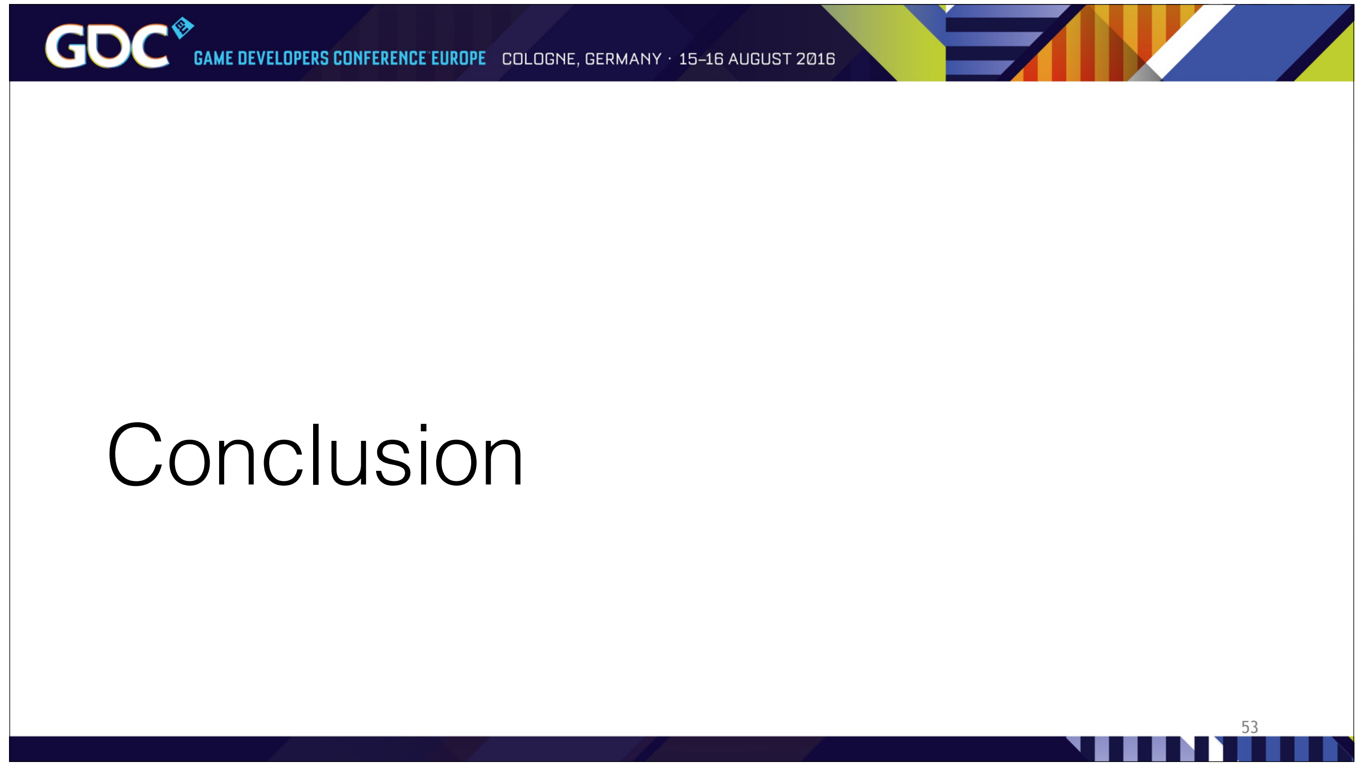
Given that our samples are interleaved in rotated grid pattern, it would feel natural to use some proper distribution when constructing the final image. We found out that simply using hardware linear sampling from each of the four frames works surprisingly well and is fast. On the left side, you can see the result of linear sampling we use in Quantum Break. On the right side, I have comparison upscale with Mitchell-Netravali using 4x16 samples. Difference against somewhat soft parametrisation is small, and for us it didn't seem like worth paying the extra for the outcome.

[Upscale to 1440p from 4 previous 720p frames.]
[This is comparison between bilinear filtering we use, and 16 samples per frame filtered result. Filter used on right side is Mitchell-Netravali with B=0.35 and C=0.325]

Periodic Noise

- We use mostly periodic noise, since our Temporal AA is average of four consecutive frames.
- Too much noise can cause problems with neighbourhood clamp.
- Damping noise down with accumulation buffer works fine with sequential noise.

As the final frame in Quantum Break is composite of four consecutive frames, its possible to hide some amount of noise by using periodic noise patterns. Average over four frames gives stable results as long as the noise is roughly within the limits of the clamping neighbourhood. On top of periodic noise, we are using classic tail accumulation in couple of locations to smooth the noise down. With tail accumulation, its easy to control how sharp the noise is, when it gets fed into temporal anti-aliasing.



Next, I'll quickly recap what I talked about, and present few ideas on where we might be going next.

Recap

- Geometry buffer with 4xMSAA. Gives 3.7M nicely distributed geometry samples for 720p image.
- Good quality downsampling is needed in order to preserve relevant data when calculating the lighting with less samples.
- Sub-pixel offsets from downsampling can be shared for other effects.
- MSAA is still relevant for geometry aliasing. Having more and better distributed samples makes sense.

I started by introducing light pre-pass architecture, and by highlighting problems we had with our MSAA implementation with it. I then presented how we first draw geometry buffers using 4xMSAA and select relevant samples for lighting calculations. While doing the selection, we store sub-pixel offset for each MSAA sample. These offsets can be used to look up the correct data per sample from downsampled buffers. Offsets can also be used with other techniques that use the same downsampled geometry information.

Recap

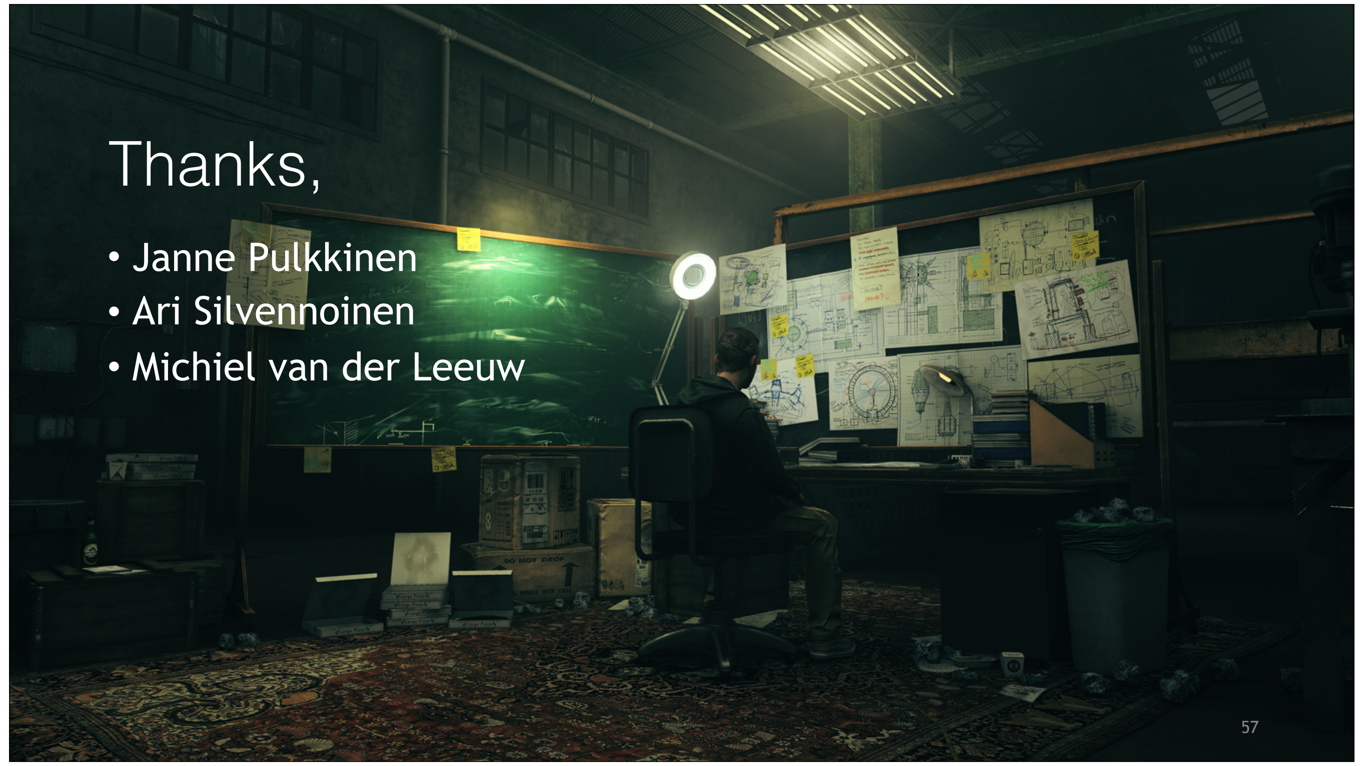
- Storing and re-projecting N previous frames gives a good framework for temporal AA and upscale.
- Feeding temporal anti-aliasing with too much variance breaks the system. Every frame needs to be stable enough.
- Temporal AA exchanges aliasing to ghosting. Tighter clamping bounds can lead to flickering.

Temporal anti-aliasing is relatively cheap and straight forward way to balance between aliasing and ghosting. By interleaving samples of multiple frames, it is possible to get towards super sampling, but with obvious limitations. For us, and depending on content, simple clamp with neighbour min and max colour isn't enough to avoid flickering. Bounds need to be extended somehow. Finally, when using any clamping based on the current frame, the picture going into temporal anti-aliasing needs to be stable enough or you will see the bounds of the clamp.

Future improvements

- Having two geometry passes is fairly expensive. Second pass mostly for diffuse colour that would make geometry buffer quite fat.
- Hash based clustering is not optimal solution but is fairly cheap. Haven’t run proper comparisons against other methods.
- Should try out Coverage AA.
- Better re-projection for temporal AA.
- Improve heuristics for temporal AA clamping range.

As always, there is ways to impove. Here I have collected few related items on what we might be looking into in future. [Talk about some items if there is time.]



Thanks for these guys on helping to put this presentation together. And especially to Janne for staring pixels with me on this subject.

Thank you for listening!

Do you have any questions?