

An Excursion in Temporal Supersampling

Marco Salvi

GDC16 – San Francisco, March 22, 2016



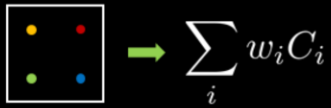
Agenda

- Introduction to temporal supersampling
- Neighborhood clipping demystified
- TAA with Variance Clipping
- Efficient post-resolve TAA for many-sample/multi-layer images
- Efficient image de-noising with large VC filters

How can we reduce aliasing?

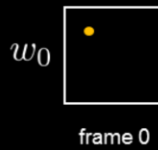
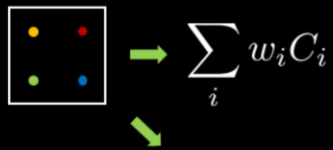
- Pre-filtering
 - No general case solution
 - Progress is being made (don't miss next talk!)
- More samples
 - Supersampling is expensive, often impractical
 - Better performance with decoupled shading techniques (e.g. MSAA, CSAA)
 - Image quality can be much worse (AA only on edge fragments)

Turn sum over space into sum over time

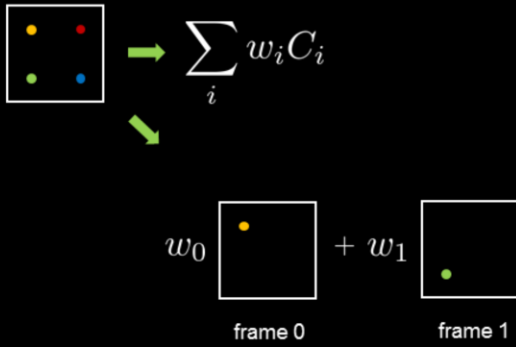


$$\sum_i w_i C_i$$

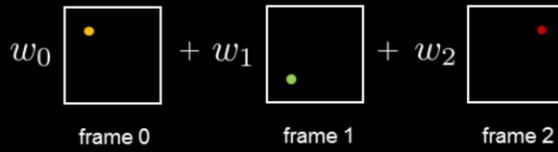
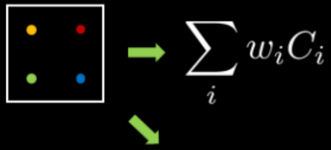
Turn sum over space into sum over time



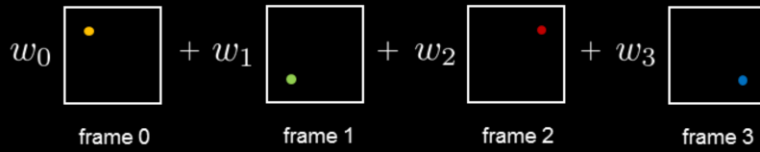
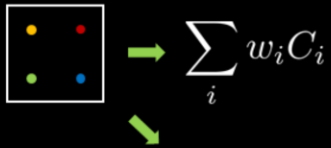
Turn sum over space into sum over time



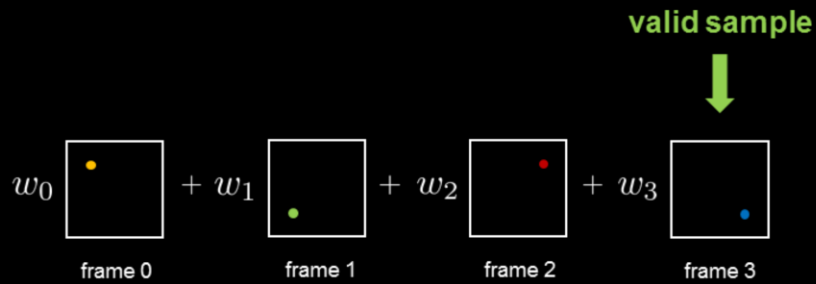
Turn sum over space into sum over time



Turn sum over space into sum over time

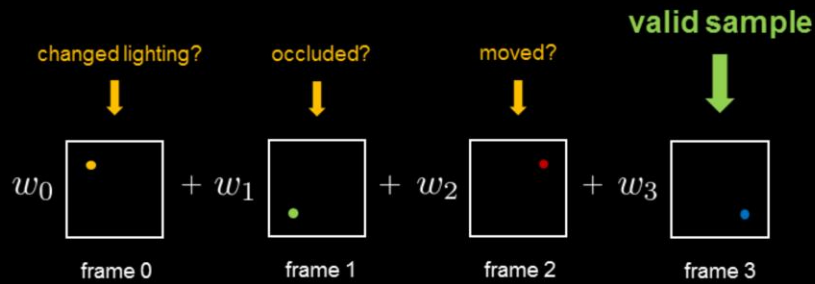


Turn sum over space into sum over time

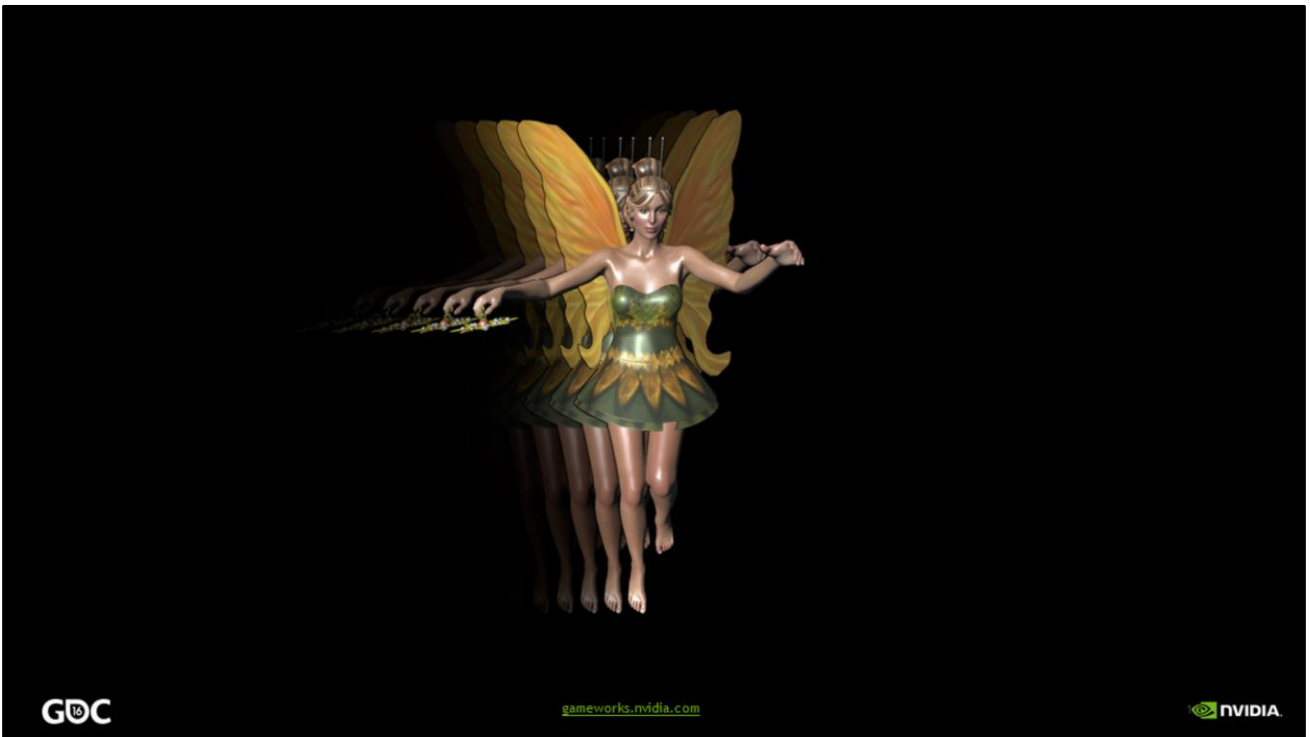


We know for certain that the last sample, shaded in the current frame, is valid.

Turn sum over space into sum over time



We cannot say whether the color of the remaining 3 samples would be the same if computed in the current frame due to motion, (dis)occlusion, change in lighting, etc..



Re-using stale/invalid samples results in quite interesting image artifacts.

In this case the fairy model is moving from the left to the right side of the screen and if we re-use every past sample we will observe a characteristic ghosting artifact.

Robust temporal supersampling is hard

- Reject old samples by testing depth, normals, material ID, etc.
- Impractical to detect all possible cases
 - Common artifacts are ghosting and trails left by moving specular highlights

There are many possible strategies to reject samples.

Most involve checking whether plurality of pre and post shading attributes from past frames is consistent with information acquired in the current frame.

In practice it is really hard to come up with a robust strategy that works in the general case. For these reasons “old” approaches have seen limited success/applicability.

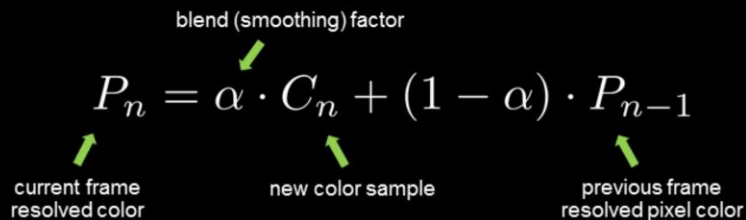
Re-use resolved color, not individual samples

- Only use previous frame
 - Lowered storage and memory bandwidth costs
 - Exponential moving average continuously integrates final pixel color [Yang et al.]

$$P_n = \alpha \cdot C_n + (1 - \alpha) \cdot P_{n-1}$$

blend (smoothing) factor

current frame resolved color
new color sample
previous frame resolved pixel color



More recent temporal supersampling methods are based on the general idea of re-using resolved color (i.e. the color associated to a “small” image region after applying a reconstruction filter), while older methods re-use individual samples. What might appear as a small difference is actually extremely important and makes possible to build much more robust temporal techniques.

First of all re-using resolved color makes possible to throw away most of the past information and to only keep around the last frame. This helps reducing the cost of temporal methods.

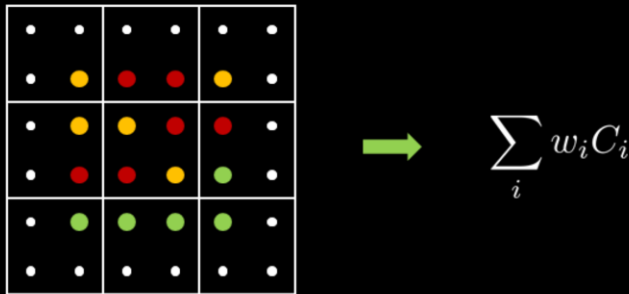
Since we don’t have access to individual samples anymore the pixel color is computed by continuous integration using a moving exponential average (EMA).

Typically we blend 10% of the current frame (1 sample) with 90% of the past frame (many samples integrated over a number of frames..).

Note that for very small values of alpha EMA behaves like an arithmetic average.

Also EMA acts as a smoothening filter and tends to damp down rapid changes in time.

Neighborhood clipping demystified [Malan12][Karis14]



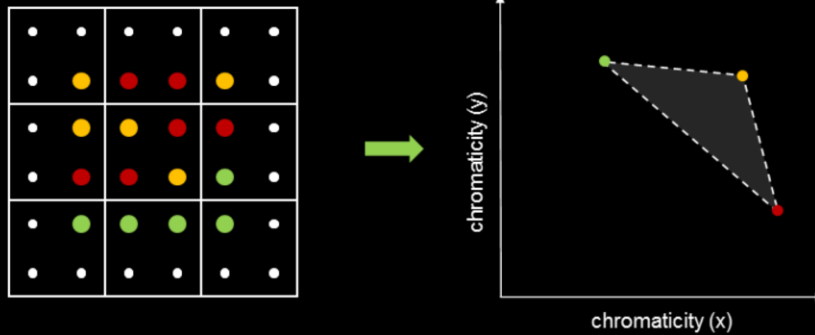
Neighborhood clipping [MALAN 2012][KARIS 2014] is the main ingredient behind the success of recent temporal supersampling methods.

The basic idea is simple: we want to identify if the resolved pixel color from the previous frame is consistent with what we know about the current frame.

If this is the case we blend the current sample color with the previous frame resolved pixel color. Conversely, we modify the resolved color from the past frame to make it consistent, then we blend it against the current color sample. We'll see the latter option is sort-of-equivalent to restarting the temporal summation from the last color sample.

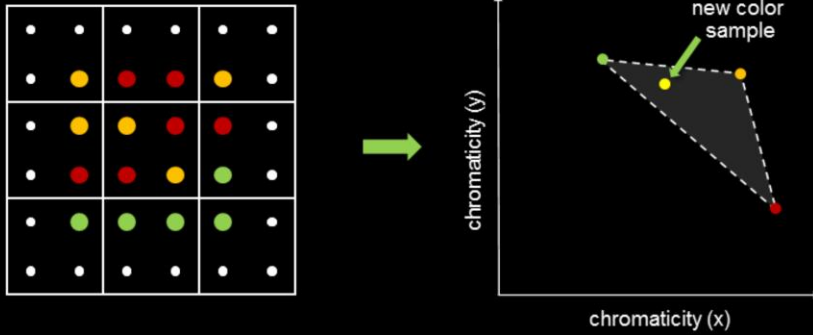
To explain how neighborhood clipping works we start with a practical example. Let's take in consideration a group of 4x4 color samples that can be resolved into a single pixel color after applying a 2x2 pixel wide reconstruction filter (e.g. tent filter). If we assume the coefficients of our reconstruction filter are positive then we can say that the resolved pixel color will lie inside the convex hull defined by the 4x4 samples (the idea also works with negative weights, but for it is easier to explain by assuming all weights are positive).

Neighborhood clipping demystified



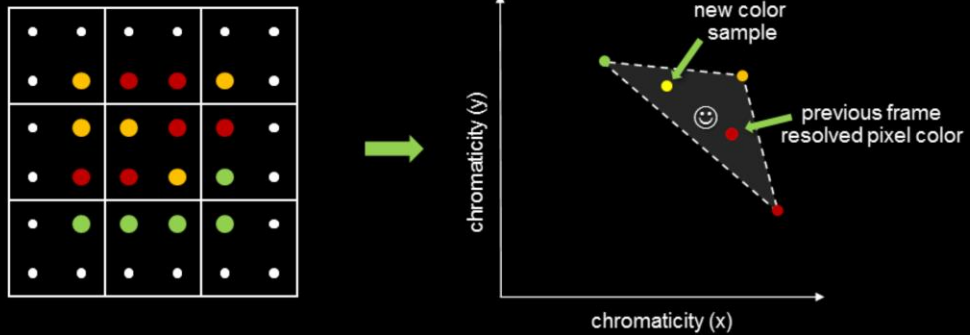
For simplicity we visualize this convex hull as a triangle over a 2D chromaticity space. In practice the convex hull is defined by 2D planes embedded in a 3D color space (e.g. RGB, YCoCg, etc.). The number of planes is variable, depending on the color sample distribution, up to the number of samples.

Neighborhood clipping demystified



By definition a new color sample computed in the current frame falls inside the convex hull.

Neighborhood clipping demystified

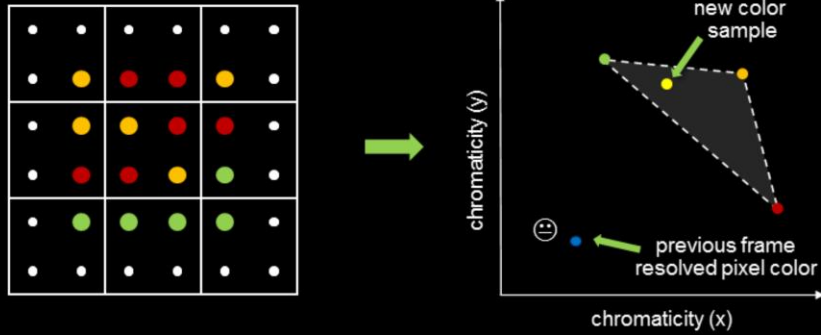


If we re-use the resolved pixel color from the previous frame we have two possibilities. If we are lucky this color falls inside the convex hull determined by current samples and everything is ok.

In this case we assume the past data is consistent with the present data. In reality this is true as long as the information we have about the current data/frame is representative of the signal we are sampling.

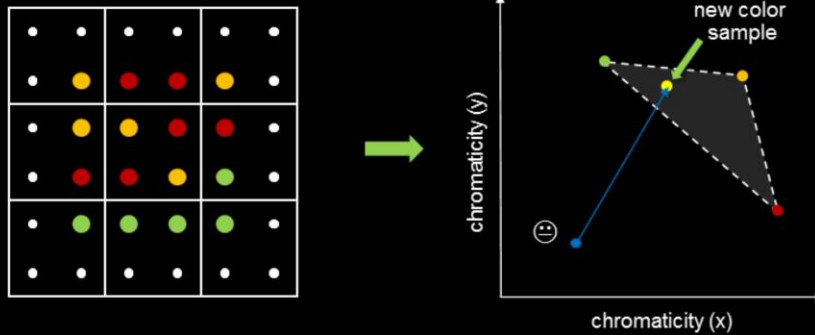
In practice we can tolerate moderate amount of aliasing and this assumption still works fairly well. We will discuss later what happens when this assumption is invalid and how to fix it.

Neighborhood clipping demystified



If the previous frame pixel color falls outside the convex hull we cannot re-use it as it's not consistent with current frame data.

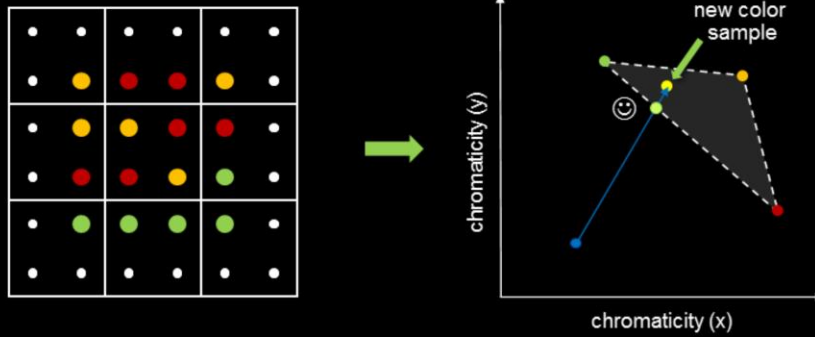
Neighborhood clipping demystified



While we could throw away the stale data from the previous frame we prefer to condition it to make it more consistent with the current frame data.

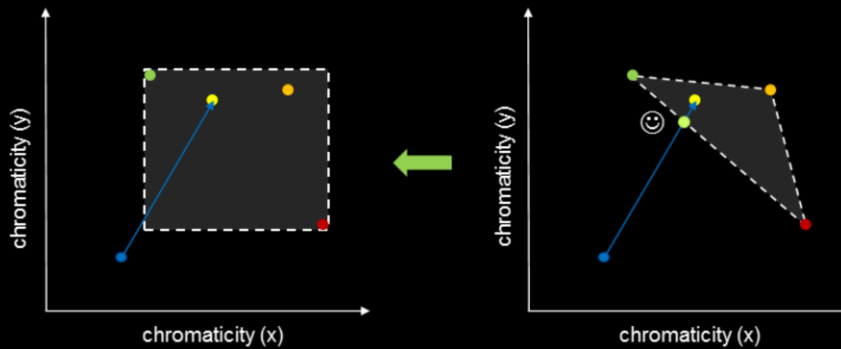
We do so by connecting with a segment the old pixel color with the new sample color. A new color value is generated by intersecting/clipping this segment against the convex hull.

Neighborhood clipping demystified



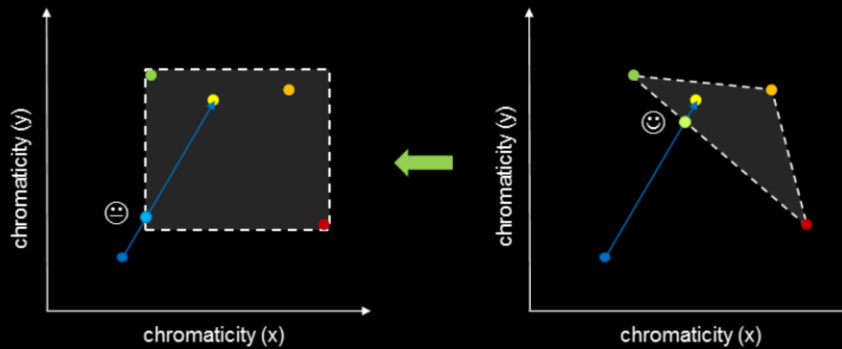
As we can see the new color value lies on the surface of the convex hull and we are now free to continuously integrate it with the current pixel color.

Neighborhood clipping demystified



Building a per-pixel convex hull and intersecting it against a segment is too expensive. A common alternative is to compute an AAB that encloses all the local current color samples and to clip the color segment against it.

Neighborhood clipping demystified



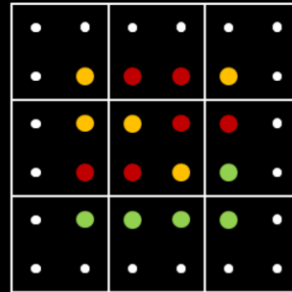
Unfortunately this can still lead to quite poor results. In this case the intersection generates a new color value that is far away from the convex hull, and therefore it is not consistent with the current frame data, causing ghosting artifacts.

Variance clipping

```
for all local samples..
m1 += color[i];
m2 += color[i] * color[i];
```



- Compute 1st and 2nd color moments



To address these cases we propose a new method called Variance Clipping (VC). We first compute the first two raw moments of the local color sample distribution from the present frame. They will be used to build an improved AABB.

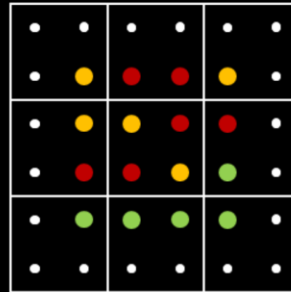
Variance clipping

- Compute 1st and 2nd color moments
- AABB from mean μ and variance σ^2
- $\mu \pm \gamma\sigma$

for all local samples..
`m1 += color[i];`
`m2 += color[i] * color[i];`

```
mu    = m1 / N;
sigma = sqrt(m2 / N - mu * mu);

minc  = mu - gamma * sigma;
maxc  = mu + gamma * sigma;
```



From the raw moments we compute the mean value μ and the standard deviation σ .

We then build an AABB centered around μ . The dimensions of the AABB are determined by σ , up to a scaling factor γ .

Larger γ s produced more temporally stable results at the cost of increased ghosting.

When γ is too small we lose the ability of integrating data over time.

Variance clipping

- Compute 1st and 2nd color moments
- AABB from mean μ and variance σ^2

- $\mu \pm \gamma\sigma$

```

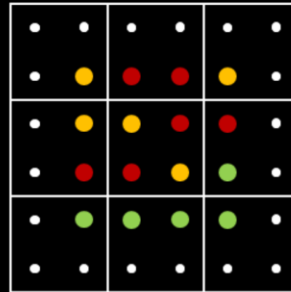
mu    = m1 / N;
sigma = sqrt(m2 / N - mu * mu);

minc  = mu - gamma * sigma;
maxc  = mu + gamma * sigma;
    
```

- Scale down σ for reduced ghosting
- $\gamma = 1$ works well
- Can clip new AABB against old AABB

```

for all local samples..
m1 += color[i];
m2 += color[i] * color[i];
    
```

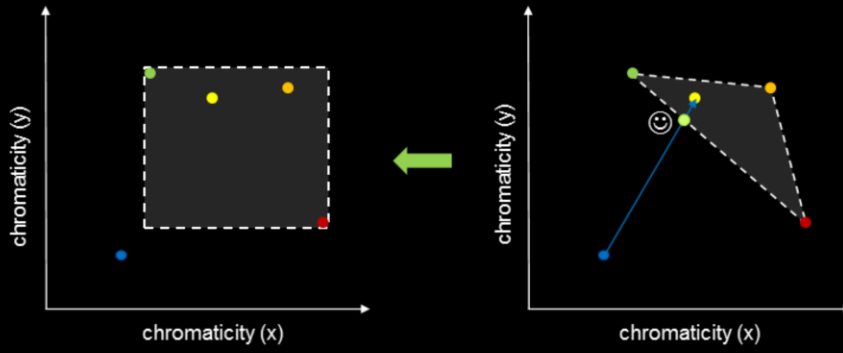


We typically use gamma = 1 for good results.

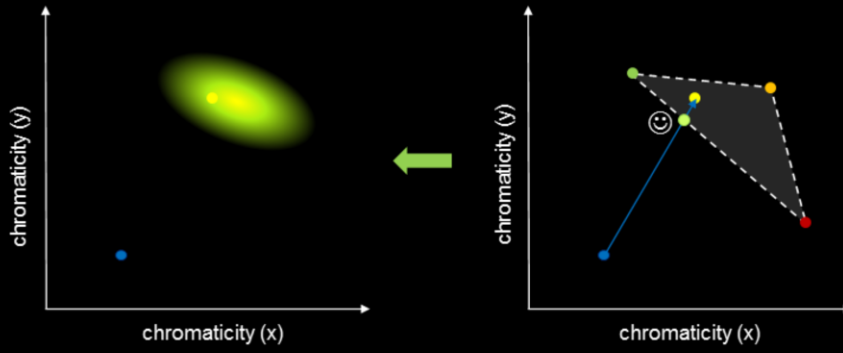
By building the AABB using a statistical method we can better eliminate outliers in the sample distribution.

To make sure our AABB is not significantly larger than the old AABB computed using min/max operations we can clamp it against the old AABB.

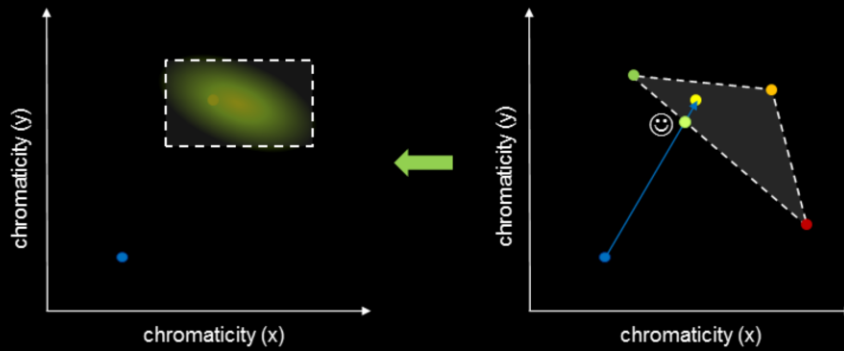
Variance clipping



Variance clipping

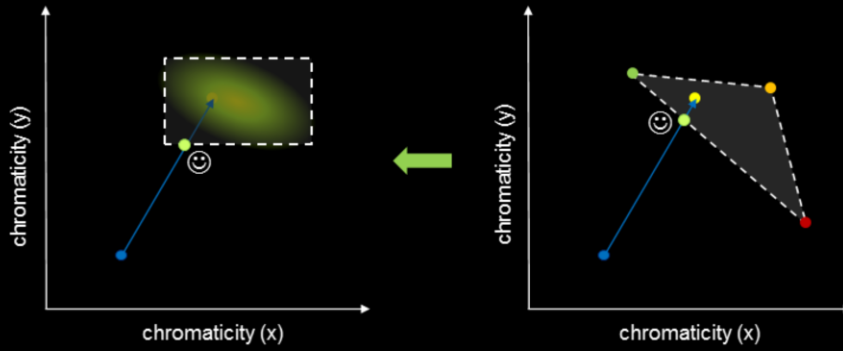


Variance clipping



The old AABB is replaced by the new computed using the first two raw moments of the color distribution

Variance clipping



The sample generated by clipping the segment against the new ABBB is now more consistent with the present frame data.

A simple recipe for anti-aliasing

1. Render 1 spp, color + motion vector
 - Integrate over pixel by jittering sub-pixel sample position over time

A simple recipe for anti-aliasing

1. Render 1spp, color + motion vector
 - Integrate over pixel by jittering sub-pixel sample position over time
2. Temporal supersampling as image post processing
 - Use motion vectors + bi-cubic filtering to sample previous frame data

A simple recipe for anti-aliasing

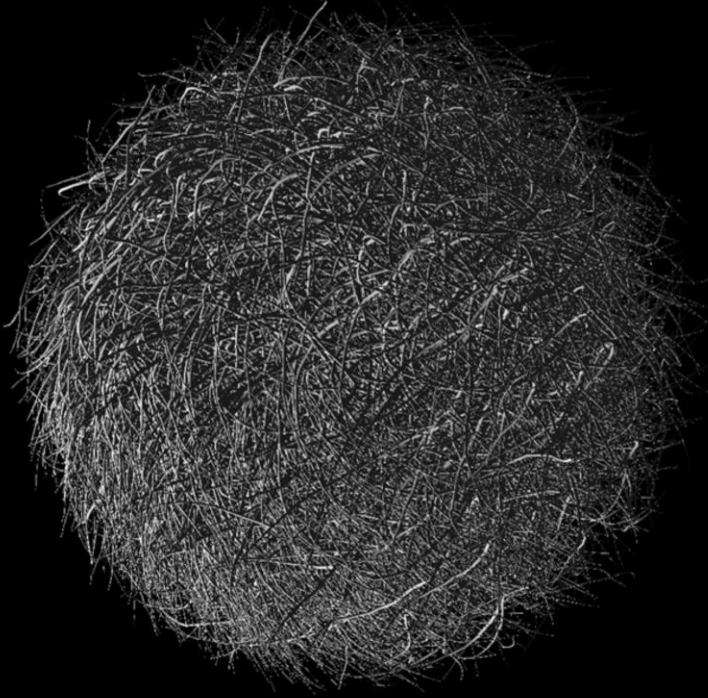
1. Render 1 spp, color + motion vector
 - Integrate over pixel by jittering sub-pixel sample position over time
2. Temporal supersampling as image post processing
 - Use motion vectors + bi-cubic filtering to sample previous frame data
 - Apply 3x3 variance clipping to remove ghosting
 - Good AABB $\rightarrow \gamma \in [0.75, 1.25]$
 - Integrate results with current sample color $\rightarrow \alpha = 0.1$

No AA

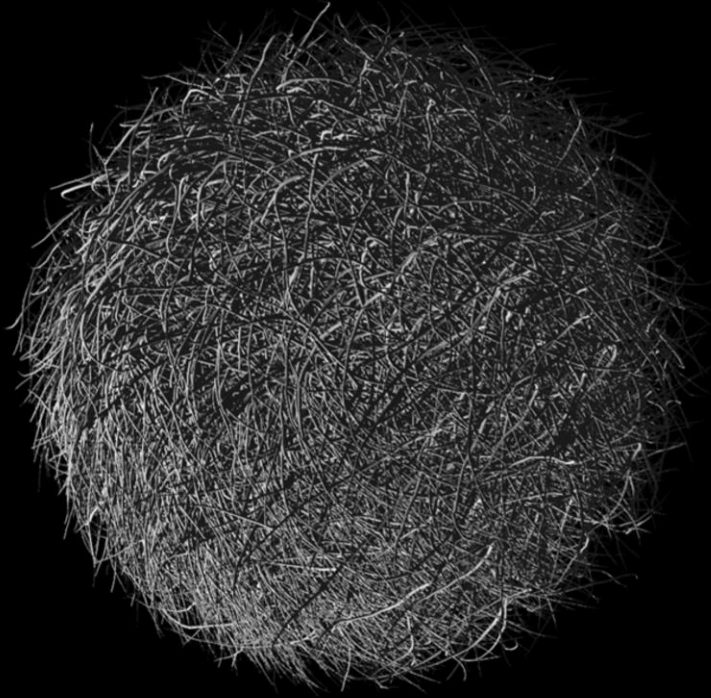




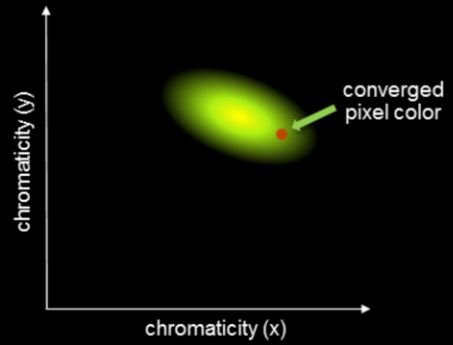
No AA



TAA

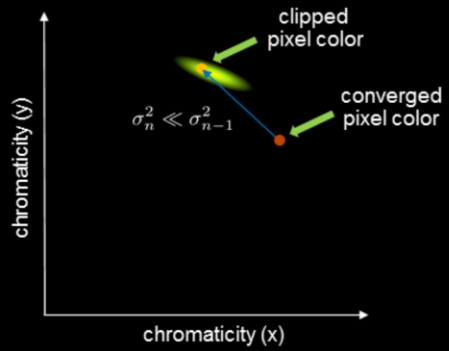


Sub-pixel details can cause color flickering



Sub-pixel details can cause color flickering

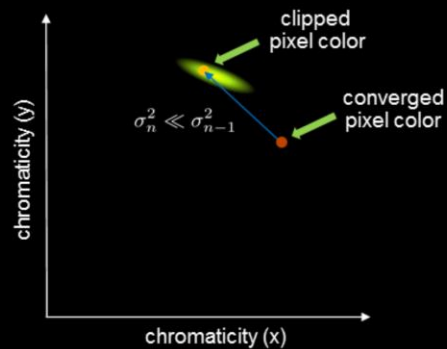
- When variance suddenly drops..
- Moving average is re-set to latest color sample
 - Exactly what we need to eliminate ghosting, but..



When

Sub-pixel details can cause color flickering

- When variance suddenly drops..
 - Moving average is re-set to latest color sample
 - Exactly what we need to eliminate ghosting, but..
- Repeated “reset events” cause flickering
 - Due to sub-pixel geometry or lighting details
 - Spatial aliasing → temporal aliasing
 - Even when nothing moves, if we use jittering ☹️



Temporal supersampling can transform spatial aliasing into temporal aliasing artifacts such as flickering.

This happens mostly when our original assumption on having a representative set of samples from the current frames breaks down due to excessive aliasing.

In other words, simply jittering the viewport might cause some extremely thin geometrical or lighting features to fall between samples, entirely erasing its color contribution from the current color distribution. When this happens variance can shrink significantly, causing every past color contribution to be clipped against the current sample color.

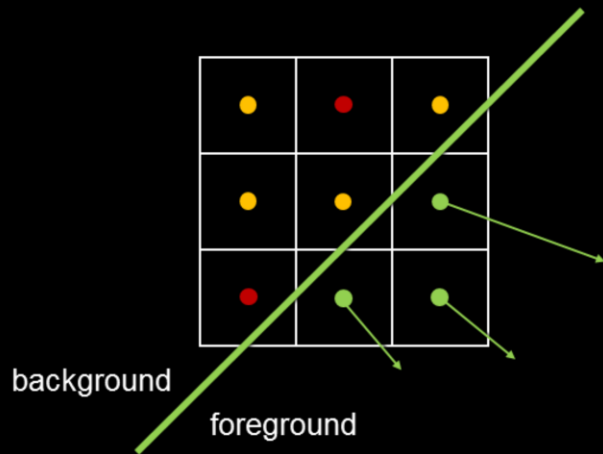
This event resets the exponentially averaged pixel color, which is great to eliminate ghosting. If these events are repeated (for instance due to jittering the viewport) they will cause flickering, even when nothing is moving on the screen.

More or better samples reduce color flickering

- More samples
 - MSAA and SSAA are effective, even when TAA is applied post-resolve
 - Hallucinating new color samples with FXAA yields temporally unstable results
- Better samples
 - Normal pre-filtering: Toksvig normals, LEAN mapping, etc.
 - Anton's new method (next talk!)
- Reduce jittering amplitude → trade off flickering for edge aliasing

Since VC-induced flickering is a byproduct of excessive spatial aliasing, we can take care of it by using other anti-aliasing methods. From taking more samples to modern pre-filtering techniques.

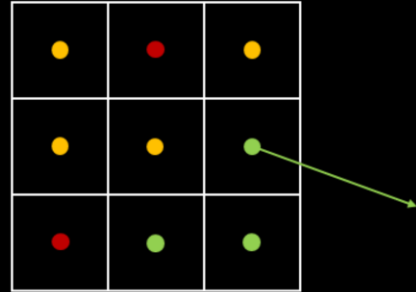
Motion vectors must track silhouettes



Integrating color over a small region that potentially includes different elements moving at different velocities requires special care. In this case, if we use the (zero) motion vector from the center of the pixel, we might completely miss moving features.

Motion vectors must track silhouettes

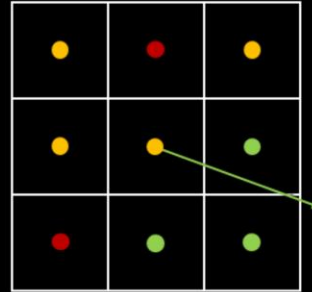
- Pick longest motion vector in 1-ring



In this case the foreground samples don't cover the center of the 3x3 region. Nonetheless we pick the longest motion vector and we apply at the center of our filter in order to track the fastest moving feature and accurately integrate over time.

Motion vectors must track silhouettes

- Pick longest motion vector in 1-ring
- Apply to center sample
- Improved silhouette under motion



TAA with many samples per pixel

- Best image quality
 - Apply TAA before MSAA/SSAA resolve
- Best performance (and still great image quality)
 - Resolve color and longest motion vector and apply TAA after resolve
 - Performance independent from number of samples per pixel

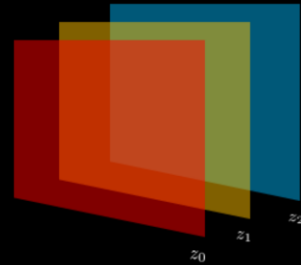
For best image quality with MSAA, CSAA or SSAA just apply TAA before the resolve pass.

It is possible to make TAA performance independent upon the number of samples per pixel by applying it post-resolve.

To use this method one has to first properly resolve the motion vector by outputting the longest one (color is resolved as usual).

Anti-aliasing for multi-layer images

- Run TAA on all layers and composite
 - Highest image quality
 - Performance depends upon number of layers

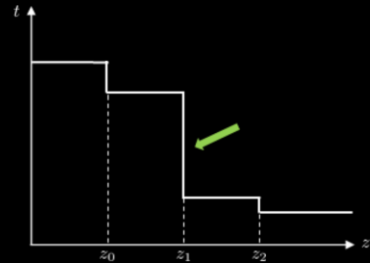
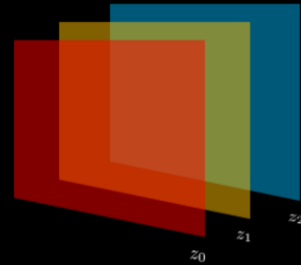


We can apply TAA on a per-layer basis for best quality with multi-layer images.

Anti-aliasing for multi-layer images

- Run TAA on all layers and composite
 - Highest image quality
 - Performance depends upon number of layers

- Run TAA after compositing layers
 - Resolve color and motion vectors
 - Pick motion vector with largest transmittance variation
 - Great image quality and performance



Similarly to the many samples per-pixel case we can also apply TAA to multi-layer images after resolving the layers into a single image. In this case we need to resolve out the motion vector that generates the largest variation in transmittance. This motion vector is associated to the layer the impacts the image the most (on a per-pixel basis).

Noise reduction for stochastic rendering

- Change random seed every frame
- TAA to reduce noise

We can also use TAA to reduce noise by stochastically integrating a function over time.

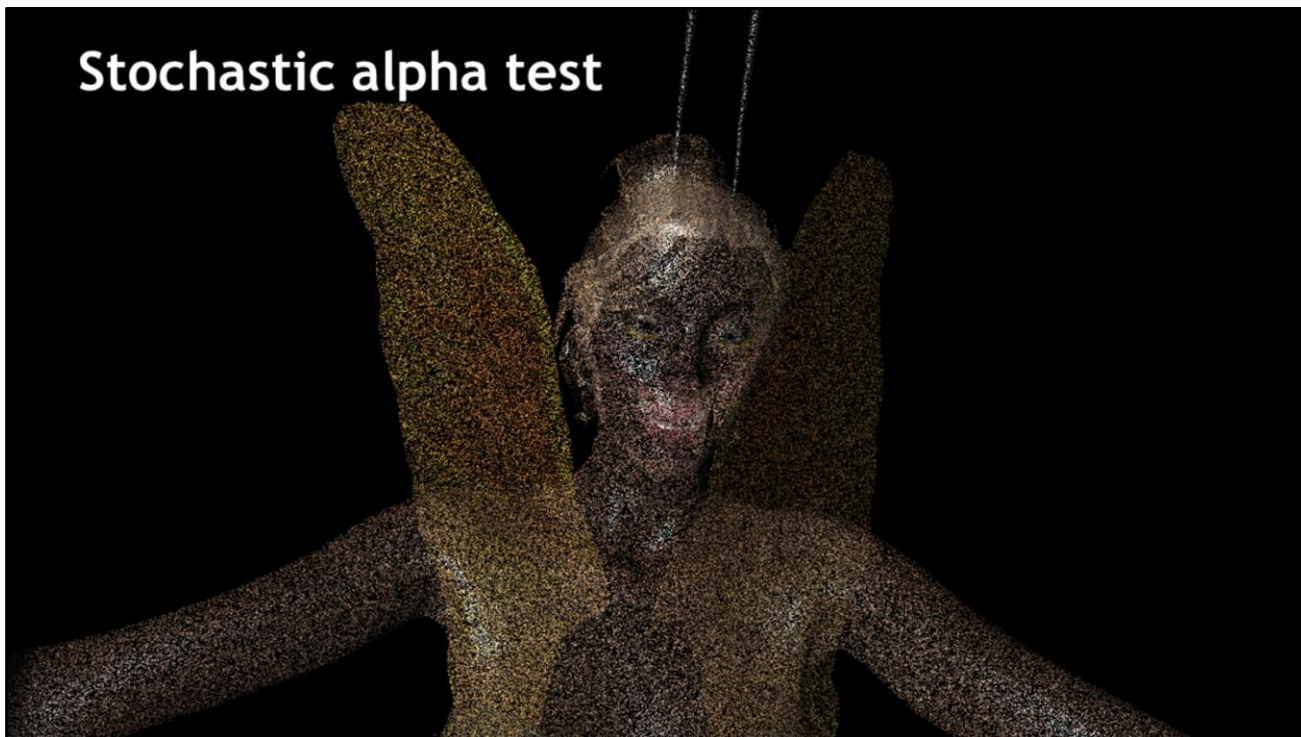
TAA

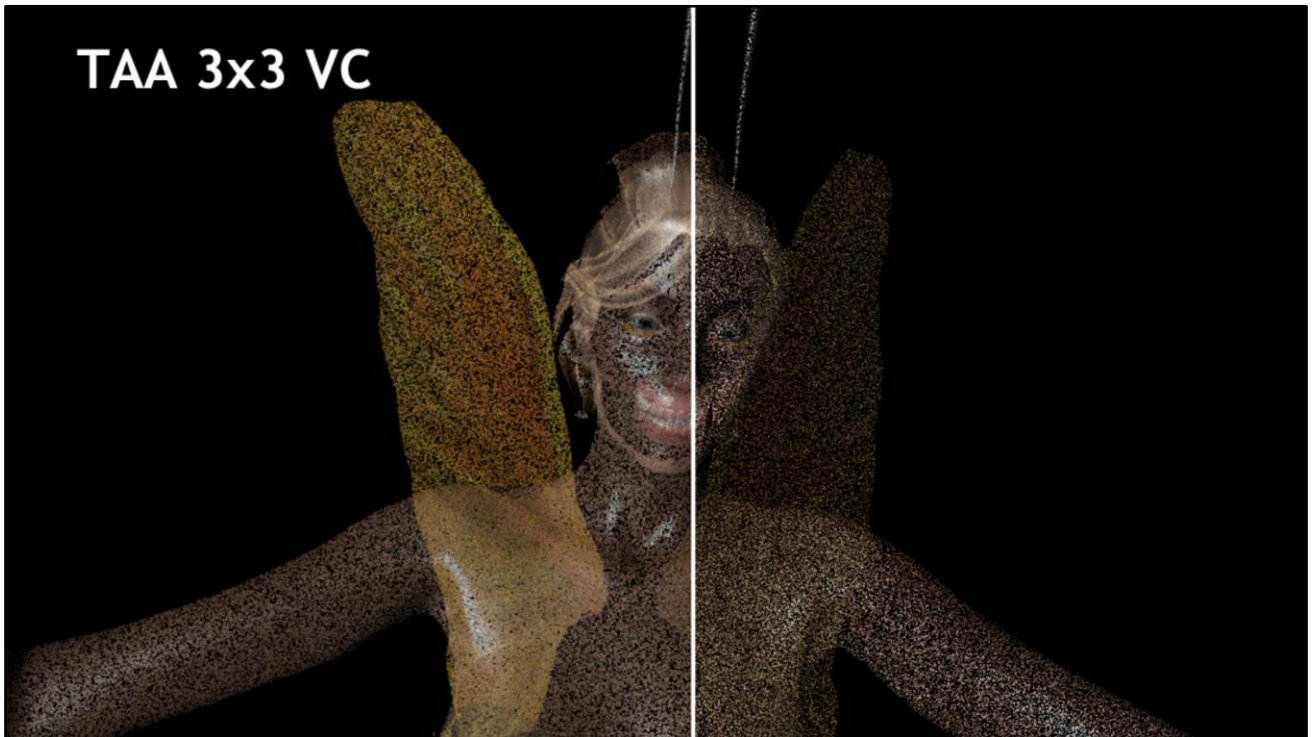


OIT (alpha = $\frac{1}{4}$)

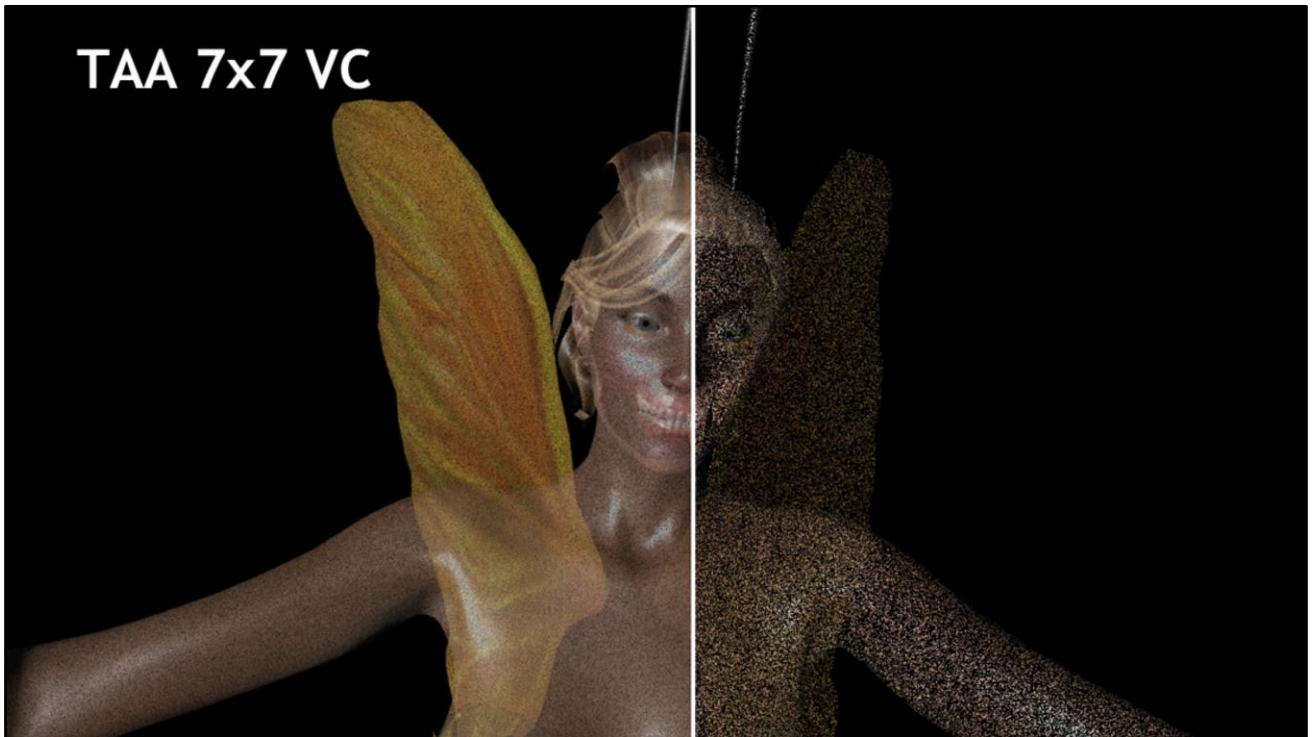


Stochastic alpha test





When we apply TAA with a 3x3 Variance Clipping window the noise is somewhat reduced, although the final result is still quite unsatisfactory. This is due to the fact that the stochastic alpha test is removing 75% of the present data (since in this example alpha is set to 25%), drastically reducing the amount of information available to Variance Clipping.



If we modify VC to work on larger window, let say 7x7 samples, then we are able to better reconstruct the original color distribution from the present frame. This is due to the fact that a larger VC window increases the likelihood of reconstructing the local sample distribution. Unfortunately it also increases the amount of ghosting we can see under motion.

Efficient variance clipping on large regions

- a) Compute 1st and 2nd color moments
 - b) Apply separable filter, repeated downsampling, etc.
 - c) Generate color AABB from filtered moments
 - Similar to variance or exponential shadow maps
- Trade off less noise reduction for more ghosting

A remarkable property of Variance Clipping is that we can apply over large windows in an efficient manner by pre-filtering the first two color moments with a blur pass or some other filter. This is very similar to what developers do to pre-filter variance or exponential shadow maps.

Summary

- Use Variance Clipping (VC) to condition previous frame color data
 - Reduced ghosting artifacts
- Applying VC post resolve still generates great looking results
 - As long as motion vectors are properly resolved
- Use VC over larger image regions for more effective noise reduction
 - Fast implementations possible by pre-filtering color moments

Q&A

E-mail: msalvi@nvidia.com

Twitter: @marcosalvi

Bibliography

1. YANG, L., NEHAB, D., SANDER, P. V., SITTHIAMORN, P., LAWRENCE J., HOPPE H. Amortized supersampling. In ACM Trans. Graph. 28 (2009), 135:1–135:12.
http://www.cs.virginia.edu/~gfx/pubs/Yang_2009_AMS/yang2009.pdf
2. MALAN, H. “Real-Time Global Illumination and Reflections in Dust 514”. Advances in Real-Time Rendering in Games course, SIGGRAPH 2012
[http://advances.realtimerendering.com/s2012/CCP/Malan-Dust_514_GI_reflections\(Siggraph2012\).pptx](http://advances.realtimerendering.com/s2012/CCP/Malan-Dust_514_GI_reflections(Siggraph2012).pptx)
3. KARIS, B. “High Quality Temporal Supersampling”, Advances in Real-Time Rendering in Games course, SIGGRAPH 2014.
<http://advances.realtimerendering.com/s2014/epic/TemporalAA.pptx>