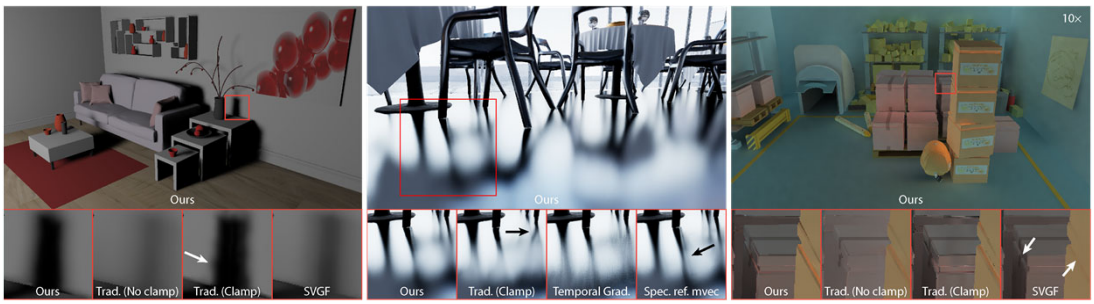


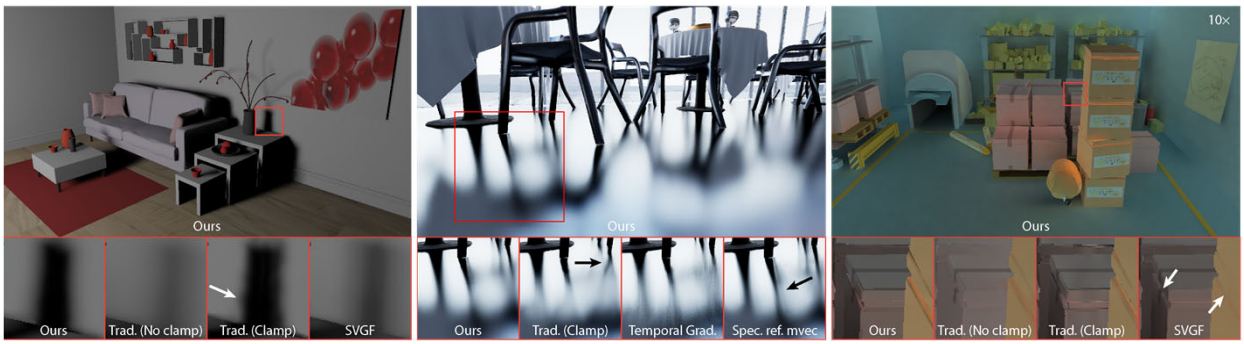
Temporally Reliable Motion Vectors for Real-time Ray Tracing

Zheng Zeng[1], Shiqiu Liu[2], Jinglei Yang[3], Lu Wang[1] and Ling-Qi Yan[3]

enjoy Vienna
EUROGRAPHICS
2021

Hello everyone!

I'm going to talk about our work called Temporally Reliable Motion Vectors for Real-time Ray Tracing,



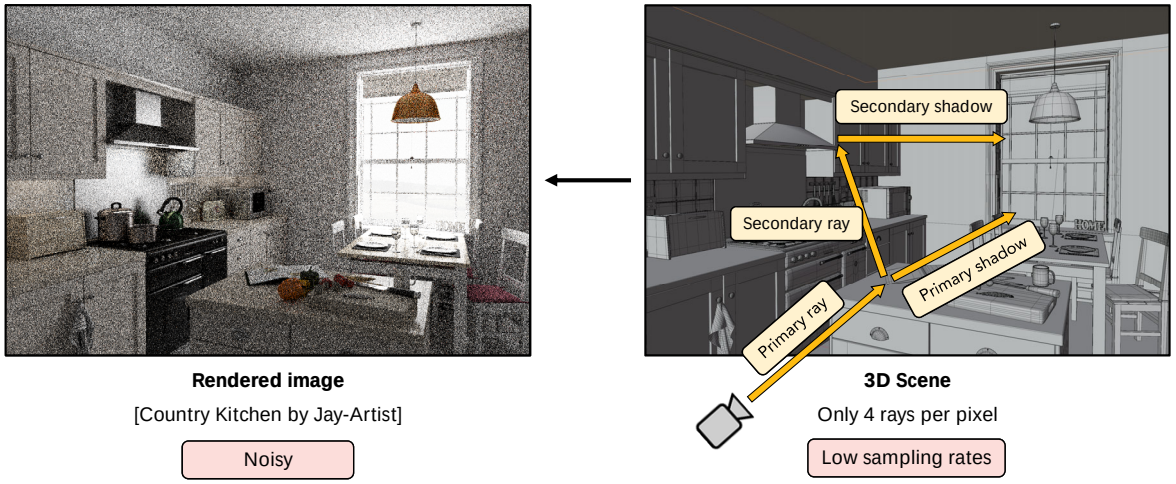
Temporally Reliable Motion Vectors for Real-time Ray Tracing

In this work, we address the ghosting and lagging artifacts in real-time ray tracing using our motion vectors.

To explain where these artifacts come from, I'll first give a little background.

Background

Real-time ray tracing



Rendered image

[Country Kitchen by Jay-Artist]

Noisy

3D Scene

Only 4 rays per pixel

Low sampling rates

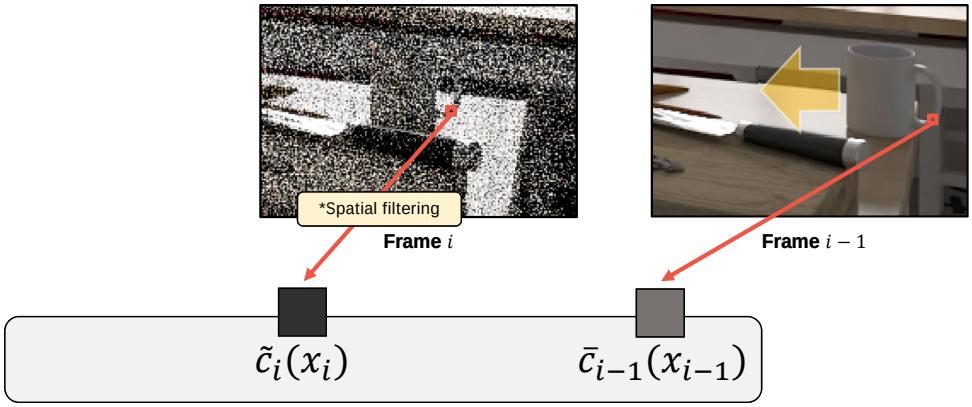
Nowadays, real-time ray tracing is increasingly being used in a variety of industries.
(click) However, despite the rapid progress, it still only allows for low sampling rates.
(click) For example, In order to guarantee real-time performance, for each pixel, only 4 rays can be traced.
(click) As a result, the rendered image will be noisy.
To improve the essential sampling rate, many research directions have been explored.

Temporal filtering

Frame i

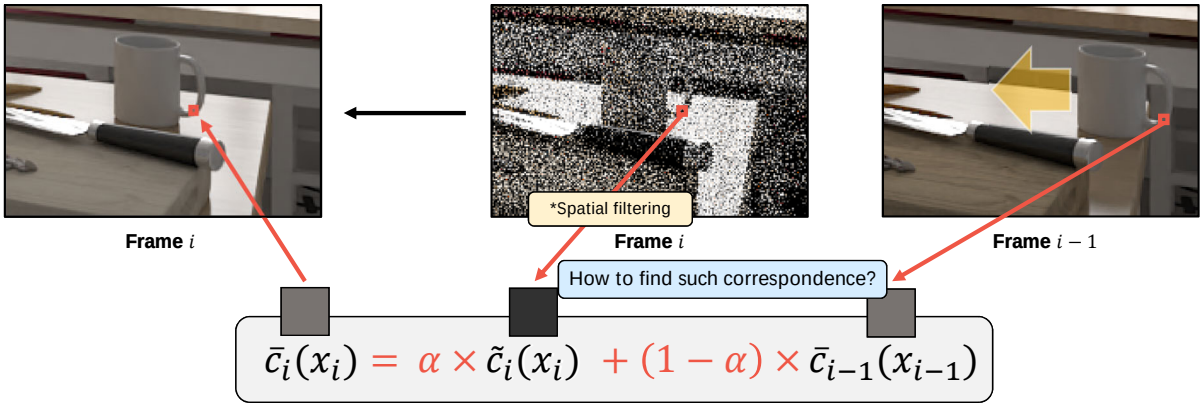
Among these directions, the temporal filtering method stands out.
For better explanation of how it works, let's zoom in a little bit.

Temporal filtering



For a pixel [x i] in current noisy frame,
(c1) It finds the corresponding pixel [x i-1] in previous frame

Temporal filtering

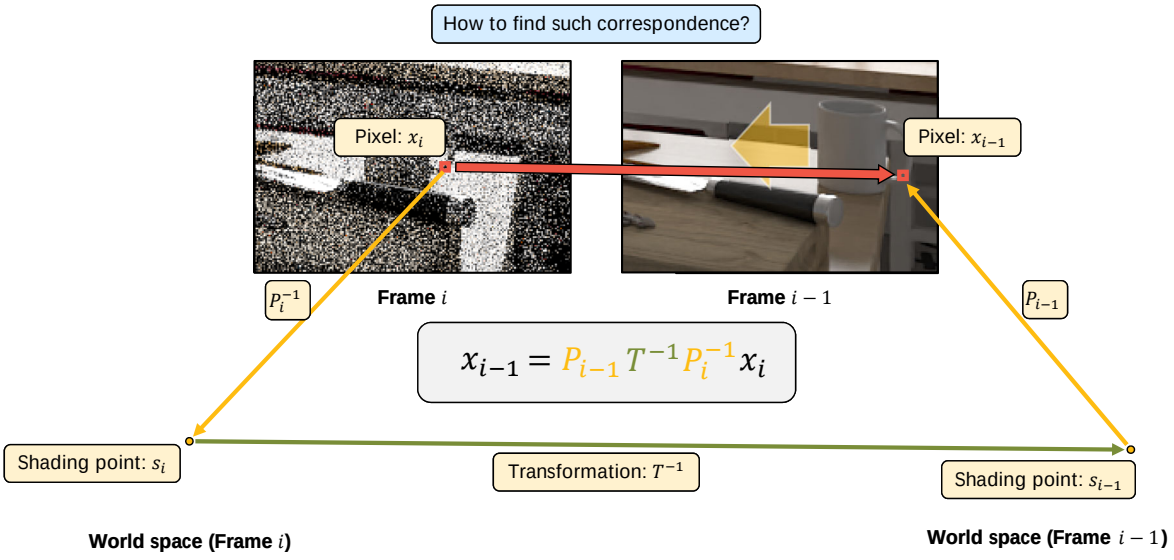


Then, it linearly blends these two pixels, using a blending factor [alpha]. The [alpha] is a factor between 0 and 1 that determines how much temporal information is trusted and used. We usually set it to 0.1 to 0.2 in practice.

(c1) In this way, the shading results are accumulated and smoothed.

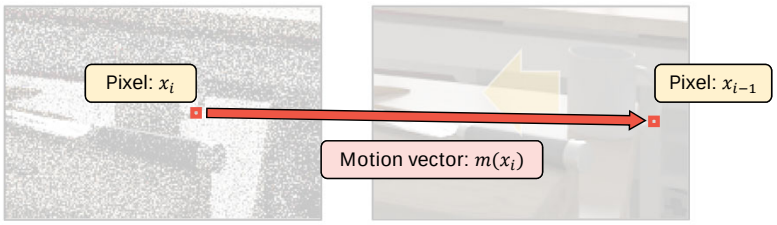
(c2) Now, the question is, how to find such correspondence?

Back-projection



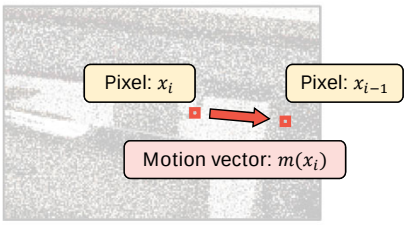
We will use the technique called the back-projection.
When two consecutive frames are given,
(c1) For each pixel [x i] in current frame,
(c2) we first project it back to its world space to get the shading point [s i].
(c3) Then, according to the movement of the geometry, we transform this shading point back to the previous frame.
(c4) Finally, we project the transformed shading point back to the image space to get the corresponding pixel [x i-1].
(c5) Now, we have found the temporal correspondence.

Motion vectors



And the red arrow here, is called the motion vector

Motion vectors

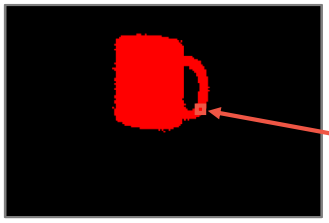


Frame i

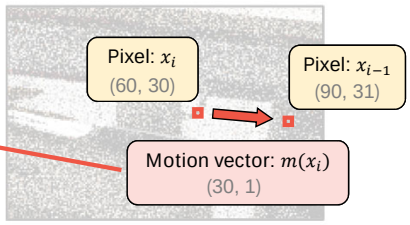
$$m(x_i) = x_{i-1} - x_i$$

It literally is a 2D vector in the image space that points from the current pixel location to its previous location.

Motion vectors



2D Texture: motion vectors



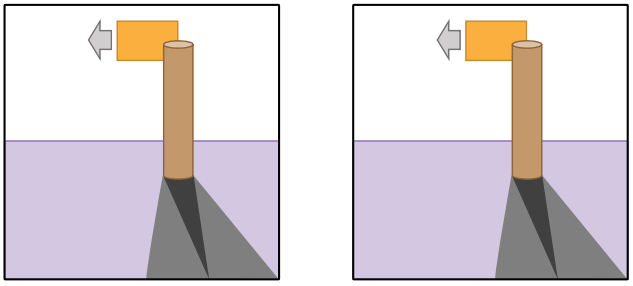
Frame i

Could we always find a valid one?

$$m(x_i) = x_{i-1} - x_i$$

The motion vectors are stored on a 2d texture like this.
Note the black pixels in this texture means their motion vectors are all zero-length. Since the geometries within these pixels are all static.
(c1) However, could it always find the right temporal correspondence for us?
The answer is no.

Typical failure 1: shadows



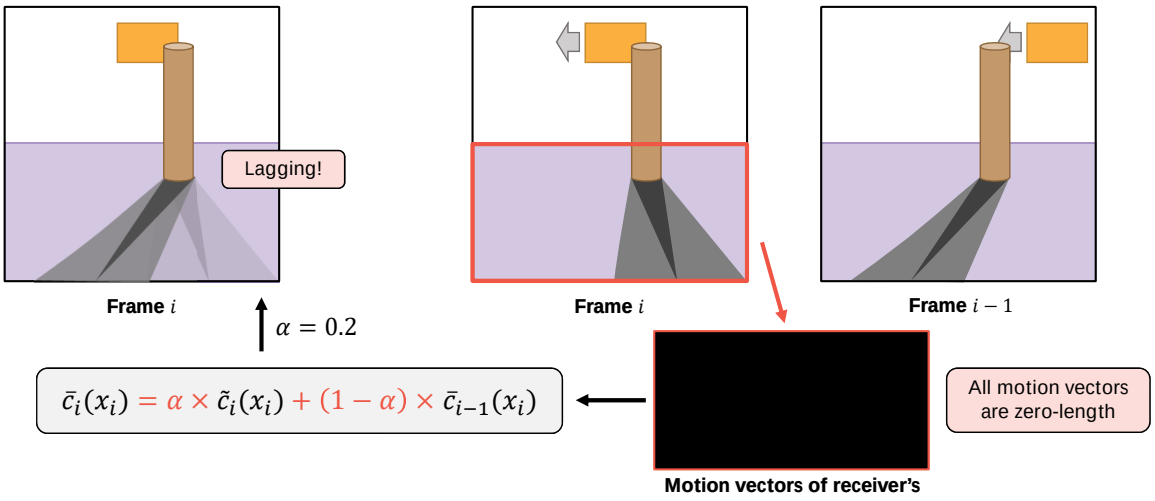
The motion vectors may be wrong under certain cases.

The first typical failure is the shadows.

Here we have a static plane,
with a static cylinder standing on it.

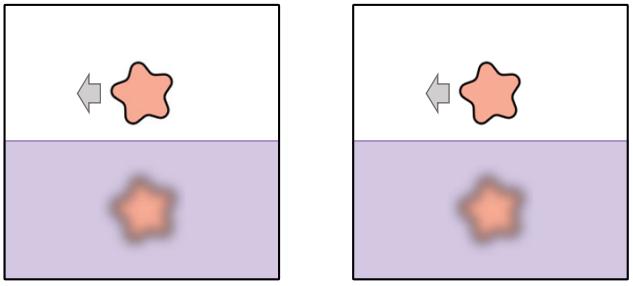
(c1) Now, there is an area light source moving to left,
(c2) while casting shadows onto the plane.
(c3) Lets rewind the time to the last frame.

Typical failure 1: shadows



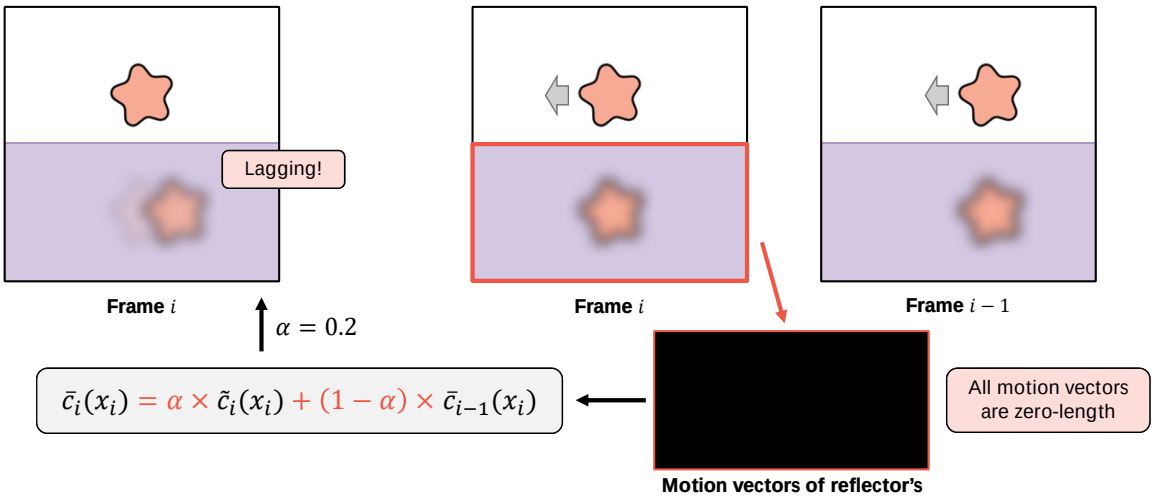
If we calculate the motion vectors of the static plane's,
(c1) we will find that the motion vectors are all zero-length, since the geometry is static.
(c2) In this case, if the temporal filtering is applied anyway, it will simply blend the pixels on the two planes.
(c3) As a result, lagging artifacts will emerge.

Typical failure 2: glossy reflections



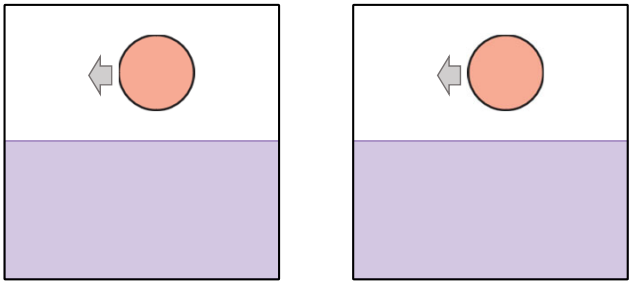
Another typical failure is glossy reflections.
Here we have an object moving to left.
(c1) And below the object, there is a glossy surface.
(c2) Again, lets rewind the time to the last frame.

Typical failure 2: glossy reflections



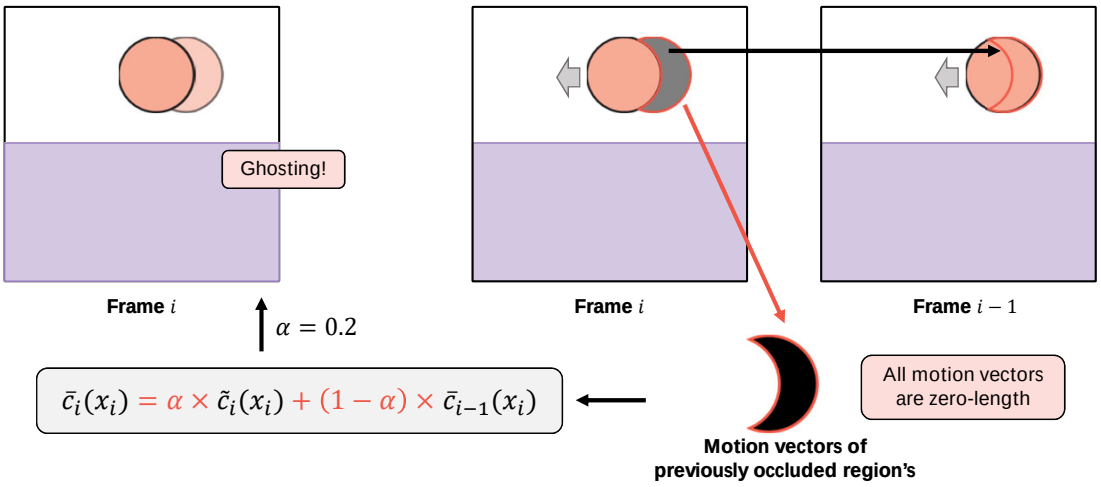
Similar to the shadow case, all of the motion vectors of this glossy plane's are also zero-length.
(c1) And again, lagging artifacts will emerge.

Typical failure 3: occlusions



The next failure case is occlusions.
We have an object moving to left, and with a white wall behind it.
(c1) Now we rewind the time to the last frame.

Typical failure 3: occlusions



For the previously occluded regions, the motion vectors are again all zero-length.
(c1) So it will simply blend the this regions with the occluder in previous frame.
(c2) As a result, ghosting artifacts will emerge.

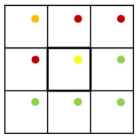
Previous Work

Various methods are designed to alleviate the temporal failures.

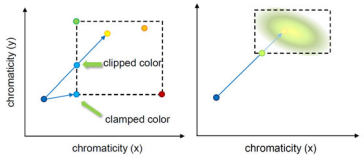
Previous work

Neighborhood Clamping (from TAA)

(History rectification)



[Karis 2014] [Salvi 2016]



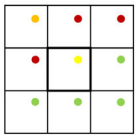
$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

The neighborhood clamping methods first gather the color information from neighborhood pixels.
(c1) Then, they compute an axis-aligned bounding box or a Gaussian distribution of the colors.
And finally, clamp or clip the history color according to it.
(c2) This kind of method aims at rectifying the history color, in order to suppress the ghosting artifacts.

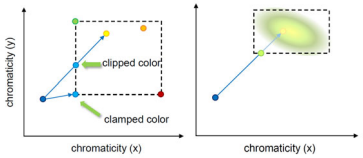
Previous work

Neighborhood Clamping (from TAA)

(History rectification)



[Karis 2014] [Salvi 2016]



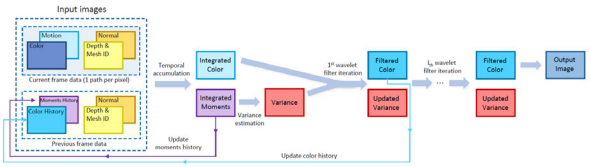
$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

SVGF

(Better spatial filtering scheme)

[Schied et al. 2017]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

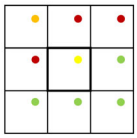


The spatiotemporal variance-guided filtering (SVGF) method analyzes both spatial and temporal variances to guide the shape and size of spatial filters. This method focuses on a better spatial filtering scheme to acquire better current result.

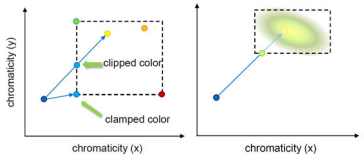
Previous work

Neighborhood Clamping (from TAA)

(History rectification)



[Karis 2014] [Salvi 2016]



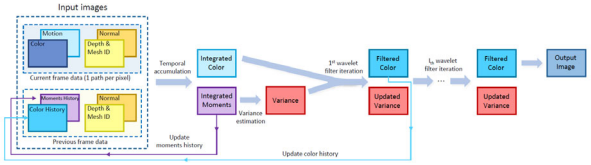
$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

SVGF

(Better spatial filtering scheme)

[Schied et al. 2017]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

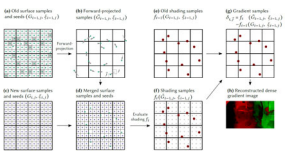


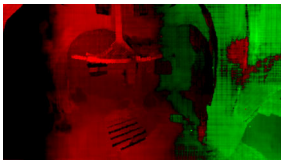
A-SVGF

(Adjust the blending factor α)

[Schied et al. 2018]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$





And the ASVGF method detects rapid temporal changes to adjust the blending factor α, to rely more or less on spatial filtering, trading ghosting artifacts for noise.

Our goal

Previous work

- Detect the temporal failures
- Either rectify or reject the wrong temporal information

SVGF
(Better spatial filtering scheme)
[Schied et al. 2017]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

Neighborhood Clamping (from TAA)
(History rectification)
[Karis 2014] [Salvi 2016]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

A-SVGF
(Adjust the blending factor α)
[Schied et al. 2018]

$$\bar{c}_i(x_i) = \alpha \times \tilde{c}_i(x_i) + (1-\alpha) \times \bar{c}_{i-1}(x_i)$$

Our goal

- Find more reliable temporal information
- For shadows, glossy reflections and occlusions

All of the previous methods are aiming at detecting the temporal failures, and either rectifying or rejecting the wrong temporal information.

(c1) In contrast to previous methods, our goal is to find more reliable temporal information.

Specifically, we will focus on the three commonly encountered temporal failure cases, shadows, glossy reflections and occlusions.

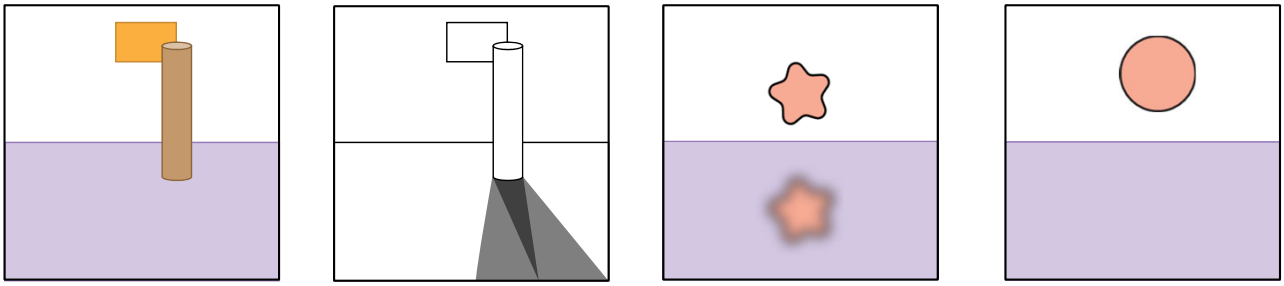
Our Method

Now let's take a look at our method.

Our method

For shadows, glossy reflections and occlusions:

Motion vector for shadows | Motion vector for glossy ref. | Motion vector for occlusions



Shading
Direct illumination without shadows

Visibility
Shadows

Glossy Reflections
Indirect illumination for specular / glossy materials

Indirect illumination
Indirect illumination for other materials

Can be approximated by Linearly Transformed Cosines (LTC) [Heitz et al. 2016]

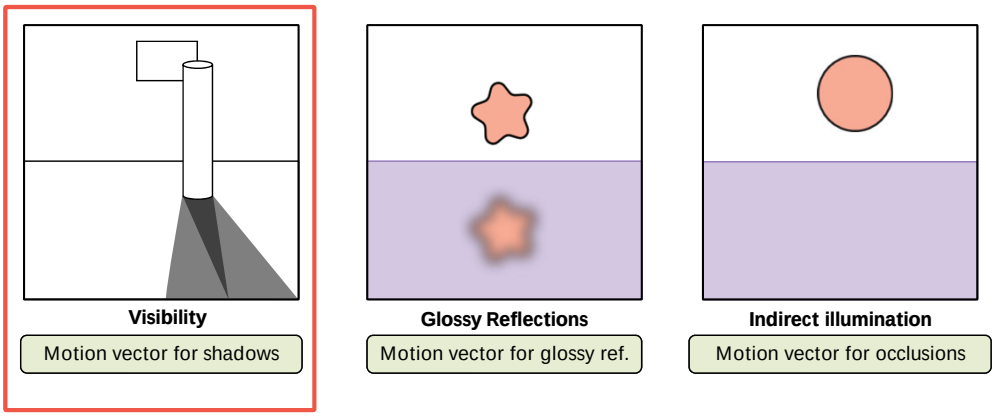
The high level idea is that we design a separate motion vector for each of these effects.

(c1) In accordance with the different types of motion vectors, we also treat our final output as a collection of separate components.

(c2) Using the LTC method, the shading part can already be reasonably approximated in a noise-free way,

Our method

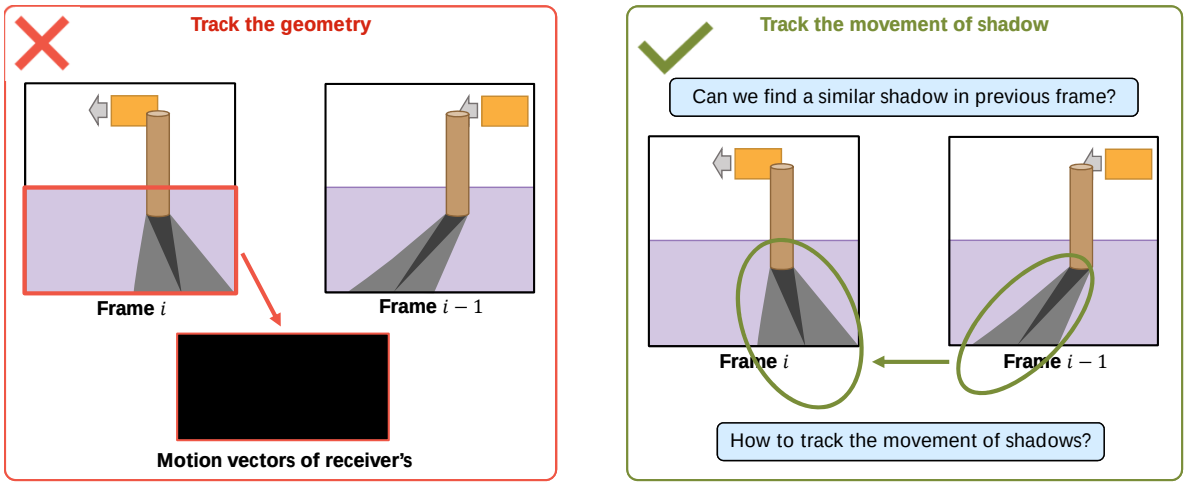
For shadows, glossy reflections and occlusions:



So now we only need to apply each of our motion vectors to the remaining three components respectively.

(c1) Let's first take a look at our motion vector for shadows.

Shadows: where is the temporal correspondence?



Remember that all we want to do is to find the right temporal correspondence for a pixel in current frame.

(c1) We already know that, for shadows, tracking the movement of geometry is not working.

(c2) Our key observation is, can we find a similar shadow in previous frame?

(c3) The answer is yes. These pixels in the previous frame should be the right temporal correspondences.

(c4) So our insight is, for shadows, it is not the geometry we want to track, but the movement of shadows.

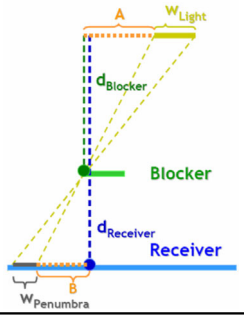
(c5) Now, the questions is, how to track the movement of shadows?

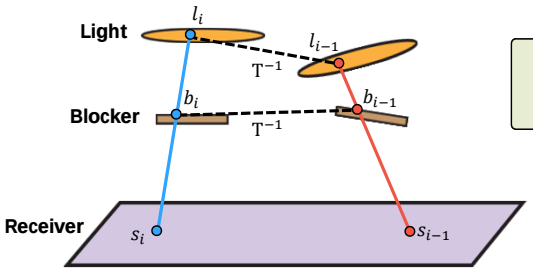
Shadow motion vector: calculation

How to track the movement of shadows?

Percentage Closer Soft Shadows (PCSS)

[Fernando 2016]





Our shadow motion vector is:

$$m^V(x_i) = x^V_{i-1} - x_i$$

Assumption: locally flat surface

What if the surface isn't flat?

$$x^V_{i-1} \leftarrow P_{i-1} = intersect((T^{-1}l_i \rightarrow T^{-1}b_i); T^{-1}Plane(s_i))$$

Inspired by the PCSS method, we propose to track the movement of the shadow by following the blocker and light positions over time.

(c1) Given two consecutive frames,

(c2) for a pixel in shadow, we know exactly the world space position of its shading point si,

the blocker position bi, and the light sample position li

(c3) Since the blocker and the light sample positions are associated with certain objects, we immediately know their positions in the previous frame.

(c4) Now, suppose that the geometry around si is a locally flat surface,

We can easily find the intersection s_i-1 between this plane and the line connecting l_i-1 and b_i-1。

(c5) Finally, we project this intersection to the image space. And this projected pixel is our tracked shadow position.

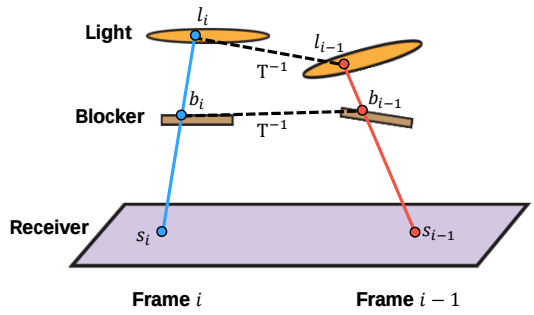
(c6) Our motion vector can be formally written as this:

Note that the only assumption we made is that the geometry is locally flat surface.

(c7) But what if the surface isn't flat?

Non-planar case

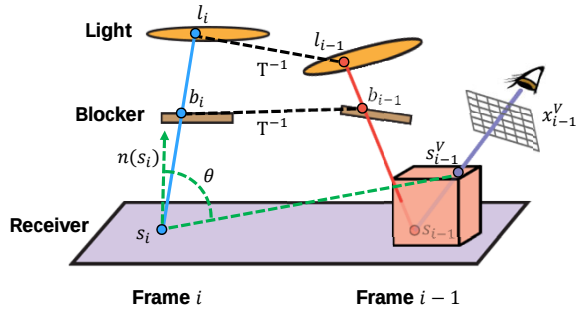
What if the surface isn't flat?



For example,

Non-planar case

What if the surface isn't flat?



- Only when θ is close to 90°,
  - fully depend on temporal result.
- Otherwise,
  - trust more on the current result.

↓

Use θ to adjust the α!

$$\alpha^V = 1 - Gaussian\left(\theta - \frac{\pi}{2}; 0, 0.1\right) \times (1 - \alpha)$$

$$\bar{c}_i(x_i) = \alpha^V \times \tilde{c}_i(x_i) + (1 - \alpha^V) \times \bar{c}_{i-1}(x_{i-1})$$

there is a box above this intersection.

(c1) Then the real shading point in previous frame is on the top of the box.

To deal with this situation, we introduce a simple but effective falloff heuristic. That is, we measure the extent of "non-planarity"

(c2) we connect these two shading points, and measure the angle [theta] between the normal of the plane and this line.

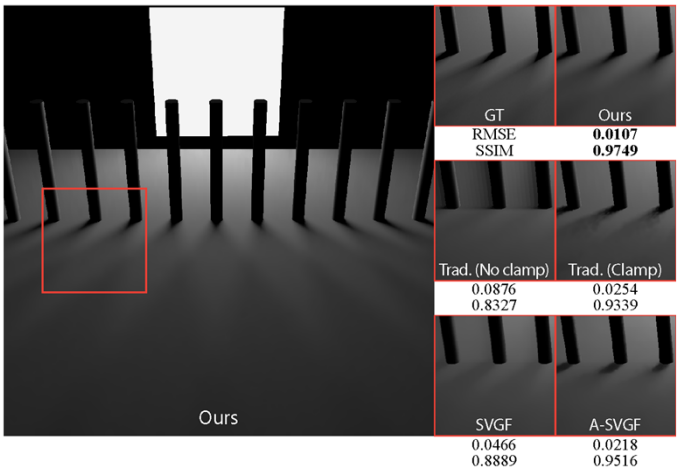
(c3) Our key observation is that only when θ is close to 90°, We can fully depend on temporal result.

(c4) Otherwise, we should trust more on the current result.

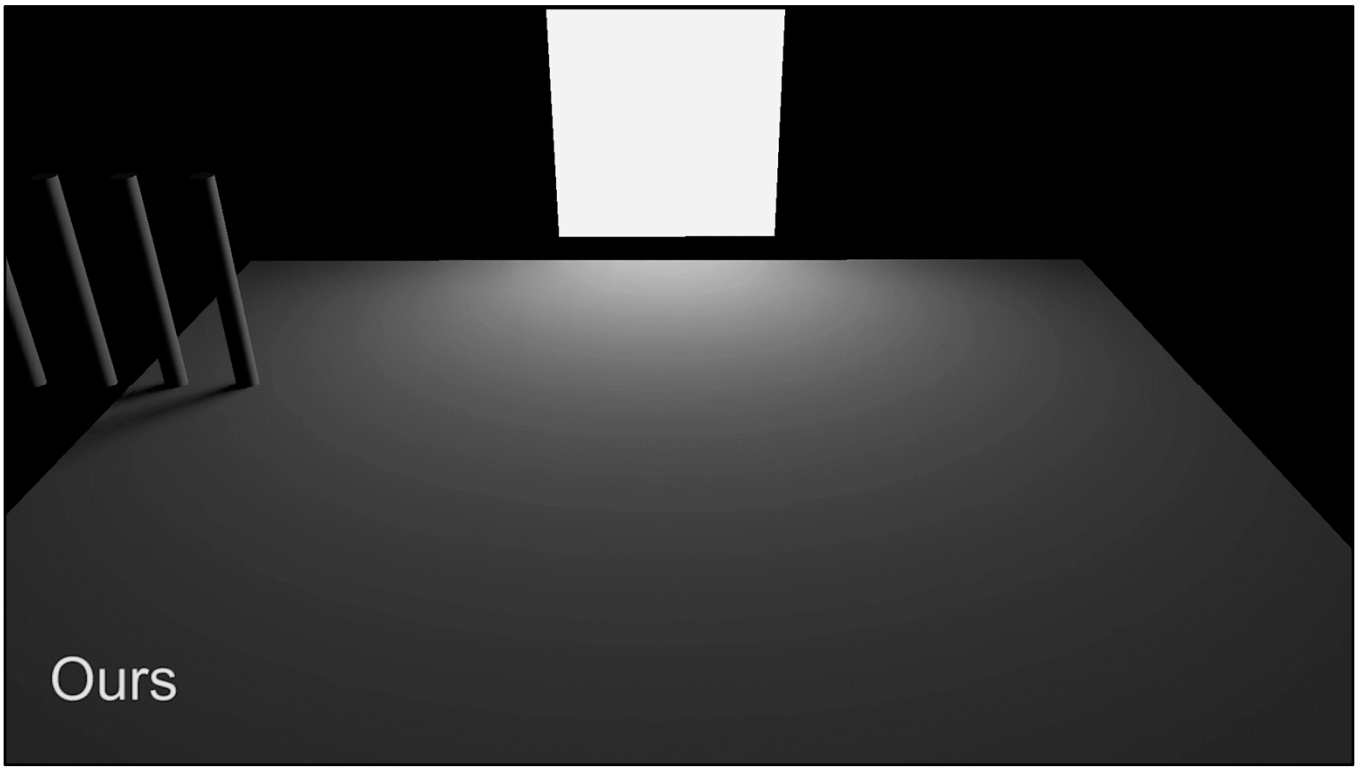
(c5) So, we use [theta] to adjust the [alpha]

Finally, we achieve high quality, non-lagging shadows.

Shadow motion vector: result



We compare our results with the ones generated using traditional motion vectors, with and without the neighborhood clamping approach used in temporal filtering. We also compare our method with the SVGF and A-SVGF.



Note that we are able to produce shadows that are closely attached to the fence.

GT: We produce the closest result to the ground truth.

Traditional motion vectors produce significant ghosting artifacts.

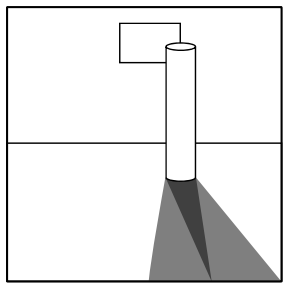
With clamping, the results are less lagging but much more noisy.

SVGF method produces over-blurred and lagging shadows.

A-SVGF method discards temporal information, resulting in noisy shadows.

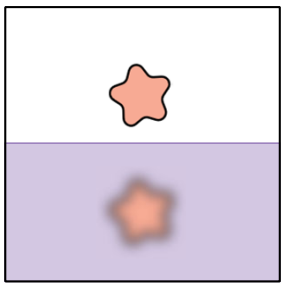
Our method

For shadows, glossy reflections and occlusions:



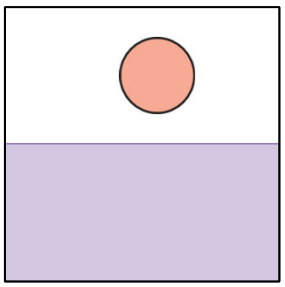
Visibility

Motion vector for shadows



Glossy Reflections

Motion vector for glossy ref.

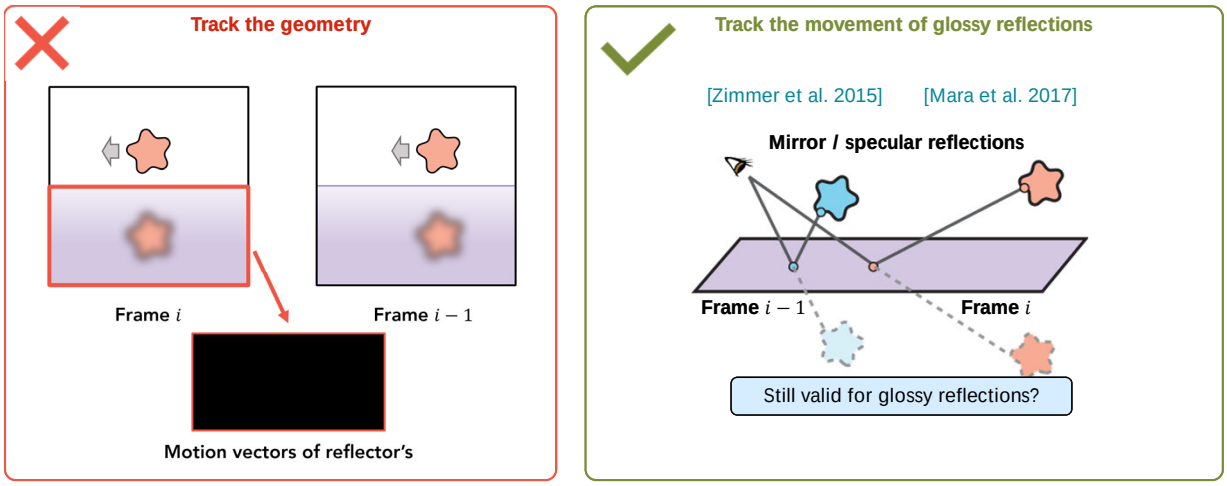


Indirect illumination

Motion vector for occlusions

Next, let's take a look at our motion vector for glossy reflections.

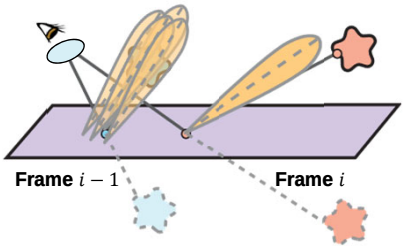
Glossy reflections: where is the temporal correspondence?



Similar to tracking the shadows, we can also track the movement of glossy reflections, instead of geometry.
(c1) This is inspired by Zimmer et al. and Mara et al. on mirror / specular reflections. They assume that the virtual image is a real object behind the mirror reflector, and calculate the motion vectors of the virtual image instead of the reflector.
(c2) However, the question is, is this still valid for glossy reflections?

Glossy reflections: where is the temporal correspondence?

Still valid for glossy reflections?



Our insight

Goal:

- Find the corresponding pixel of x_i

Observation:

- Glossy BRDFs model is a non-delta distribution
- Multiple shading points may reflect to the same hit point
- Multiple pixels from the previous frame that correspond to x_i
- Form a finite area in the screen space.

→ Design a stochastic motion vector, aiming at finding one pixel from this finite area at a time.

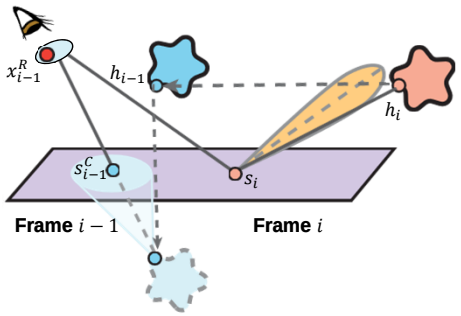
The answer is no.
(c1) Since we trace 1 secondary ray per pixel, we can only importance sample the glossy lobe to decide where to bounce.
And the bounced direction can certainly be different to the specular reflected direction.
(c2) Our insight is that no matter if we are using the specular or sampled direction,
(c3) what we need to do is still to find the corresponding pixel in the previous frame
(c4) However, since glossy BRDFs model a non-delta distribution,

(c5) Multiple shading points may reflect to the same hit point

(c6) And there are Multiple pixels from the previous frame that correspond to xi

(c7) forming a finite area in the image space.
(c8) So, our goal is to design a stochastic motion vector, aiming at finding one pixel from this finite area at a time.

Stochastic glossy ref. motion vector: calculation



Design a stochastic motion vector, aiming at finding one pixel from this finite area at a time.

Our stochastic glossy ref. motion vector is:

$$m^R(x_i) = x_{i-1}^R - x_i$$

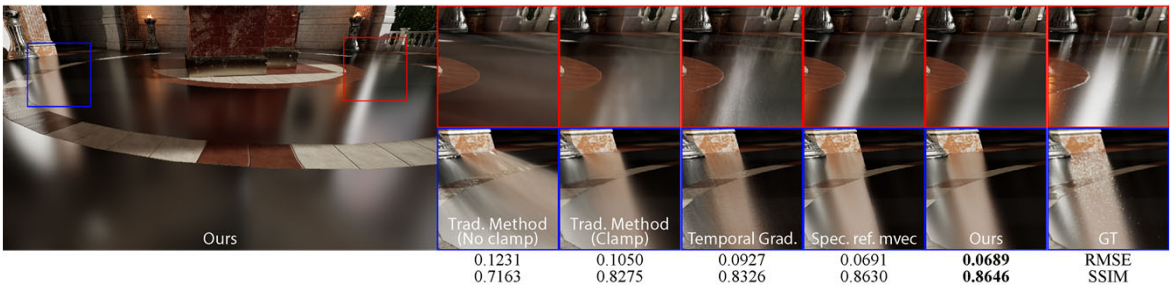
$x_{i-1}^R = \text{random sample}(P_{i-1} \equiv mirror[T^{-1}h_i, T^{-1}plane(s_i)], \Sigma)$

We start from the importance sampled secondary ray at the shading point [si] and the secondary hit point [hi] in the world space.
(c1) We then transform [hi] to the previous frame
Since the center of a glossy BRDF lobe is usually the strongest, here we can first assume that the glossy BRDF degenerates to pure specular,
(c2) now we can immediately find find its mirror-reflected image,
(c3) next, we project it towards the screen to get [sC i−1]
Then our insight is that, as the glossy lobe gradually emerges,
(c4)a region will appear around sC i−1, in which all the points are able to reflect to the same hit point hi−1.
This region can be approximated by tracing a glossy lobe from the virtual image towards sC i−1.

(c5) In practice, there is no need to trace any cones, and we simply assume that the glossy lobe is a Gaussian in directions, and that the intersected region is a Gaussian in positions as well as in the image space. Then our stochastic motion vector can randomly sample one corresponding pixel from the gaussian function.

(c6) It can be formally written as this:

Stochastic glossy ref. motion vector: result



Similar to shadows, we again compare our results with these methods. Specially, we compare with method using specular reflection motion vectors, also with clamping.

Ours: our glossy reflection motion vectors do not introduce ghosting artifacts.

Spec: The results of specular motion vector look plausible, but we can also find discontinuous artifacts around the edges of the reflected objects.

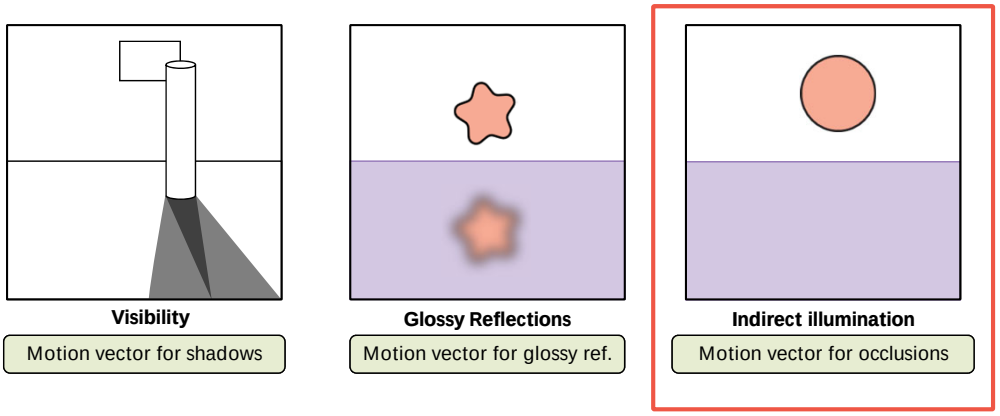
Trad: with traditional motion vectors, naive filtering produces significant ghosting

Trad clamp: Clamping relieves the lagging but introduces discontinuous artifacts.

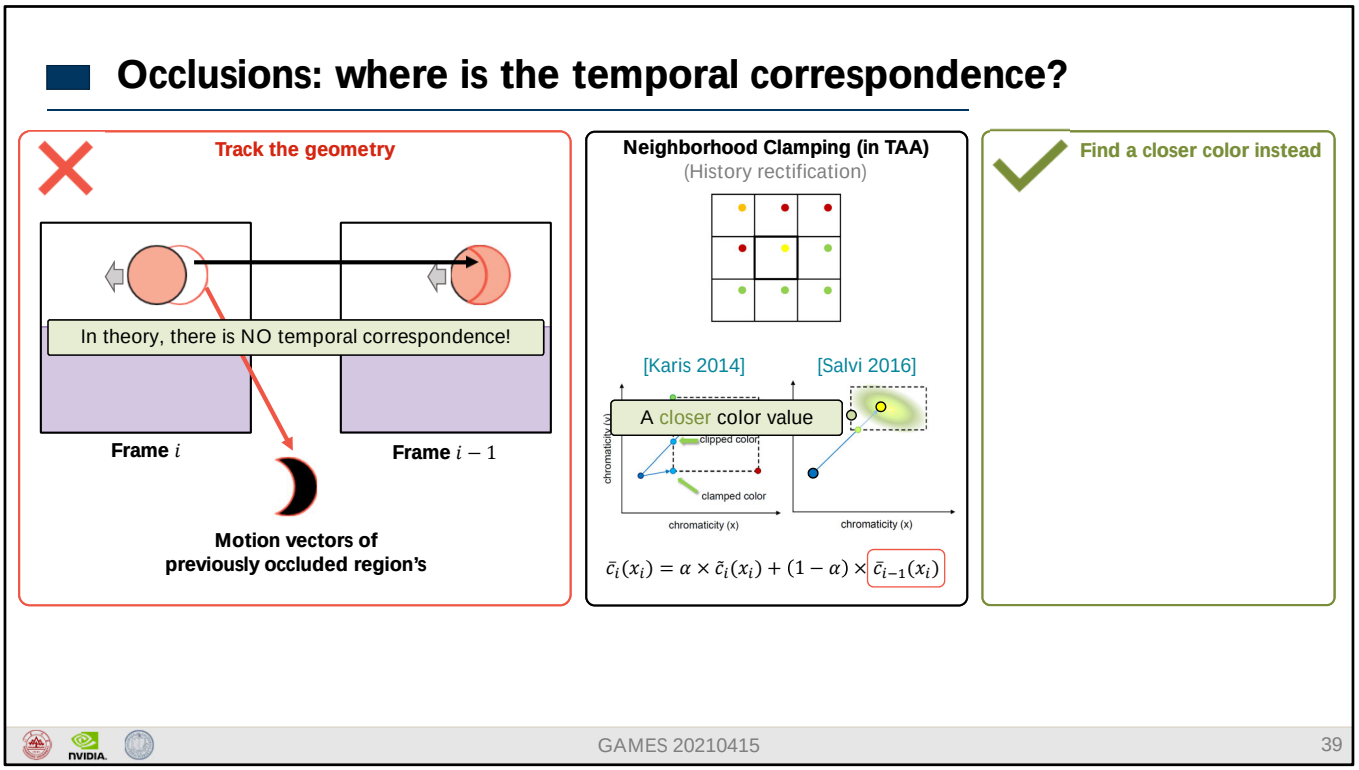
ASVGF: Finally, A-SVGF will always result in noisy

Our method

For shadows, glossy reflections and occlusions:

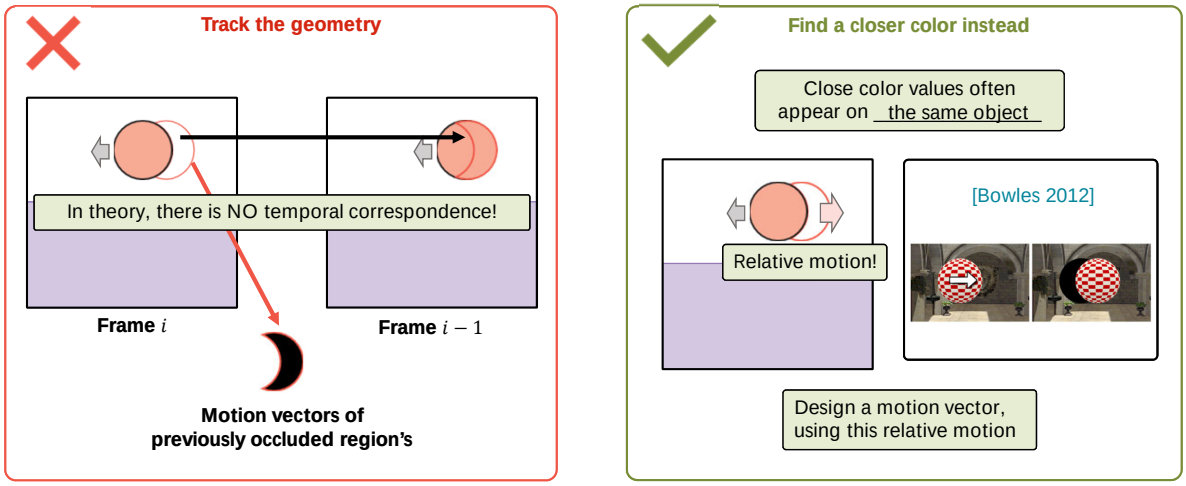


Finally, let's take a look at our motion vector for occlusions.



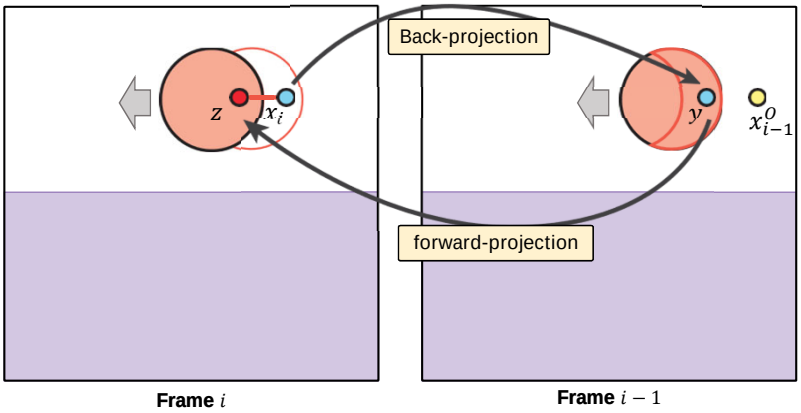
Tracking the geometry is still not working.
However, different from shadows and glossy reflections,
(c1) When occlusion happens, in theory there are no temporal correspondences of pixels in the occluded regions.
(c2) The back-projected motion vectors of these pixels will always land on the occluders, thus the previous pixel values cannot easily be used.
(c3) To alleviate this issue, we start from the clamping methods.
(c4) They clamp the history pixel values of occluders to the current value, but is still prone to ghosting artifacts, since usually the occluders have completely different colors.
(c5) Our insight is that if the history value is closer to the current value in the occluded regions, the issues produced by the clamping method can be better resolved.
(c6) So, we propose to find a closer color instead.

Occlusions: where is the temporal correspondence?



We observe that the close color values often appears on
(c1) the same object.
(c2) In this case, close color values appear on the background.
So when the object moving to left, we temporally reuse the color values from the right side.
(c3) This is a relative motion!
(c4) So, Inspired by Bowles et al., we are going to design a motion vector, using this relative motion.

Dual motion vector: calculation



Design a motion vector, using this relative motion

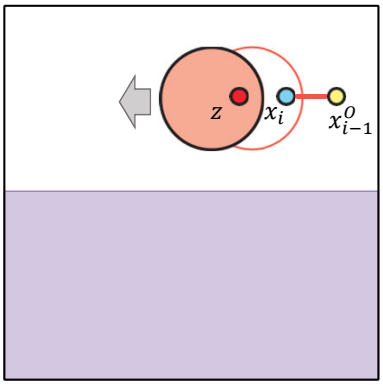
$$x_{i-1}^O = y + (x_i - z)$$

Our dual motion vector is:

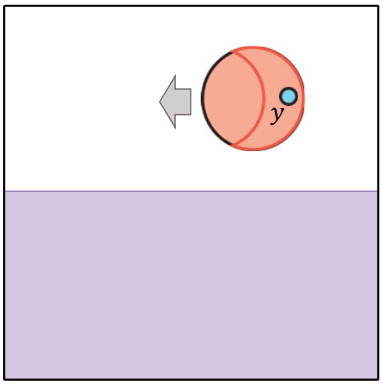
$$m^O(x_i) = x_{i-1}^O - x_i$$

For each pixel xi in current frame,
(c1) the traditional motion vector using back-projection gives the xi to y
(c2) We then continues to track the movement of y to z, using the motion vector of occluder's, that is a forward-projection.
(c3) Then, based on the relative positions of xi and z,
(c4) we are able to find the [xO i−1] in the previous frame.
(c5) Since we have applied a back-projection followed by a forward-projection, essentially using two motion vectors, we name our approach "dual motion vectors"

Dual motion vector: calculation



Frame i



Frame $i-1$

Design a motion vector, using this relative motion

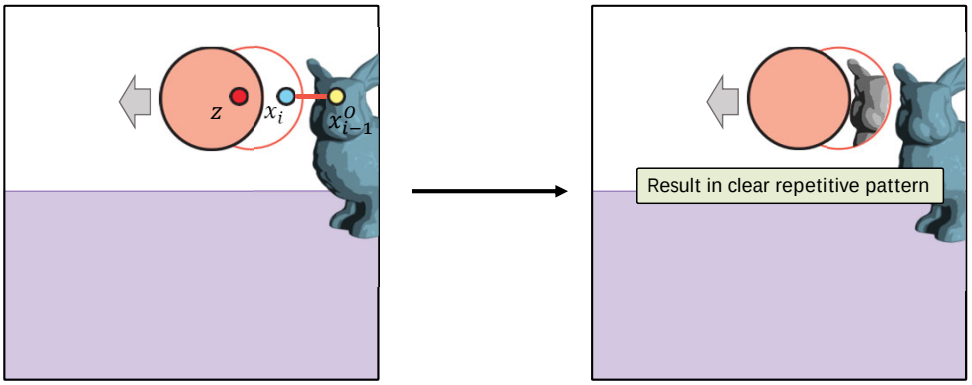
$$x_{i-1}^o = y + (x_i - z)$$

Our dual motion vector is:

$$m^o(x_i) = x_{i-1}^o - x_i$$

So far, we are able to find a much closer color value to xi on the background.

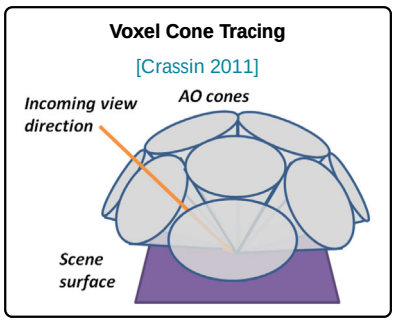
Reusing color vs. incident radiance

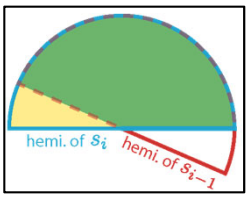


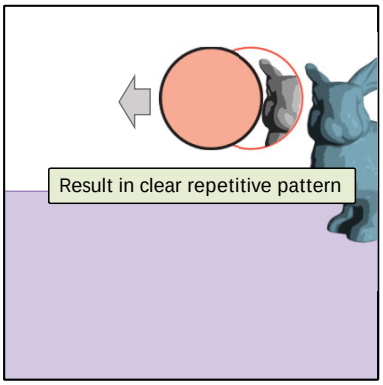
Frame i **Frame** i

However, if the corresponding pixel land on another object,
(c1) simply applying the color values will result in clear repetitive pattern. because it is essentially copy-pasting image contents.

Reusing color vs. incident radiance



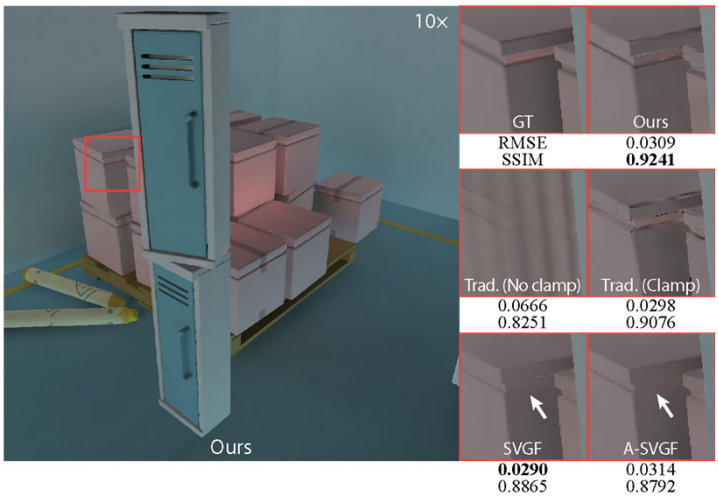




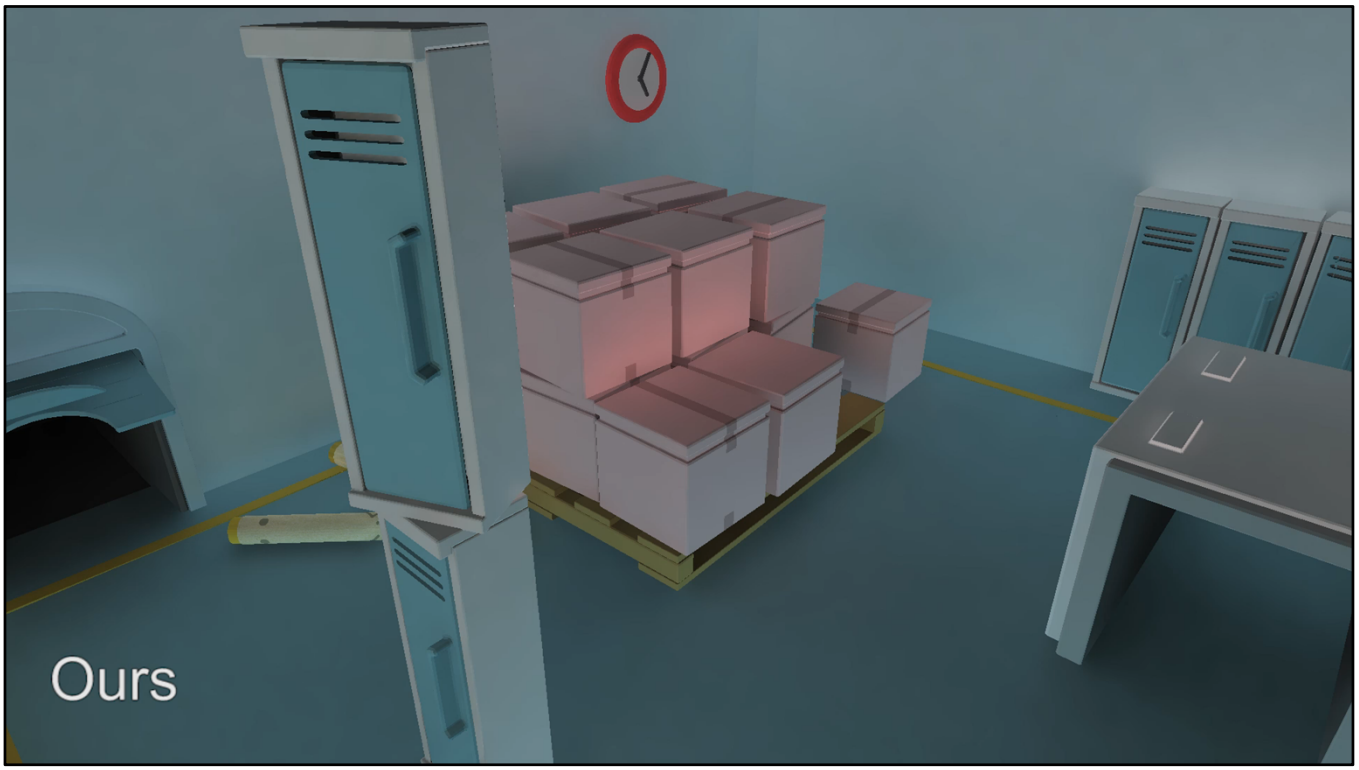
To address this issue, we propose to temporally reuse the incident radiance instead of the shading result.
Specifically, for the application of diffuse indirect illumination, we record the 2D indirect incident radiance per pixel.

(c1) For efficiently store the 2D incident radiance per pixel, we refer to the representation in the voxel cone tracing approach.
(c2) We subdivide a hemisphere into 6 cones.
(c3) And during spatial or temporal filtering, instead of averaging the shading result, we filter for each cone individually, using the overlapping solid angles between each pair of cones as the filtering weight.

Dual motion vector: result



We again compare our results with these methods



Ours: Our motion vectors avoid noise and ghosting, preserving details and suppressing overblur

GT: We produce the closest result to the ground truth

SVGF: SVGF method still results in significant amount of overblur

A-SVGF: A-SVGF method appears to be smeared spatially and loses temporal stability

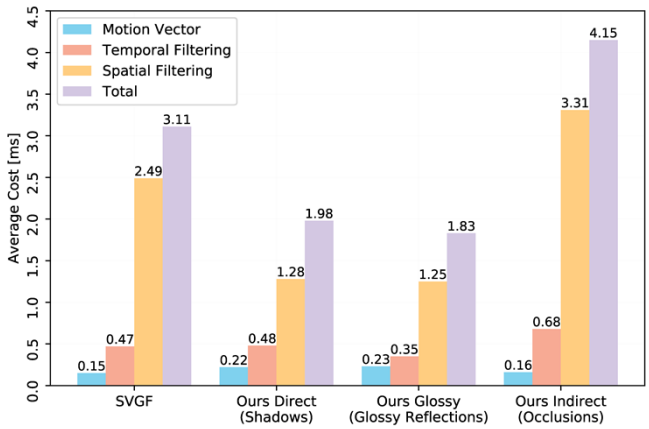
Trad (no clamp): The traditional motion vectors introduce ghosting artifacts.

Trad (clamp): With clamping, the results are less ghosting but much more noisy.

Discussion

Now let's move to the discussion.

Performance



Here we show the performance of our motion vectors.

We compared with SVGF using traditional motion vectors.

Different from previous work, our main contribution is not a spatial filtering approach or system, but the different types of motion vectors.

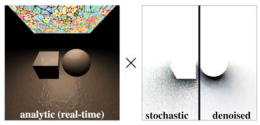
As one would expect from their simple computation, in practice, we observed only a negligible performance cost by replacing with our motion vectors.

Potential improvements

Gain Potential improvements with orthogonal approaches:

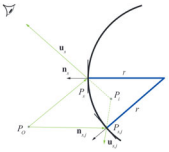
Better spatial filter

Blue noise sampling patterns
[Heitz et al. 2019]



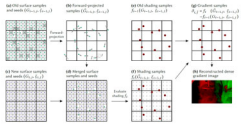
Better shadow estimators
[Heitz et al. 2018]

Help us to find more accurate motion vectors



Finding the mirror-reflected images by curved reflectors
[Hirvonen et al. 2019]

Dynamically select blending factor α



A-SVGF
[Schied et al. 2018]

...

Our method could gain potential improvements with orthogonal approaches.

For example,

(c1) any orthogonal method that improves spatial filtering is potentially helpful to our method

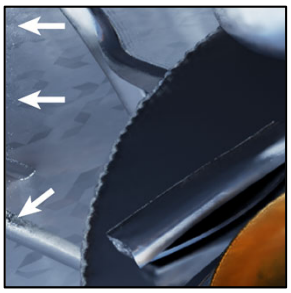
(c2) Also, there are better methods that help us to find more accurate motion vectors,

(c3) Moreover, adaptive methods that dynamically select temporal blending weights such as A-SVGF can also be combined with our method immediately for better temporal robustness.

Limitations

Ghosting due to the sudden change of local irradiance

The out of image plane issue



Failure due to drastic temporal change

...

There are some limitations of our method.

For example,

(c1) When filtering the occlusions, sometimes our method will still introduce faintly visible ghosting due to the intensely varied irradiance in the local area.

(c2) Also, currently, we do not explicitly handle the cases where the motion vectors point to regions outside the image plane.

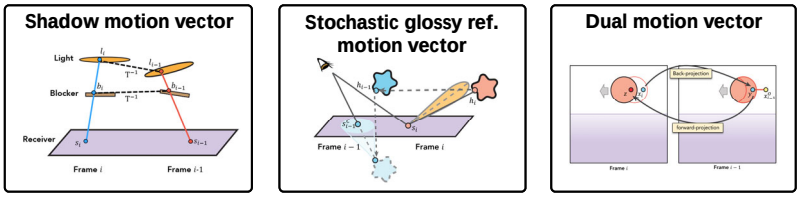
(c3) Moreover, When the temporal change is too drastic, there is barely any temporal information that we can use. One typical case is the sudden switch of scenes.

Conclusion

Now, let's wrap up.

Conclusion

- Three types of motion vectors



- Temporally reliable
- Negligible performance overhead
- Production-ready

We have proposed multiple types of motion vectors for better utilization of temporal information in real-time ray tracing.
(c1)We show that our motion vectors are temporally more reliable than traditional motion vectors,
(c2)with negligible performance overhead.
(c3)And they are production ready.

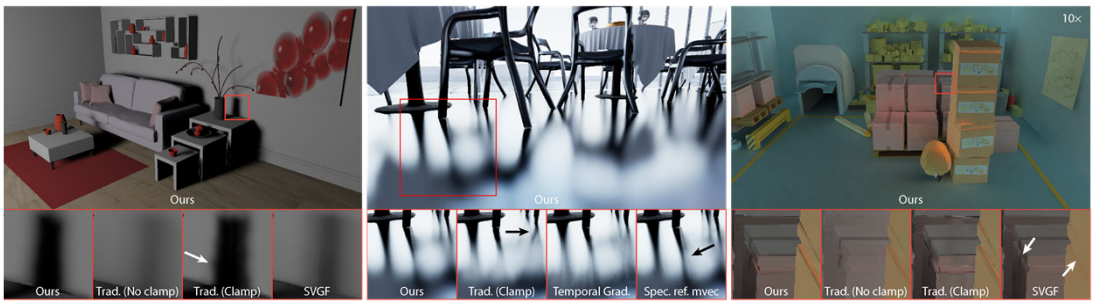
Future work

- Temporally reliable motion vector for distribution effects
  - Motion blur, depth of field, etc.
- Study non-exponential temporal falloffs
- Machine learning

In the future, we would like to design temporally reliable motion vector for distribution effects, such as motion blur and depth of field effects.

(c1)It would also be interesting to keep multiple previous frames to study non-exponential temporal falloffs.

(c2)
Exploiting machine learning approaches to better summarize temporal correlations from examples could also be a promising direction.



Temporally Reliable Motion Vectors for Real-time Ray Tracing

Zheng Zeng[1], Shiqiu Liu[2], Jinglei Yang[3], Lu Wang[1] and Ling-Qi Yan[3]

enjoy Vienna
EUROGRAPHICS
2021

Thanks for your attention, and I'd be happy to take any questions now.