PACIFIC GRAPHICS 2016
OCT. 11-14 OIST Conference Center
OKINAWA

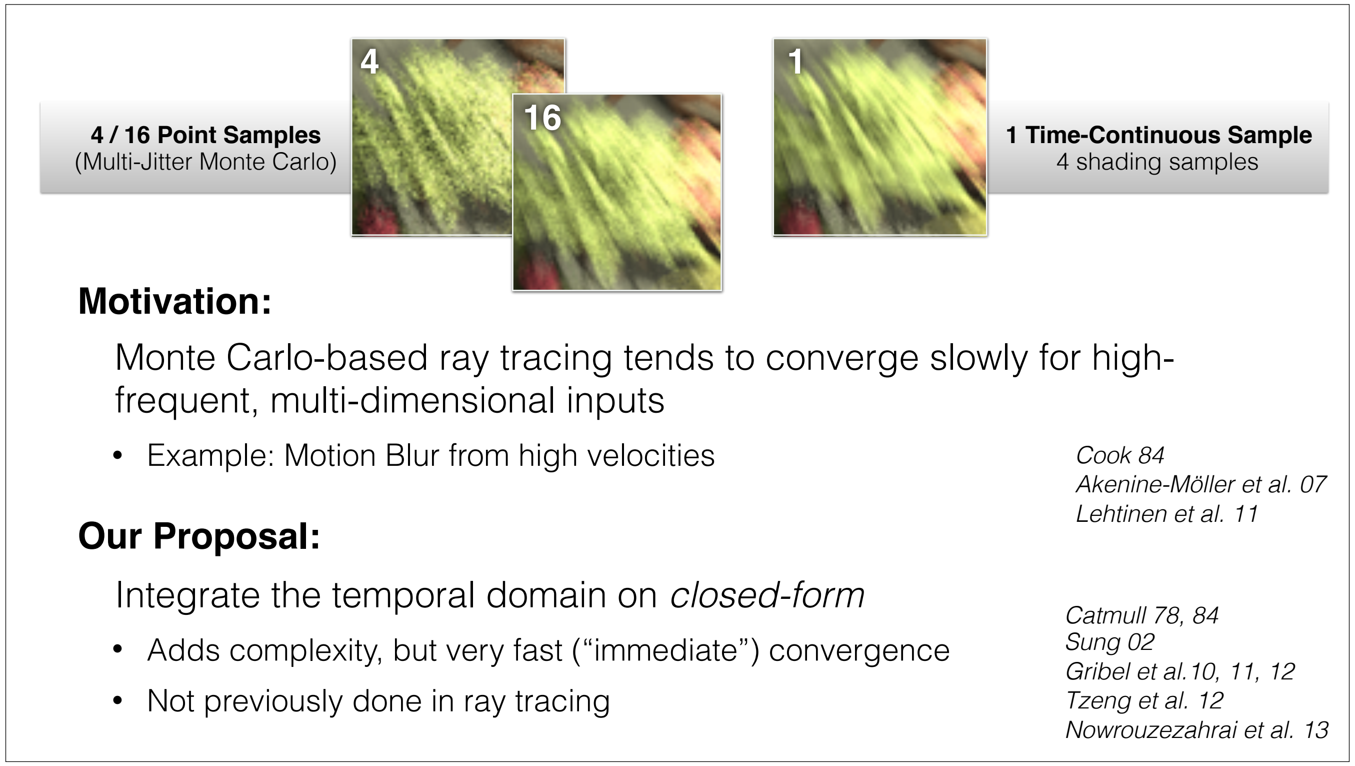
Time-Continuous Quasi-Monte Carlo Ray Tracing

Carl Johan Gribel

Malmö University
Lund University

Tomas Akenine-Möller

Intel Corporation
Lund University



4 / 16 Point Samples
(Multi-Jitter Monte Carlo)

1 Time-Continuous Sample
4 shading samples

Motivation:

Monte Carlo-based ray tracing tends to converge slowly for high-frequent, multi-dimensional inputs

- Example: Motion Blur from high velocities

Cook 84
Akenine-Möller et al. 07
Lehtinen et al. 11

Our Proposal:

Integrate the temporal domain on *closed-form*

- Adds complexity, but very fast ("immediate") convergence
- Not previously done in ray tracing

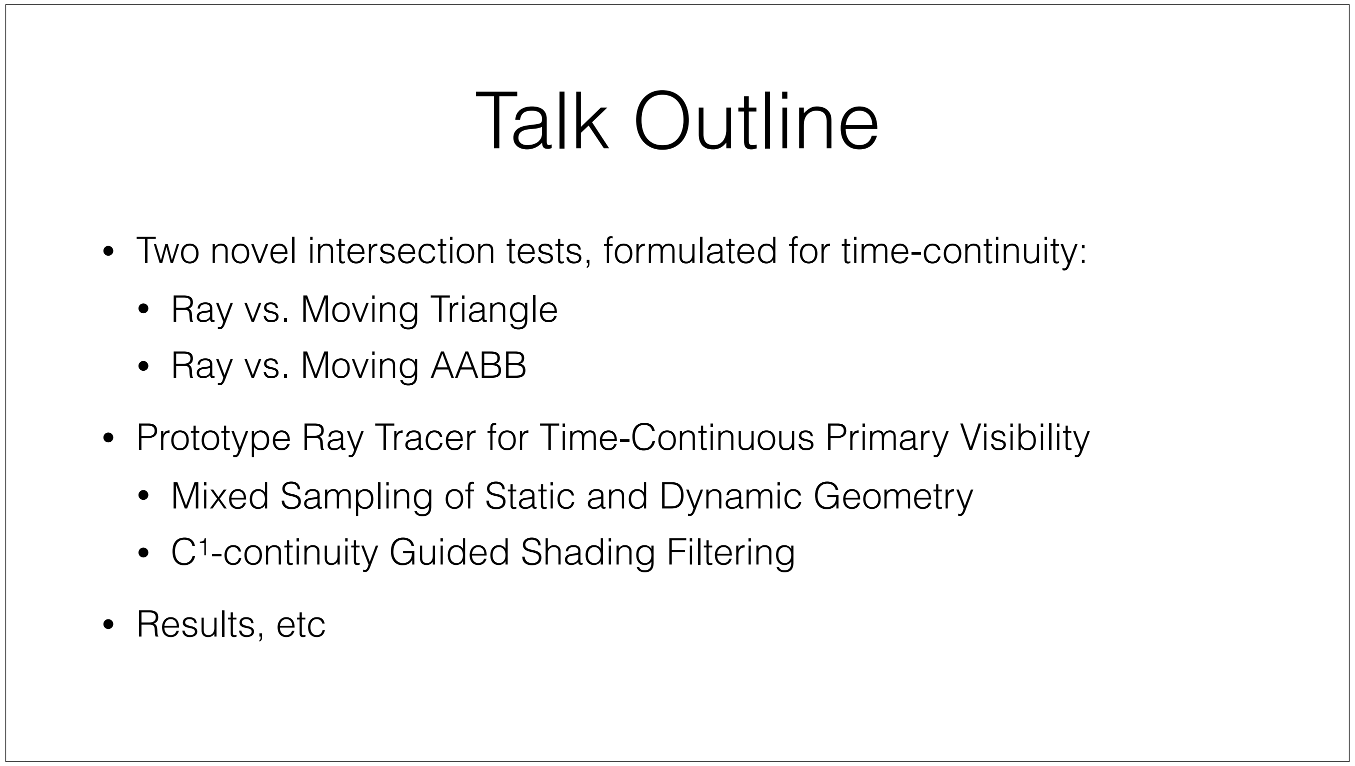
Catmull 78, 84
Sung 02
Gribel et al.10, 11, 12
Tzeng et al. 12
Nowrouzezahrai et al. 13

The motivation for this paper is –
variance reduction for ray traced motion blur.

Point sampling, such as Monte Carlo –
in rasterization and ray tracing alike –
do extend gracefully to high order effects such as motion blur,
but tends to converge slowly for high-frequencies such as small or fast-moving geometry.

It has been shown in the context of rasterization that analytical methods
do address this problem with efficiency –
and this paper explores this approach of analytical, or closed-form, integration of the temporal domain in the context of ray tracing,
which, to our knowledge, has not been done before.

In the end we show that this type of algorithm *can* be integrated in a high-end ray tracer,
and produce competitive or exceeding performance.



Talk Outline

- Two novel intersection tests, formulated for time-continuity:
  - Ray vs. Moving Triangle
  - Ray vs. Moving AABB
- Prototype Ray Tracer for Time-Continuous Primary Visibility
  - Mixed Sampling of Static and Dynamic Geometry
  - C^1-continuity Guided Shading Filtering
- Results, etc

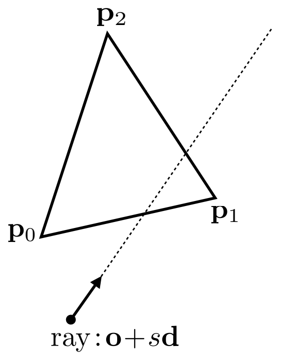
The major pieces of content for this talk are as follows.

We derive two, new intersection tests, adapted for time-continuity.
Ray versus moving triangle, and ray versus moving AABB.
These are executed at the leaf– and node–level during BVH traversal, respectively.

We also present a prototype ray tracer with a special time-continuous traversal kernel.
This ray tracer has a few auxiliary features,
including a mixed sampling heuristic,
to support static as well as dynamic geometry,
and a cluster mechanism to improve shading filtering.

Ray vs. Static Triangle

- Möller-Trumbore intersection test [Möller and Trumbore 97]
- Allow early-out termination, highly optimizable
- *s* = hit depth, *(u, v)* = barycentric coordinates of hit point



$$\begin{pmatrix} s\\ u\\ v \end{pmatrix} = \begin{pmatrix} -\mathbf{d}\\ \mathbf{p}_1-\mathbf{p}_0\\ \mathbf{p}_2-\mathbf{p}_0 \end{pmatrix}^{-\mathrm{T}} (\mathbf{o}-\mathbf{p}_0)$$

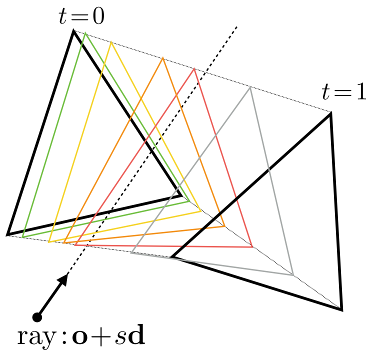
Intersection if $\left.\begin{array}{r} u\\ v\\ 1-u-v \end{array}\right\} \geq 0$

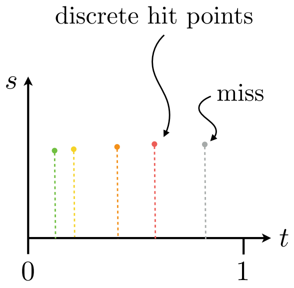
The ray triangle test is based upon the well known Möller-Trumbore test.
This test basically parameterize the triangle plane in barycentric coordinates, u and v, then sets this plane equation equal to the equation of the ray, then finally inverts the matrix of this system.

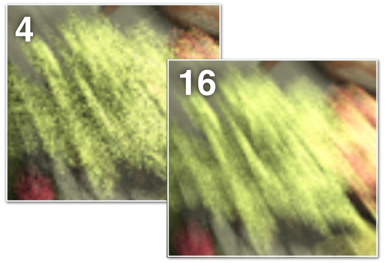
This results in expressions for u and v, along with certain intersection criteria.
This test is highly optimizable and has the benefit of
potential early-outs, whenever u and/or v intermediately is found to violate the intersection criteria.

Ray vs. Moving Triangle

- Monte Carlo Motion Blur: Assign discrete times to each ray
- In effect: interpolate triangle, intersect as if it were static





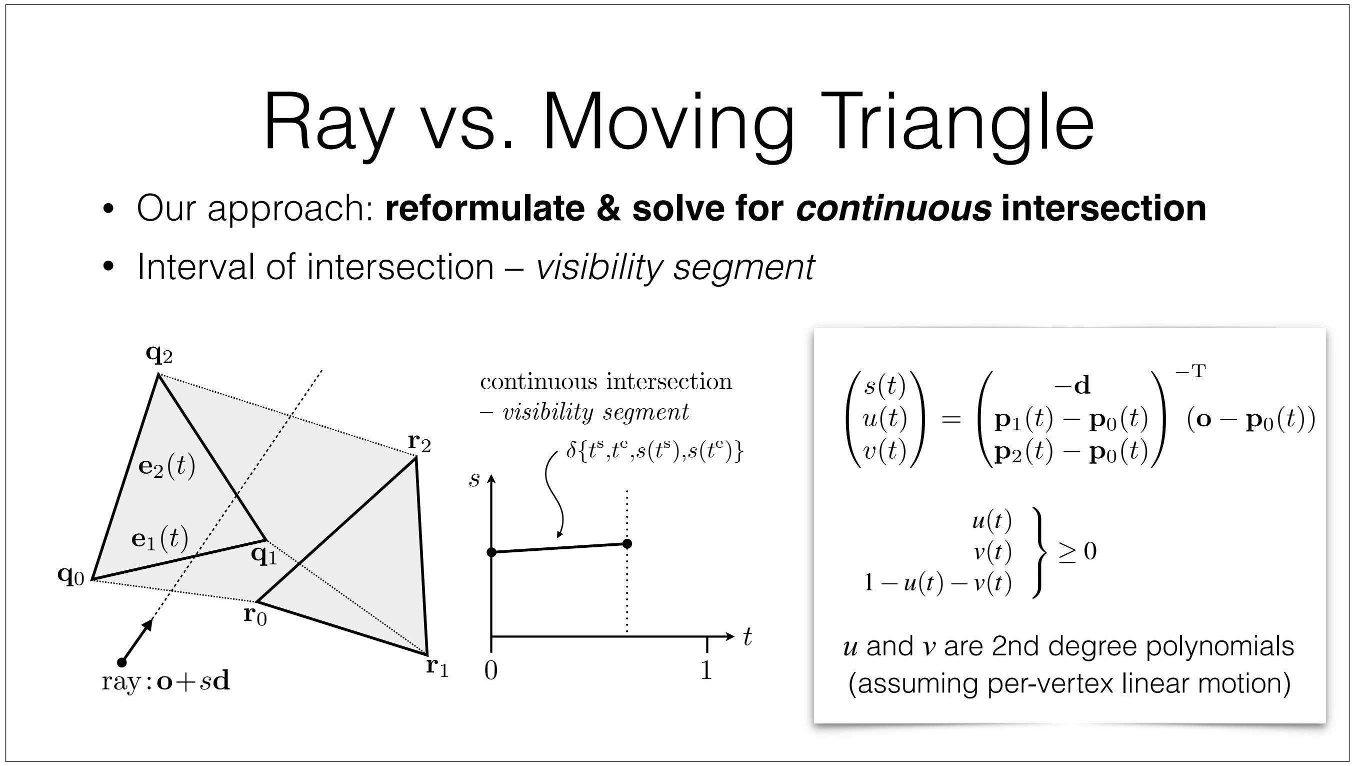


Point sampled Motion Blur is achieved by introducing
time and movement to the geometry, typically movement which is per-vertex linear.
Time is then discretized and each ray assigned a time value.

Upon intersection, the triangle is, in effect, interpolated to its position at that particular time, and then intersected using the same test.

All rays are then filtered in some way and we get motion blur.

But, as I said in the beginning, if the triangle is small or moving fast,
we may need a large number of rays to properly approximate this integral.



Ray vs. Moving Triangle

- Our approach: **reformulate & solve for *continuous* intersection**
- Interval of intersection – *visibility segment*

$$\begin{pmatrix} s(t) \\ u(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} -\mathbf{d} \\ \mathbf{p}_1(t)-\mathbf{p}_0(t) \\ \mathbf{p}_2(t)-\mathbf{p}_0(t) \end{pmatrix}^{-\mathrm{T}} (\mathbf{o}-\mathbf{p}_0(t))$$

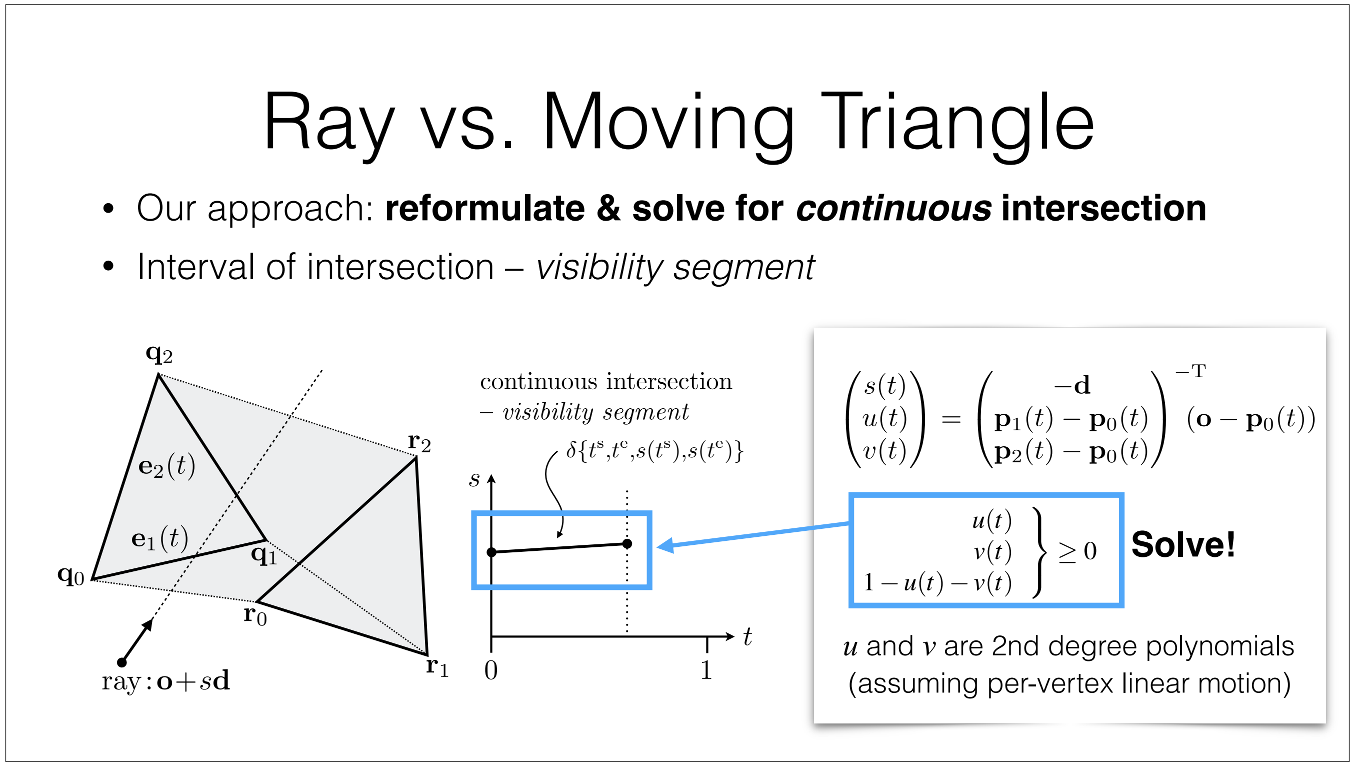
$$\left.\begin{array}{r} u(t) \\ v(t) \\ 1-u(t)-v(t) \end{array}\right\} \geq 0$$

u and v are 2nd degree polynomials
(assuming per-vertex linear motion)

We extend this test first and foremost by re-deriving it theoretically with time and vertex movement included in the formulation.

The final form, shown here – and I kindly refer you to the paper for details – is similar to the original form,
but we now know that the expressions for u and v are time-dependent, and, that they are quadratic polynomials.

This leads us to the concept of time-continuity.
Whereas Monte Carlo techniques essentially *samples* these polynomials, we _solve_ them, on closed-form, thus obtaining exact points in time for *when* a ray begins, and seizes, to intersect the triangle.



Ray vs. Moving Triangle

- Our approach: **reformulate & solve for *continuous* intersection**
- Interval of intersection – *visibility segment*

$$\begin{pmatrix} s(t)\\ u(t)\\ v(t)\end{pmatrix} = \begin{pmatrix} -\mathbf{d}\\ \mathbf{p}_1(t)-\mathbf{p}_0(t)\\ \mathbf{p}_2(t)-\mathbf{p}_0(t)\end{pmatrix}^{-\mathrm{T}} (\mathbf{o}-\mathbf{p}_0(t))$$

$$\boxed{\left.\begin{array}{r} u(t)\\ v(t)\\ 1-u(t)-v(t)\end{array}\right\} \geq 0}$$

Solve!

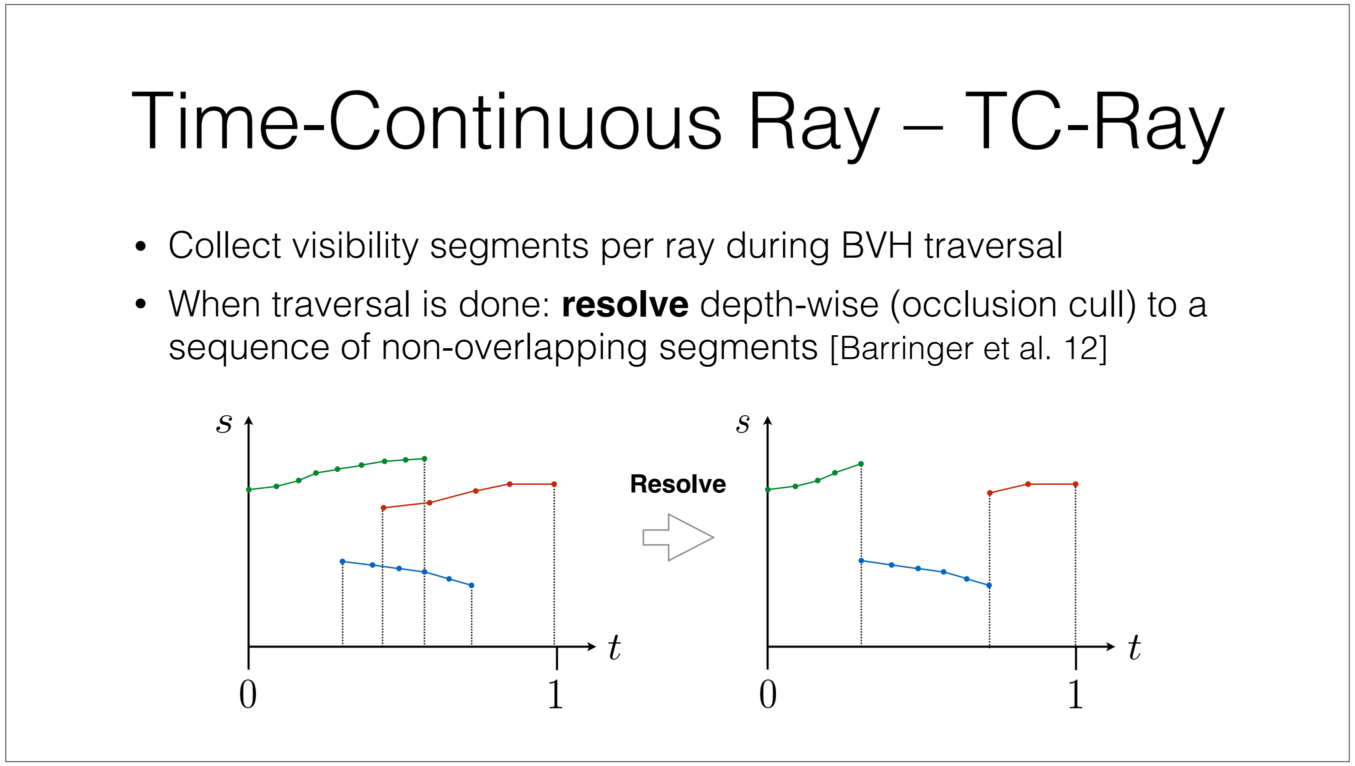
u and v are 2nd degree polynomials
(assuming per-vertex linear motion)

The obtained roots,
when combined and made to satisfy the inside criteria,
form not points of intersection,
but intervals of intersection.

We call these intervals *visibility segments*,
and they are actually an exact,
or at least pseudo-exact,
solution to the integral,
regardless of triangle size and velocity.

The part that is not exact is the depth function,
which is linearized.

We furthermore call the special type of rays,
which utilize this extended test,
time-continuous rays,
or TC-rays.



Time-Continuous Ray – TC-Ray

- Collect visibility segments per ray during BVH traversal
- When traversal is done: **resolve** depth-wise (occlusion cull) to a sequence of non-overlapping segments [Barringer et al. 12]

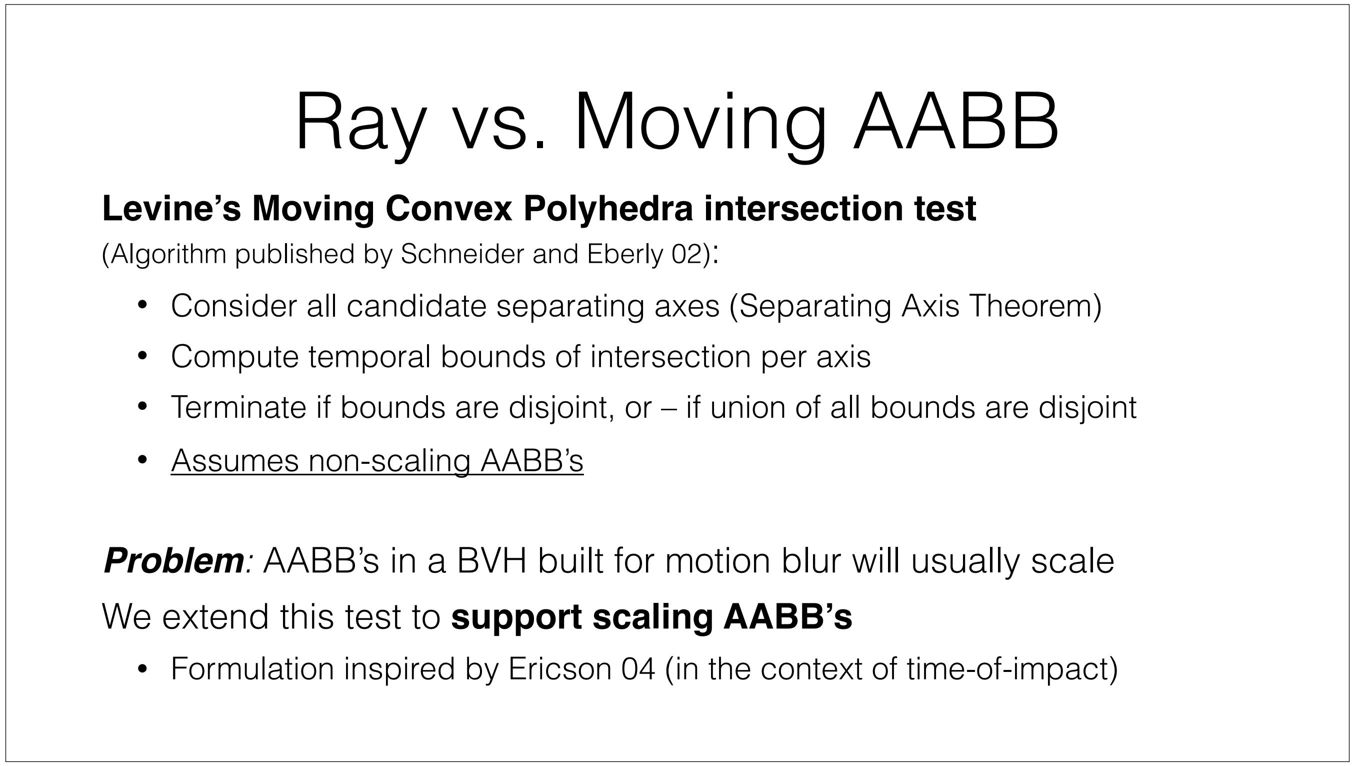
TC-ray are responsible for collecting and storing all resulting visibility segments during BVH traversal.

When traversal is complete,
the collection of visibility intervals is *resolved*,
meaning that occluded segments are culled or, if necessary, clipped,
according to Barringer et al.s algorithm.

The resulting set of visibility segments is a sequential,
non-overlapping list of segments which describe —
perpetually over the temporal domain —
which triangles are visible, for how long, and at which depth.

This set constitute the final,
temporally continuous,
visibility data for this ray.

[pause]



Ray vs. Moving AABB

Levine's Moving Convex Polyhedra intersection test

(Algorithm published by Schneider and Eberly 02):

- Consider all candidate separating axes (Separating Axis Theorem)
- Compute temporal bounds of intersection per axis
- Terminate if bounds are disjoint, or – if union of all bounds are disjoint
- Assumes non-scaling AABB's

***Problem**:* AABB's in a BVH built for motion blur will usually scale

We extend this test to **support scaling AABB's**

- Formulation inspired by Ericson 04 (in the context of time-of-impact)

Moving on to the ray versus box test,
which is executed between rays and BVH nodes.
This test is also an extension/adaptation to a previous algorithm,
namely Levine's test for a moving polyhedron against another polyhedron.

The clever thing Levine does is to utilize the separating axis theorem and essentially treat time as another potential separating axis, beyond the spatial ones.

The problem for us is that this test does not allow scaling of the moving polyhedra.
And we really need that since in a BVH with moving geometry,
the start- and end- node-boxes are very unlikely to be of equal size.

For this reason we extend Levine's test to support just that –
moving and scaling AABB's.

Ray vs. Moving AABB

- Candidate Separating axes for a ray $r = \mathbf{o} + s\mathbf{d}$ and an AABB:

$$\mathbf{n}_\mathrm{i} = \mathbf{u}_\mathrm{i} \times \mathbf{d} \qquad \text{where} \qquad \mathbf{u}_\mathrm{i} = \{(1,0,0),(0,1,1),(0,0,1)\}$$

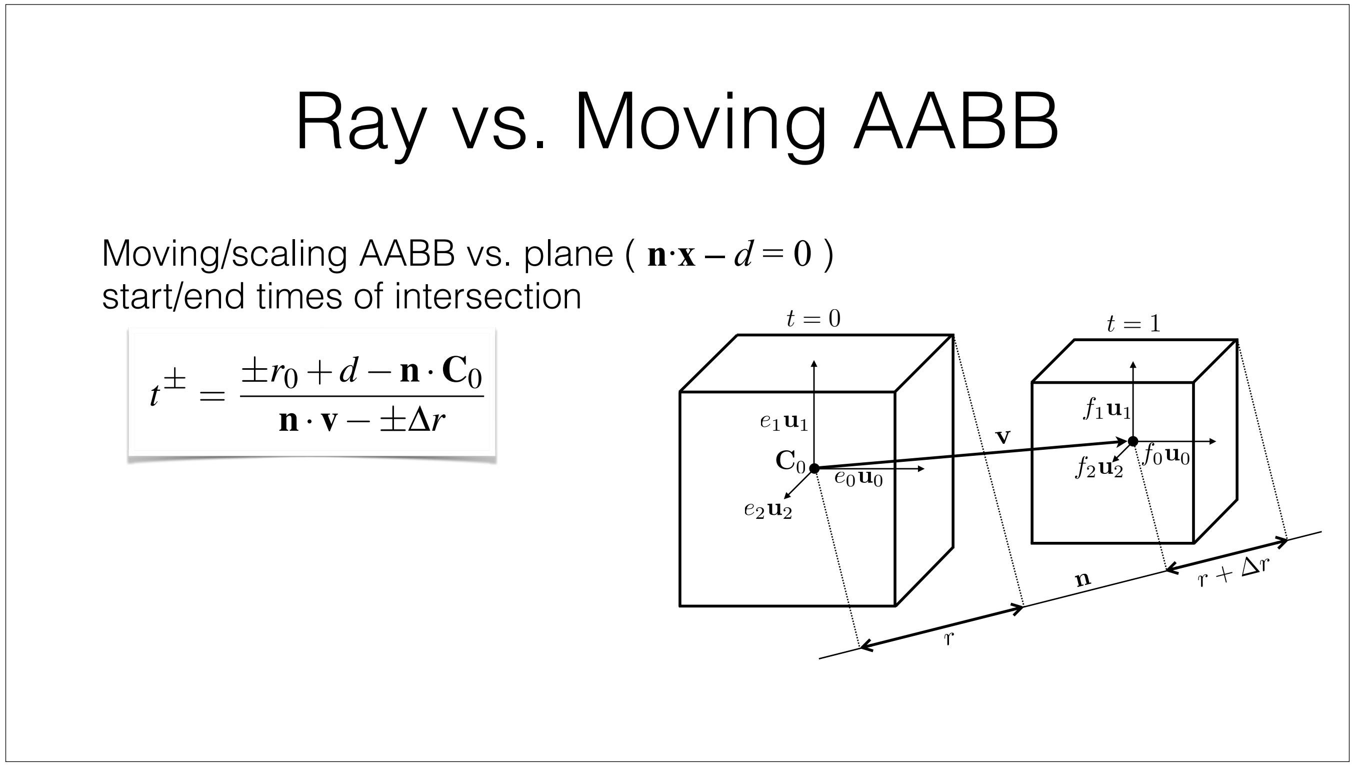
- These axes correspond to separating *planes*

$$\boxed{\pi_\mathrm{i} : (\mathbf{u}_\mathrm{i} \times \mathbf{d}) \cdot (\mathbf{x} - \mathbf{o}) = 0}$$

First we need to establish the set of candidate separating axes.

In this case there are three such axes.
These are the cross products between the ray edge,
i.e. the ray direction,
and the box edges,
i.e. the three cardinal axes.

These candidate axes are, in turn,
normals to corresponding separating *planes*,
and we handle these planes one at at a time.



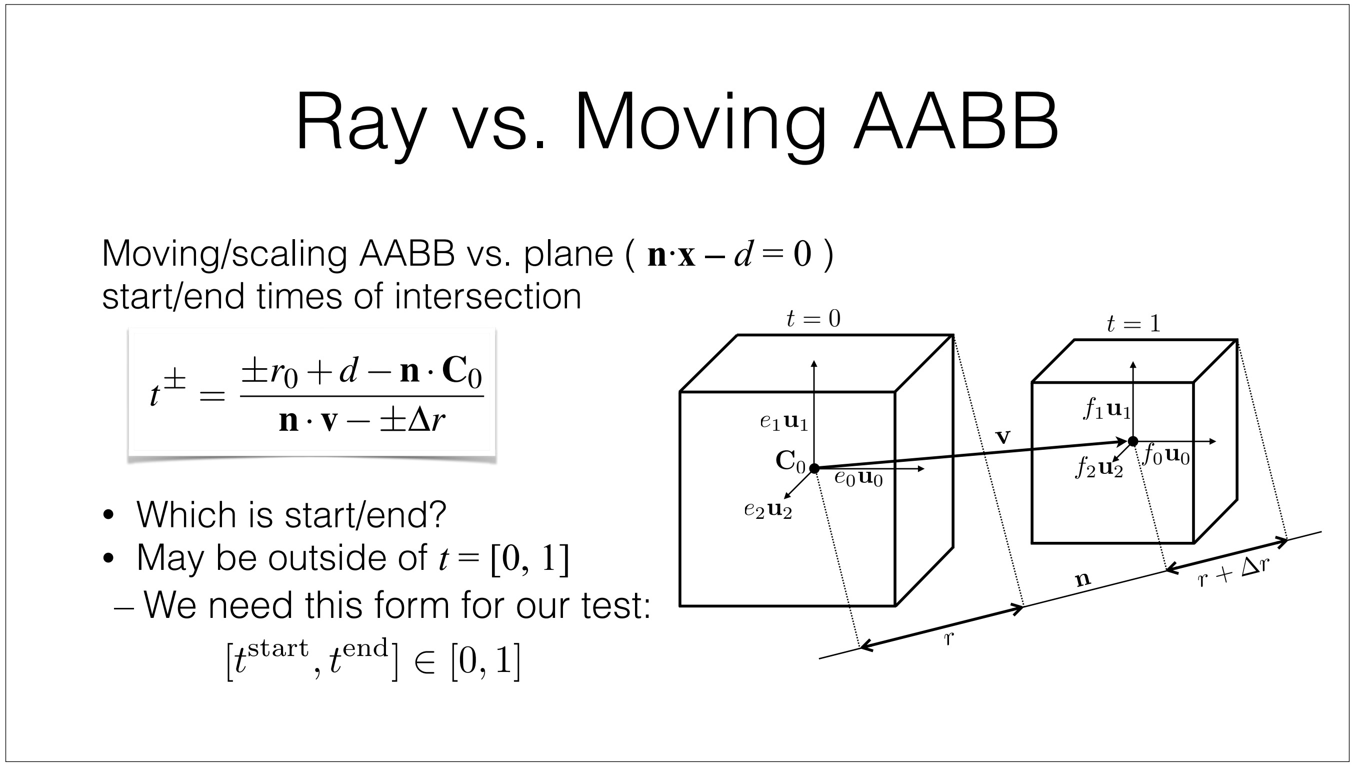
Ray vs. Moving AABB

Moving/scaling AABB vs. plane ($\mathbf{n}\cdot\mathbf{x} - d = 0$)
start/end times of intersection

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n}\cdot\mathbf{C}_0}{\mathbf{n}\cdot\mathbf{v} - \pm\Delta r}$$

So for one particular potential separating plane,
and a moving, scaling AABB,
we derive the following expression for two distinguished points in time,
t+ and t-,
for when the AABB either starts or seizes to intersect the plane.
(Again, for time reasons, I have to refer to the paper for a full derivation.)

Note that this box is allowed to scale,
as it moves from a start to an end location over time.



Ray vs. Moving AABB

Moving/scaling AABB vs. plane ($\mathbf{n}\cdot\mathbf{x} - d = 0$)
start/end times of intersection

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n}\cdot\mathbf{C}_0}{\mathbf{n}\cdot\mathbf{v} - \pm\Delta r}$$

- Which is start/end?
- May be outside of $t = [0, 1]$

– We need this form for our test:

$$[t^{\text{start}}, t^{\text{end}}] \in [0, 1]$$

Now, in order to express bounds which are chronologically correct and also clamped to the intended domain or shutter interval,
we have to make a closer investigation of how the
box actually interacts with the plane.

Ray vs. Moving AABB

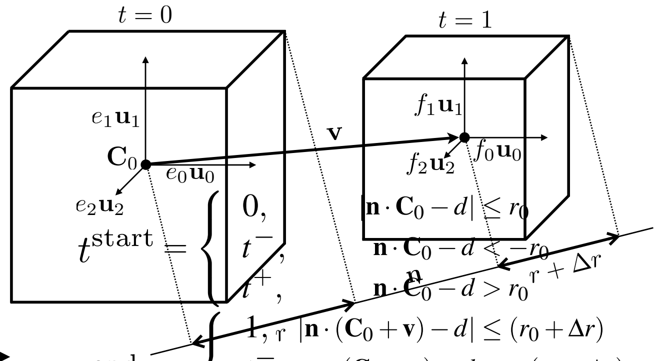
Moving/scaling AABB vs. plane ($\mathbf{n}\cdot\mathbf{x} - d = 0$)
start/end times of intersection

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n}\cdot\mathbf{C}_0}{\mathbf{n}\cdot\mathbf{v} - \pm\Delta r}$$

- Which is start/end?
- May be outside of $t = [0, 1]$

– We need this form for our test:

$$[t^{\text{start}}, t^{\text{end}}] \in [0, 1] \longrightarrow$$



$$t^{\text{start}} = \begin{cases} 0, & |\mathbf{n}\cdot\mathbf{C}_0 - d| \le r_0 \\ t^{-}, & \mathbf{n}\cdot\mathbf{C}_0 - d < -r_0 \\ t^{+}, & \mathbf{n}\cdot\mathbf{C}_0 - d > r_0 \end{cases}$$

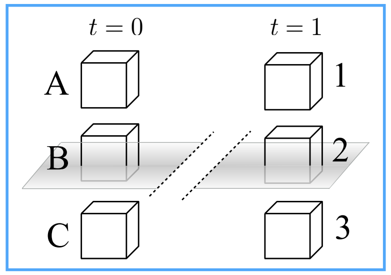
$$t^{\text{end}} = \begin{cases} 1, & |\mathbf{n}\cdot(\mathbf{C}_0 + \mathbf{v}) - d| \le (r_0 + \Delta r) \\ t^{-}, & \mathbf{n}\cdot(\mathbf{C}_0 + \mathbf{v}) - d < -(r_0 + \Delta r) \\ t^{+}, & \mathbf{n}\cdot(\mathbf{C}_0 + \mathbf{v}) - d > (r_0 + \Delta r) \end{cases}$$

In doing so, we find a number of relations between the two time-values,
t+ and t-,
along with conditions related to the distance between the box and plane.

Ray vs. Moving AABB

Moving/scaling AABB vs. plane ($\mathbf{n}\cdot\mathbf{x} - d = 0$)
start/end times of intersection

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n}\cdot\mathbf{C}_0}{\mathbf{n}\cdot\mathbf{v} - \pm\Delta r}$$



- Which is start/end?
- May be outside of $t = [0, 1]$
  - We need this form for our test:

$$[t^{\mathrm{start}}, t^{\mathrm{end}}] \in [0, 1] \longrightarrow$$

$$t^{\mathrm{start}} = \begin{cases} 0, & \boxed{\text{B}} \\ t^{-}, & \boxed{\text{C2 or C1}} \\ t^{+}, & \boxed{\text{A2 or A3}} \end{cases}$$

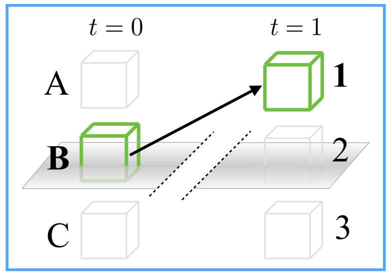
$$t^{\mathrm{end}} = \begin{cases} 1, & \boxed{\text{2}} \\ t^{-}, & \boxed{\text{A3 or B3}} \\ t^{+}, & \boxed{\text{B1 or C1}} \end{cases}$$

Put more simply,
these conditions are simply state flags for wether the box is inside, outside or in the plane,
at its start and end position,
respectively.

Ray vs. Moving AABB

Moving/scaling AABB vs. plane ($\mathbf{n}\cdot\mathbf{x} - d = 0$)
start/end times of intersection

$$t^{\pm} = \frac{\pm r_0 + d - \mathbf{n}\cdot\mathbf{C}_0}{\mathbf{n}\cdot\mathbf{v} - \pm\Delta r}$$



- Which is start/end?
- May be outside of $t = [0, 1]$

– We need this form for our test:

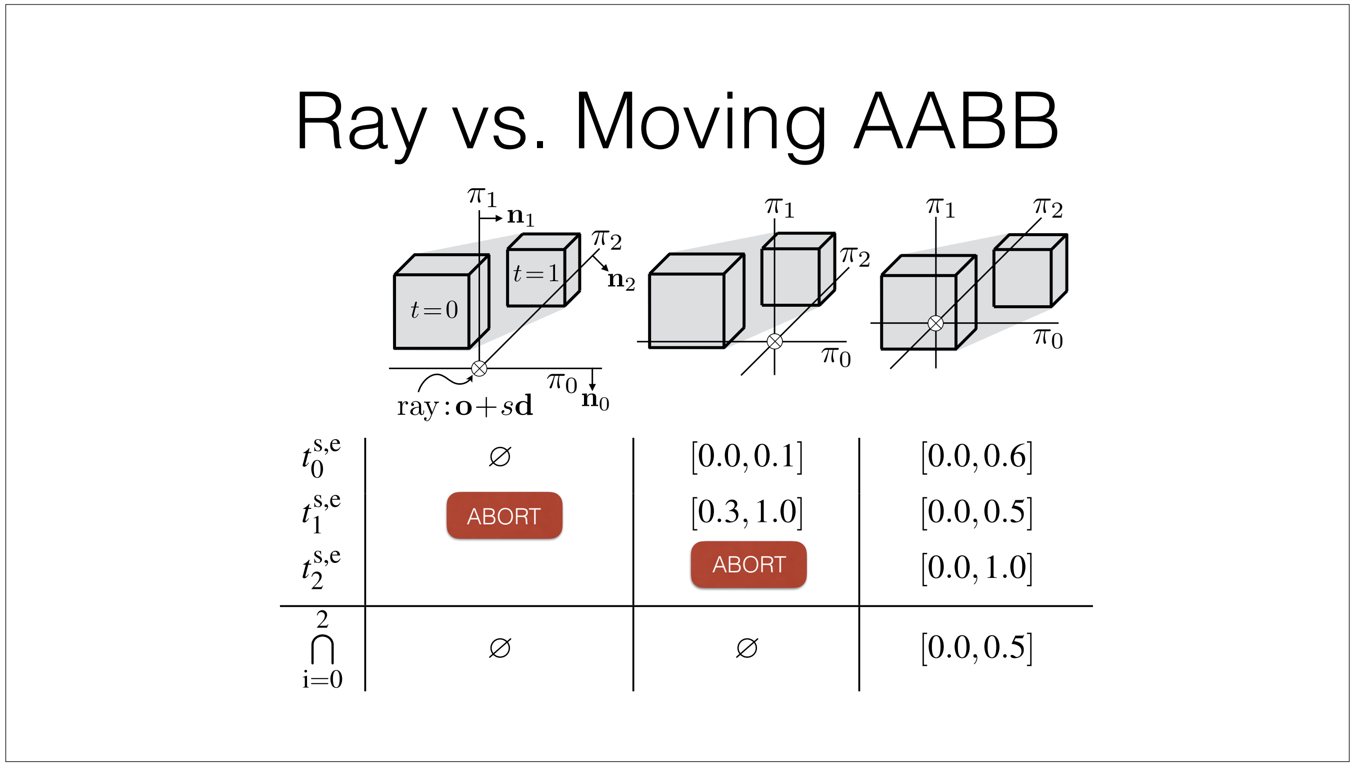
$$[t^{\text{start}}, t^{\text{end}}] \in [0, 1] \longrightarrow$$

$$t^{\text{start}} = \begin{cases} 0, & \textbf{B} \\ t^-, & \text{C2 or C1} \\ t^+, & \text{A2 or A3} \end{cases}$$

$$t^{\text{end}} = \begin{cases} 1, & 2 \\ t^-, & \text{A3 or B3} \\ t^+, & \textbf{B1}\text{ or C1} \end{cases}$$

For example,
this box is initially IN the plane, and then moves outside it.
This results in bounds starting at zero and ending in t+,
per the stated definition.

So, this heuristic may seem like a high-level thing but these cases involve just a few computations
and are straightforward to vectorize using bit masks.
We implemented this entire test using SIMD extensions with good performance.



Ray vs. Moving AABB

$t_0^{s,e}$	$\varnothing$	$[0.0, 0.1]$	$[0.0, 0.6]$
$t_1^{s,e}$	ABORT	$[0.3, 1.0]$	$[0.0, 0.5]$
$t_2^{s,e}$		ABORT	$[0.0, 1.0]$
$\bigcap_{i=0}^{2}$	$\varnothing$	$\varnothing$	$[0.0, 0.5]$

The resulting bounds for one plane is then carried over and combined with the next plane, and then the next, if necessary.

Here is an example of where all three planes are considered, for a moving, scaling box and three different rays, pointing into the figure.

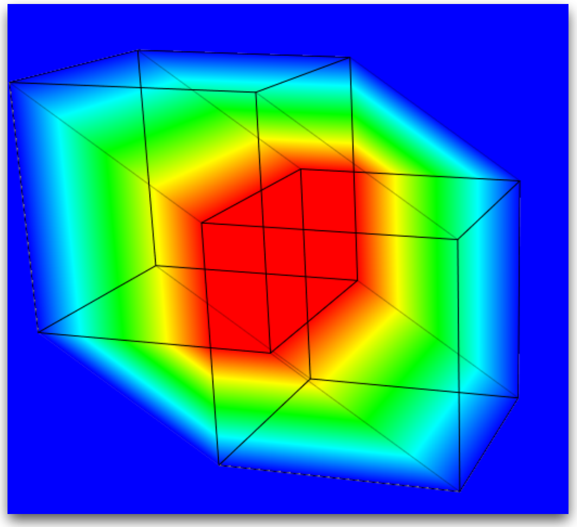
I the first case, already the first plane, pi-0, is separated from the box, so the test is aborted with no intersection.

In the second case, no plane is separating, but the bounds from the first two planes are disjoint, so the test can, again, be aborted early.

In the final case, again no plane is separating, but now combined bounds are not empty,
so here we do end up with an intersection, and we can see that it takes place between time 0 and 0.5.

Ray vs. Moving AABB

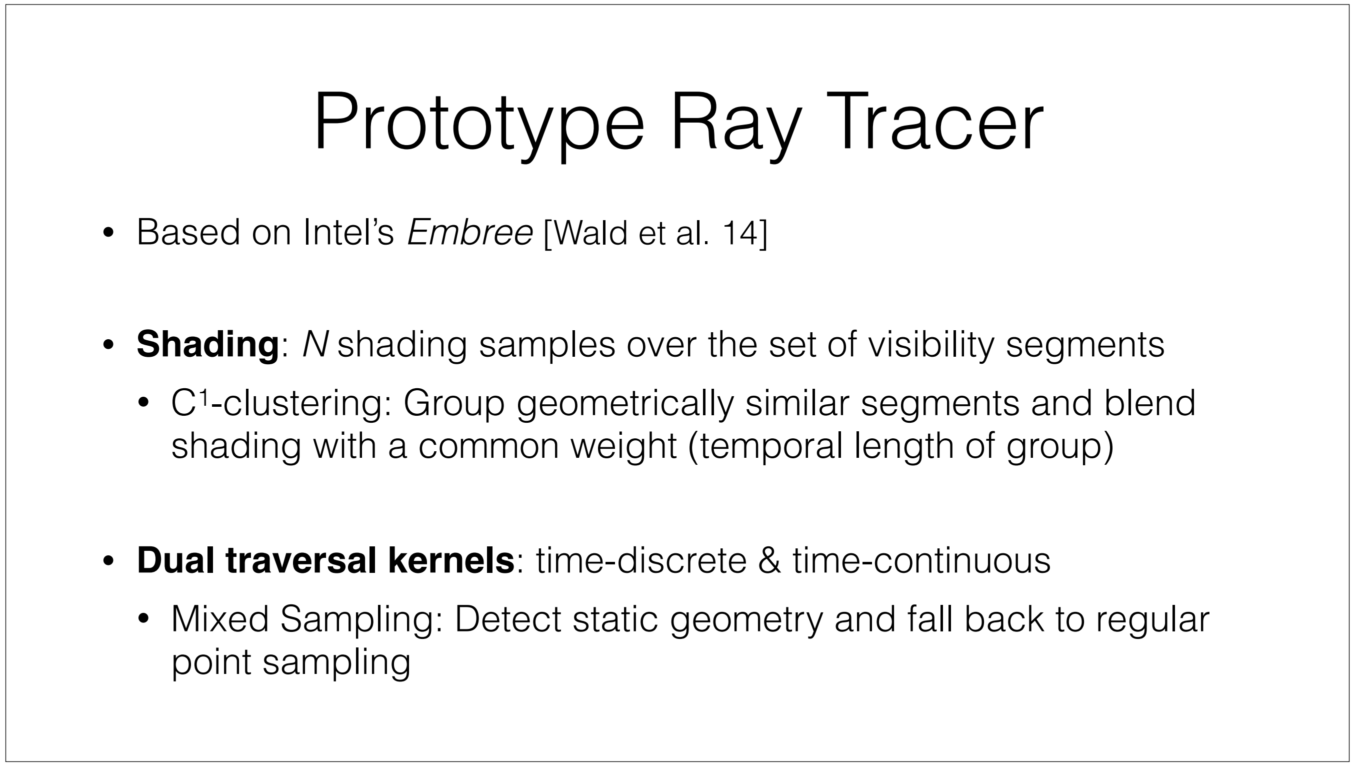
- Bound accuracy
  - Blue: empty bounds
  - Red: [0,1]



Finally a visualization of the tightness of the computed bounds,
This moving box is ray traced, using eye rays, and colorized based on the duration of intersection.

Blue means that the duration is zero,
and red means that it is one,
i.e. the ray intersects the box throughout its path.

This kind of tightness or is of course critical for performance because every false hit results
in unnecessary computations as the ray will
traverse more nodes and leafs without ever hitting something.



Prototype Ray Tracer

- Based on Intel's *Embree* [Wald et al. 14]

- **Shading**: *N* shading samples over the set of visibility segments
  - C^1-clustering: Group geometrically similar segments and blend shading with a common weight (temporal length of group)

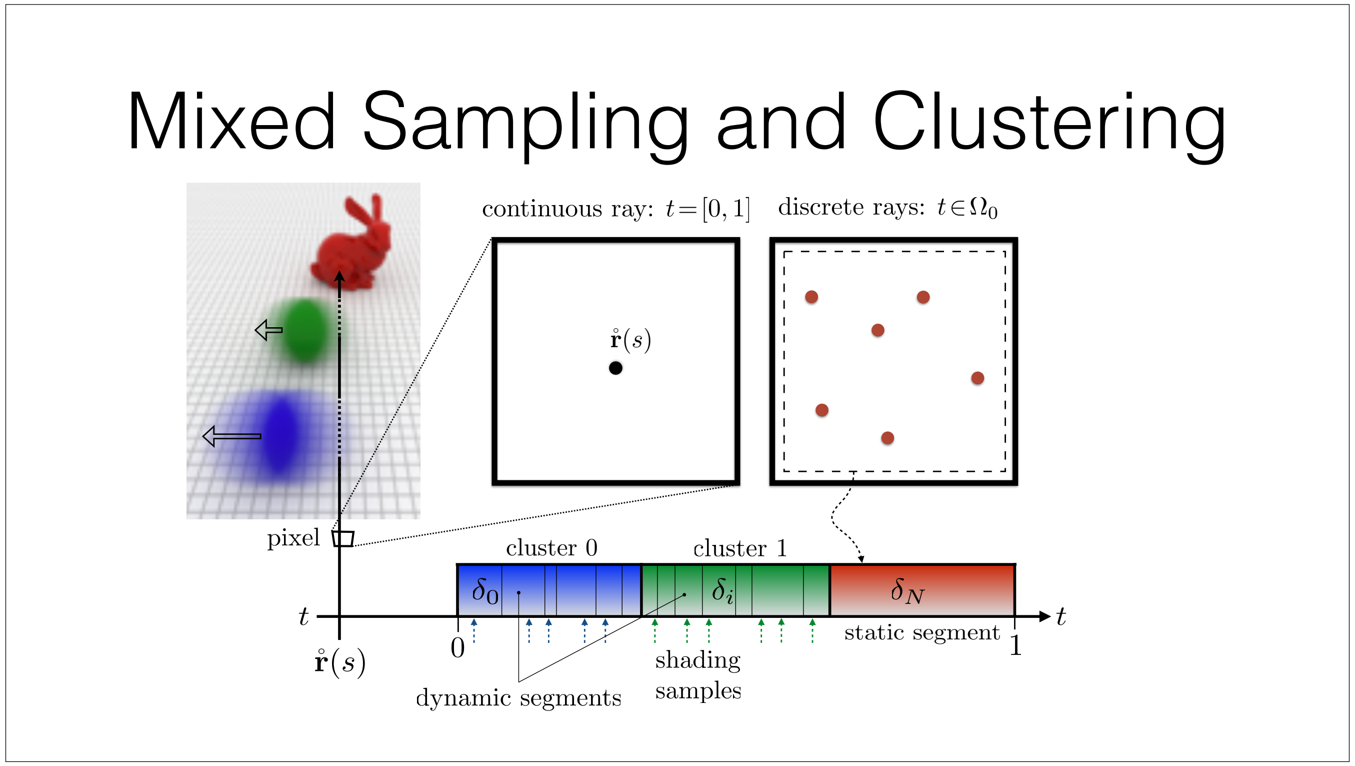
- **Dual traversal kernels**: time-discrete & time-continuous
  - Mixed Sampling: Detect static geometry and fall back to regular point sampling

Like I said in the beginning we have constructed a prototype ray tracer around the presented algorithms, based on the Embree Path Tracer from Intel.

We borrow and modify,
among other things,
Embree's SIMD-vectorized BVH for motion blur.

This prototype ray tracer contains two kernels,
one for regular, time-discrete traversal,
and one for time-continuous traversal.
In the latter kernel, shading is computed by sampling the visibility segments at a fixed baseline rate.

This tracer also has a couple of auxiliary functions which are either needed or beneficial,
and though I wont have time to describe them in-depth,
I would like to mention two of them —
and these are *segment clustering* and *mixed sampling*.



Mixed Sampling and Clustering

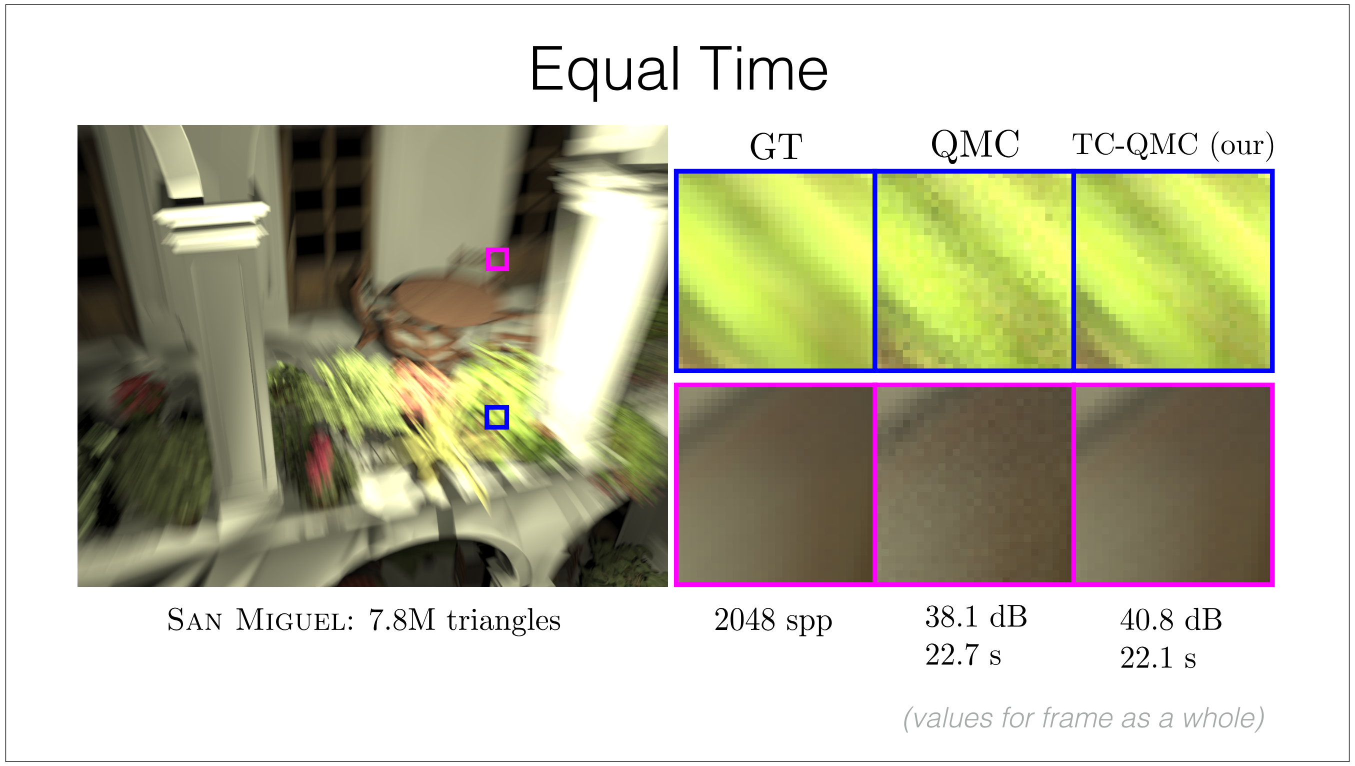
continuous ray: $t=[0,1]$
discrete rays: $t\in\Omega_0$
$\mathring{\mathbf{r}}(s)$
pixel
cluster 0
cluster 1
δ_0
δ_i
δ_N
t
t
0
1
static segment
$\mathring{\mathbf{r}}(s)$
shading samples
dynamic segments

So clustering, in short,
means that we attempt to identify and group segments originating from triangles of the same mesh.
For example,
the blue and green spheres, in this particular scene.

For such groups, shading is combined and weighted according to the temporal length of that group.
Experimentation shows that this technique causes image quality to converge faster.

Then we have the static bunny in the background,
which is visible during roughly the last third of the shutter interval.
Here there is no temporal variance,
and hence no reason at all to use our algorithm.

Here we make it so that traversal is shifted to the discrete kernel,
using discrete rays with a distribution over time – limited to this slot where it is visible – as well as, and more importantly, space,
so that this object receives proper sampling in space, which is its main source if alias.



Equal Time

SAN MIGUEL: 7.8M triangles

2048 spp

38.1 dB
22.7 s

40.8 dB
22.1 s

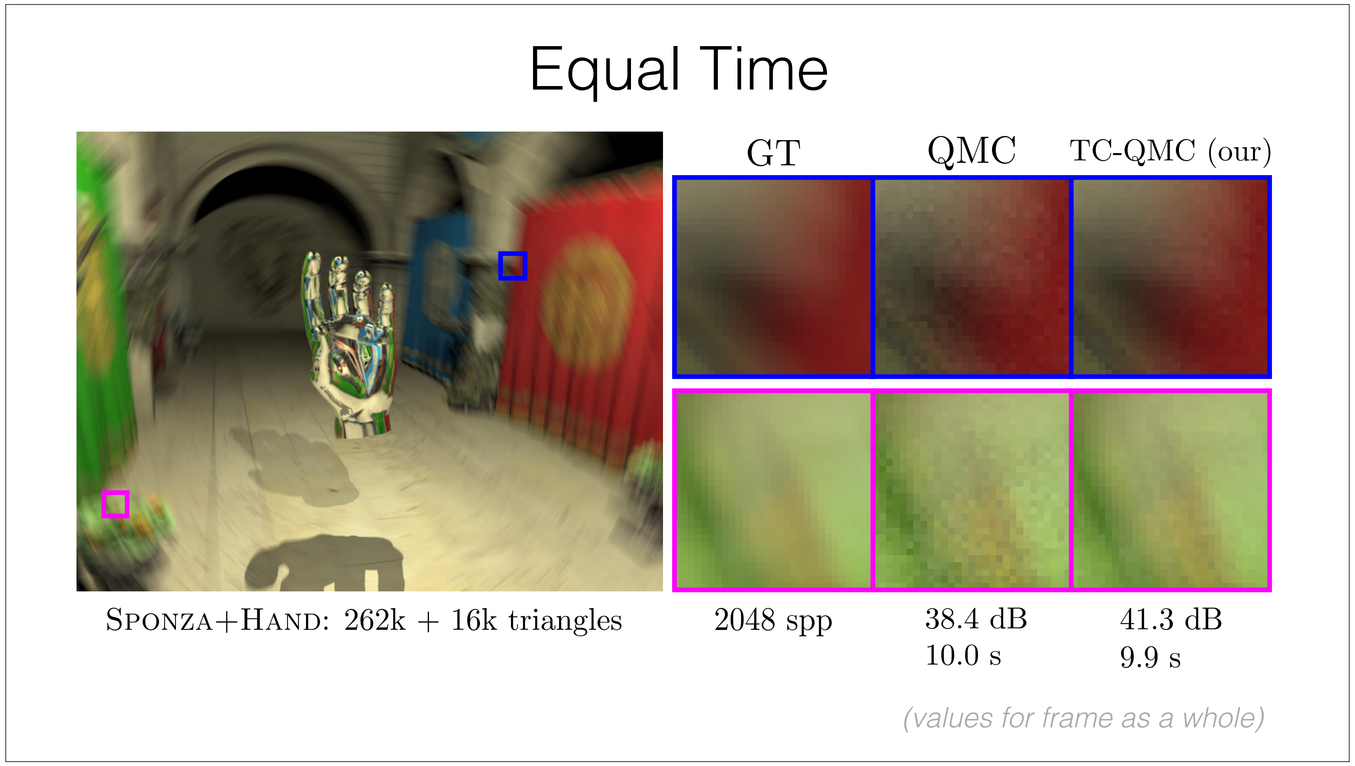
(values for frame as a whole)

On to results.

We compare our algorithm with Quasi-Monte Carlo sampling,
specifically, multi-jittered sampling.

This is San Miguel, about 8 million triangles, with fairly aggressive camera motion into the scene towards the balcony.

We see that our method produces much less temporal noise, as highlighted on the chair and a plant in the foreground.



Equal Time

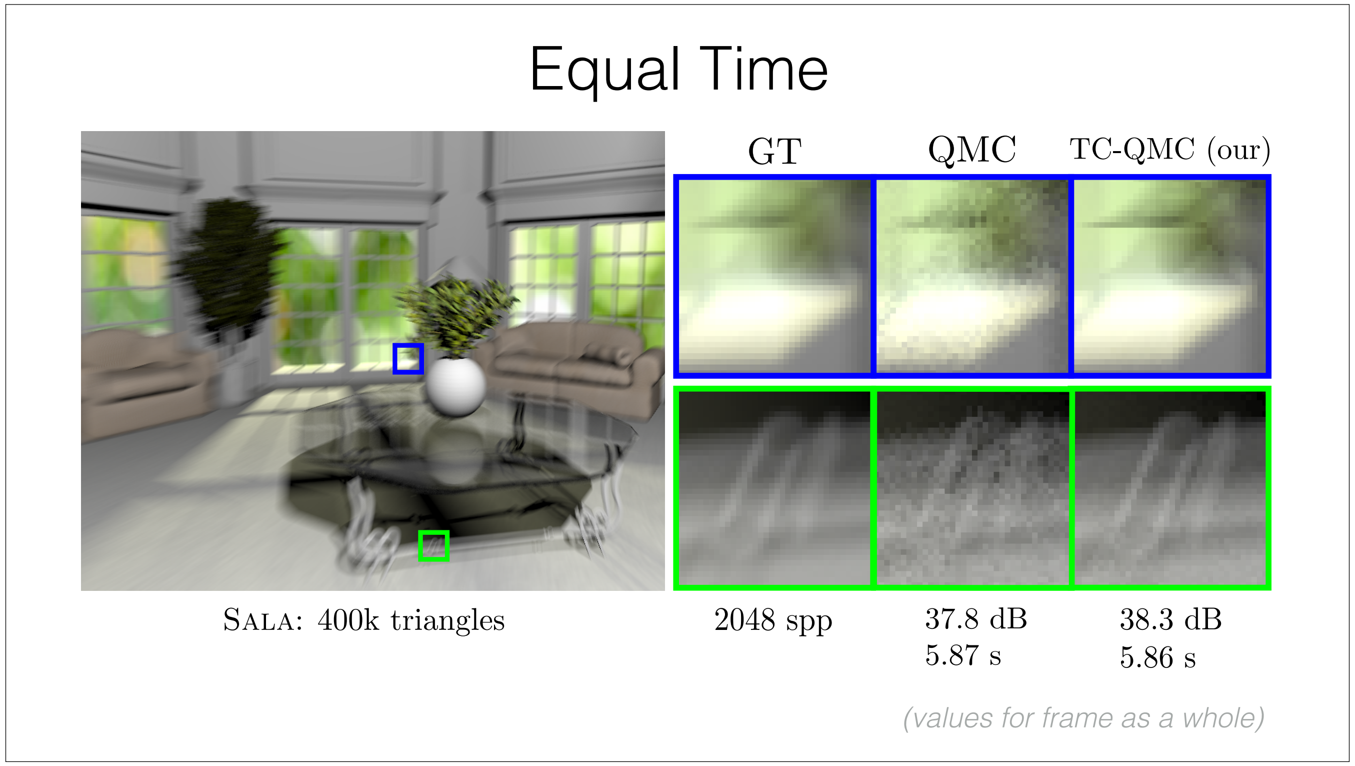
	GT	QMC	TC-QMC (our)
Sponza+Hand: 262k + 16k triangles	2048 spp	38.4 dB 10.0 s	41.3 dB 9.9 s

(values for frame as a whole)

The same effect is visible in this scene with the hand from Utah Animation Repository merged into Crytek Sponza.

One thing to note here is that the hand is completely static with respect to the camera,
meaning that mixed-sampling will kick in,
and sampling of the hand will fall back on the discrete traversal kernel,
to make sure that rays are properly distributed over space.

This also means that the hand itself is actually identical between our method and QMC.



Equal Time

	GT	QMC	TC-QMC (our)
SALA: 400k triangles	2048 spp	37.8 dB 5.87 s	38.3 dB 5.86 s

(values for frame as a whole)

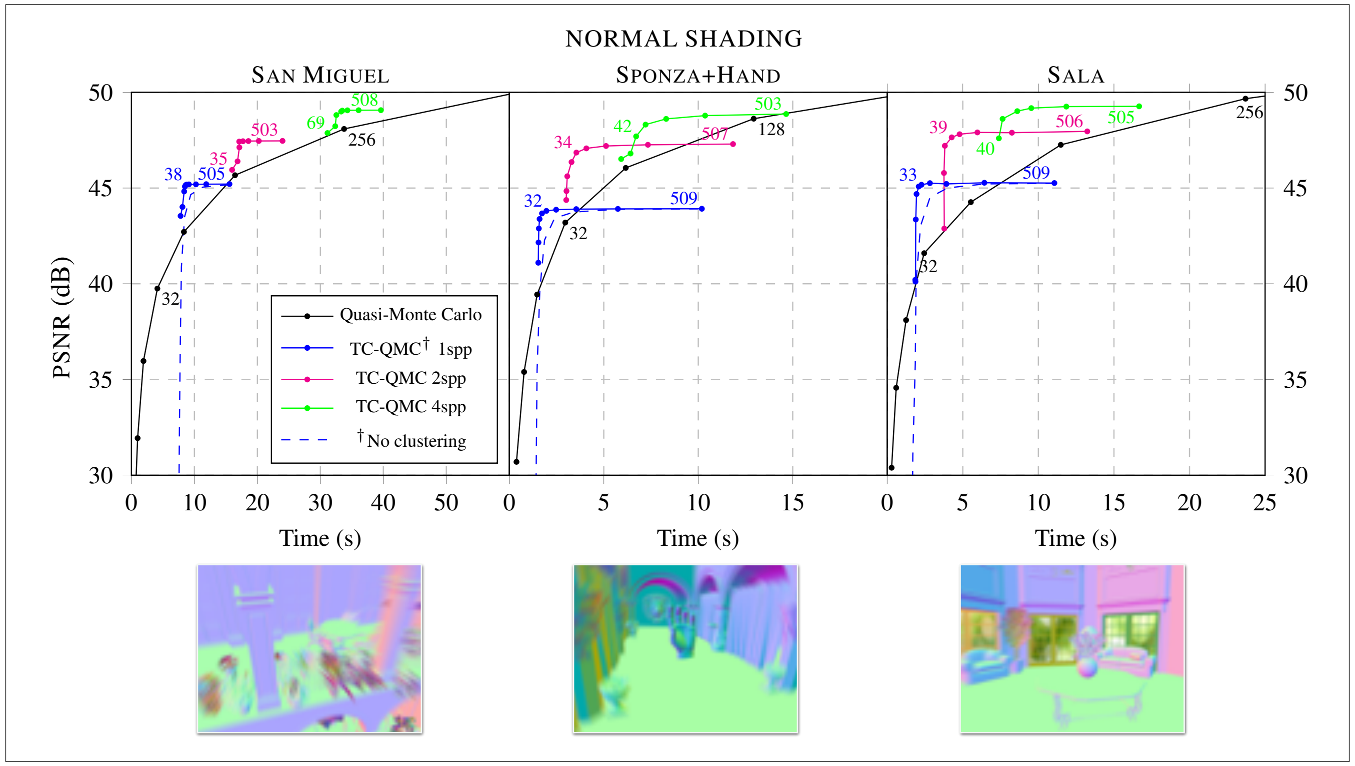
Sala exhibits much of the same features,
especially on the fine details such as the table.

There is slightly *less* difference here,
between our method and QMC,
in overall image quality.

This which might be because there are many flat surface,
which aren't really advantageous to us,
as well as slightly less motion.

Another factor which plays in here is the glass tabletop,
which is made out of a reflective-refractive material,
meaning that all its shading is a result of secondary rays, and beyond.

And since our method considers only primary rays,
in its current implementation,
there is no effect of time-continuity here –
our method uses time-discrete rays for secondary visibility,
just like Quasi-Monte Carlo.



We have some hard data measurements as well.

This is quality, PSNR, as a function of rendering time,
for QMC, the black line, and for our method with one, two and four TC-rays per pixel, colored blue, magenta and green, respectively.
QMC utilizes an increasing number of rays per pixel,
and our method utilizes an increasing number of shading samples.

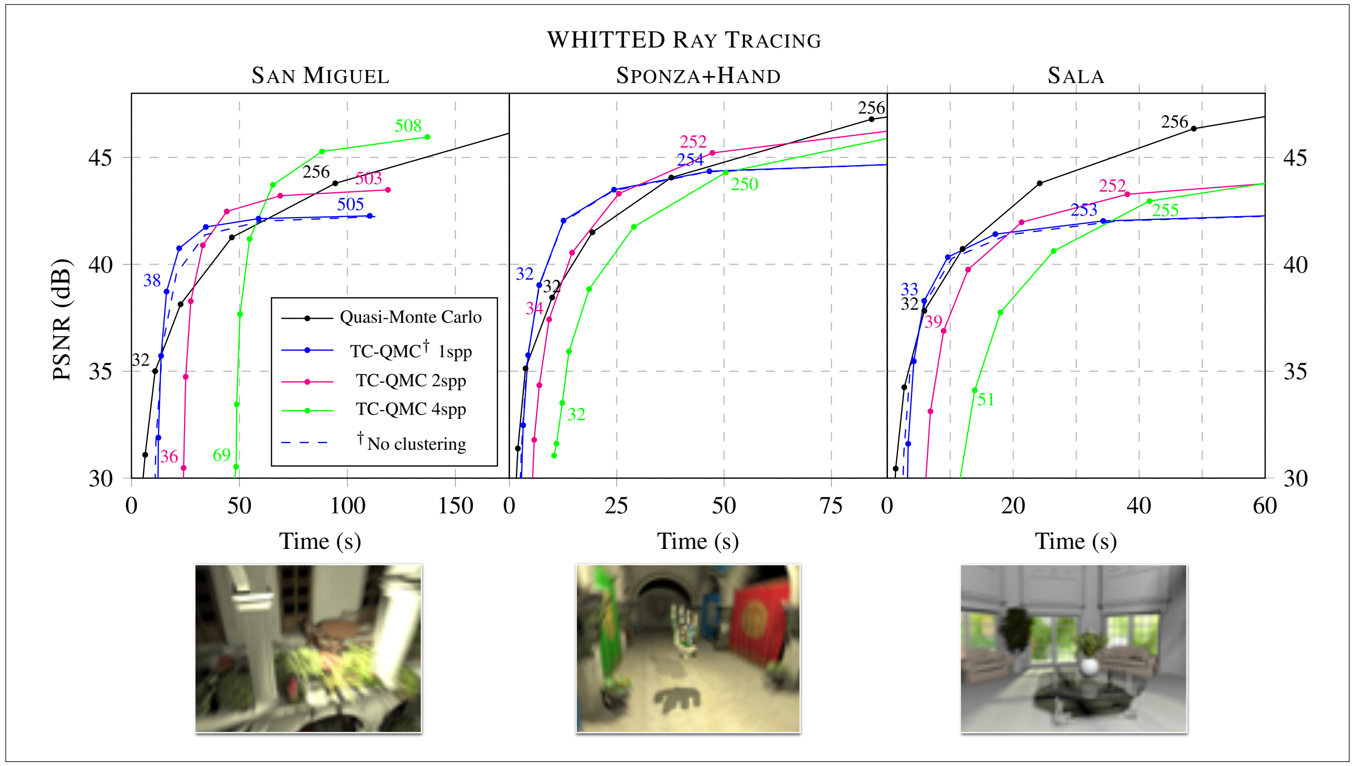
In this set we use simple normal shading to better highlight just visibility.

The main features to point out here is that while QMC increases in quality in a familiar, logarithmic pattern,
our method, once the extra overhead is taken into account,
rises very steeply, almost immediately, in quality.

Our curve then plateaus, because of the low distribution over space.

By adding more TC-rays per pixel, the entire curve shifts to the right,
i.e. due to more overhead,
but also up, towards better quality.

So we can see that, depending on the number of TC-rays used per pixel, there is a clear sweet-spot, or sweet range,
where our method clearly produces better quality at equal time.



In this data set we use Whitted shading instead,
and even though the same benefits are visible,
they are somewhat smaller.

This is most clearly in Sala, where QMC takes over and is more and more efficient, compared to our method,
for more samples and longer rendering times.

This is quite interesting because the underlying geometrical problem is exactly the same, the only difference is shading.
So clearly, and expectedly, shading introduce additional noise which our method is not always fully capable of handling.

On a closing note, I think this is maybe the main avenue for future work.
I.e. to better utilize this hard-earned, high-quality visibility information that we obtain through time-continuity, and use that to reduce variance in shading as well.

Thanks!

Thanks to

Dept. of Computer Science, Malmö University

Intel’s Advanced Rendering Technology team

Rasmus Barringer for SIMD–fu

CGF Reviewers for valuable feedback

…and to my family …and to You!

😃🇯🇵

Scenes

Hand: Utah 3D Animation Repository

Crytek Sponza: Marko Dabrovic/Frank Meinl

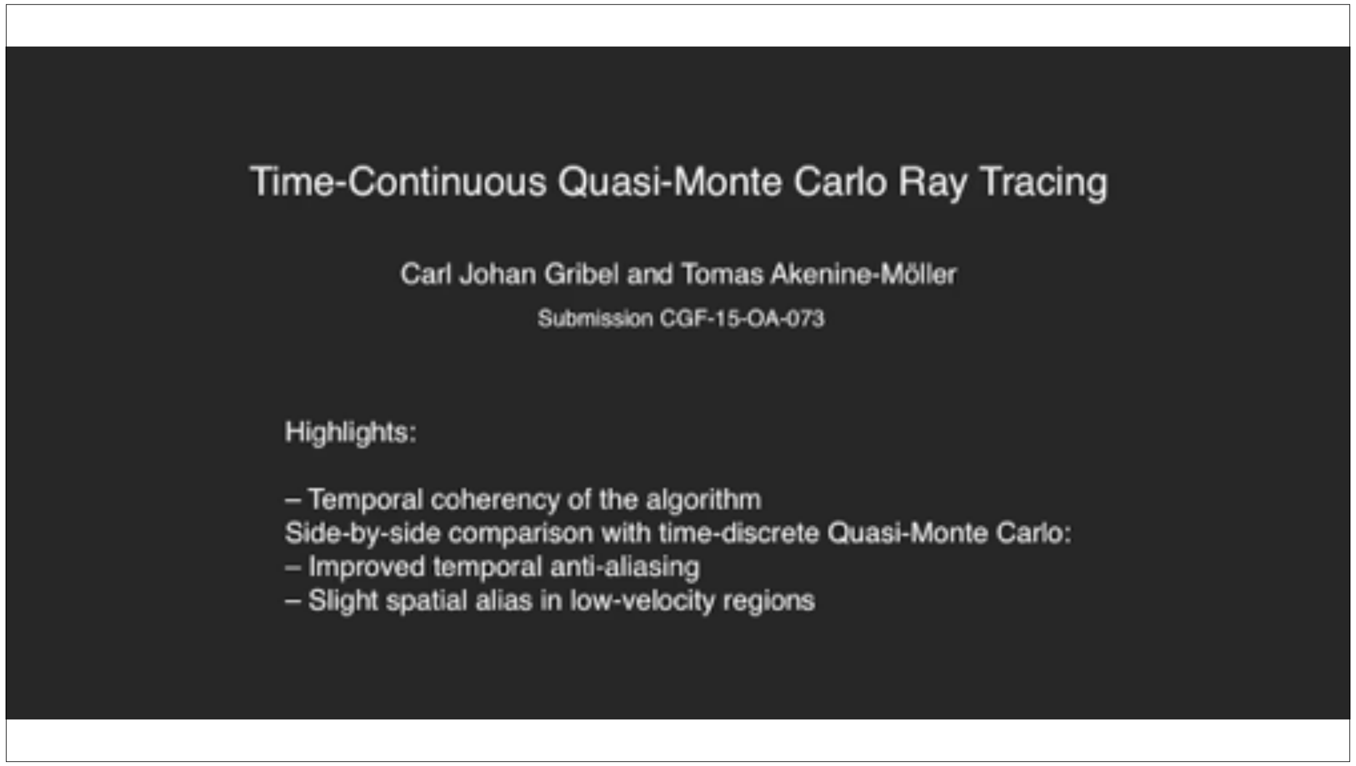
San Miguel: Guillermo M. Leal Llaguno

MALMÖ UNIVERSITY

LUND UNIVERSITY

intel

Backup



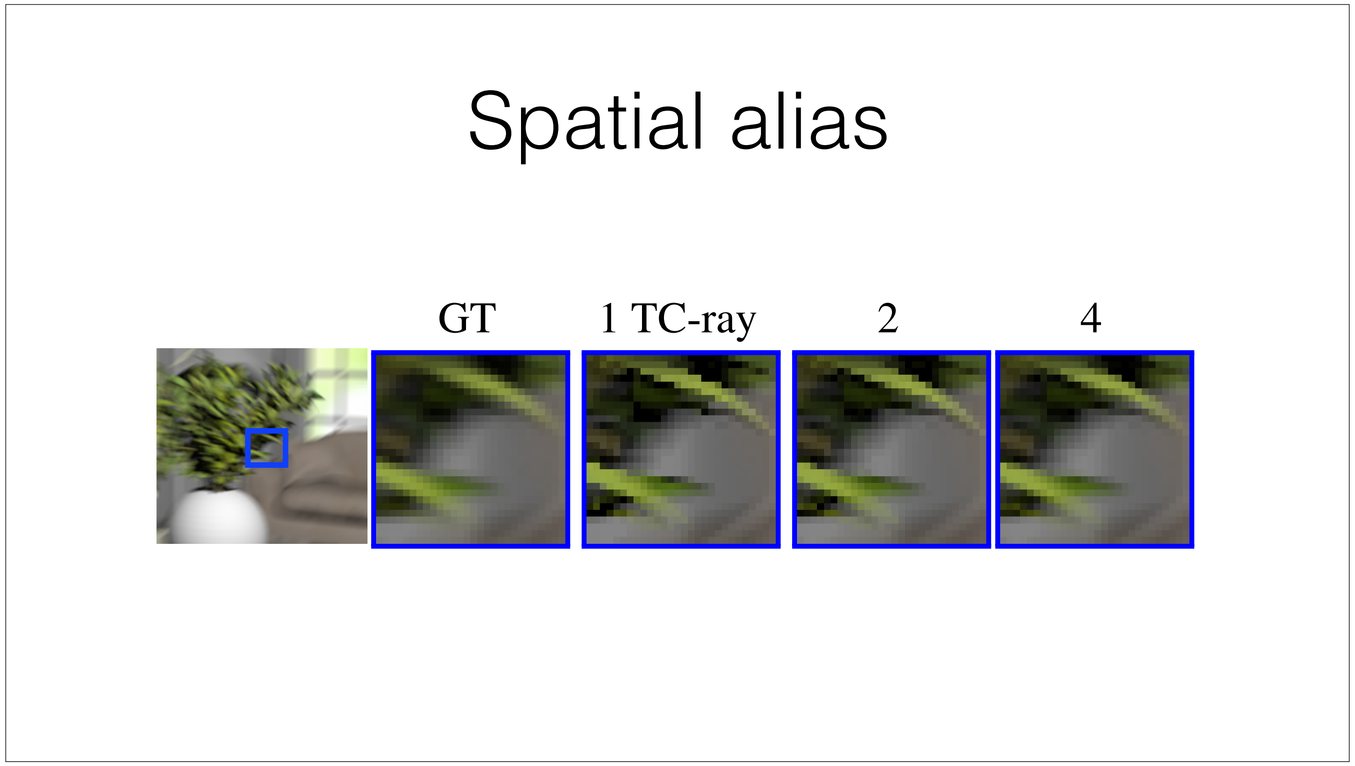
Time-Continuous Quasi-Monte Carlo Ray Tracing

Carl Johan Gribel and Tomas Akenine-Möller

Submission CGF-15-OA-073

Highlights:

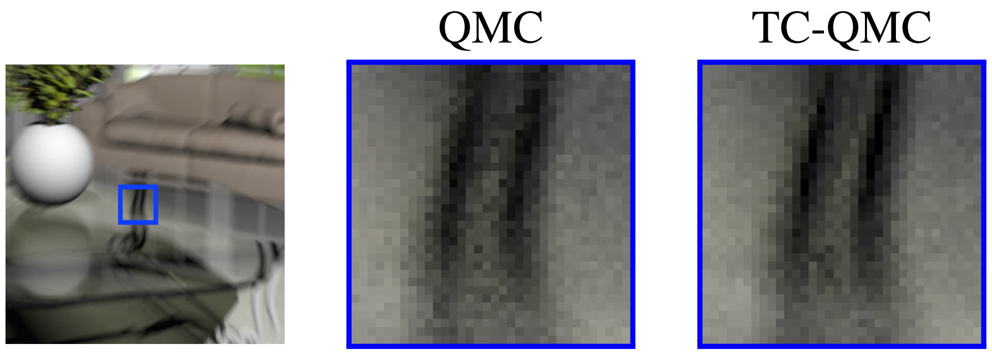
– Temporal coherency of the algorithm
Side-by-side comparison with time-discrete Quasi-Monte Carlo:
– Improved temporal anti-aliasing
– Slight spatial alias in low-velocity regions



Spatial alias

Another drawback is related to the way we space, i.e. each pixel.
One of the fundamental ideas is to use fewer, but more elaborate rays with respect to the temporal domain.
This is very effective against temporal alias,
but the lack of distribution in space shines through sometimes in the form of alias.
And especially on edges which are parallell to the direction of motion.
Here is an example of that, on a couple of leafs, also in Sala.
This alias of course diminishes when more time-continuous rays are used,
because then we introduce a greater distribution in space.

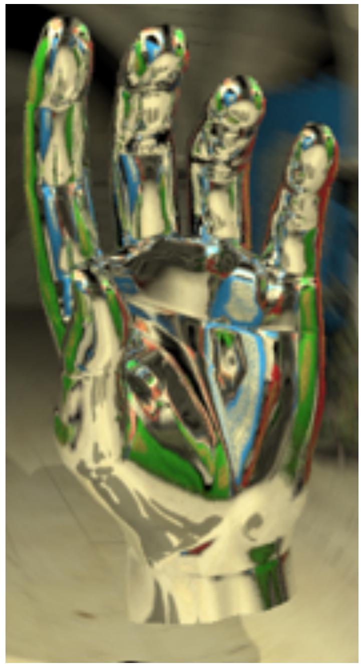
Secondary visibility



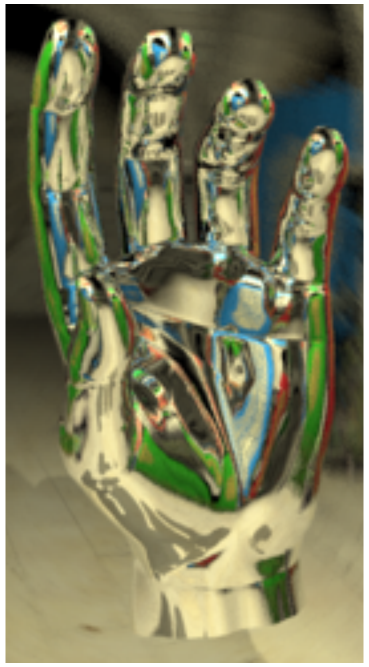
QMC

TC-QMC

This is a dielectric, reflective/refractive material,
which means that all colouring, all shading, comes from somewhere else,
from secondary rays.
These rays are, as of now, discrete, ordinary rays, they are not time-continuous.
This means everything we see reflected or refracted is, in effect,
point sampled, in terms of shading and visibility.



Our, N=32



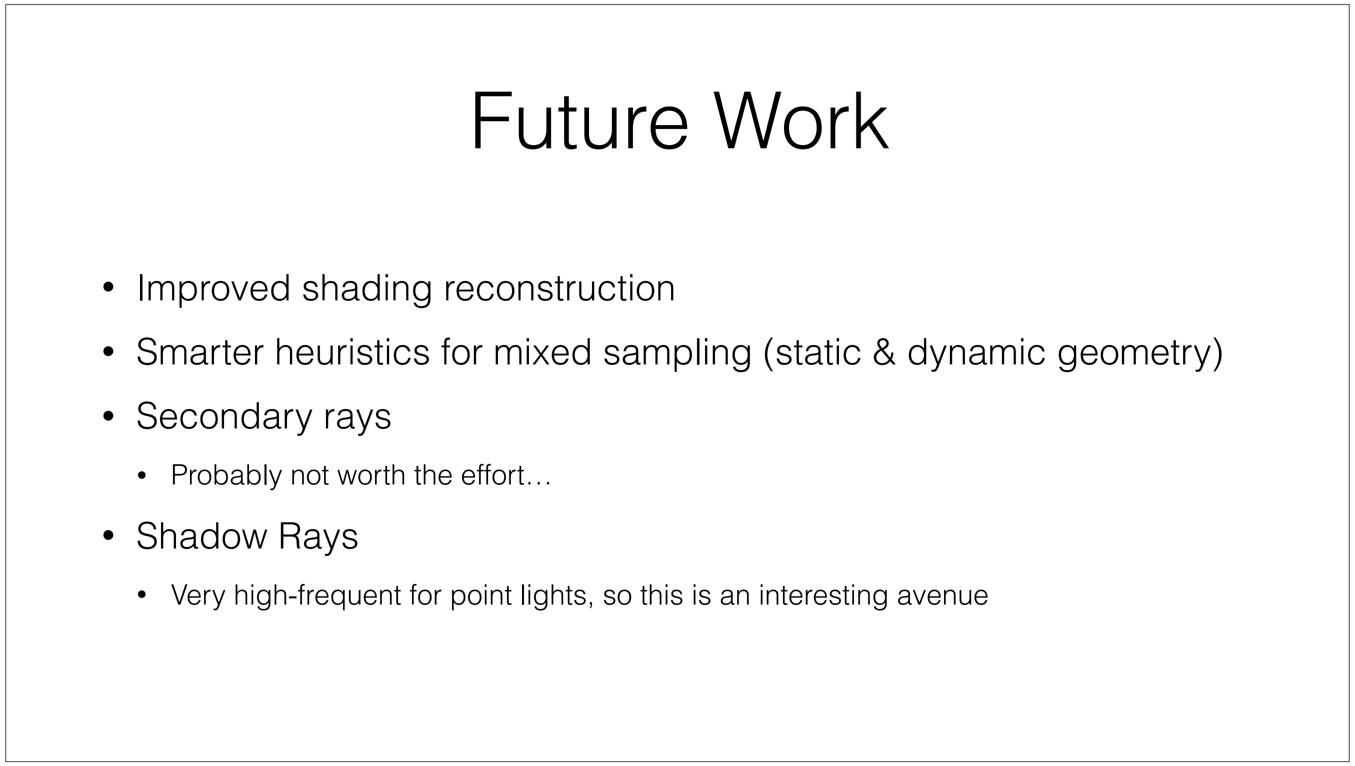
QMC, N = 32

Comparison setup: at a glance

- Our: **Time-Continuous Quasi-Monte Carlo (TC-QMC)**
  - 1, 2 or 4 TC-rays per pixel, *N* shading samples
  - TC-rays only at the primary level
- Reference: **Quasi-Monte Carlo (QMC)**
  - Stochastic sampling with *N* multi-jittered samples

- Shading Models: **Normal Shading, Whitted Ray Tracing**

- Presentation: **Quality as a function of rendering time** (growing *N*)
  - Quality metric: PSNR – Peak-Signal to Noise Ratio (dB)
  - Ground Truth: 1024-2048 spp Quasi-Monte Carlo



Future Work

- Improved shading reconstruction
- Smarter heuristics for mixed sampling (static & dynamic geometry)
- Secondary rays
  - Probably not worth the effort…
- Shadow Rays
  - Very high-frequent for point lights, so this is an interesting avenue

However, this work is by all means preliminary and there is much room for improvement.
For all its benefits, continuity-based integration leads to more expensive intersections tests, higher memory requirements and

Our current heuristic for mixed sampling, i.e. for when to employ regular point sampling and when to switch to time-continuous samlpling is rather crude and could be improved in order to make a more qualified decision for when to use either algorithm.

Another area for improvement is shading -- how to treat shading, which is and probably will remain non-continuous, over a set of continuous visibility data.
Our proposed C1-continuity-based filtering kernel improves convergence, but there is most likely a lot of room for improvement here.

Yet more avenues for future work include treating other domains as continua instead of, or in combination with, the proposed domain, that is time.
Such domains could be space, one- or two-dimensional space, lens, for depth-of-field, and so on.
So let's get to work.