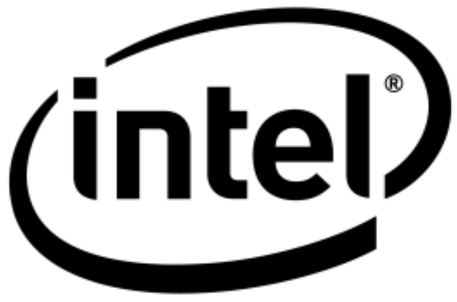
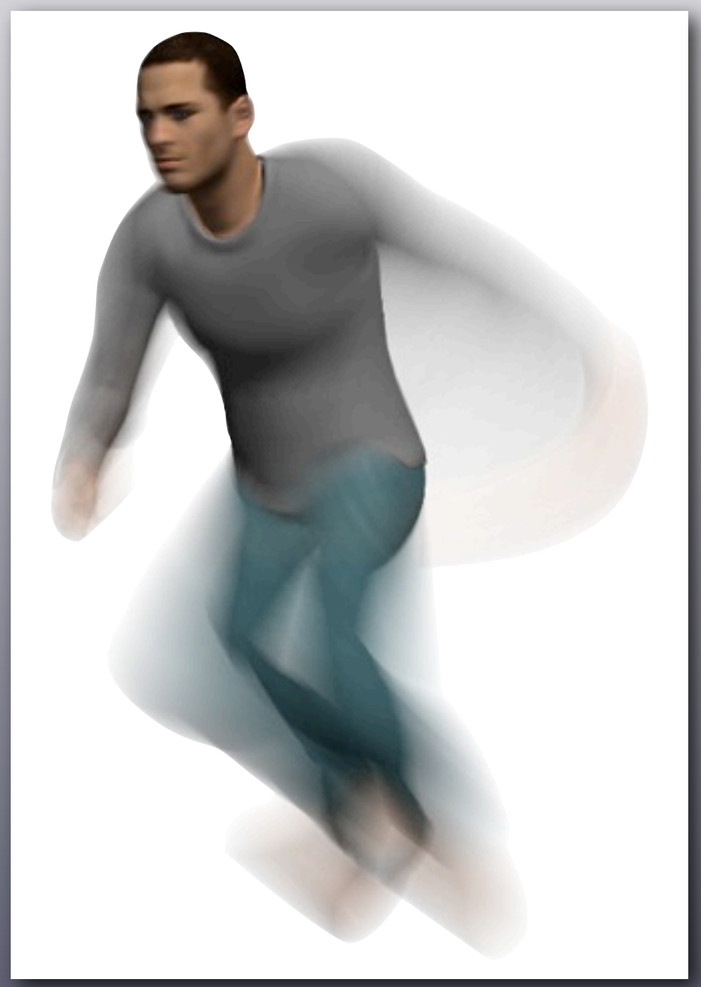
Theory and Analysis of Higher-order Motion Blur Rasterization

Carl Johan Gribel[1] Jacob Munkberg[2] Jon Hasselgren[2] Tomas Akenine-Möller[1,2]

[1]Lund University [2]Intel Corporation

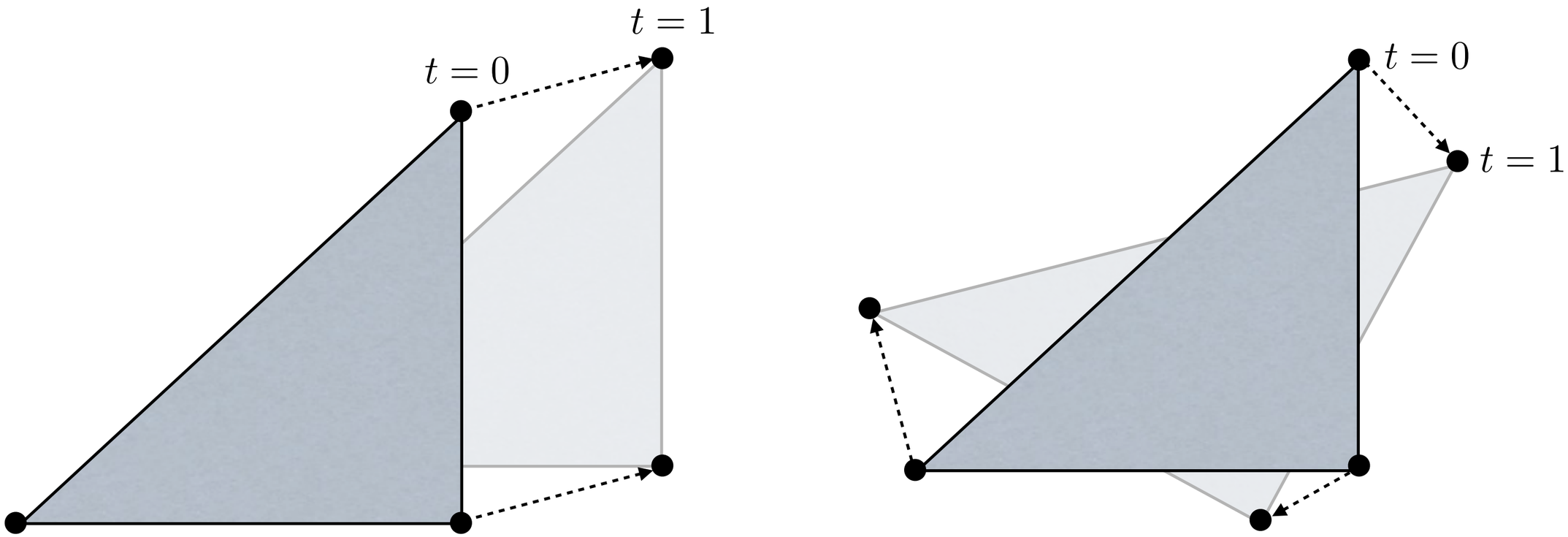




As suggested by the title, this paper investigates some of the theoretical and practical implications of high-order motion in temporal rendering.

It is pretty well established by now that temporal rendering, or motion blur, is an important effect in many applications of rendering.
Motion blur provides an intuition about velocity, direction and overall time-dependency of objects in a scene.

Per-vertex linear motion

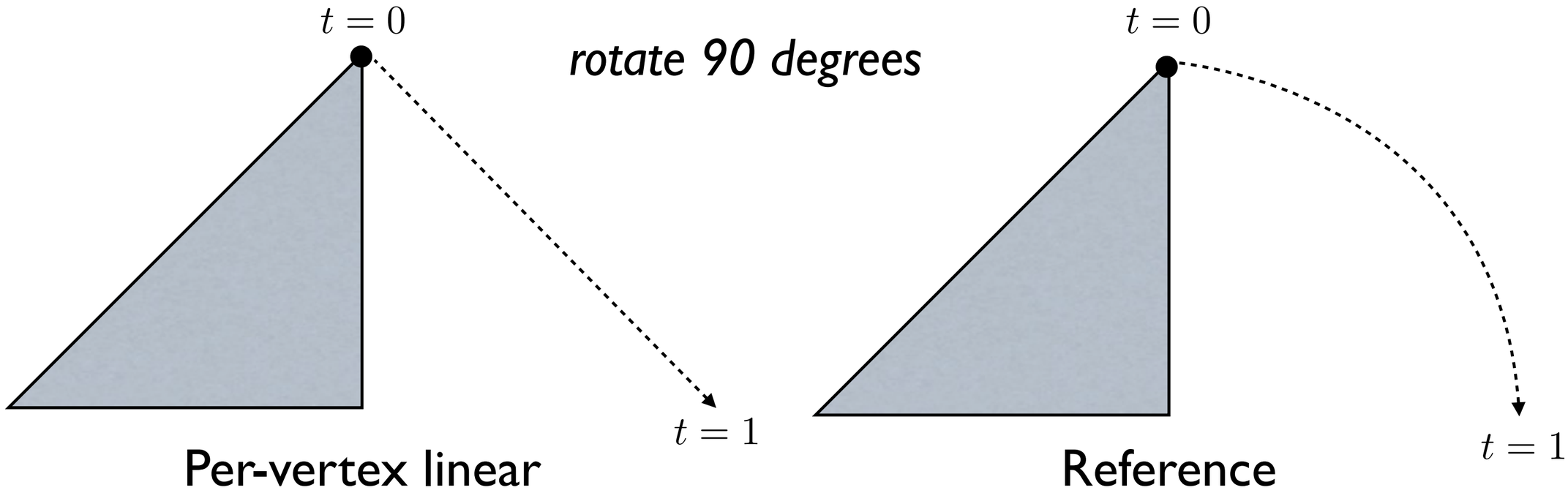


The timeframe over which an image is rendered is often associated with the shutter interval of a camera.
They way in which motion is represented during this timeframe has a big impact on rasterization.

The pervading assumption is that motion is approximatively per-vertex linear during the shutter interval.
This makes rasterization simple, and since the shutter interval is usually short in practical settings, say one 24:th of a second, such an assumption seems reasonable.

But even though per-vertex linear motion is enough to approximate most types of object movement,
it is easy to come up with exceptions.

Linear vs curved motion

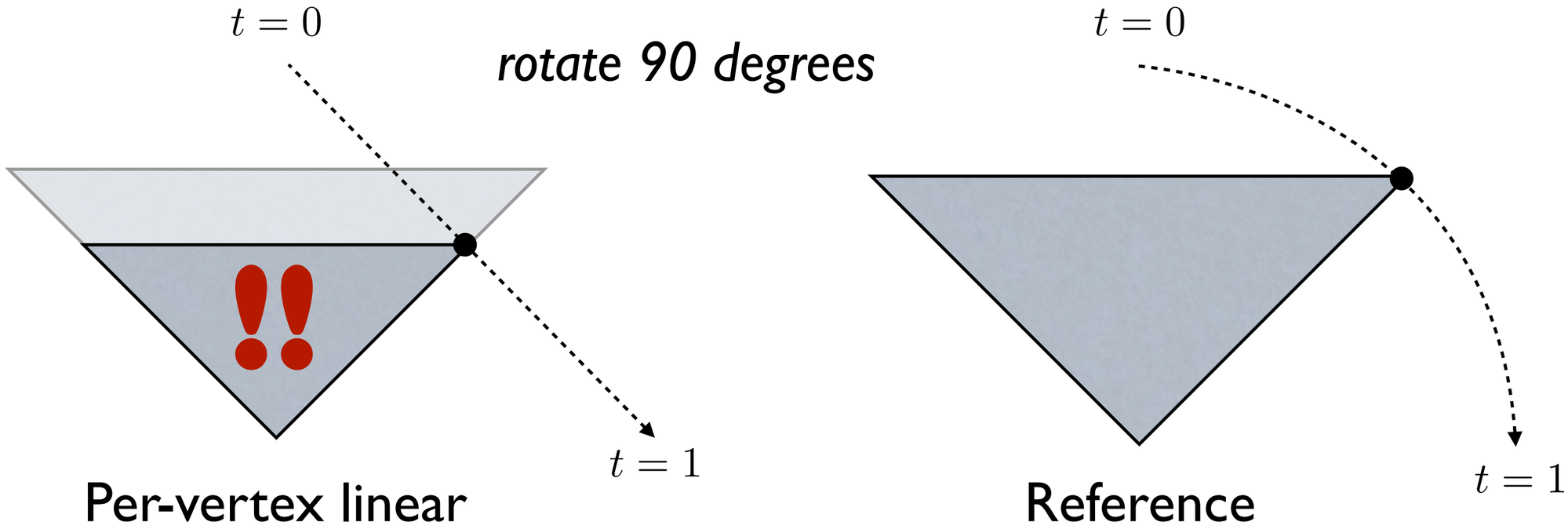


Consider a triangle that rotates 90 degrees.

When we compare linear with the reference curved motion …

… it is obvious that something goes wrong.
The triangle becomes skewed because the actual vertex paths are circular and not linear.

Linear vs curved motion

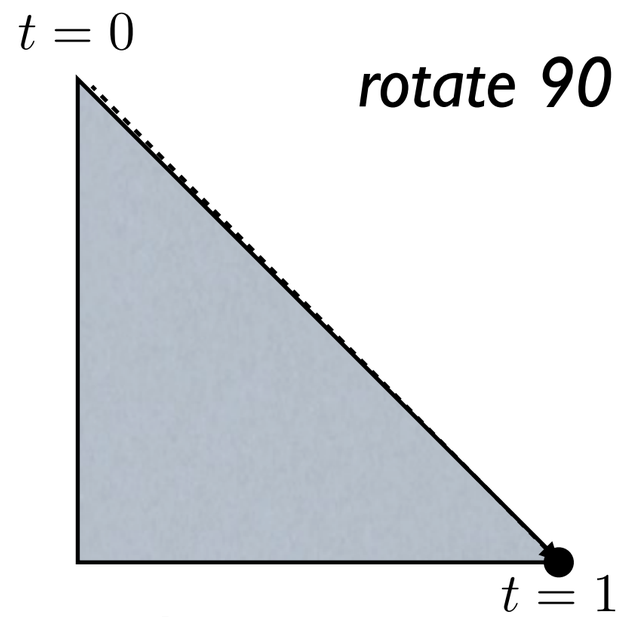


Consider a triangle that rotates 90 degrees.

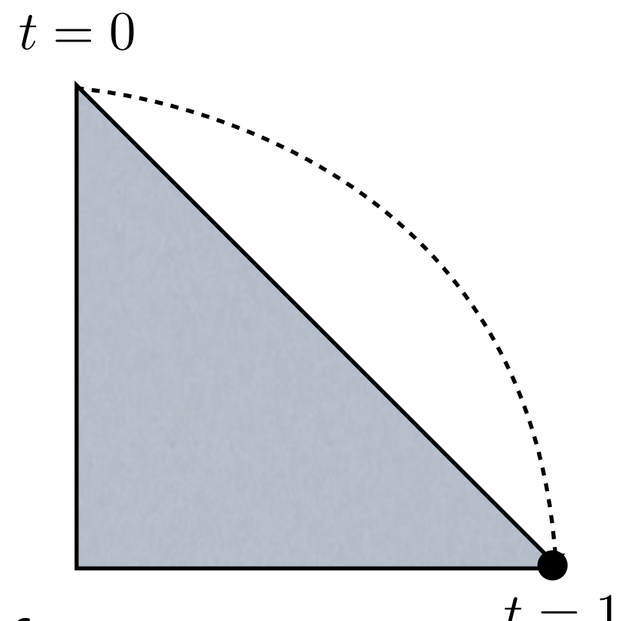
When we compare linear with the reference curved motion …

… it is obvious that something goes wrong.
The triangle becomes skewed because the actual vertex paths are circular and not linear.

Linear vs curved motion



rotate 90 degrees



Per-vertex linear

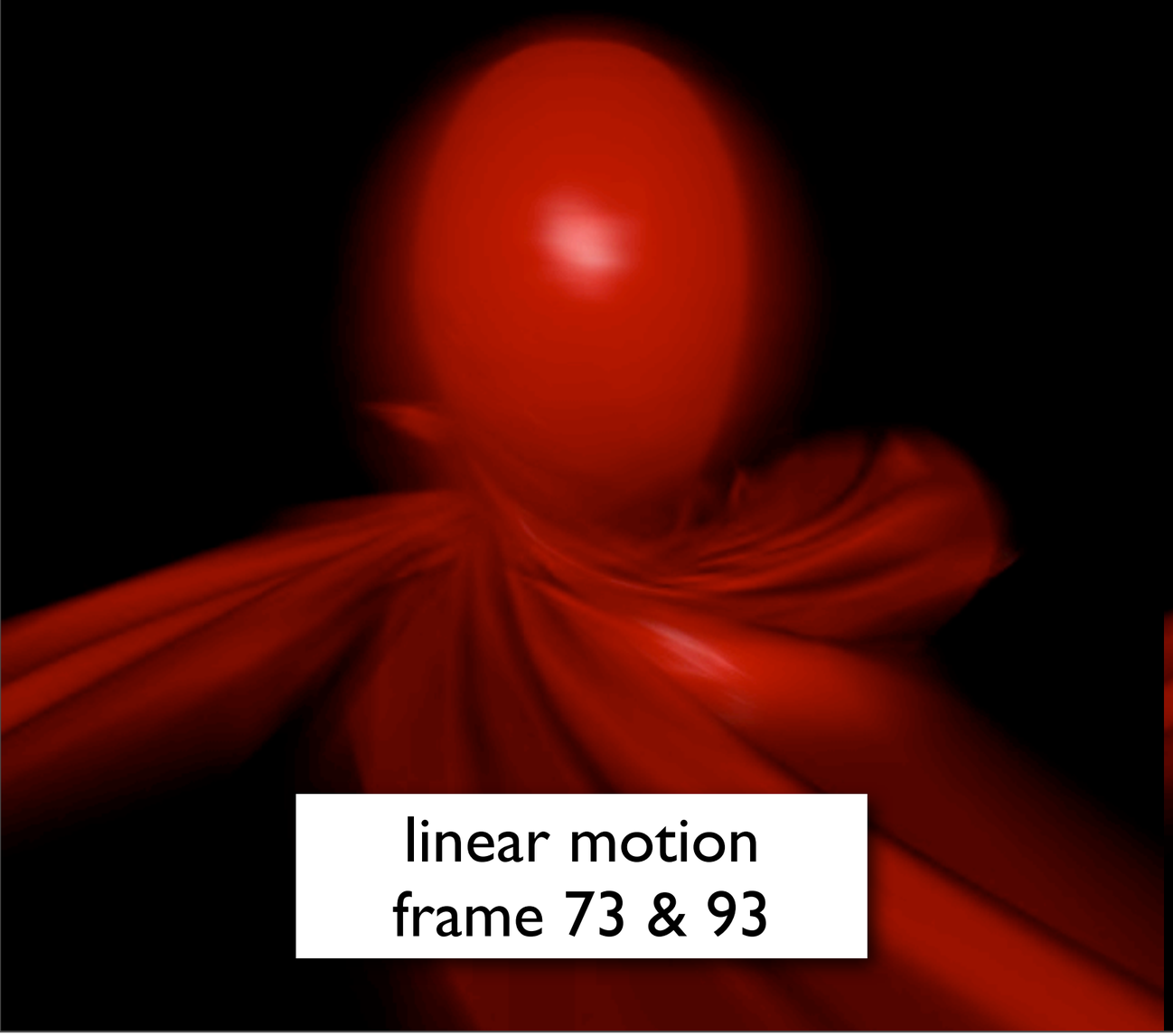
Reference

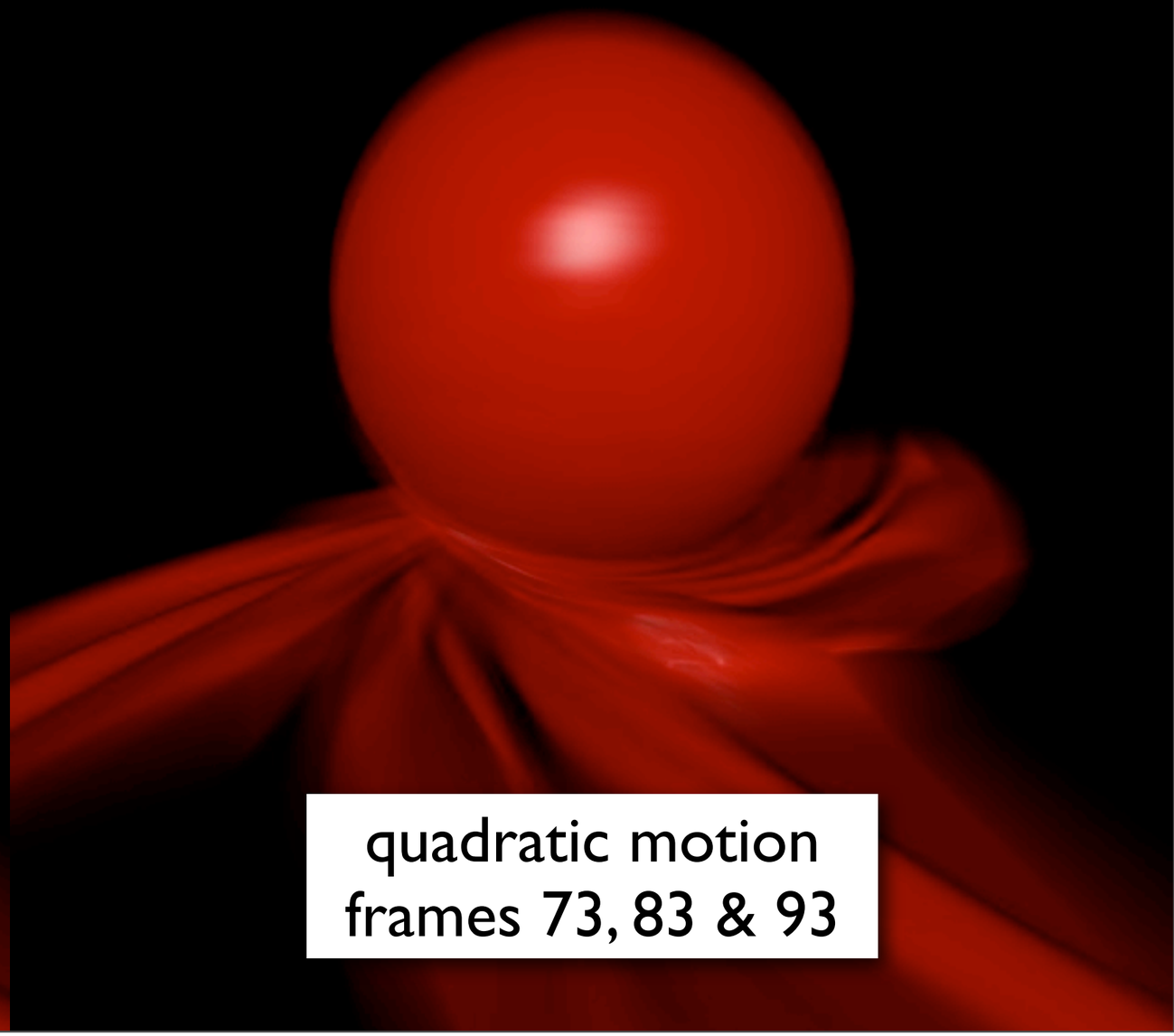
Consider a triangle that rotates 90 degrees.

When we compare linear with the reference curved motion …

… it is obvious that something goes wrong.
The triangle becomes skewed because the actual vertex paths are circular and not linear.

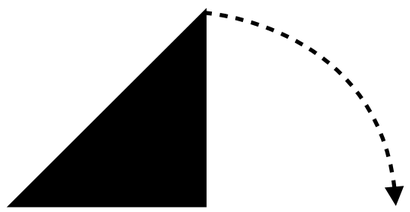
Linear vs curved motion

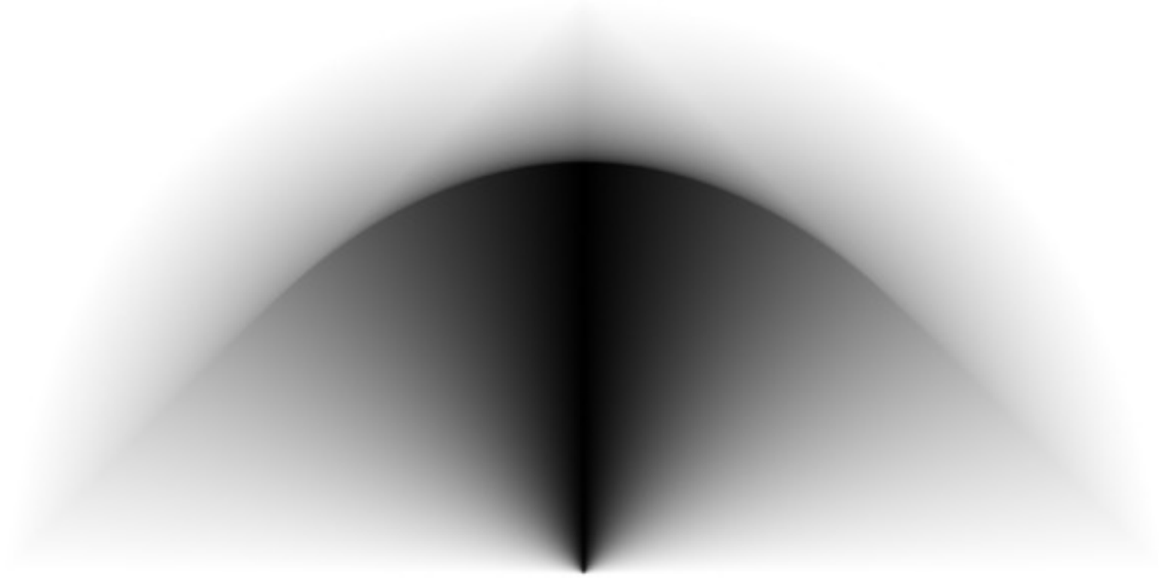




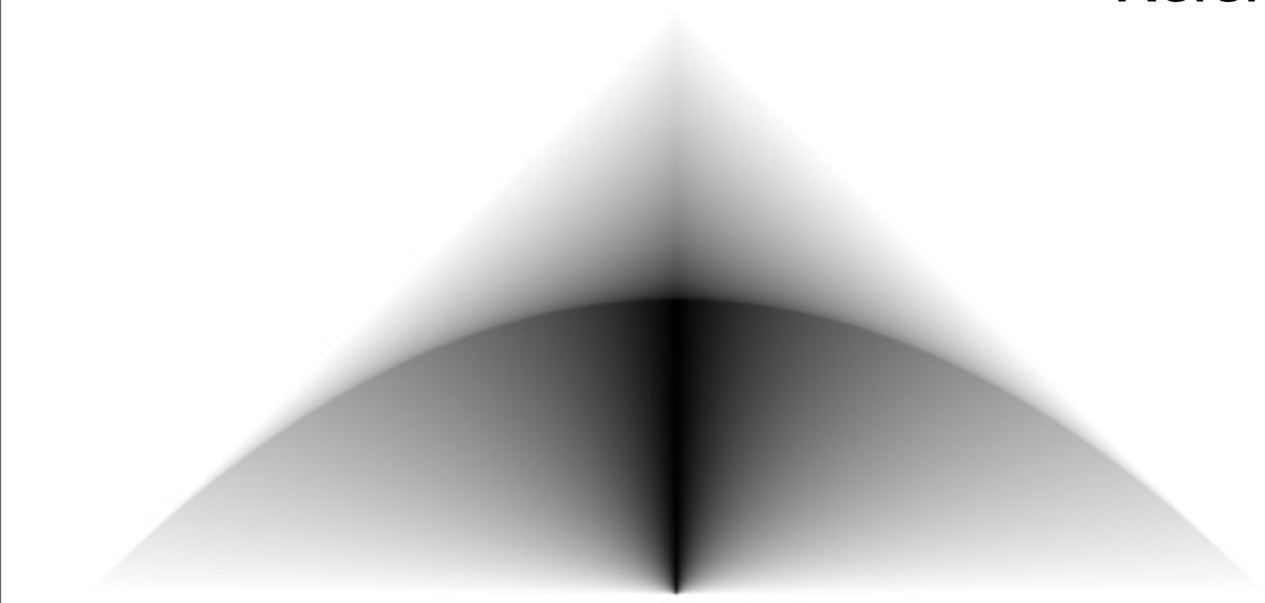
This rendering from the clothball scene displays the same artifact.
As the cloth is spun rapidly onto the sphere, it gets distorted due to linearization.

The right rendering is created by our new rasterizer that can handle curved motion.
It uses three frames instead of two as control locations,
and each vertex path is set up as a quadratic Bézier curve.

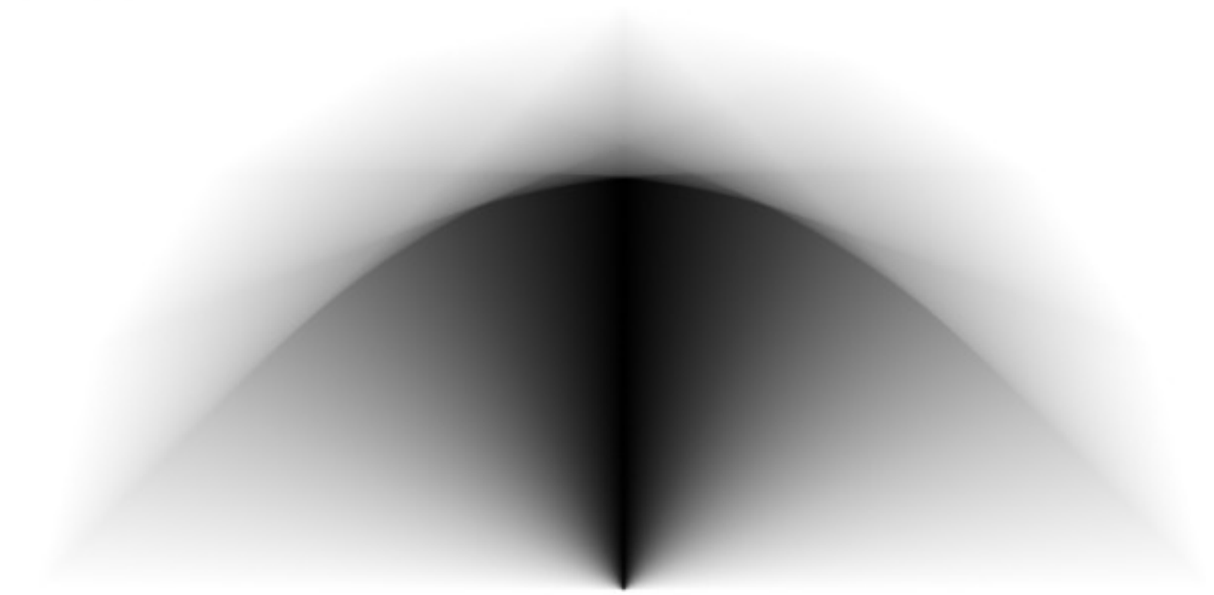




Reference



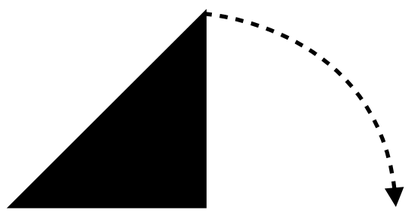
Linear motion



4x Linear

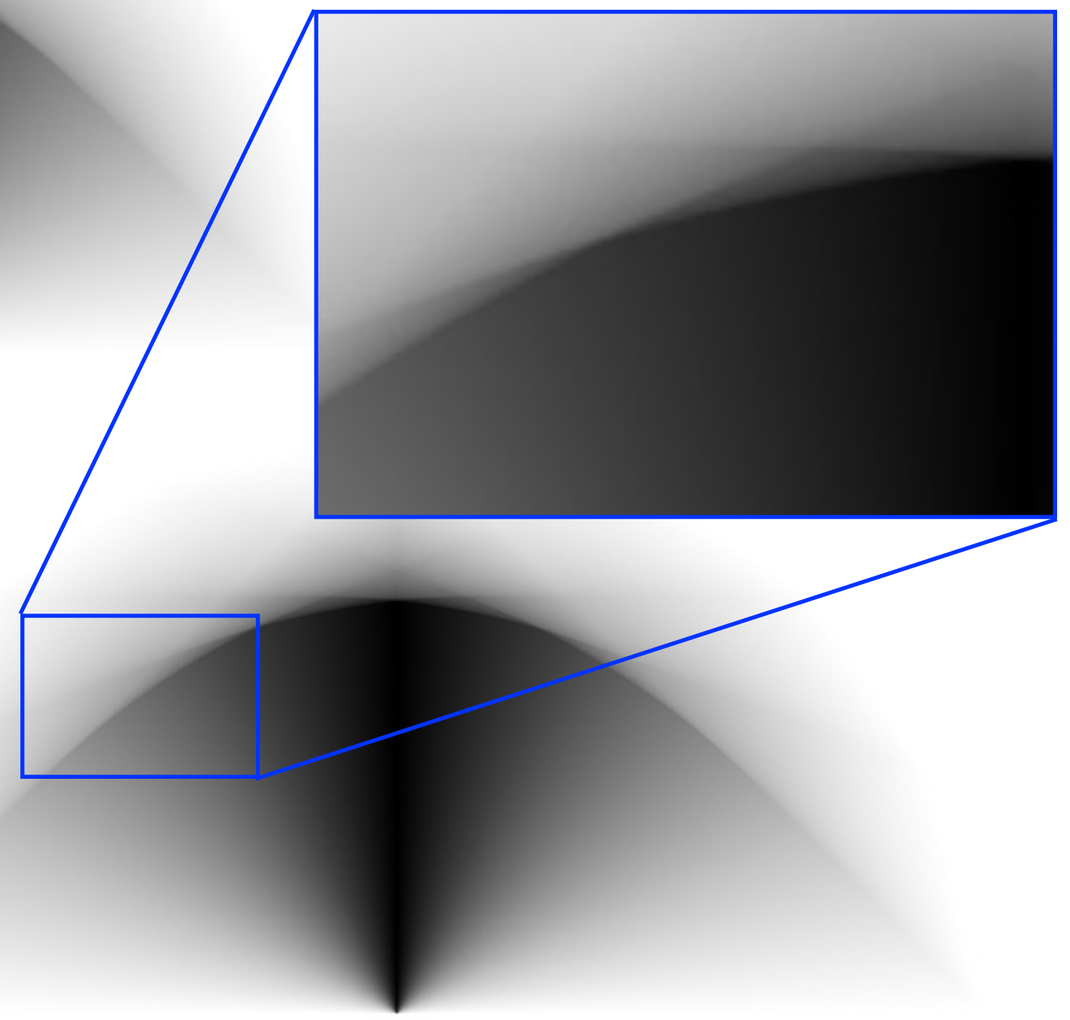
But what about just using multiple linear segments, like Renderman does for instance?

The rendering at the bottom right, which uses four linear segments instead of one,
IS an improvement compared to the reference ...



Reference

Linear motion



4x Linear

… but upon looking closer, we still see small scale linearization artifacts.
With even more segments, these artifacts will eventually be eliminated, but how many splits are needed? And how is this decided?
Splitting and maintaining segments this way is not necessarily neither trivial nor practical.

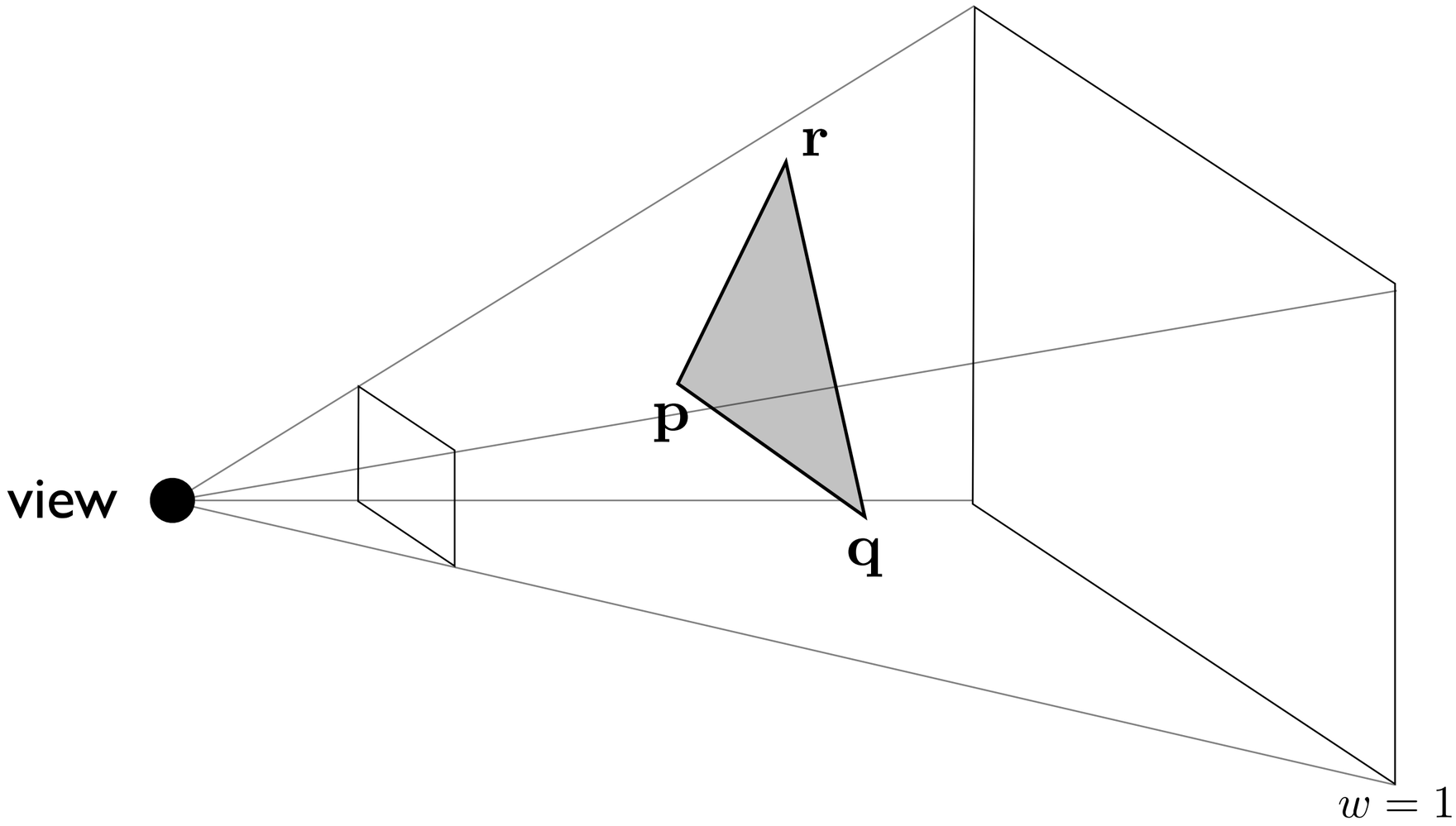
So, while it is true that per-vertex linear motion will suffice i most normal cases,
our scope is to consider those cases with aggressive, rotational or non-linear motion,
where this assumption breaks down and a high-order representation becomes a plausibility.

Overview

- Generalized edge equations for higher-order motion
  - Efficient traversal for higher-order motion
    - Use-case: semi-analytical rasterizer

My talk will follow these three main points:
i will derive edge equations for generalized motion,
then talk about algorithms that make traversal more effective under these settings,
and then talk a little bit about an additional, semi-analytical rasterizer.

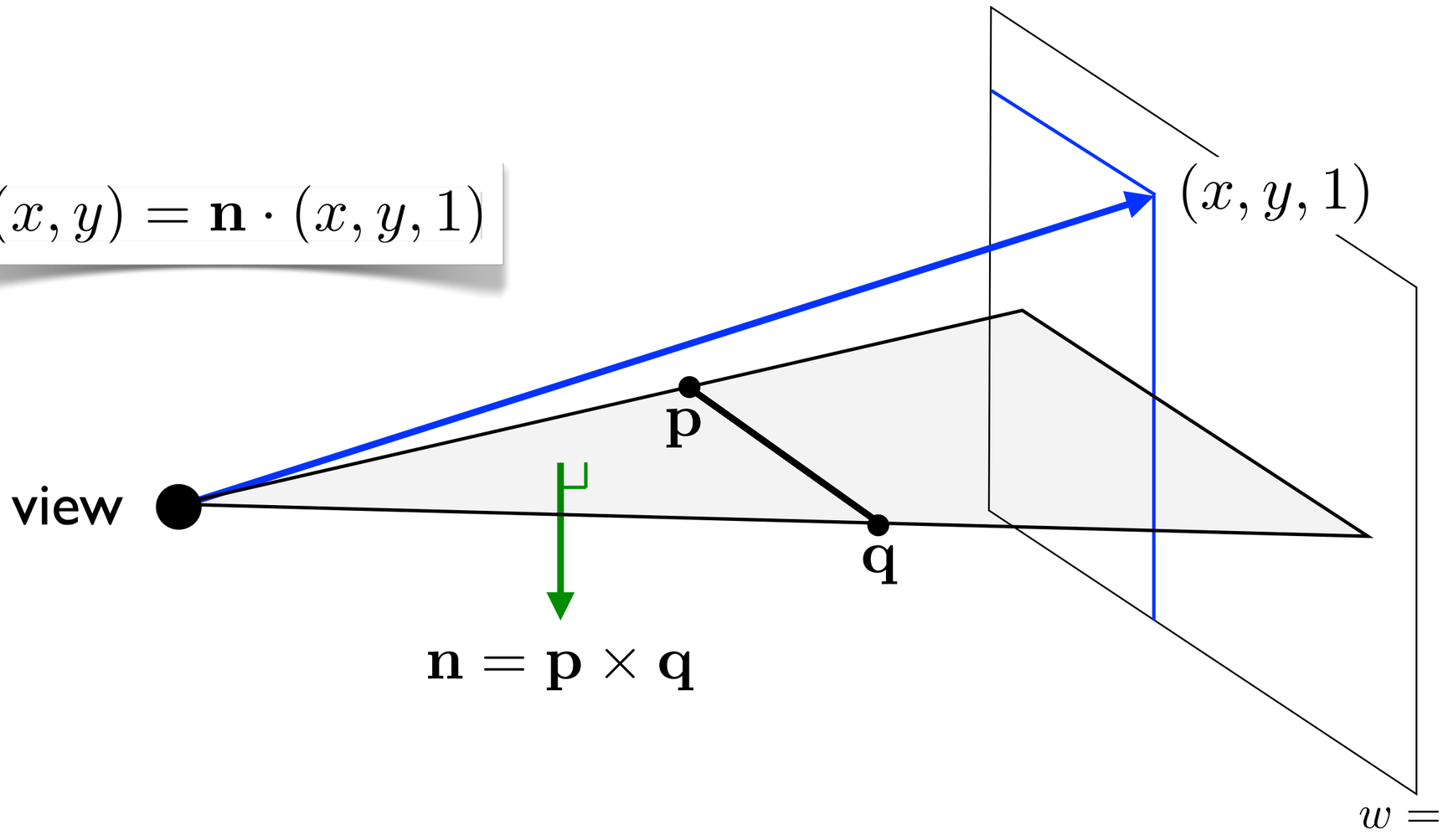
Edge Equations: static triangle



To derive the generalized edge equations, we start with a static triangle in homogEneous space.

Edge Equations: static triangle

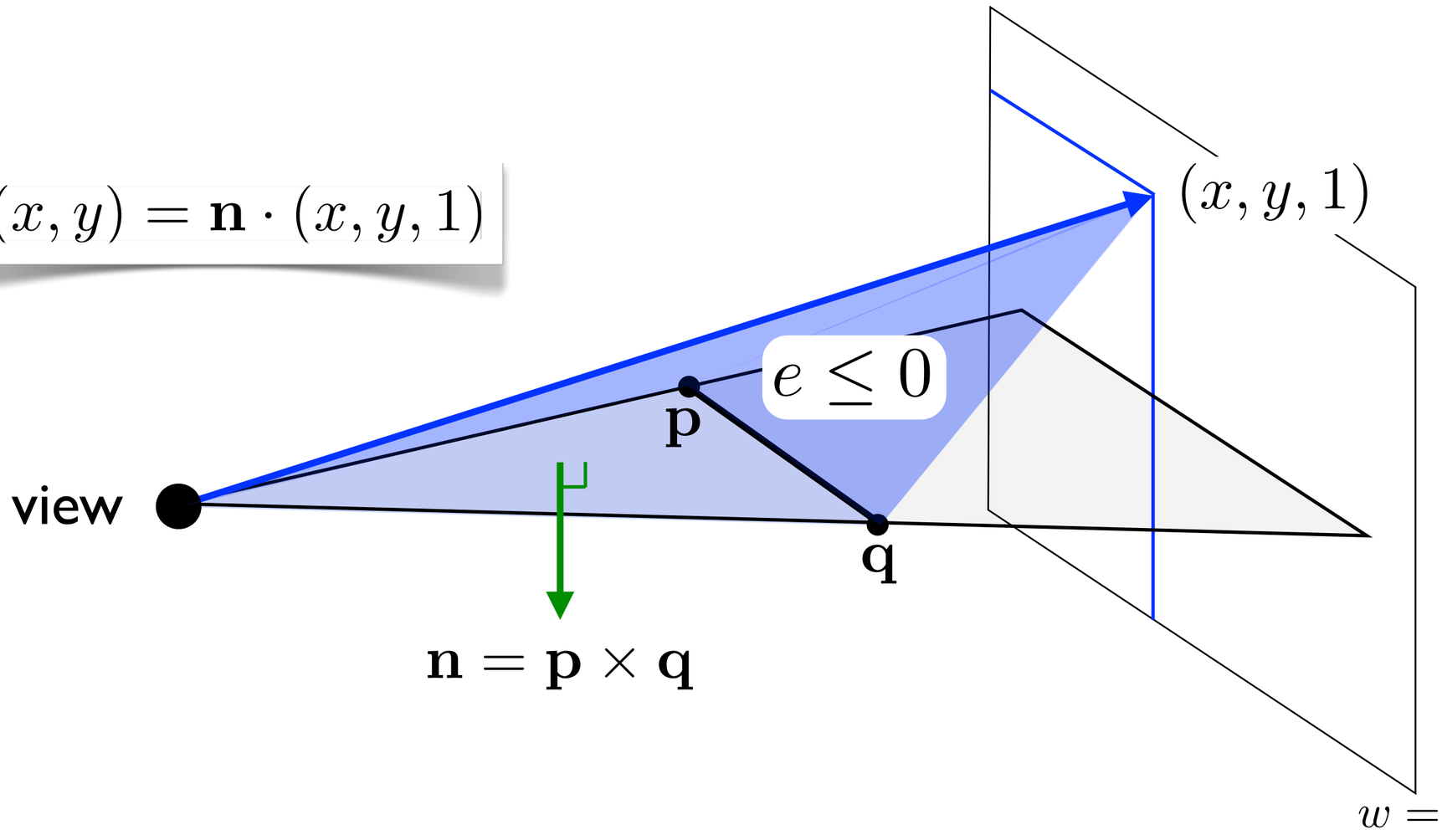
$$e(x, y) = \mathbf{n} \cdot (x, y, 1)$$



The edge equation for the bottom edge is a function of the blue and green vectors.
The blue vector represents the current sample point,
and the green vector can be interpreted as the normal to the plane that spans from the view point and through the edge.

Edge Equations: static triangle

$$e(x, y) = \mathbf{n} \cdot (x, y, 1)$$

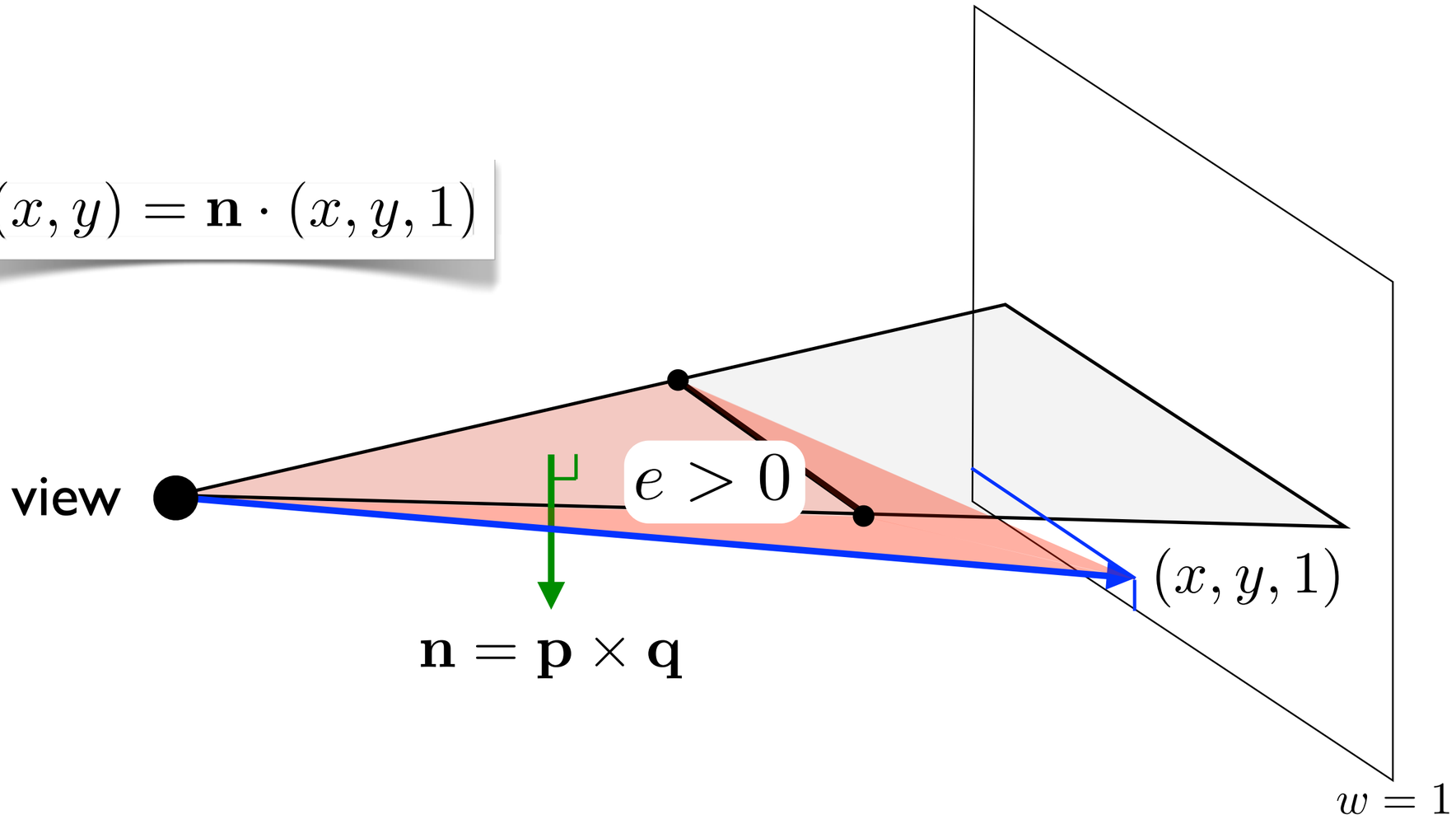


The value of the edge equation can be interpreted as the volume of the polyhedra enclosed by the view point, edge and sample point.

If this volume is negative, as it is here, the sample is considered to lie inside the edge.

Edge Equations: static triangle

$$e(x, y) = \mathbf{n} \cdot (x, y, 1)$$

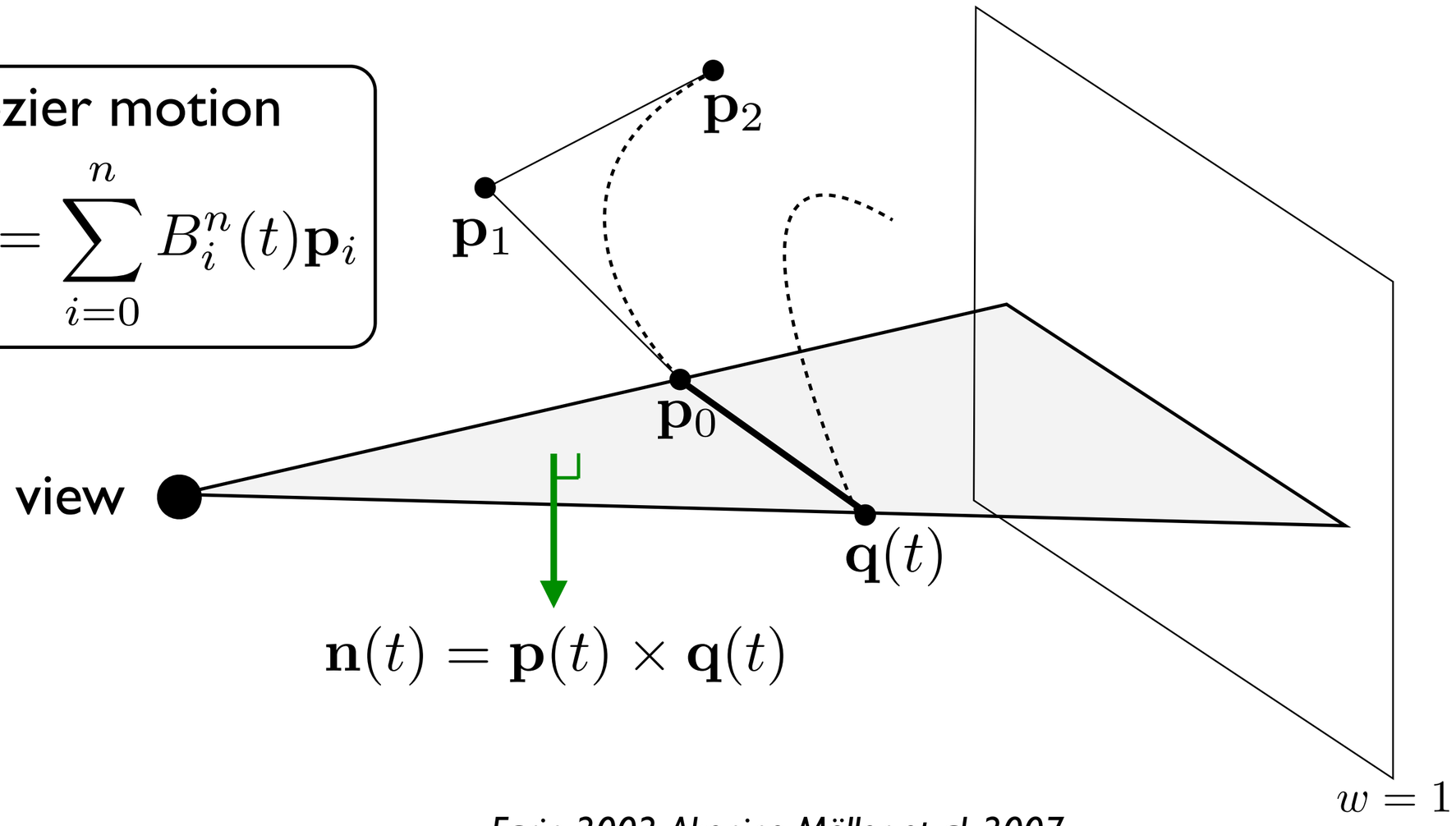


If it's positive, like here, the sample is considered to lie outside the edge.
If this happens, the coverage test has failed and no more edges needs to be tested against.
If all three edge equations are negative however, the sample lies inside the triangle.

Edge Equations: moving triangle

Bézier motion

$$\mathbf{p}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i$$



Farin 2002, Akenine-Möller et al. 2007

For a moving triangle the vertices have a time dependency.
As stated in the beginning, the prevailing assumption is that this dependency is linear.

Our first contribution is to consider a general representation of motion based on Bézier curves.
This way, motion is defined from an arbitrary number of spatial control points,
and can be of any complexity.

Generalized Edge Equation

$$e(x, y, t) = \sum_{k=0}^{2n} B_k^{2n}(t)\mathbf{c}_k$$

- Bernstein polynomial of order 2*n*
- 2*n*+1 scalar control points
- *n* = order of motion

with control points:

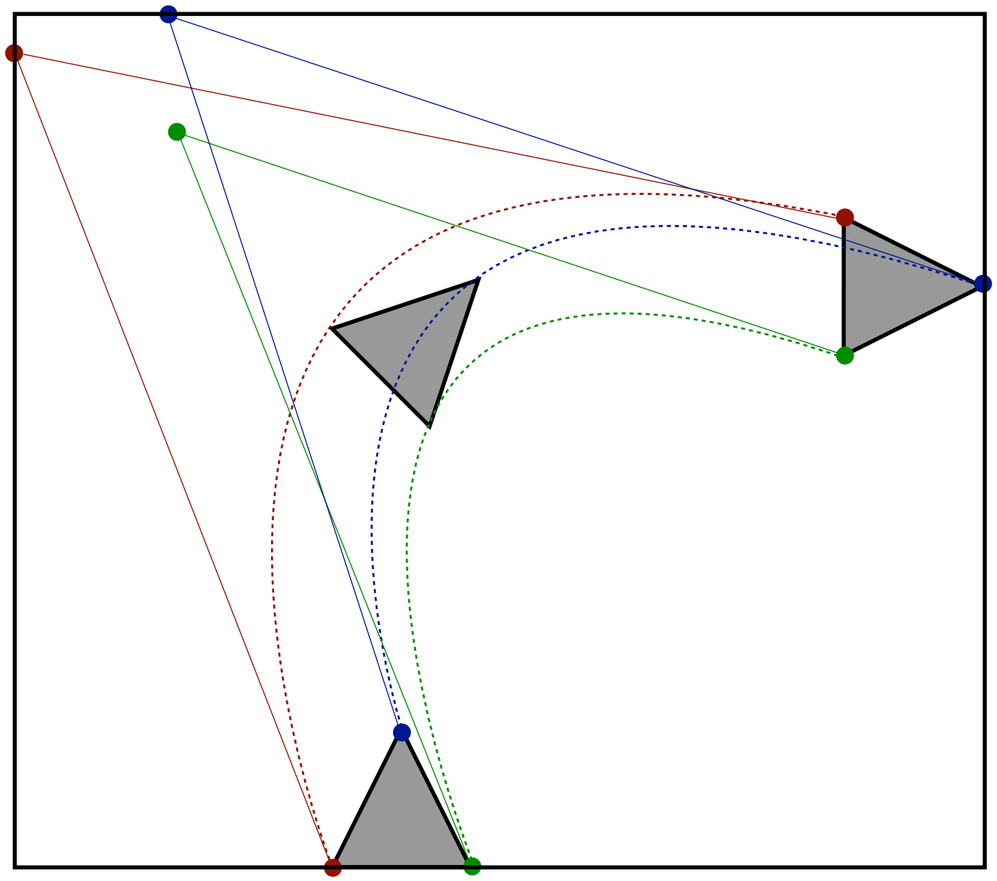
$$\mathbf{c}_k = \Big(\sum_{i+j=k} \frac{\binom{n}{i}\binom{n}{j}}{\binom{2n}{i+j}} \mathbf{p}_i \times \mathbf{q}_j \Big) \cdot (x, y, 1)$$

The resulting, generalized edge equation takes the following form.
For a full derivation I refer you to the paper.

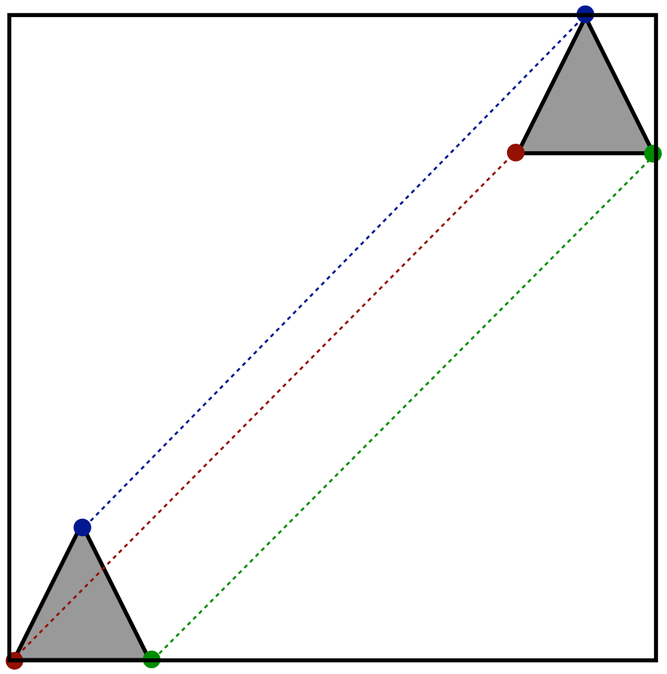
It may look a bit nasty, but the important observation is that is really just a Bernstein polynomial
of twice the motion order.
For linear motion, they thus become quadratic, which has been shown before.
For quadratic motion, which we will use in our test scenes, they are of order four.
That means, for instance, that a sample point may enter or exit an edge up to four times during one timeframe.

Traversal

Quadratic motion



Linear motion



Laine et al. 2011
Munkberg et al. 2011
Fatahalian et al. 2009
INTERVAL: *Pixar*

Now with edge equations for high-order motion in place, let's proceed with rasterization.

To rasterize efficiently, it is important to limit the number of coverage tests as much as possible.

Note, in the left figure for quadratic motion, how large the bounding box of the control points can become.
Clearly, a lot of samples will be wasted unless we use them carefully.

Traversal

- Generalized INTERVAL
- Tiled traversal: TILE
  - Triangle – Tile
  - Edge – Tile

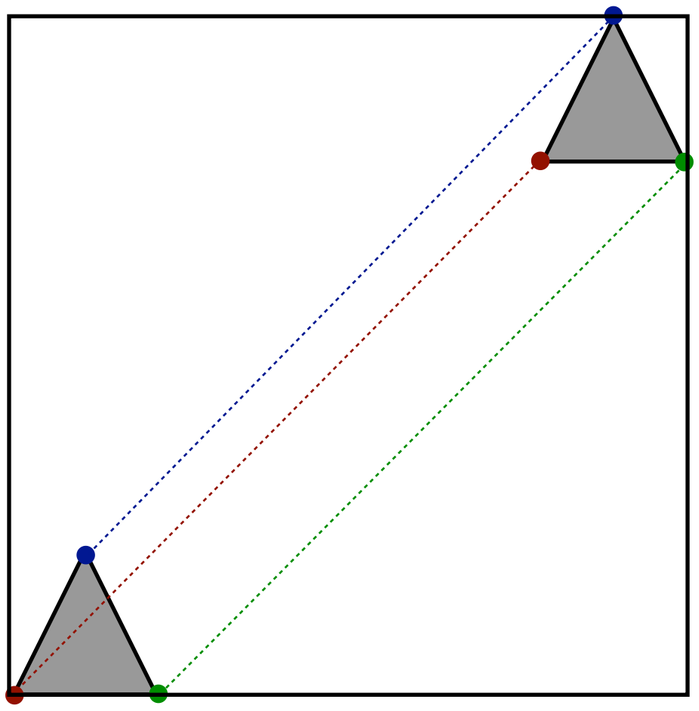
We present two algorithms which improve traversal for high-order motion.
The first algorithm is a generalization of the Pixar algorithm INTERVAL,
in which time is split into a number of subintervals.

We then present two tile tests for how to either cull tiles entirely,
or compute conservative temporal bounds for the intersection.

Traversal: INTERVAL

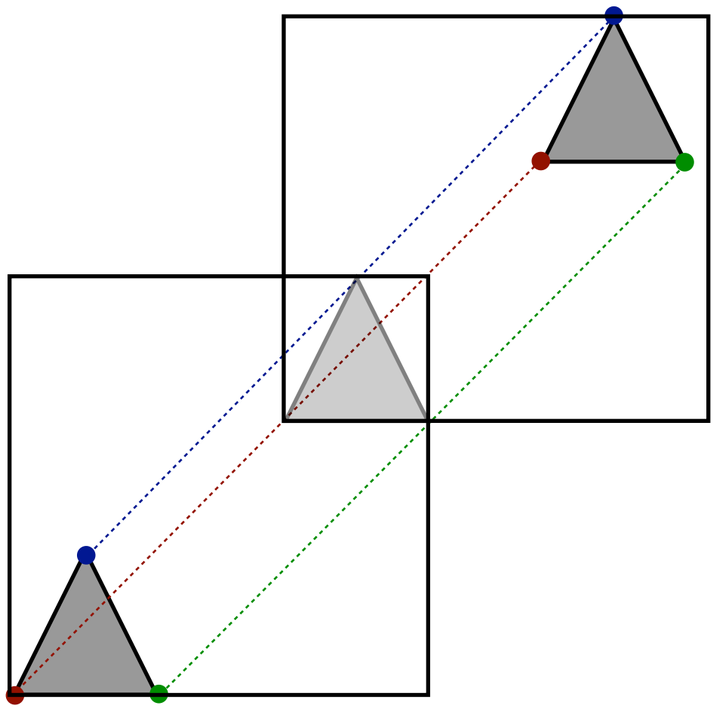
one interval

$[0, 1]$



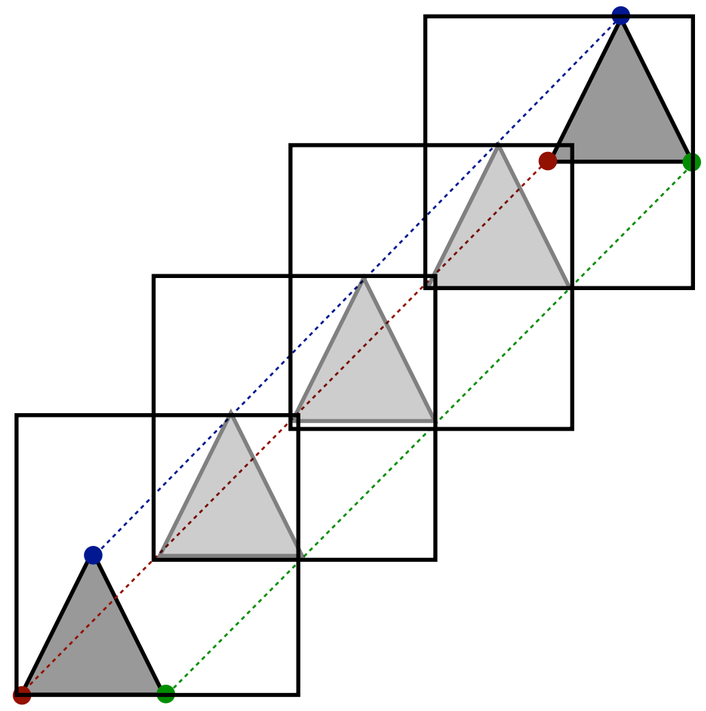
2 Intervals

$[0, 0.5],\ [0.5, 1]$



4 Intervals

$[0, 0.25],\ [0.25, 0.5],\ [0.5, 0.75],\ [0.75, 1]$



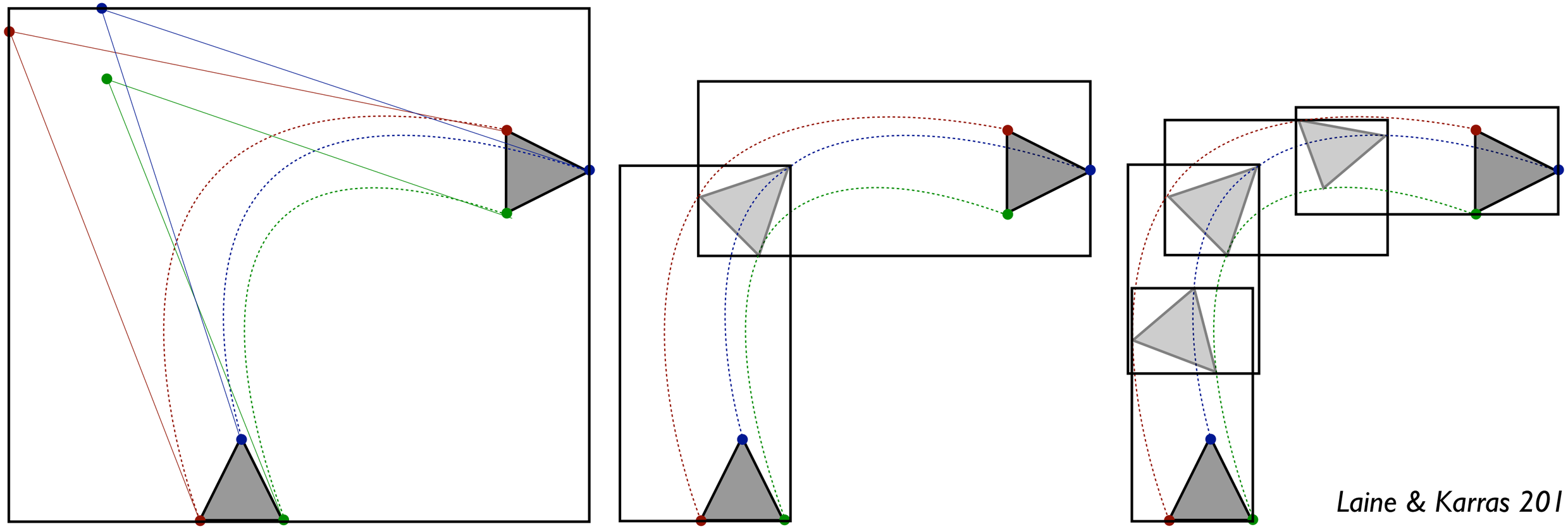
The INTERVAL algorithm works by splitting time uniformly,
and then create multiple bounds for the moving triangle within each subinterval.
These new bounds are typically much smaller than the original bound,
making traversal more effective.

Traversal: generalized INTERVAL

one interval
$[0, 1]$

2 intervals
$[0, 0.5],\ [0.5, 1]$

4 intervals
$[0, 0.25],\ [0.25, 0.5],\ [0.5, 0.75],\ [0.75, 1]$



Laine & Karras 2011

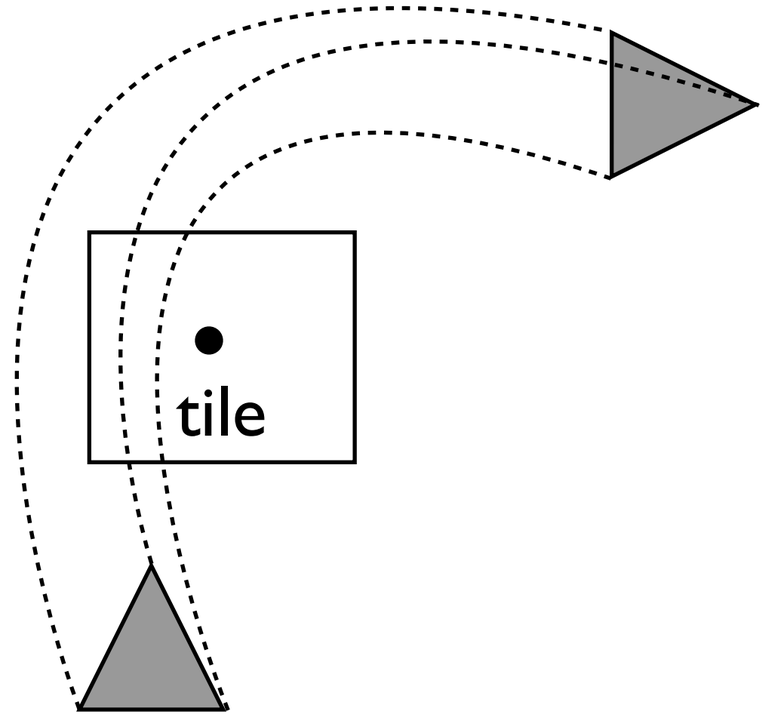
We generalize this algorithm to high-order motion by reparAmeterizing the motion curve to each subinterval.
We then use the Laine Karras algorithm to compute screen-space bounds from the control points of the curves from each subinterval.

As more intervals are used, the motion within each interval becomes less and less curved,
and the spatial bounds tighter and tighter.

The most efficient way to rasterize is then to position the triangle at the point in time associated with each sample,
and then use the corresponding local bounds for the spatial samples.

TILE: tiled traversal

- Triangle – Tile
- Edge – Tile

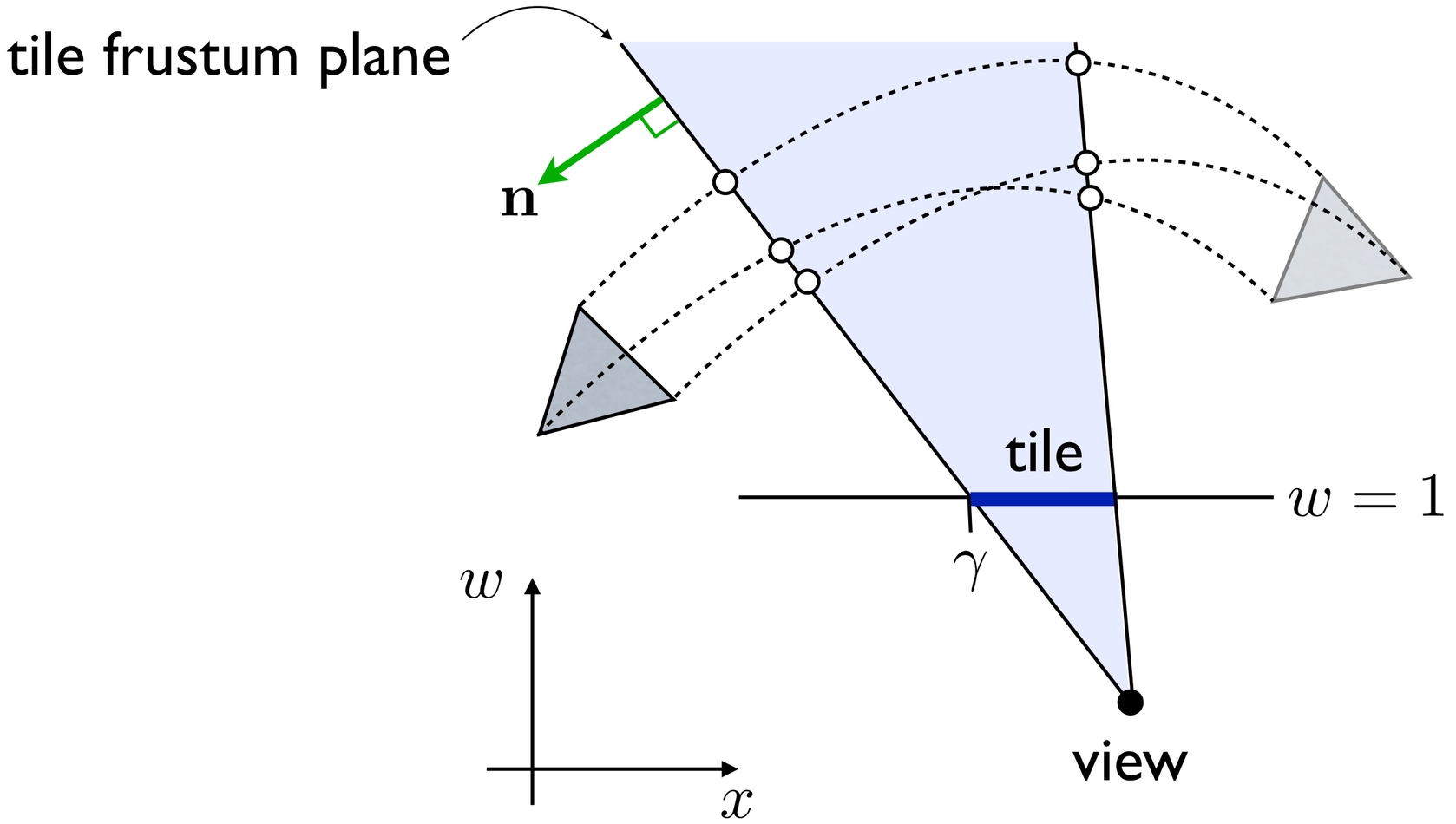


In tiled rasterization, the triangle is tested against a screen-space rectangle of some size.

The objective here is to either cull the tile entirely,
if we can rule out that it intersects any part of the moving triangle,
or come up with conservative, temporal bounds for the intersection.

These temporal bounds are then used to discard coverage tests that fall outside of them.
We combine two tests for this purpose, one for triangles and on for individual edges.

Triangle – Tile test

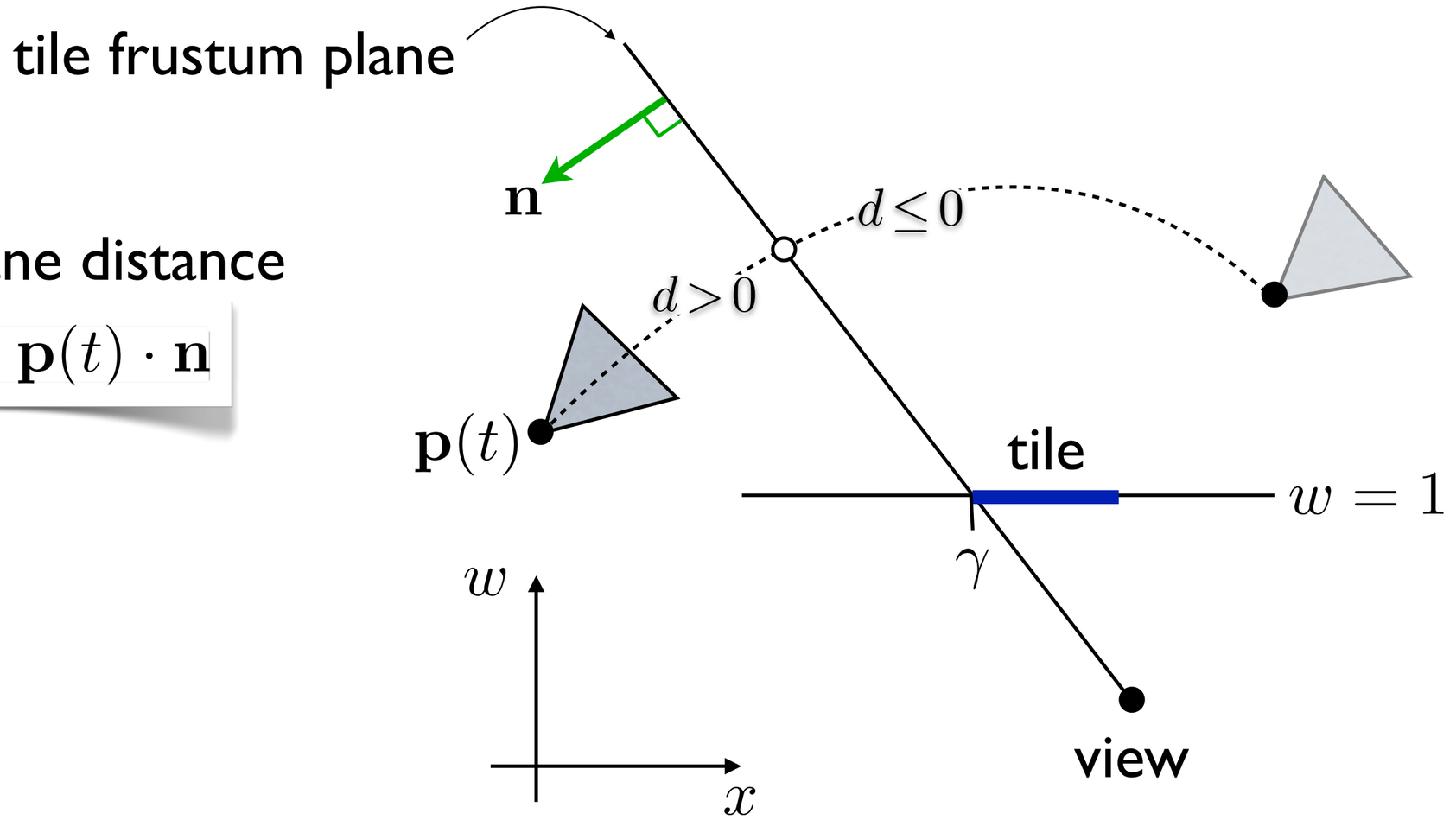


In homogEneous space, each tile is defined by a frustum comprised by four planes that originate from the view point.
In this 2D version, a triangle moves quadratically past two tile planes.

Triangle – Tile test

Vertex – plane distance

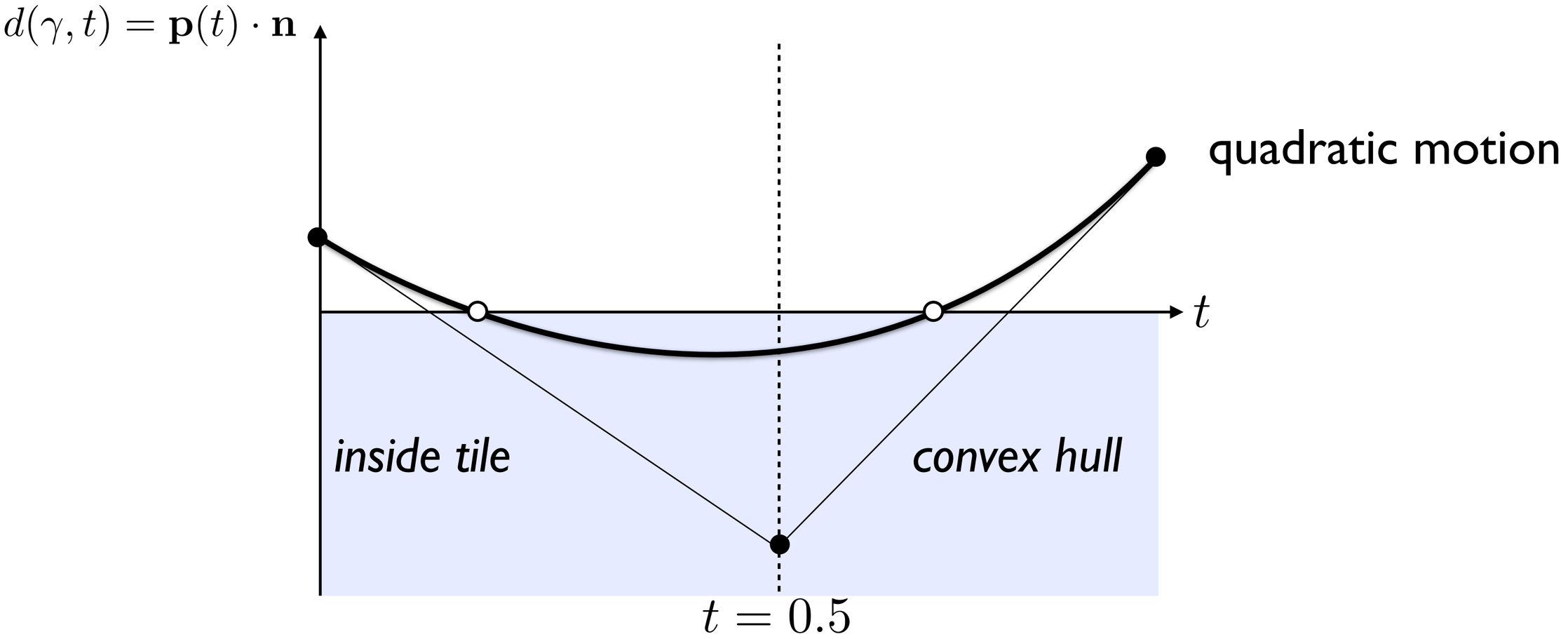
$$d(\gamma, t) = \mathbf{p}(t) \cdot \mathbf{n}$$



The distance between one of the vertices and the left plane is the dot product between the vertex and the plane normal.
This is a polynomial of the same order as the motion, which switches sign as the vertex moves in or outside of the plane.

Lets focus on this distance polynomial for a while.

Vertex – Tile distance



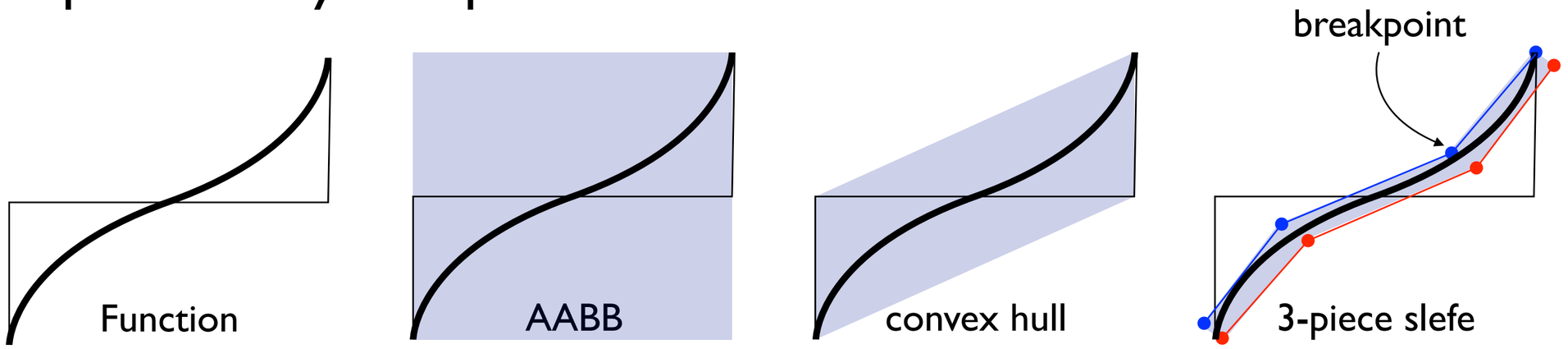
Here, distance is plotted over time for a vertex that moves quadratically.
In this example, the vertex enters the tile after a short while and then exits a bit later.

Ideally we would want the EXACT intersection points, when the distance equals zero,
but for motion of order two or more, finding these roots are expensive and impractical.

We could use the convex hull of the control points, but it bounds poorly when motion is heavily curved.
Instead we use a tighter bounding construct called slefes.

Slefes [Peters 2003]

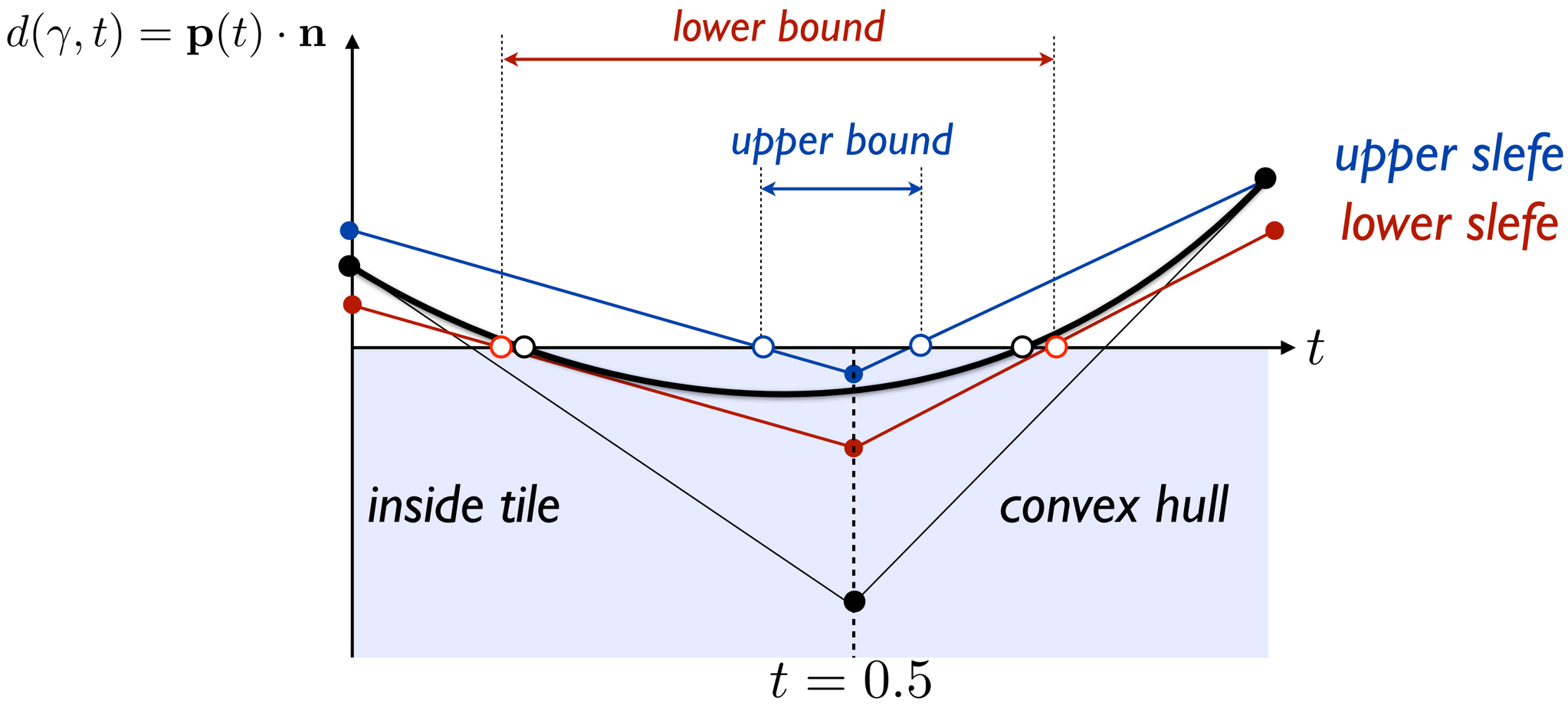
- *Subdividable Linear Efficient Function Enclosures*:
  Efficient bounding construct
- "Sandwiches" e.g. polynomials between upper & lower piecewise-linear bounds
- Computationally cheap



And slefes are, to sidestep for a moment, bounding constructs that sandwich non-linear functions,
such as polynomials, by upper and lower, piecewise linear bounds.
The control points for these piecewise functions are called breakpoints.

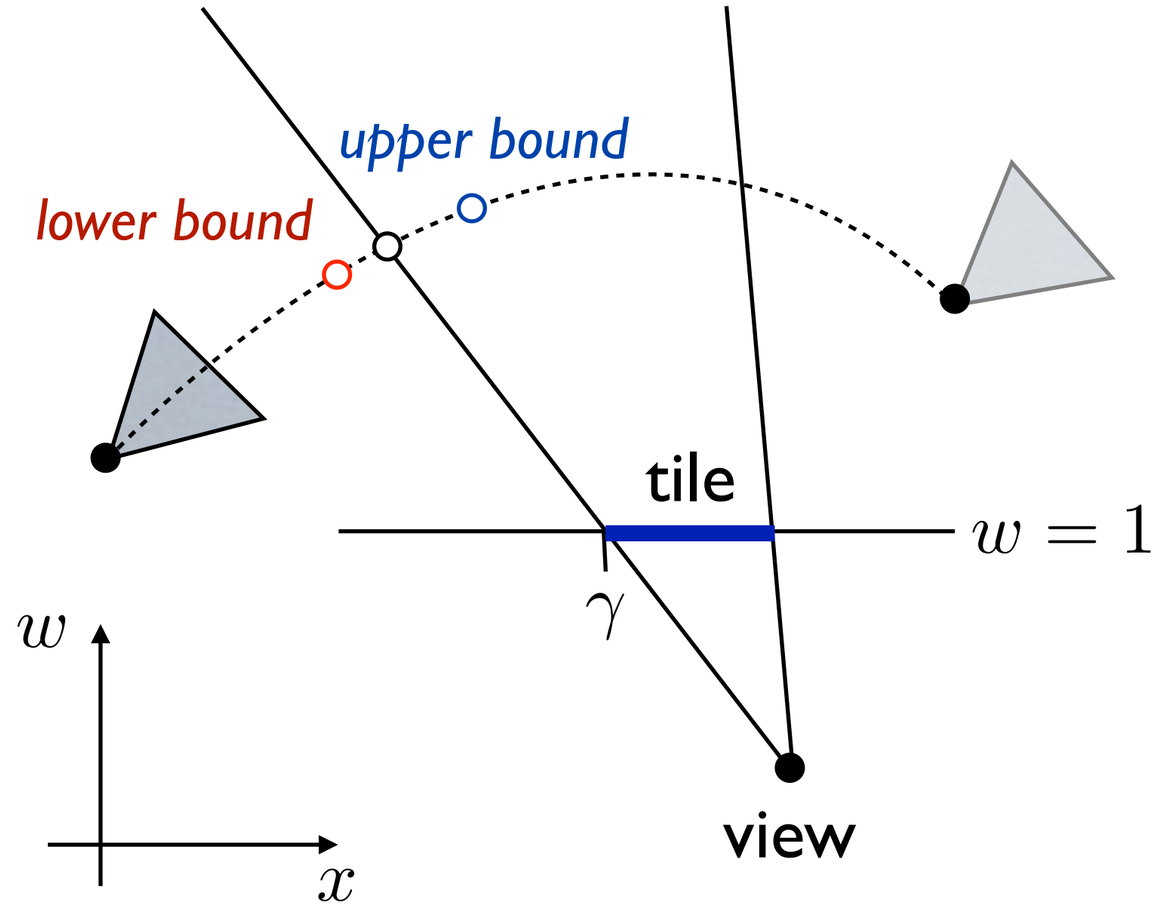
Slefes are inexpensive to compute and provide much tighter bounds than the convex hull for highly curved functions.

Vertex – Tile bounds



With slefes we get tighter bounds of the distance polynomial and each linear segment is easily intersected with the time-axis.
In this case there are two time-values for the upper and lower bound respectively,
forming upper and lower bounds within which the vertex is inside the tile plane.

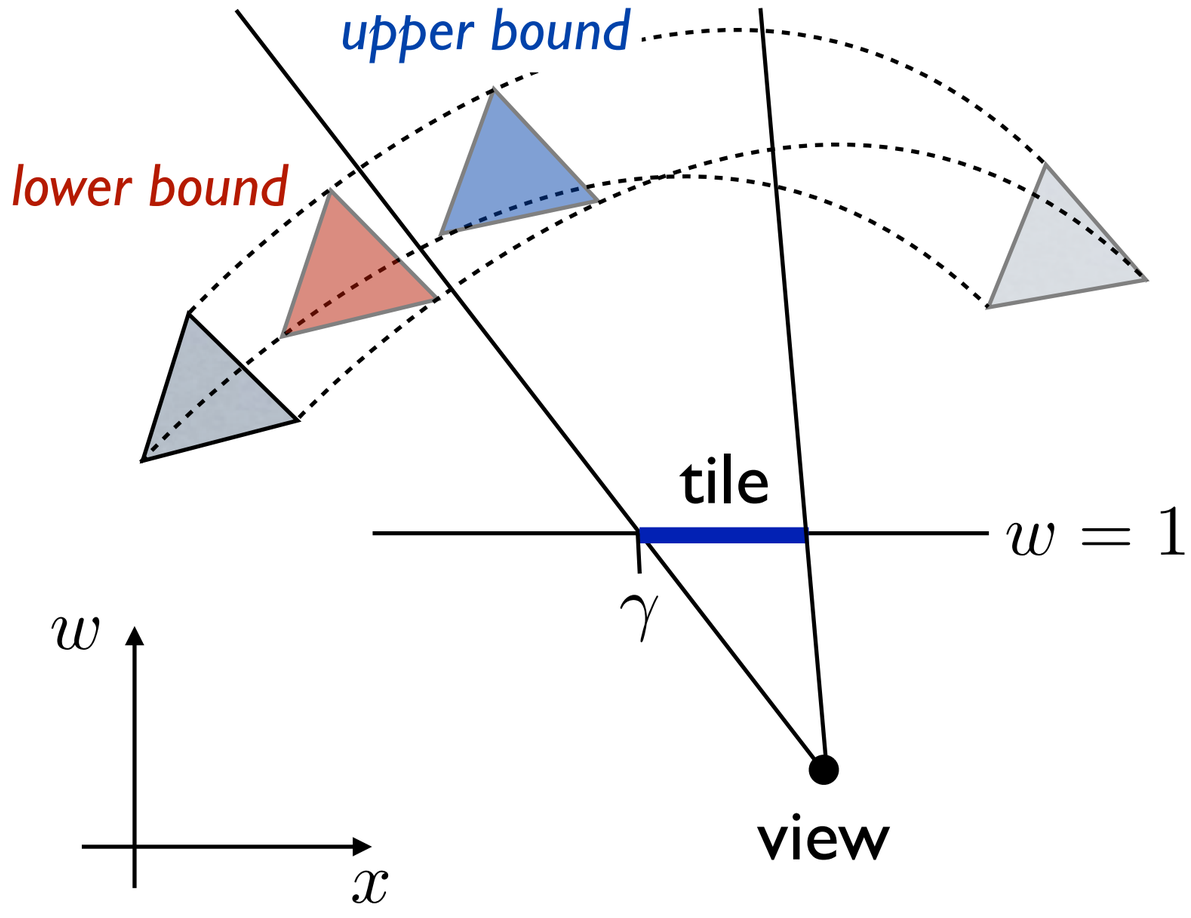
Triangle – Tile test



Here’s how such bounds could look for one of the vertices and one of the tile planes in the 2D example.

To incorporate the entire triangle, slefes for all three vertices are considered.
By picking the minimum breakpoints from the lower slefe,
and the maximum breakpoints from the upper slefe,
we obtain conservative bounds that are valid for the entire triangle.

Triangle – Tile test

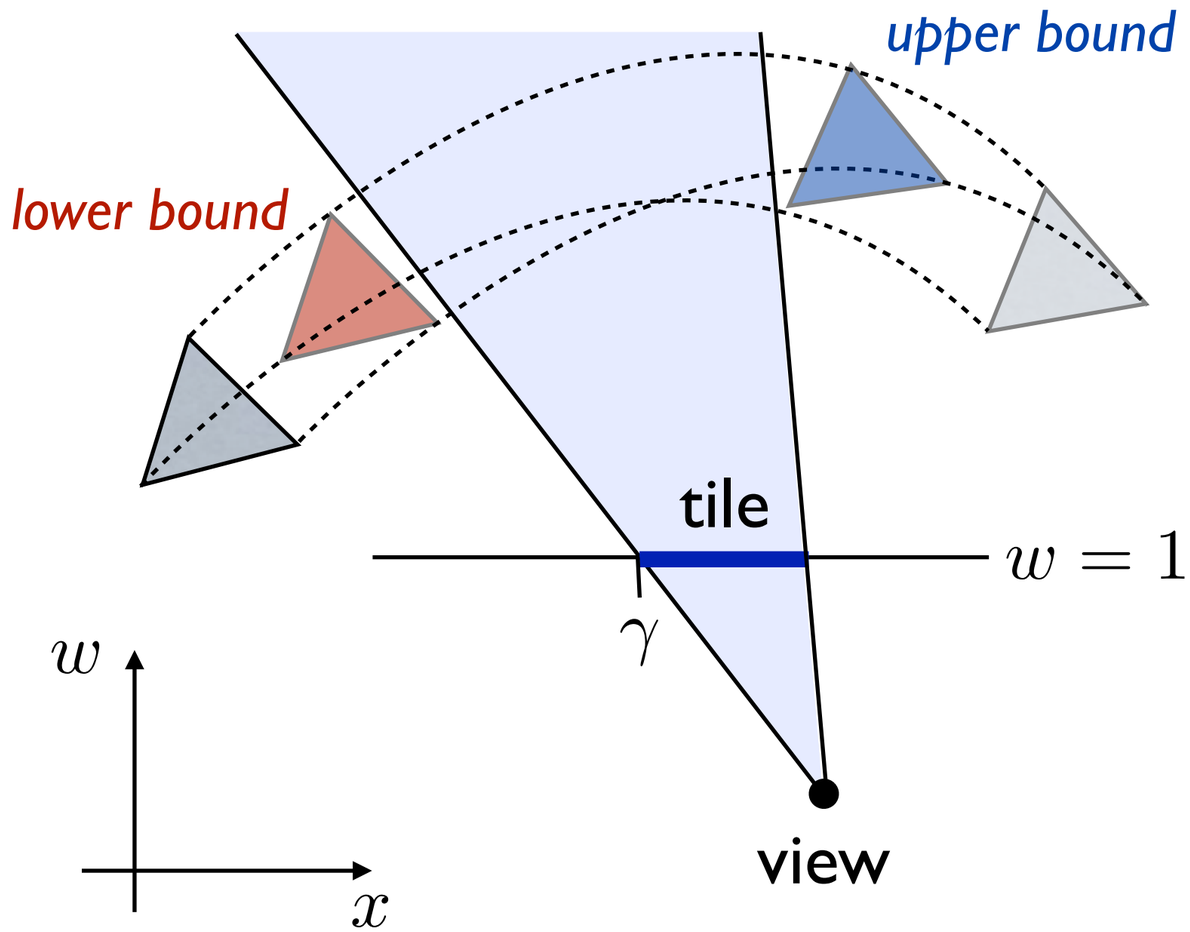


The final step is to extend the bounds further to account for the second tile plane as well.

This can be done by computing new bounds for that tile in the exact same way,
but there is one observation that makes this a lot simpler.

That is that the distance polynomial is linear over the screen-space dimensions, in this case the x-axis.
This means that instead of computing new bounds for every single plane, we can pre-compute
outer bounds for the entire axis at triangle setup,
and then interpolate bounds for each tile location in-between.

Triangle – Tile test

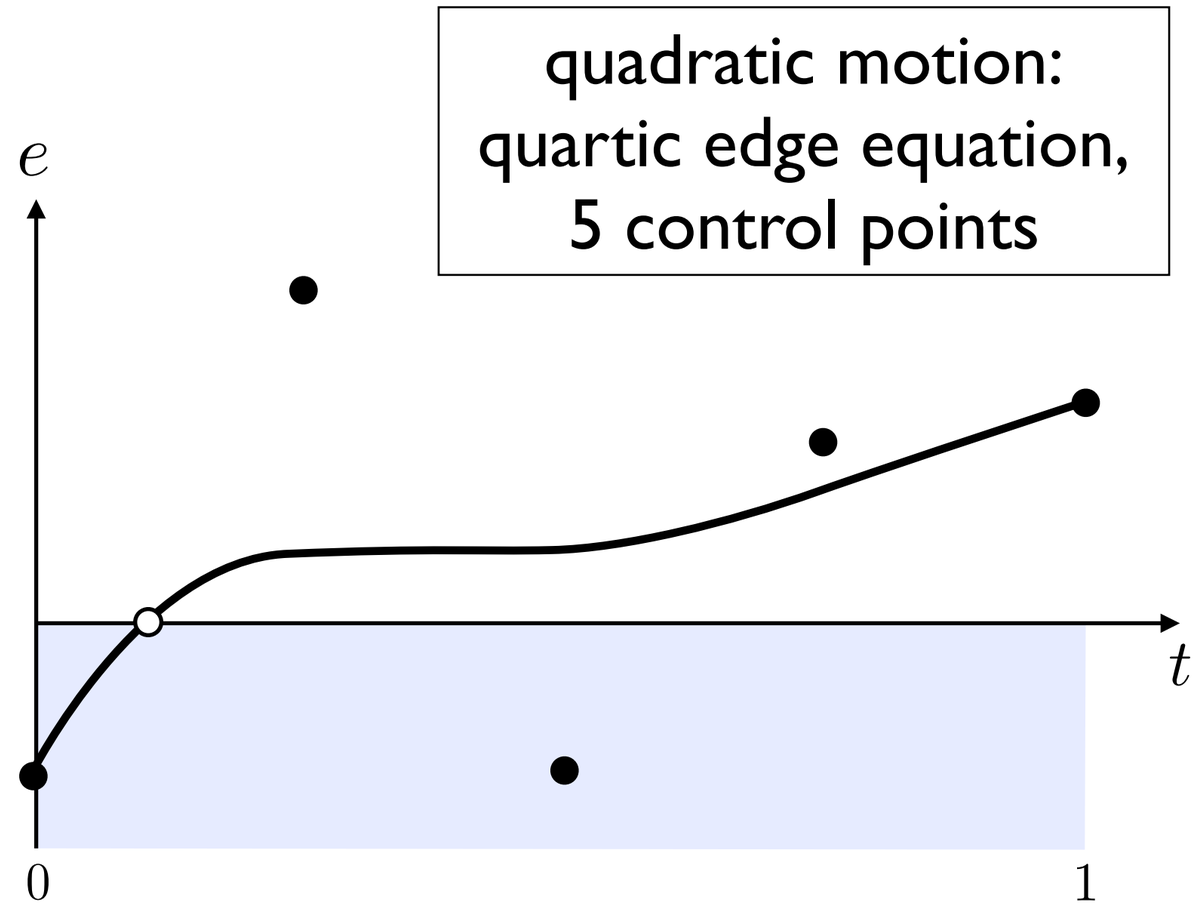


This way we obtain a new bound that accounts for the entire triangle over both tile planes.

In the full 3D case, all that remains is to include the two y-axis planes as well.

Edge – Tile test

- Recall, edge equation: $e(x, y, t)$
  - Bernstein polynomial of order 2*n*
  - 2*n*+1 control points (as functions of edge vertices)
  - Pad control points with half the tile size
- Perform test once at center of tile



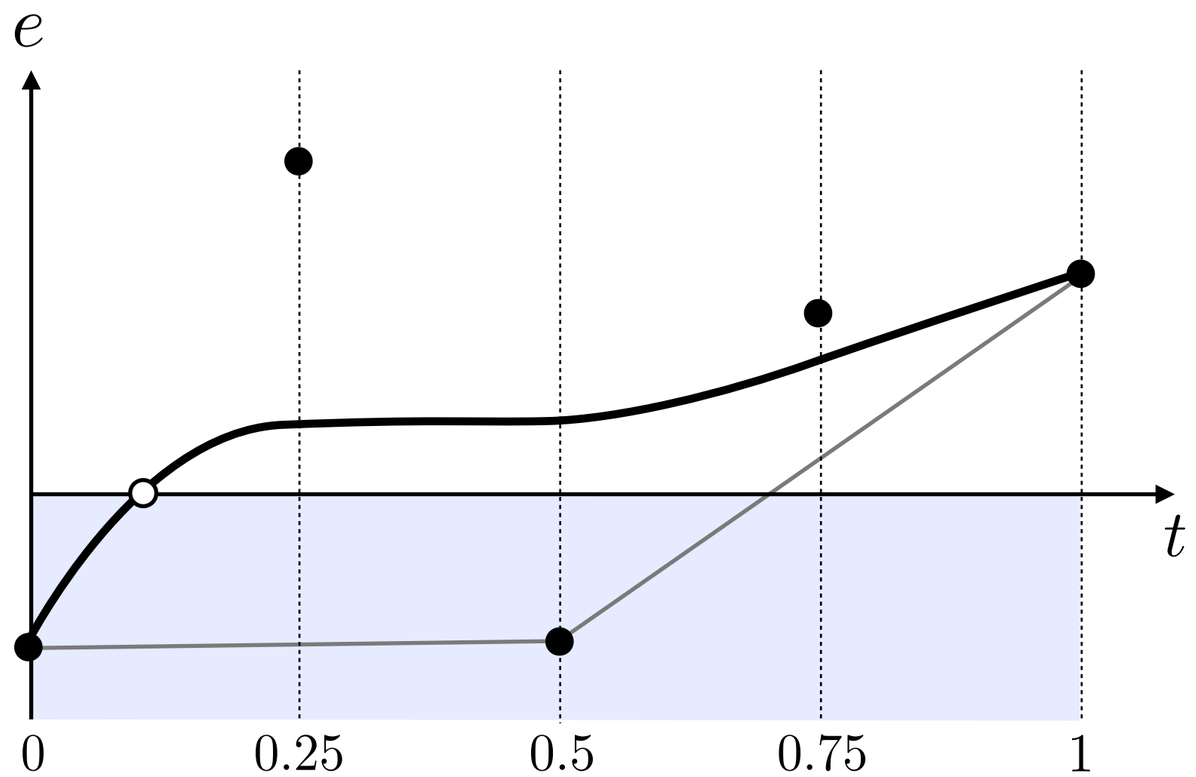
The edge – tile test works in a very similar way, but this time it is the edge equation polynomial that is used.

The curve in the figure illustrates an edge equation for quadratic motion, evaluated with respect to a tile center.

In order to make the resulting temporal bounds valid for the entire tile, and not only its center,
the control points of the edge equation are padded with half the tile-size.

Edge – Tile test

- Cull tile if all control points are > 0



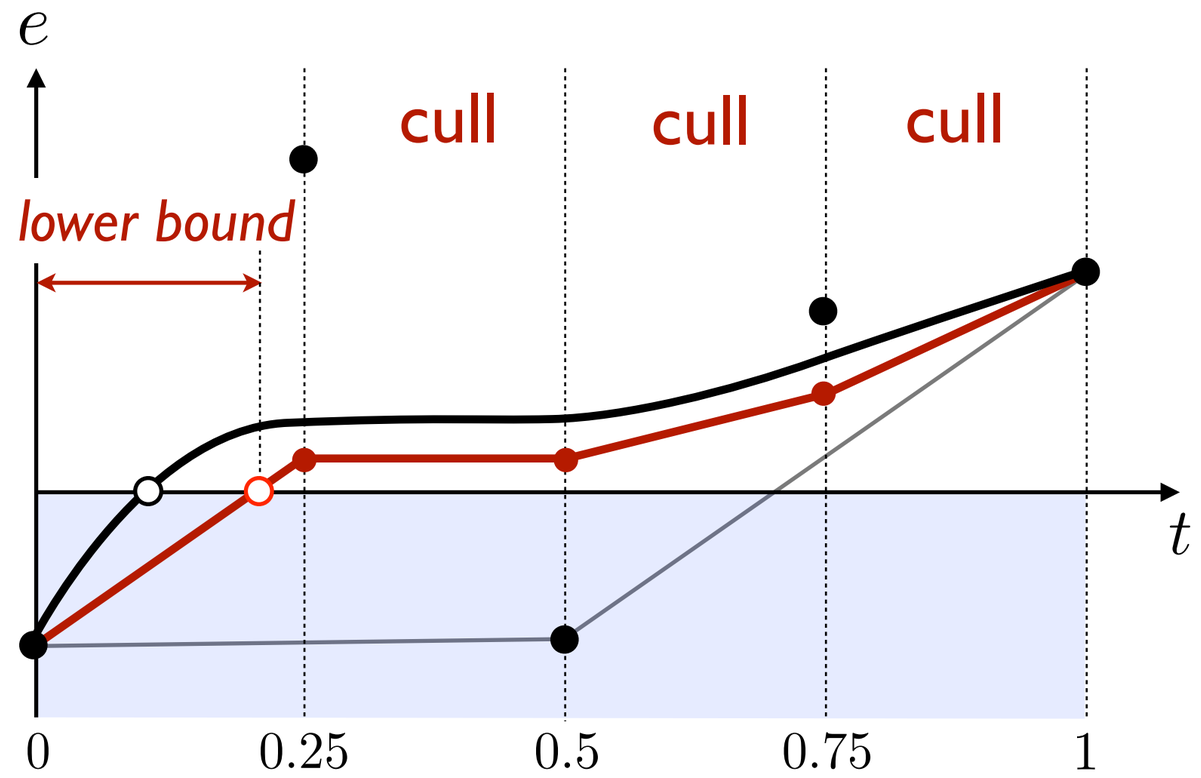
convex hull

We then go through a number of conditions.

First, if all control points are positive, the tile can be culled directly.
This is not the case here ...

Edge – Tile test

- Cull tile if all control points are > 0
- Cull tile if all slefe breakpoints are > 0
- If two consecutive breakpoints are > 0: cull this timespan
- Otherwise: intersect line segment with *t*-axis for conservative bound



convex hull

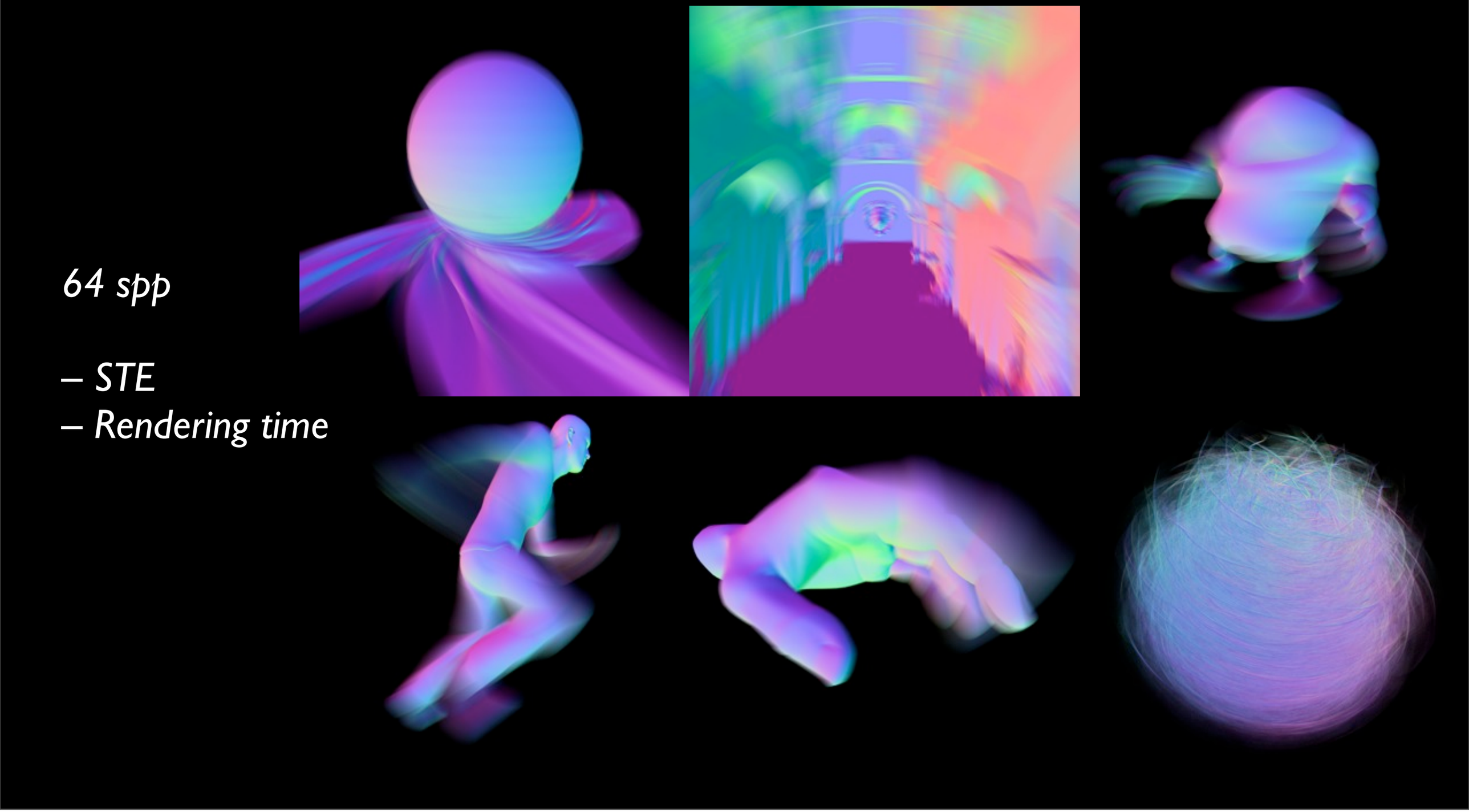
... so we again turn to slefes, drawn in red, which provide a tight lower bound.

Now, we can safely cull the three rightmost intervals, and for the remaining, leftmost interval,
a conservative bound can be computed by intersecting the slefe segment in that range.

This marks the final lower bound produced by this edge – tile test.
This bound is then combined with the bounds from the triangle – tile test,
before rasterization proceeds with the actual coverage tests, and so on..

64 spp

– STE
– Rendering time

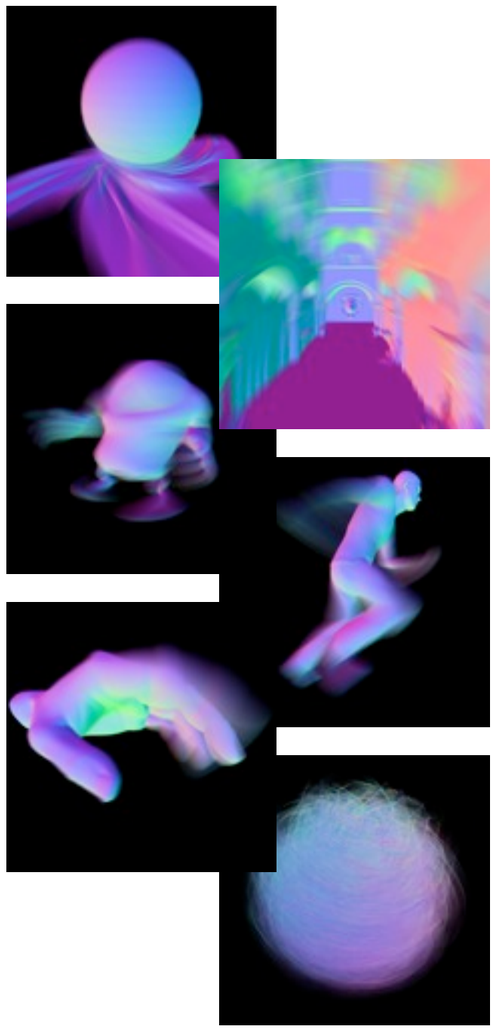


We tested the INTERVAL and TILE algorithms on a standard set of scenes with a scalar, single-threaded rasterizer.

The motion is quadratic, and is induced either by camera rotation, object-space rotation or using an animation sequence.

Two measurements were made, STE or sample test efficiency, which is the rate by which primitives are hit by the coverage tests, and performance.

Traversal: Sample test efficiency (STE)



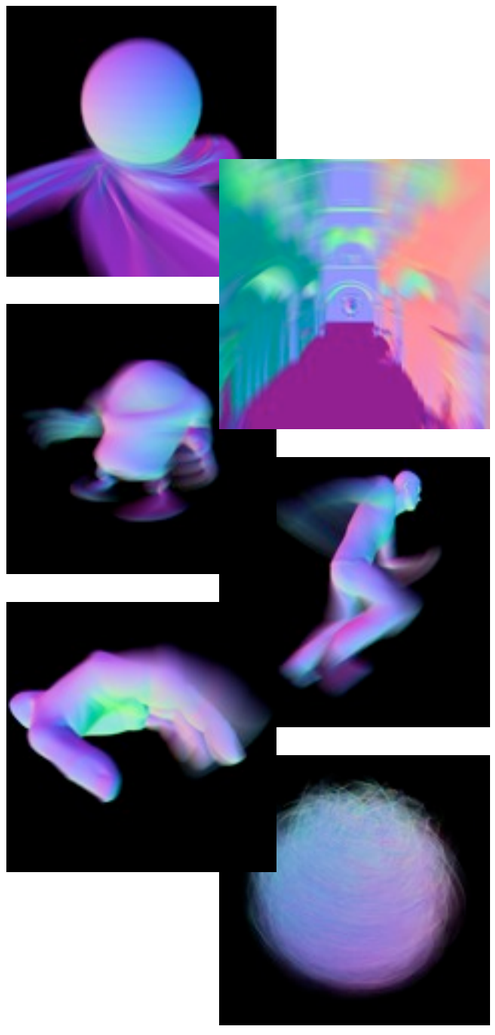
higher is better

Scene	Tile: Tile size 4 × 4	Tile: Tile size 2 × 2	Tile: Tile size 1 × 1	Interval: Num intervals 4	Interval: Num intervals 16	Interval: Num intervals 64
ClothBall	11	18	**24**	1.2	6.4	15
Sponza	47	65	**79**	10	18	21
BigGuy	17	27	**36**	3.1	11	18
Ben	12	21	**34**	0.68	5.2	14
Hand	30	46	**62**	7.8	19	24
Hairball	3.5	7.5	**14**	0.75	2.8	4.8

As for STE, the TILE algorithm as expected displays higher rates for smaller tile-sizes,
while INTERVAL displays higher rates with a greater number of intervals.
In comparing the algorithms, TILE has consistently better rates by a pretty safe margin.

However, these measurements don't account for the actual cost of the tests.
The tile tests in particular may actually account for a considerable fraction of the total execution time,
especially for long motion trails where many tiles need to be considered.

Traversal: Relative performance



lower is better

Scene	Tile: Tile size 4×4	Tile: Tile size 2×2	Tile: Tile size 1×1	Interval: Num intervals 4	Interval: Num intervals 16	Interval: Num intervals 64
ClothBall	1	1.38	3.5	4.3	0.93	**0.61**
Sponza	**1**	1.0	1.7	2.8	1.8	2.0
BigGuy	1	1.2	2.4	2.9	1.1	**0.77**
Ben	1	1.6	4.3	7.1	1.1	**0.52**
Hand	**1**	1.1	1.7	1.1	1.2	1.1
Hairball	1	1.0	2.0	2.8	0.87	**0.65**

So in terms of performance numbers, the outcome is more mixed.
INTERVAL is faster in five of the scenes, in particular those with many small triangles.
TILE is faster for SPONZA and HAND which contain more large triangles that are efficiently culled by the Edge – Tile test.

Overall, it’s not clear that there is just one winner.
Generalized INTERVAL might be the best choice for a software-based implementation, like ours,
since it is faster in pure rendering time.
However, for a parallel hardware implementation, it’s very likely that the TILE algorithm is actually better suited due to aspects such as cache coherence, ability to incorporate occlusion culling, and so on.

Semi-analytical rasterizer

- *Solve* edge equations w.r.t. to time (quartics)
  - Analytical solver: Neuman/Ferrari
    Numerical solver: Bézier clipping with root deflation
- Visibility: interval lists with resolve per sample
- Refined depth approximation
- Super-sampled shading

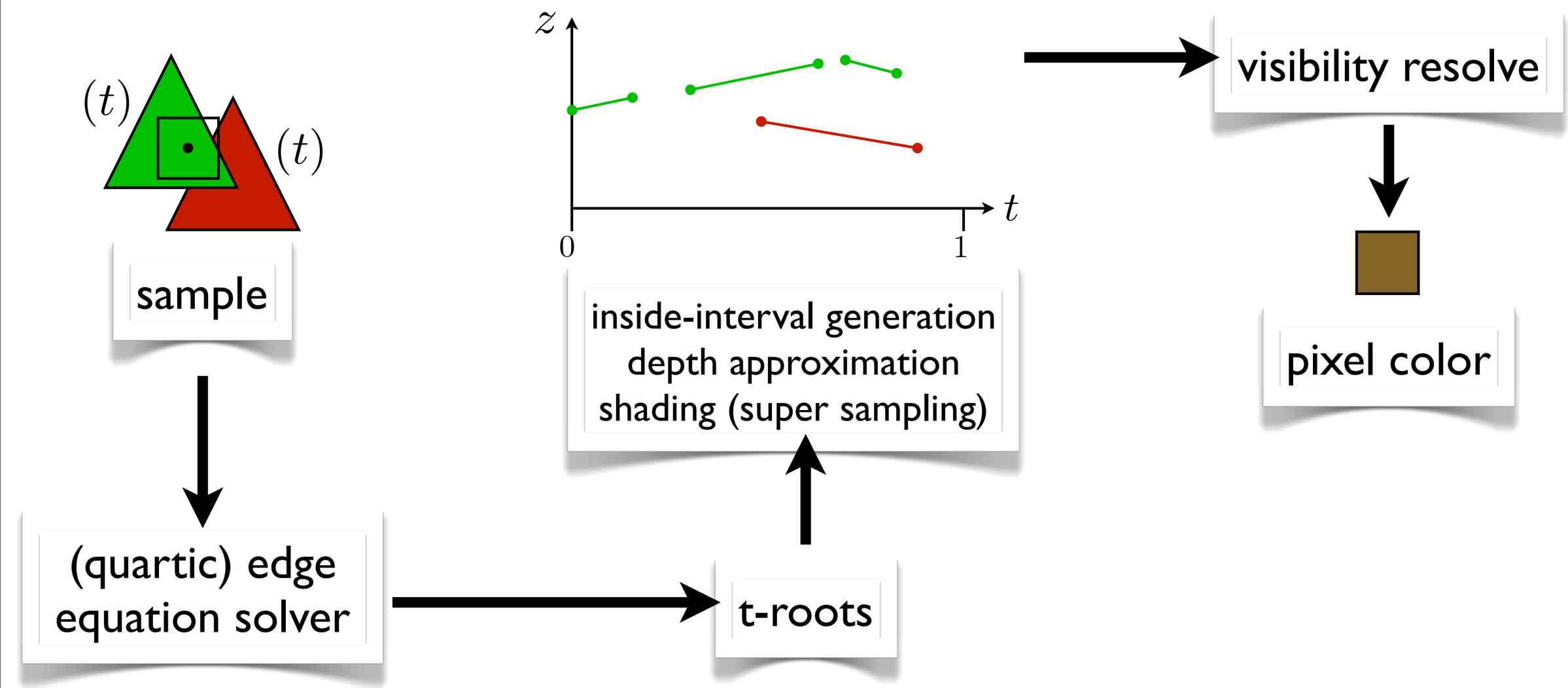
Sung et al. 2002
Glassner 1990
Paeth 1995
Sederberg and Nishita 1990
Press et al. 2007
Gribel et al. 2010 & 2011
Tzeng et al. 2012
Barringer et al. 2012

As an additional use-case we also implemented a semi-analytical rasterizer where time is treated as a continuum instead of stochastically.

There are more details about this implementation in the paper, but the main approach is to SOLVE, instead of sampling, the edge equations, and use the roots to create “visible time-intervals” that are stored in lists per sample. It’s called SEMI-analytical since the depth function is still linearized.

Some extra work is required for this rasterizer, but the benefit is that temporal alias, which is especially prevalent in settings with aggressive motion, is heavily reduced.

Semi-analytical rasterizer: primer

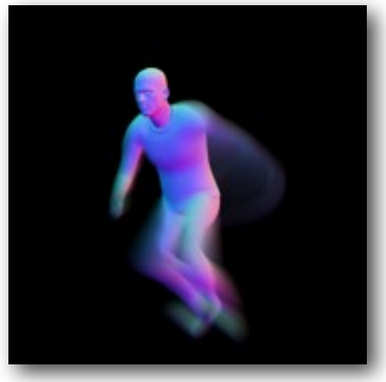


Here's an overview of the rasterizer.

Compared to previous work, we add two things.
First, a high-order representation of motion and a brief evaluation of two quartic solvers.
Secondly, since motion is expected to be more complex,
we provide a simple scheme that refines the depth approximation to avoid visibility errors.

Semi-analytical rasterizer





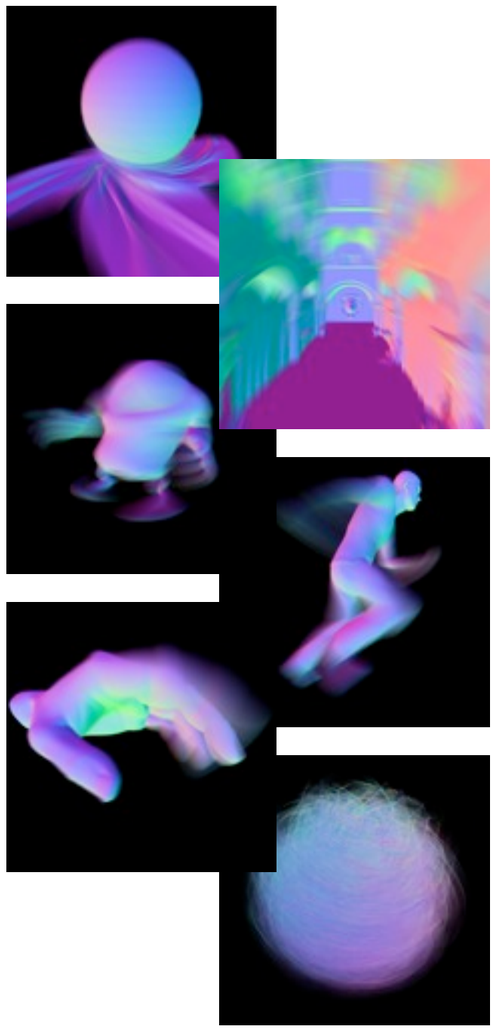

	Sponza rot 12°	Sponza rot 20°	Ben Frame 1-5-10	Hand Frame 1-5-10
PSNR, Power solver :	42.7dB	41.1dB	49.1dB	50.7dB
PSNR, Bézier clipping:	42.7dB	41.1dB	49.1dB	50.7dB
Vis.Acc. Power solver:	98.3%	97.4%	97.7%	99.5%
Vis.Acc. Bézier clipping:	98.3%	97.4%	97.8%	99.6%

higher is better

We gauged the quality of this rasterizer compared to a stochastic reference using direct signal resemblance, PSNR, and a new metric we call visibility accuracy, which samples visibility and measures the rate by which the correct primitive is rendered.

SPONZA scores a bit lower in PSNR compared to the BEN and HAND scenes,
which can be explained by the camera rotation which induces motion, and thus errors, over a larger portion of the screen.
But overall the numbers are quite high and seem to indicate that analytical approaches are a feasible option for high-order motion.

Semi-analytical rasterizer + TILE



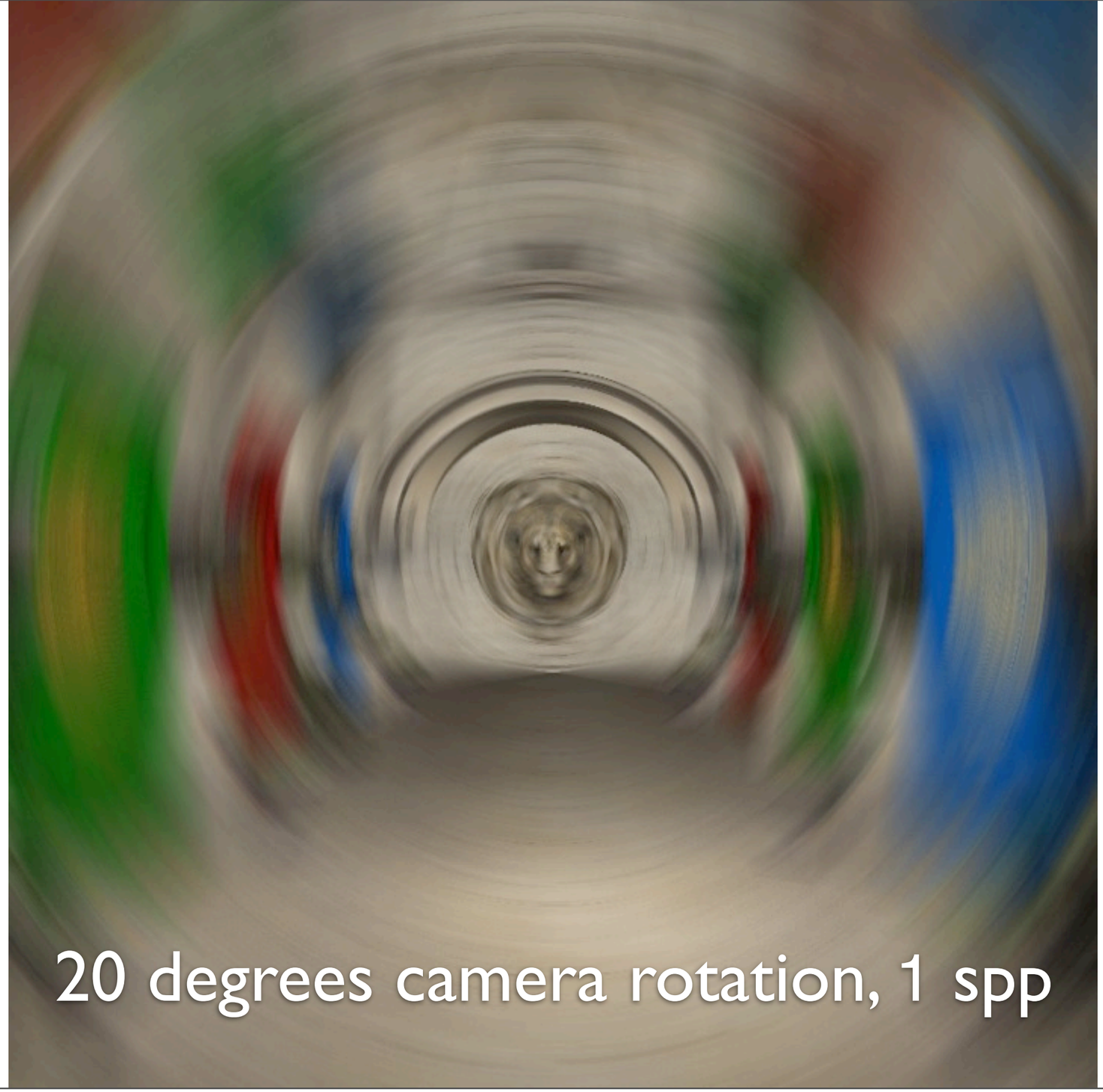
Performance improvement
compared to Gribel et al. 2010: $\times 1.7 - \times 4.9$

Furthermore, the TILE algorithm easily adapts for the semi-analytical rasterizer,
and since each coverage test is very expensive here, culling can save much time.

Compared to this earlier algorithm with naive, screen-space bounding box traversal,
the performance gains are up to almost five times.
This is actually a prime example where sample test efficiency is extremely important.

Crytek Sponza

262k triangles



Finally, a larger, and shaded version of the Crytek Sponza scene.

Since motion is perpendicular to the z-axis, one can actually see the circular motion paths in the image.
Also note the absence of temporal alias, even with just one spatial sample per pixel.

This kind of motion is obviously a bit excessive, almost disorienting.

Summary

- Edge equations for generalized motion
- Traversal
  - Generalized INTERVAL
  - TILE: Triangle – Tile, Edge – Tile
- Semi-analytical rasterizer

In summary, we derive new edge equations for generalized motion, of any order,
and present two efficient traversal algorithms.
Generalized INTERVAL is overall faster, especially in a software rasterizer,
while the TILE algorithm provides better STE and is more parallel friendly.

In addition, we provide a use-case in the form of a semi-analytical rasterizer,
and show that the theory can be applied to this context. as well

Our final note, and this is something we are clear about in the paper and the reviewers seemed to agree,
is that even though high-order motion has a narrow scope of application, and is more cumbersome to implement,
still in those cases when it does matter, with aggressive, non-linear motion, it IS quite powerful and should be considered as an option.

Thanks to...

Lund University
Intel's Advanced Rendering Technology team
The reviewers for their valuable input

Utah 3D Animation Repository (Ben, Hand)
UNC Dynamic Scene Benchmarks (Clothball)
Samuli Laine (Hairball)
Bay Raitt (Big Guy)
Marko Dabrovic/Frank Meinl (Sponza)

...and to *You* for listening!

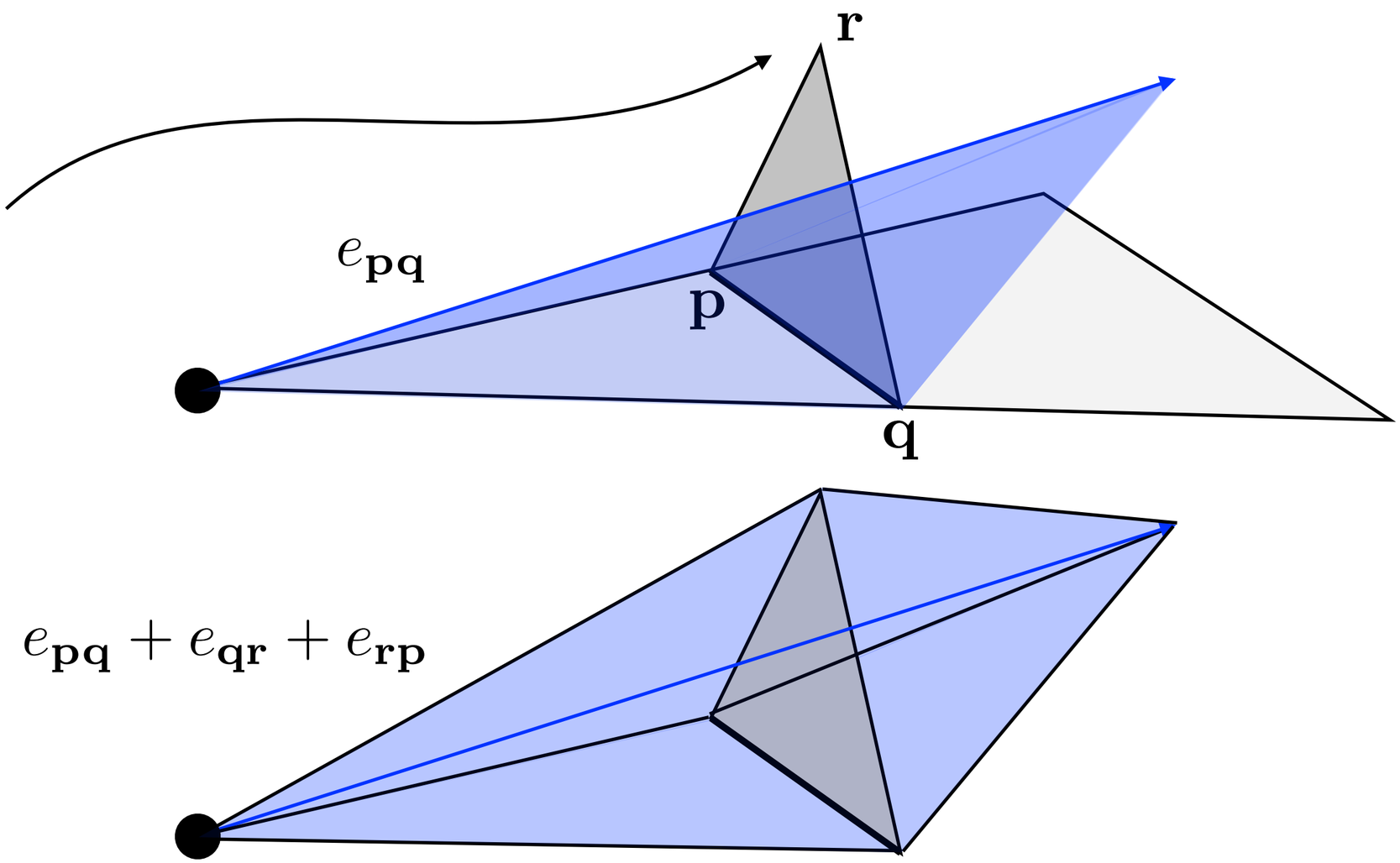
Thank you and enjoy the conference.

Backup

Edge Equations: barycentric coordinates

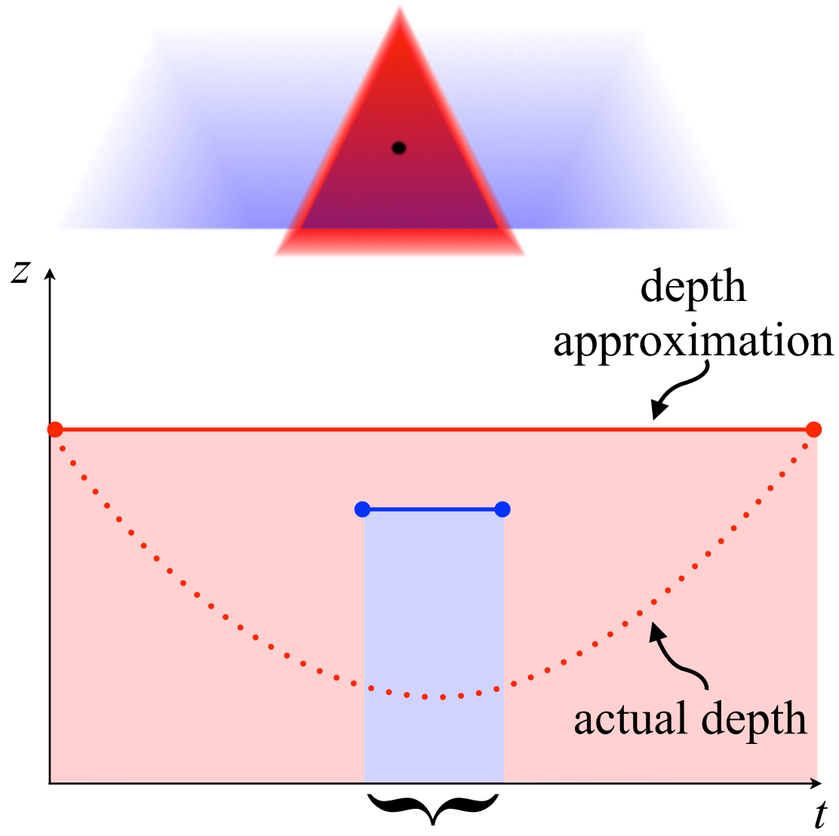
$$b_{\mathbf{r}} = \frac{e_{\mathbf{pq}}}{e_{\mathbf{pq}} + e_{\mathbf{qr}} + e_{\mathbf{rp}}}$$

barycentric weight for **r**



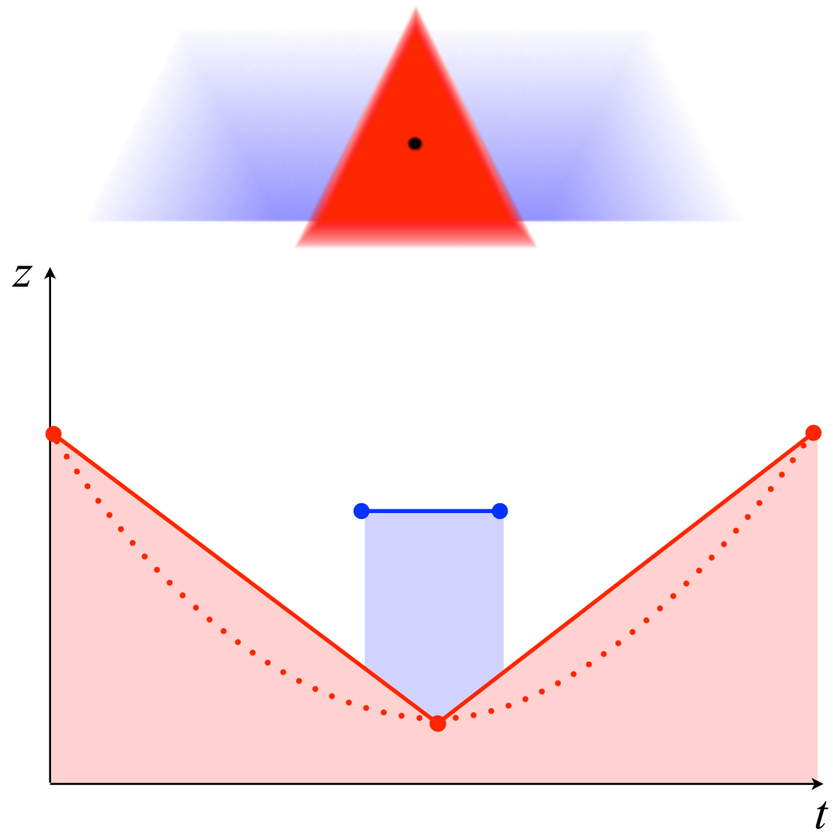
Depth approximation

no splitting



incorrect occlusion

error tolerance splitting



correct visibility

Hand

frames 1, 5, 10
16k triangles

Ben

frames 1, 5, 10
78k triangles