OptFlowCam

A 3D-Image-Flow-Based Metric in Camera Space for Camera Paths in Scenes with Extreme Scale Variations

Lisa Piotrowski, Michael Motejat, Christian Rössl, and Holger Theisel

EUROGRAPHICS CONFERENCE 2024

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

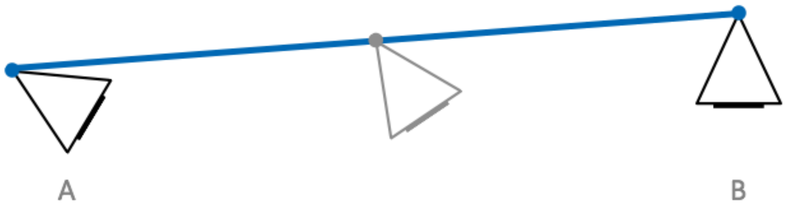
April 26, 2024, Limassol, Cyprus

Motivation

Given two virtual camera poses, how do we connect them?

Simplest Idea: Linear Interpolation

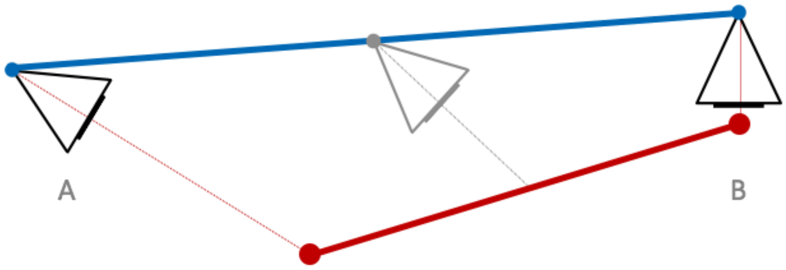
of **eye point** and **angle of rotation**

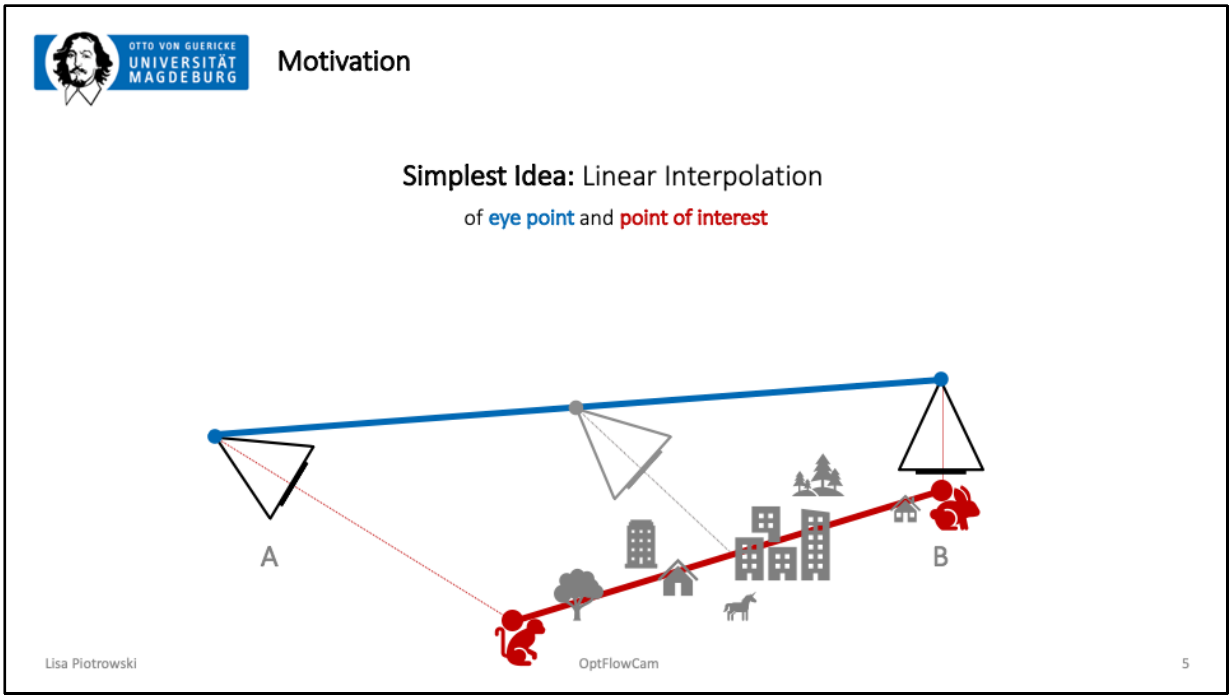


Motivation

Simplest Idea: Linear Interpolation

of **eye point** and **point of interest**



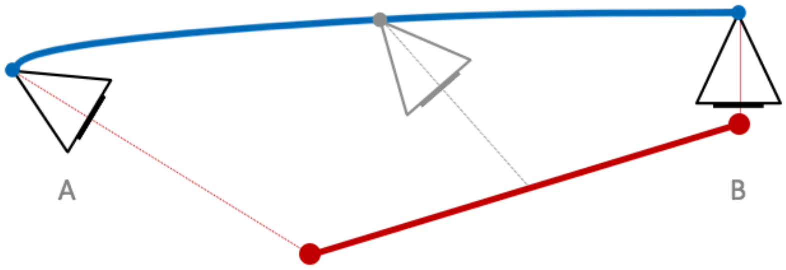


In the scene there can be anything or nothing between those two camera poses and points of interest. We assume nothing about the scene here.

Motivation

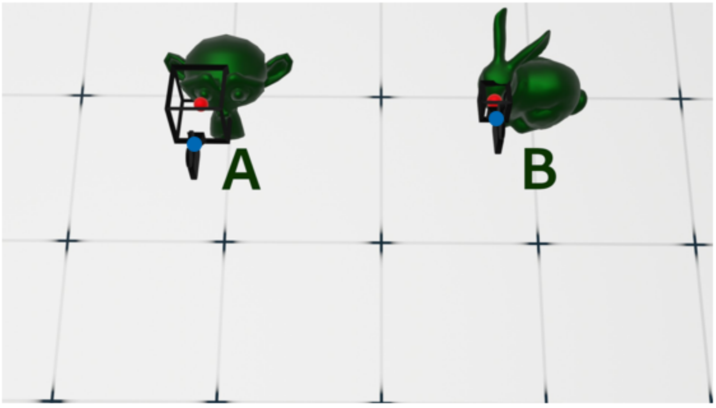
Simplest Idea: Linear Interpolation

of **point of interest,**
distance to the point of interest
and **angle of rotation**



Motivation: What is the problem?

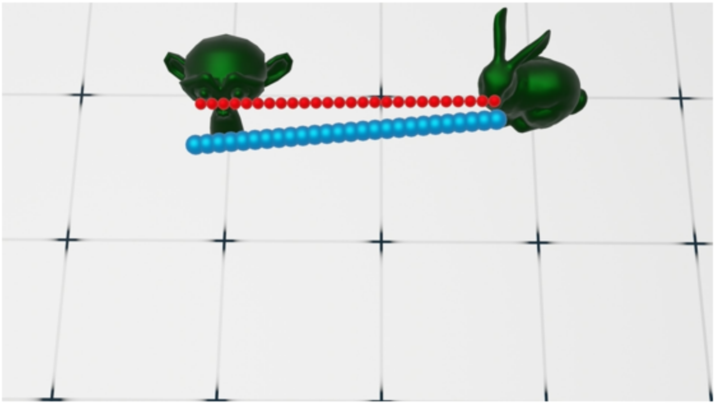
For a scene like this...



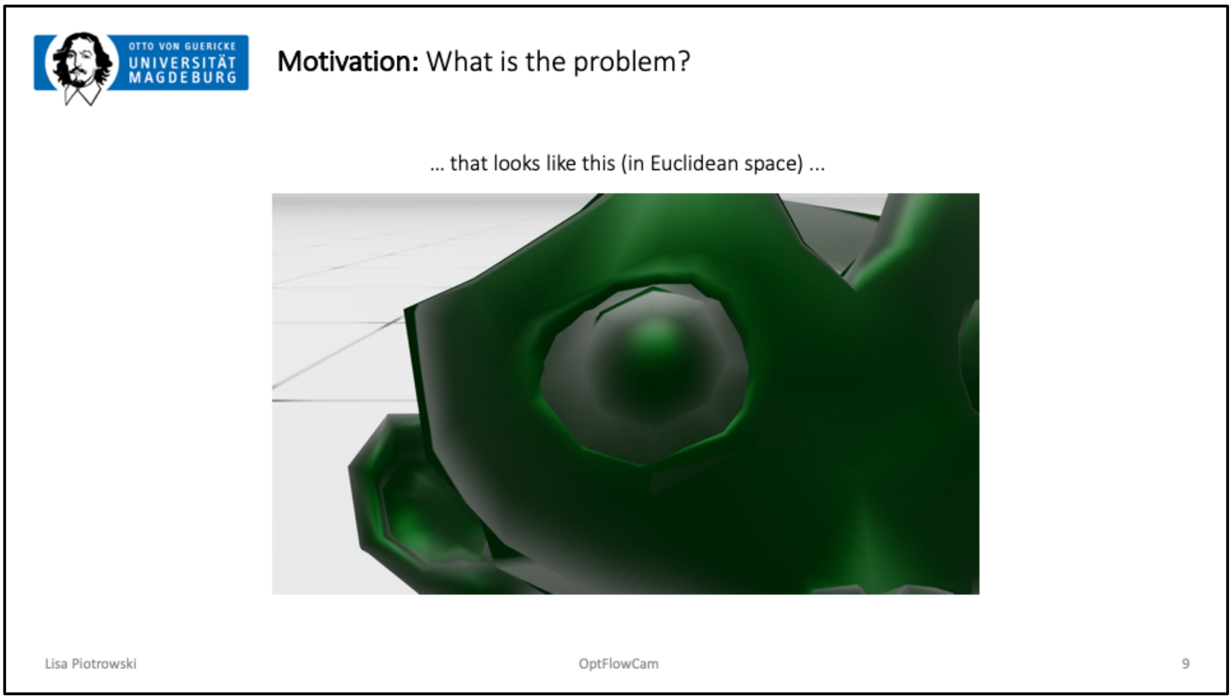
In the scene, the blue dots are the eye point of the camera and the red dots the points of interest. The cube is an approximation of the view frustum (i.e. about the extend we can see in the camera view)

Motivation: What is the problem?

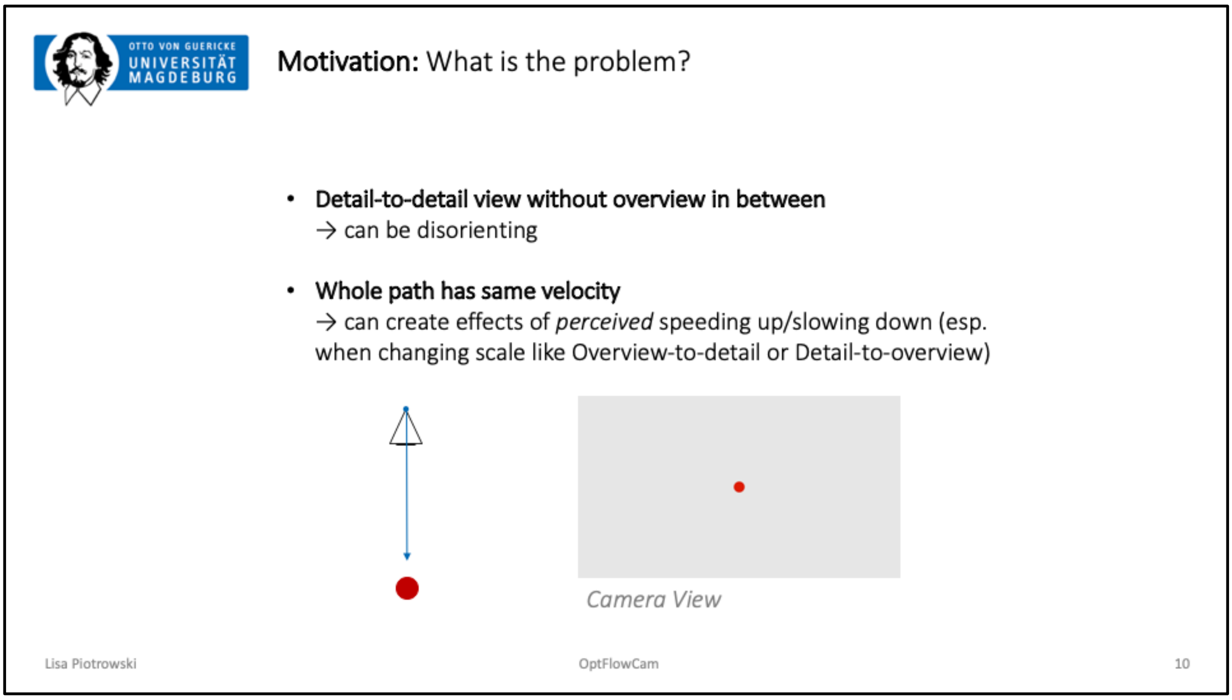
… we get a path like this …



POI and eye point are linearly interpolated. If sampled at equidistant parameter t, the blue and red dots have the same distance to each other. This doesn’ t look too bad from the outside, but…



When look at from the camera view, the result is rather jarring. The objects slide in and out of view while being really close to them and half of the time we can’ t see either of them.



The second point couldn't be seen that well in the first video but is very apparent in this example. We seem to speed up approaching the red dot although the camera has a constant ”real world” velocity.

Motivation: What would be preferrable?

- **Transition between detailed viewpoints with overview in between**
  → creates context
- **Adapt velocity to current scale**
  → similar perceived velocity over the path
- **Closed-form solution of interpolation scheme**
  → easier to integrate into existing systems
  (and faster than performing an optimization routine)

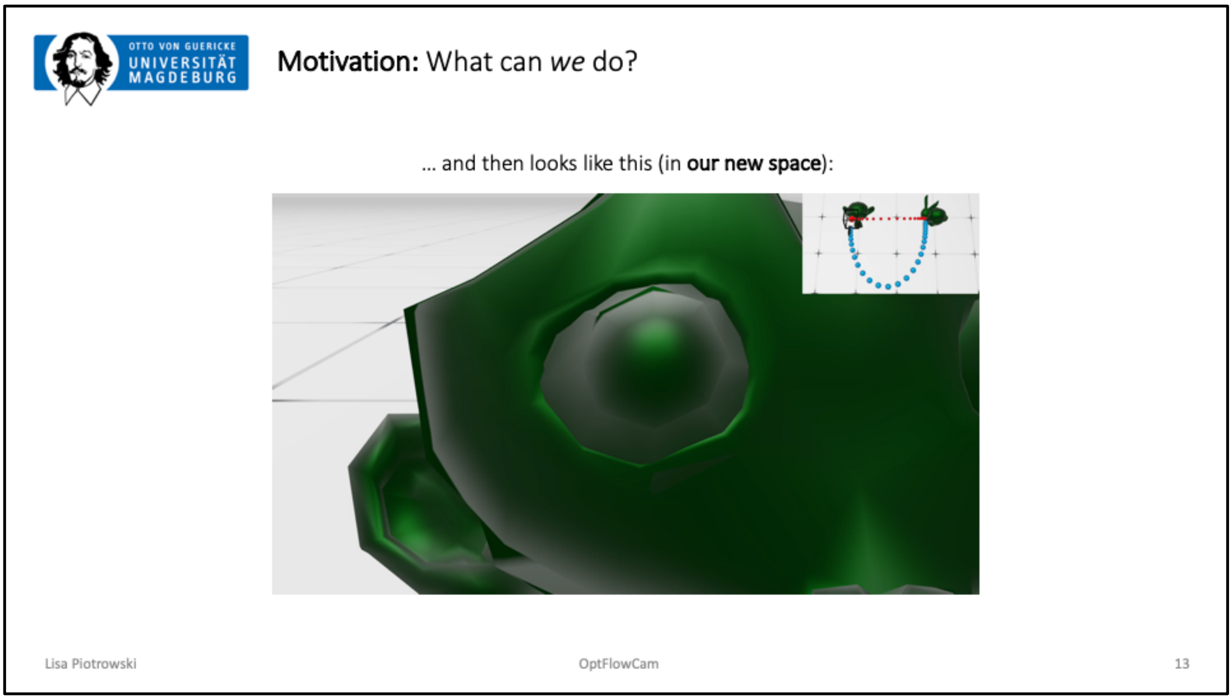
Motivation: What can we do?

Observation: Problem is the *Euclidean space*.

→ **Objective**

- introduce Riemannian metric in *camera space*
- do linear interpolation in *new space*

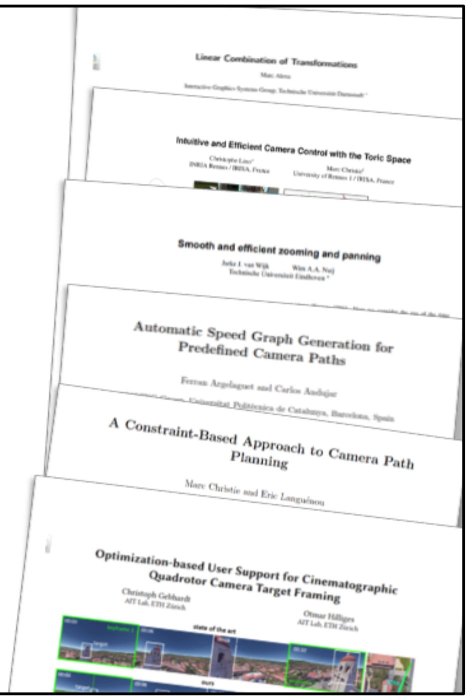
Euclidean space = world space/geometric space



This is much better than the previous example in Euclidean space because it shows all of the desired properties we talked about in the previous slide.

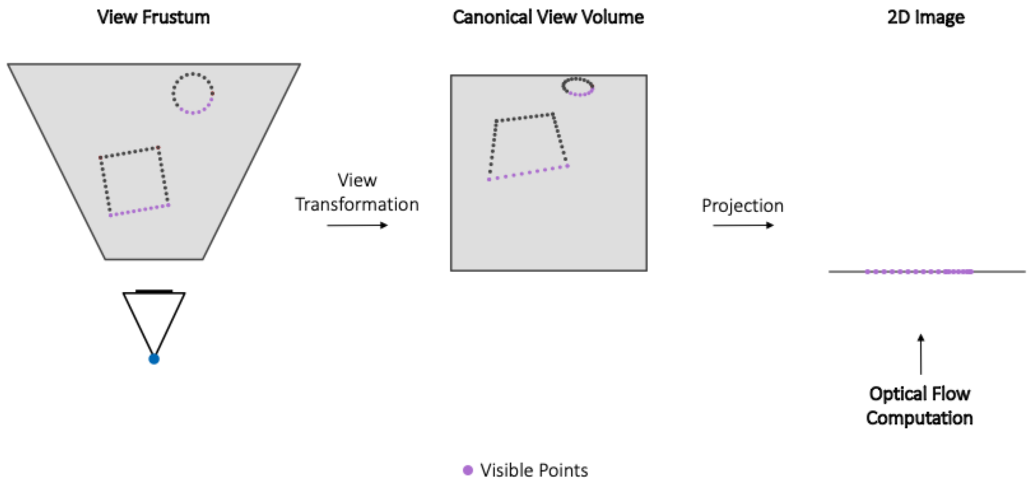
Related Work

- **Interpolation techniques** (linear interpolation of camera parameters or camera transformations [Ale02])
- **Toric Space** [LC15]
- **Zooming and panning in 2D** [vN03]
- **Camera path planning** (lots of different works)



We are firmly in the interpolation techniques section. We do NO path planning or collision avoidance or cinematographic optimization (see paper for more comparisons)

Idea: Base camera path on optical flow

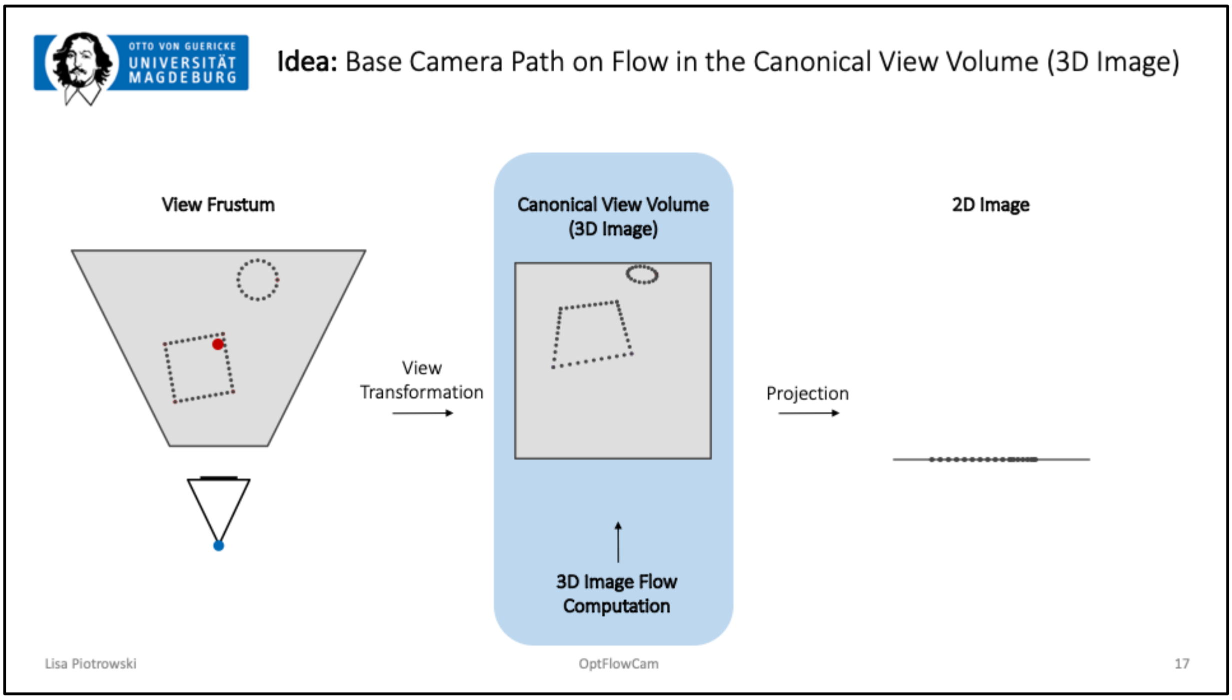


Idea: Base camera path on optical flow

Optical Flow depends on visible surfaces which means…

- … it needs an optimization routine on a discretized path
- … but objects entering and exiting the view create discontinuities (which makes it difficult to optimize)
- … and collisions can derail the optimization routine.

→ need something that is mostly agnostic to the (visible) scene for a closed-form solution

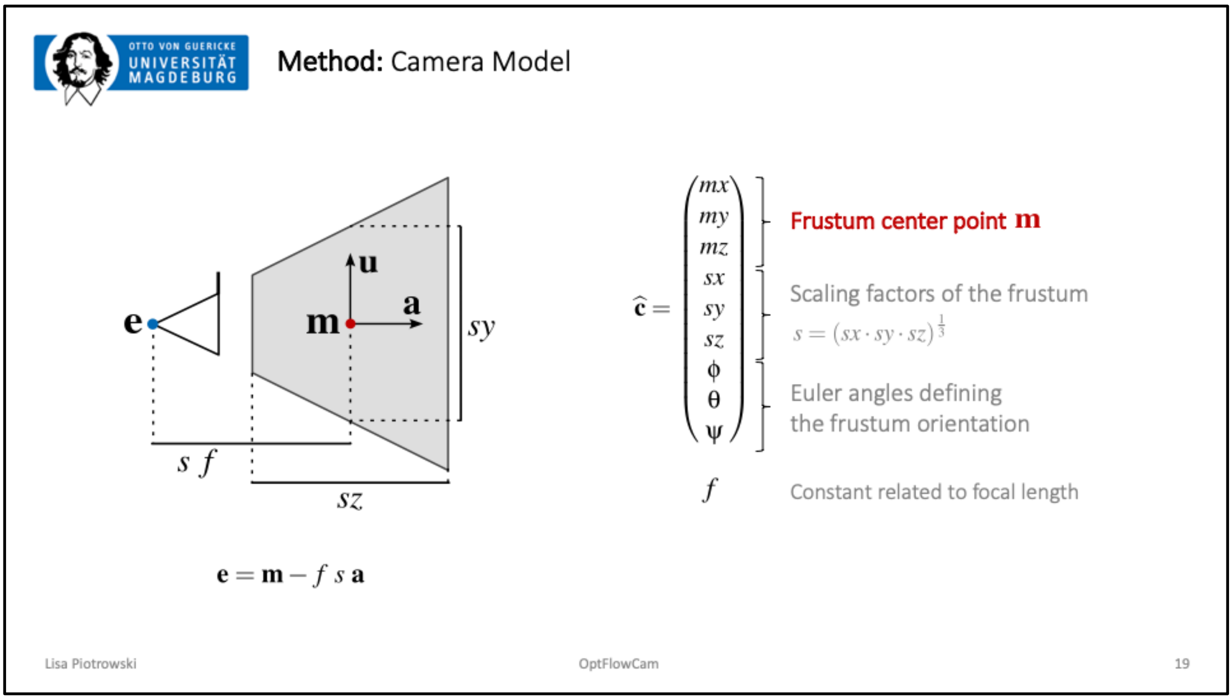


All our schematic examples are in 2D so the projection is to 1D. Of course for a 3D camera the projection is a 2D image on the screen

Idea: Base Camera Path on Flow in the Canonical View Volume (3D Image)

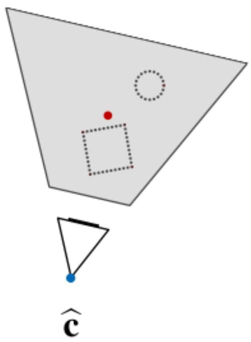
3D image flow …

- … does not depend on objects in the scene but the position of the **point of interest** in relation to the **eye point of the camera**
- … which implicitly defines if it is a detail view or overview
- … and can be expressed as a closed-form solution.

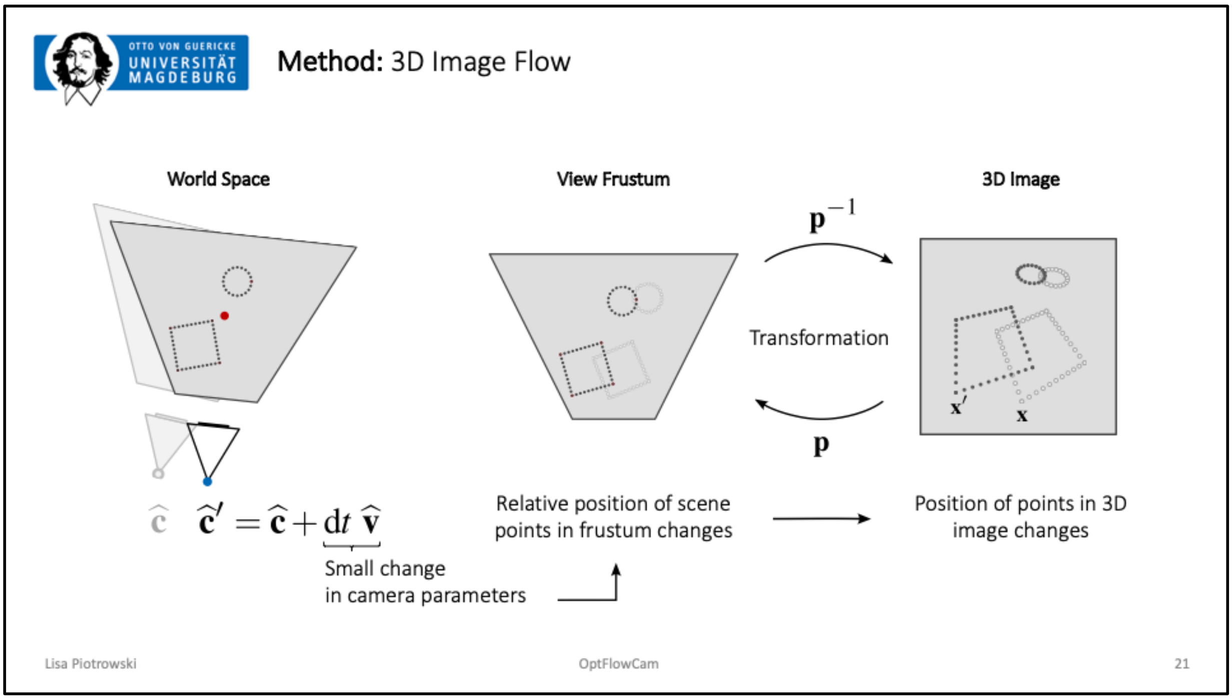


The eye point (blue dot) is a computed property here. It is possible for most camera models to convert it to our model and vice-versa.

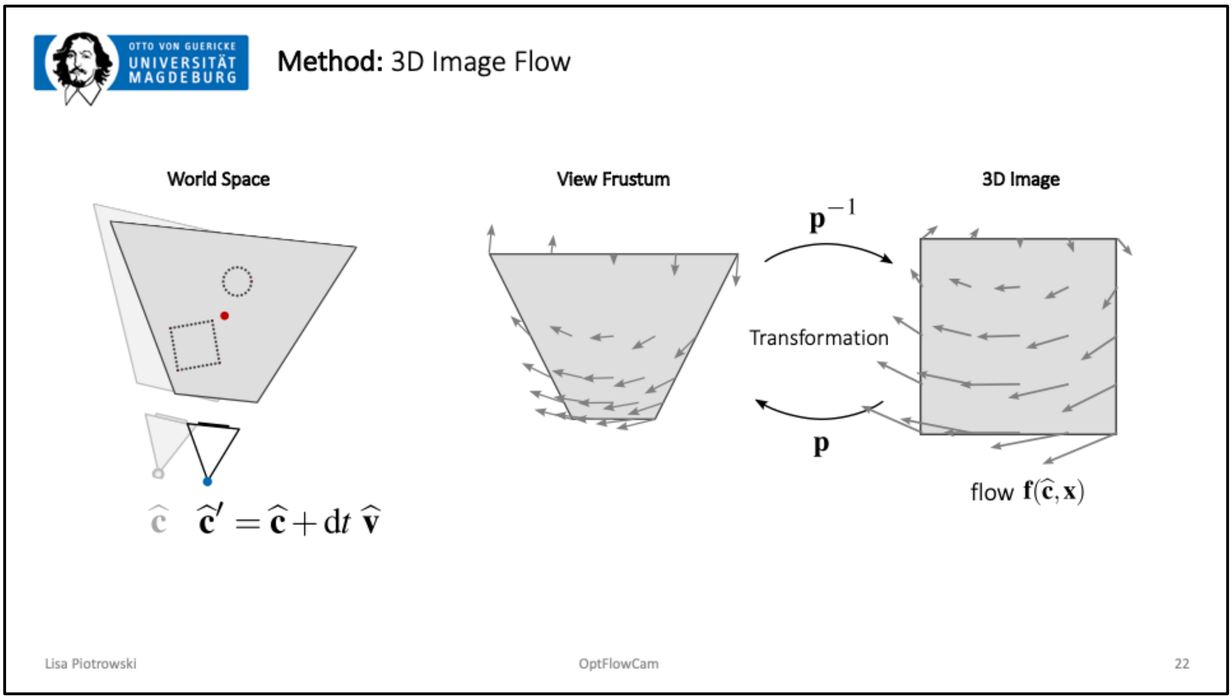
Method: 3D Image Flow



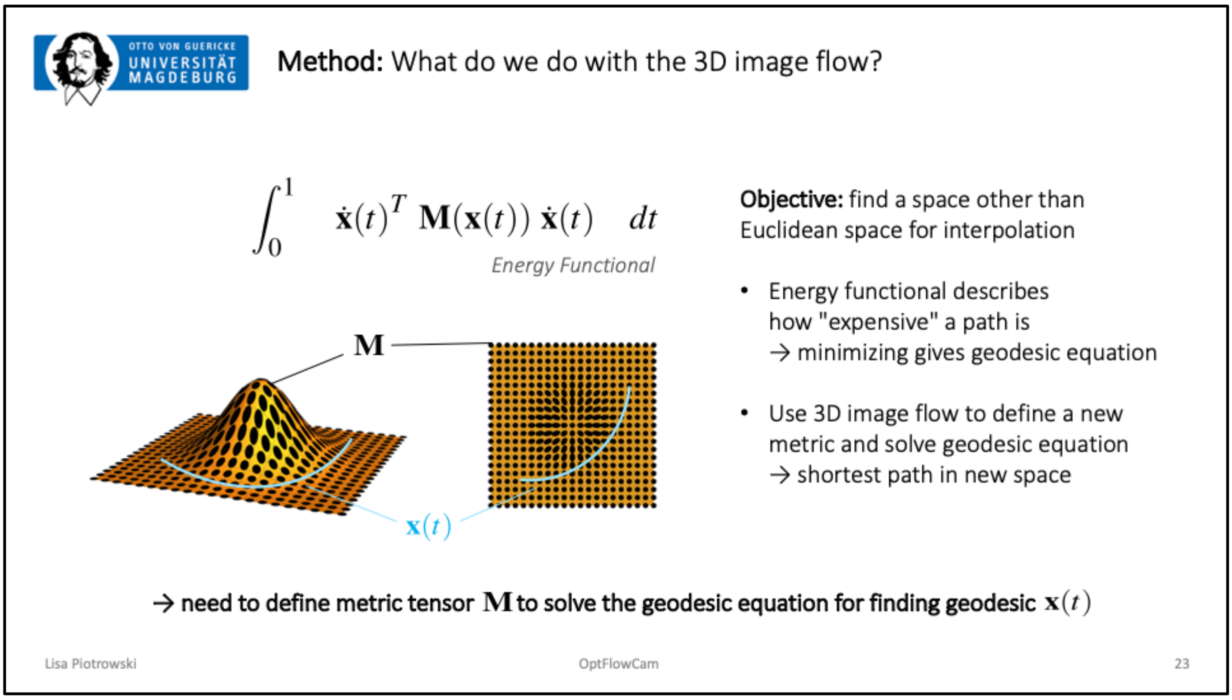
We start with a camera looking at a scene



Moving that camera a little bit changes the relative position of the scene objects in the view frustum. This change is also present in the 3D image if we transform the view frustum to it



By going infinitesimally small with the camera change (dt -> 0) we define a flow.



The flow can then be used in the energy functional. The energy functional can measure the length of a path (here x(t)) if given a metric (here M(t)). The metric M defines at every point of a space, how the local lengths look like (here, the ellipses representing the metric in the image say that going up or down the bump make the path longer because they are longer in that direction). So to minimize the path length (or energy we need to travel along it) we minimize the functional. That gives us the geodesic equations (a set of differential equations) which we can solve to get the shortest path (the geodesic).

For us this means we have a two step process:

- Define metric tensor M(x(t))
- Solve geodesic equation for x(t)

Method: 3D-Image-Flow-Based Metric

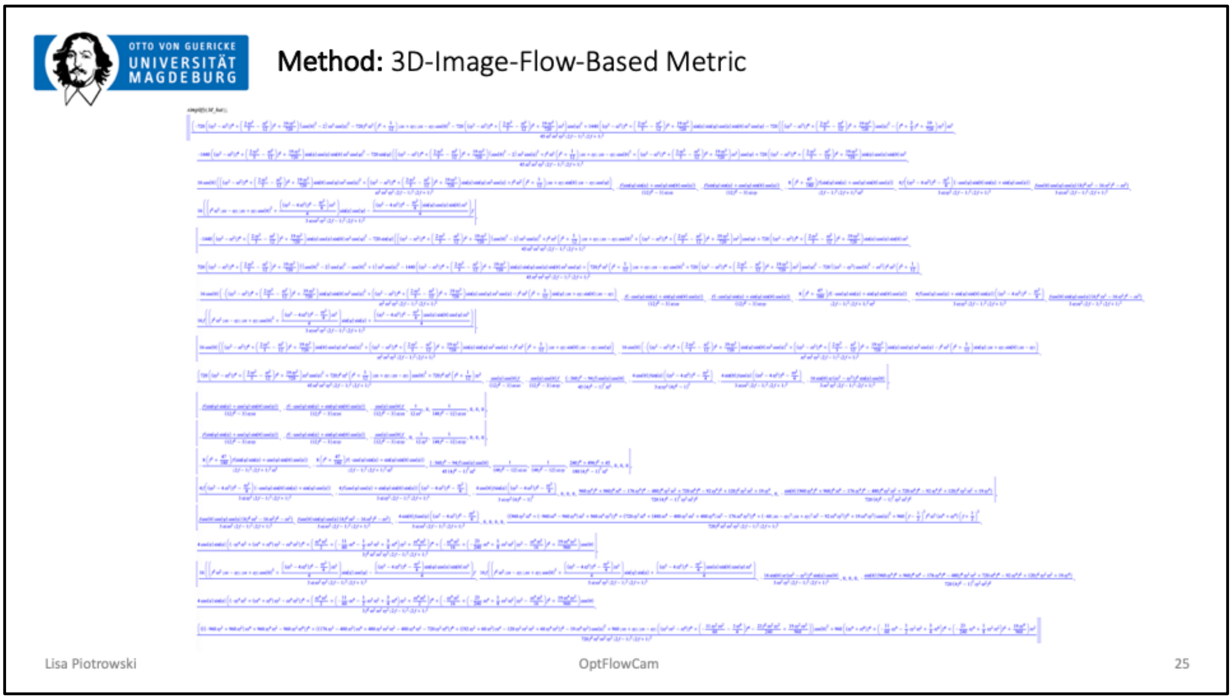
$$\left.\int_{\Omega} \mathbf{f}^T \mathbf{f} \quad d\mathbf{x}\right\} \text{Energy functional } \textit{(squared 3D image flow magnitude over whole 3D image volume)}$$

$$= \widehat{\mathbf{v}}^T \underbrace{\widehat{\mathbf{M}}(\widehat{\mathbf{c}})}_{\text{3D-Image-Flow-Based Metric}} \widehat{\mathbf{v}} \quad \Big\} \text{ can be written in terms of metric}$$

→ **has a** (complicated) **closed-form solution!**

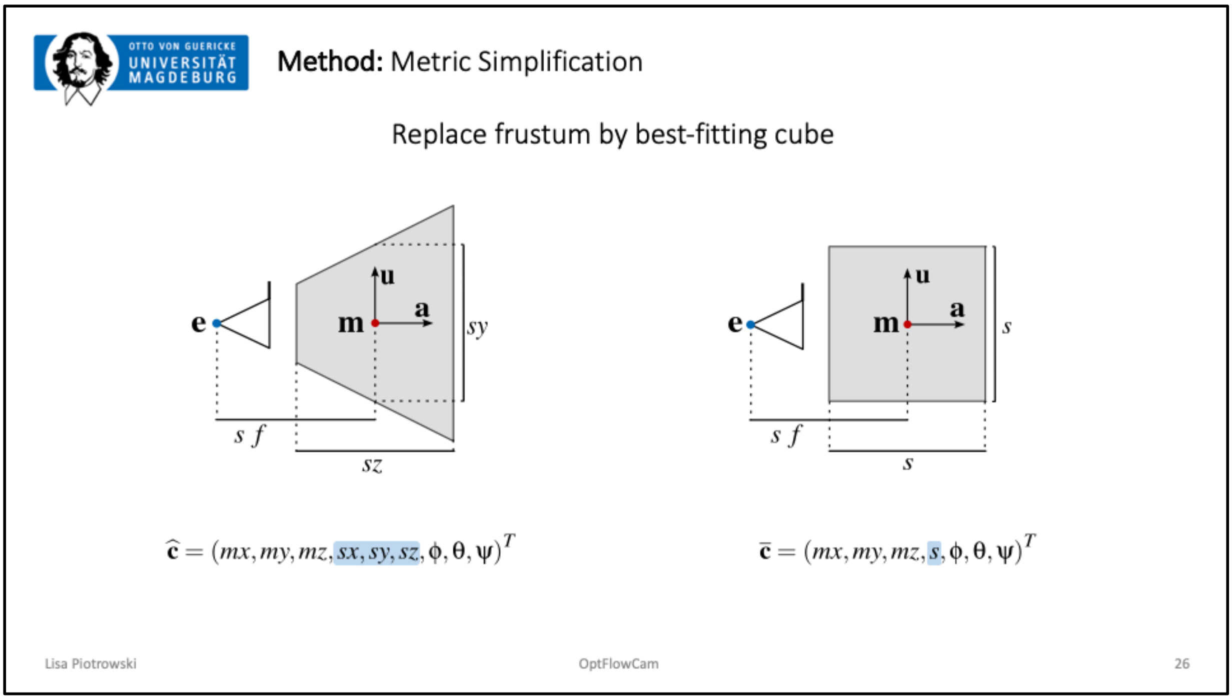
Lisa Piotrowski OptFlowCam 24

We use the flow magnitude to judge the length of the path. By expanding this, we can rearrange the terms to get out the metric which, with the current camera model, has a very complicated solution



Geodesic equation: second order differential equation

-> basically impossible to solve with this metric, so having the metric is only half the battle



Simplifying the camera space simplifies the metric a lot. We also tested that calculating the energy functional with simplified metric gives a similar result to the original metric -> can conclude the simplification is justified

Method: Metric Simplification

Replace frustum by best-fitting cube

$$
\overline{\mathbf{M}}(\overline{\mathbf{c}}) = \begin{pmatrix}
\frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{4s^2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{6} & 0 & -\frac{\sin\theta}{6} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\
0 & 0 & 0 & 0 & -\frac{\sin\theta}{6} & 0 & \frac{1}{6}
\end{pmatrix}
$$

translation + scale | rotation

→ **use metric to solve geodesic equation**
translation and scale can be handled independently of rotation

Have two independent blocks which allow us to solve the geodesic equation independently for each block

Method: Geodesics - Translation and Scale

$$\overline{\mathbf{M}}(\overline{\mathbf{c}}) = \underbrace{\begin{pmatrix} \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4s^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & -\frac{\sin\theta}{6} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & -\frac{\sin\theta}{6} & 0 & \frac{1}{6} \end{pmatrix}}_{\text{translation + scale}}$$

Method: Geodesics - Translation and Scale

Some helpful definitions

$$\begin{cases} d = \|\mathbf{m}_1 - \mathbf{m}_0\|, \quad \mathbf{d} = \dfrac{\mathbf{m}_1 - \mathbf{m}_0}{d} \\ b_i = \dfrac{s_1^2 - s_0^2 + (-1)^i\, 4d^2}{4\, s_i\, d} \\ r_i = \ln\left(-b_i + \sqrt{b_i^2 + 1}\right) \end{cases} \quad \text{for } i = 0, 1$$

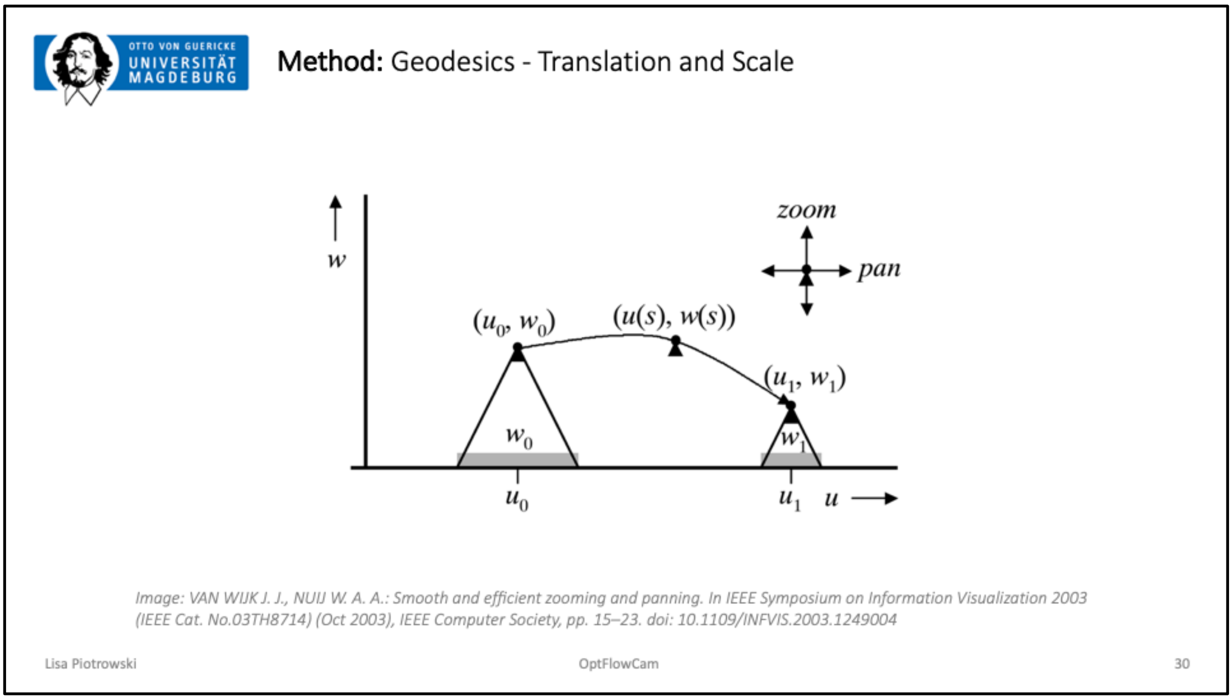
Scale changes faster
if current scale is higher
(i.e. camera further away from POI)

$$s(t) = \frac{s_0 \cosh(r_0)}{\cosh((r_1 - r_0)t + r_0)}$$

Frustum position changes faster
if current scale is high

$$\begin{cases} m(t) = \dfrac{s_0}{2} \cosh(r_0) \tanh((r_1 - r_0)t + r_0) - \dfrac{s_0}{2} \sinh(r_0) \\ \mathbf{m}(t) = \mathbf{m}_0 + m(t)\, \mathbf{d}. \end{cases}$$

→ slower movement when close to POI, faster movement when further away
zooming in/out in between viewpoints depending on distance and scale

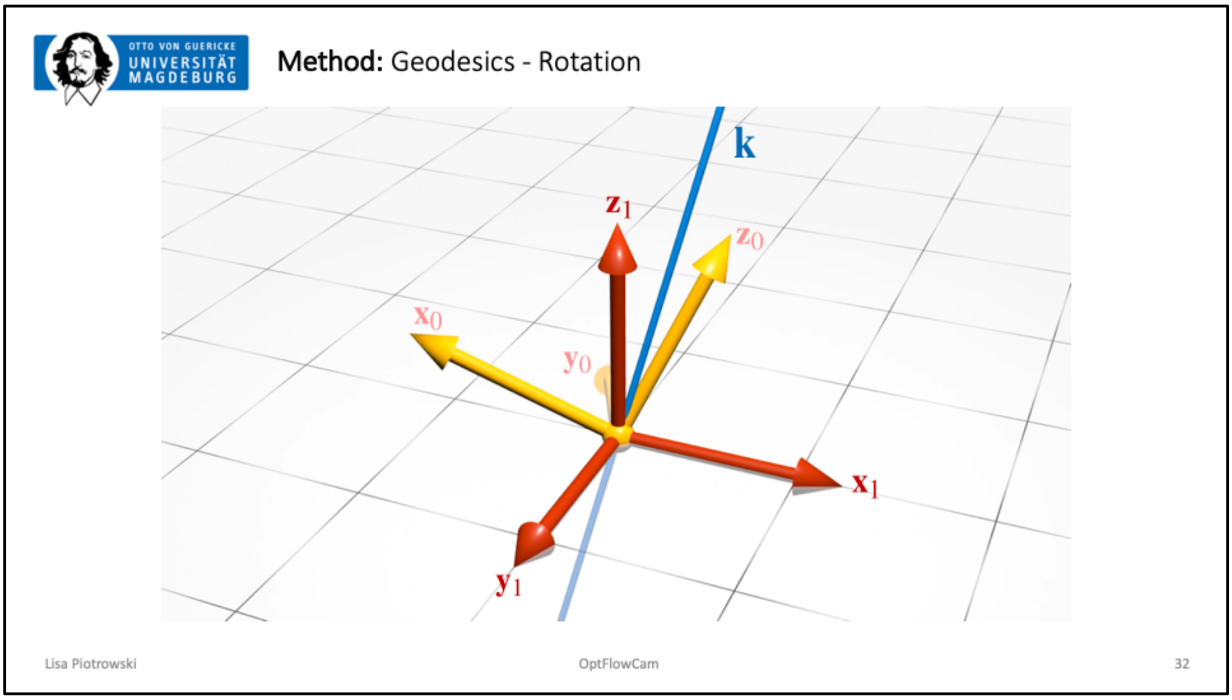


Very similar work to ours for 2D zooming and panning in images.

Method: Geodesics - Rotation

$$\overline{\mathbf{M}}(\overline{\mathbf{c}}) = \begin{pmatrix} \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4s^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & -\frac{\sin\theta}{6} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & -\frac{\sin\theta}{6} & 0 & \frac{1}{6} \end{pmatrix}$$

rotation



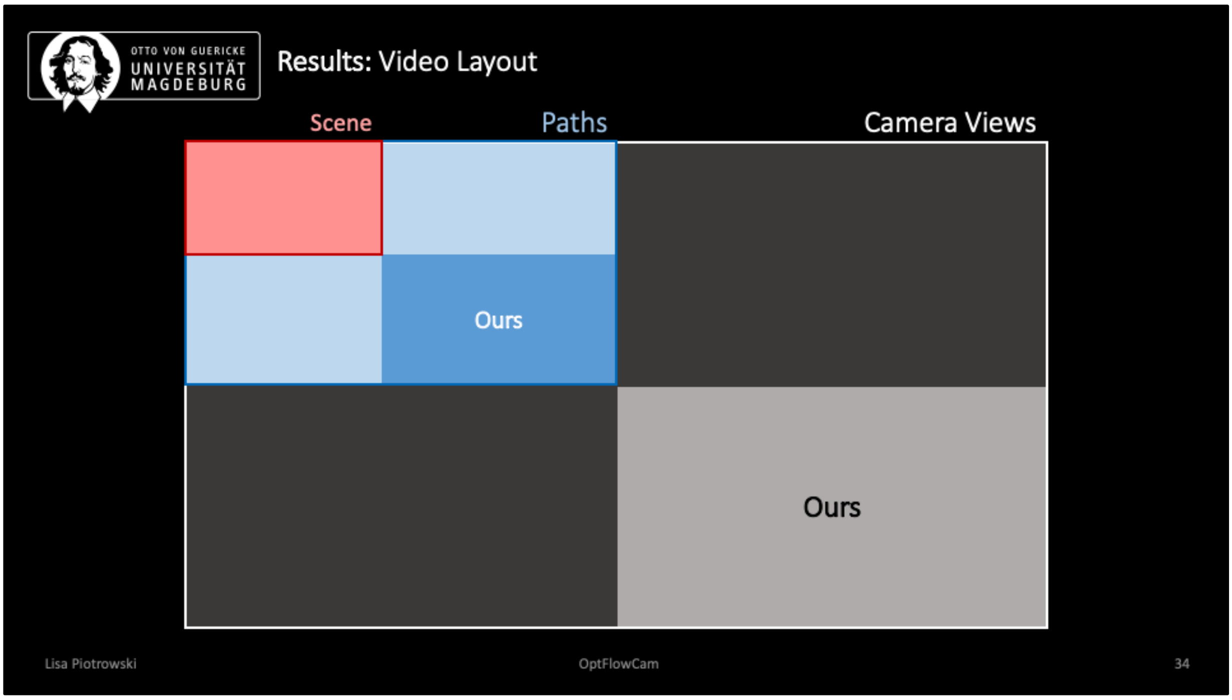
Rotation turns out to be a constant speed rotation around a fixed axis of rotation.
This axis of rotation (k) can be calculated by solving an eigenvalue problem in 3D.

Method: Geodesics - Rotation

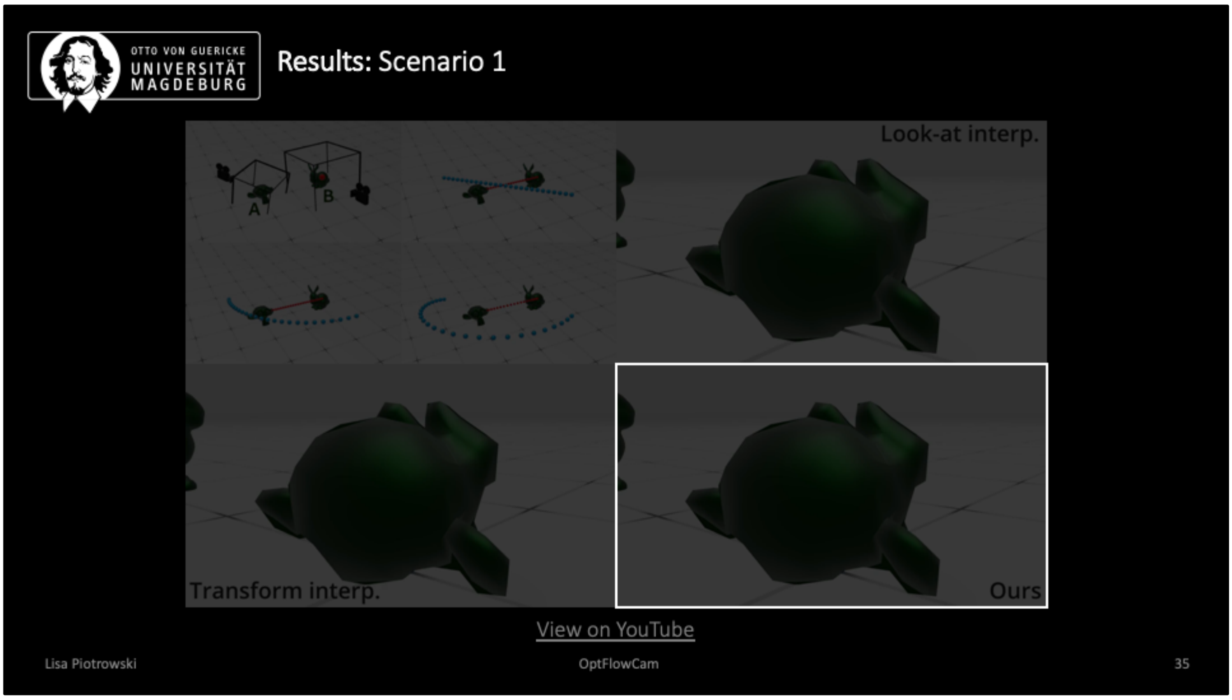
$$\mathbf{R}(t) = \overbrace{\mathbf{R_k}}^{\text{Rotation matrix transforming axis of rotation } \mathbf{k} \text{ to/from the z-axis (solution of an eigenvalue problem)}} \; \underbrace{\mathbf{R}_\xi(t)}_{\text{Rotation matrix rotating by an angle of } t\xi \text{ around the z-axis}} \; \overbrace{\mathbf{R_k}^T} \; \underbrace{\mathbf{R}_0}_{\text{Initial orientation matrix}}$$

$$\underbrace{\mathbf{R}(t)}_{\text{Final orientation matrix}}$$

→ **final rotation is around a constant axis with constant speed**

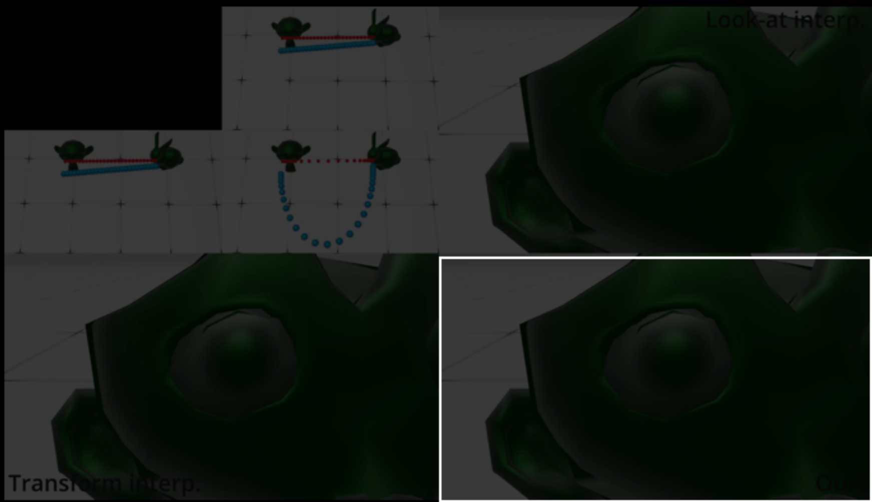


Layout of the video examples because there is so much going an in them that it’s easy to get overwhelmed.



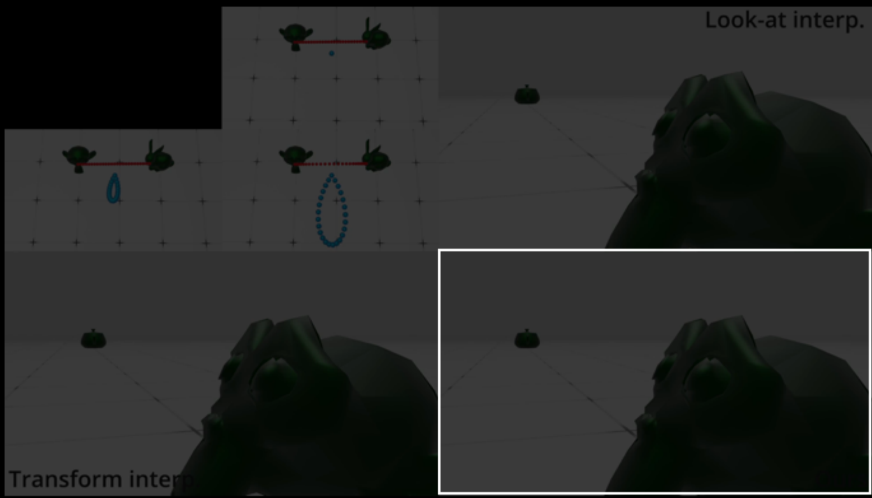
Nice zooming out/in in for out method. Alternative methods in the upper right and lower left loose sight of the scene objects

Results: Scenario 2



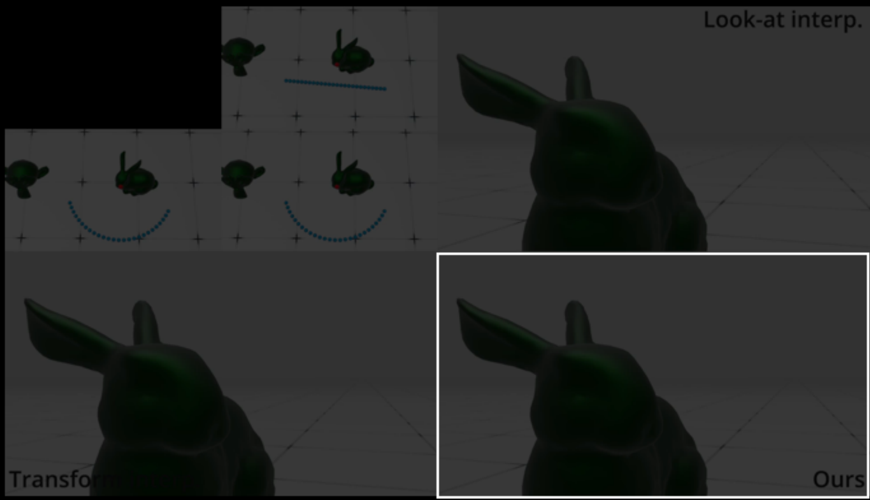
View on YouTube

Results: Scenario 3

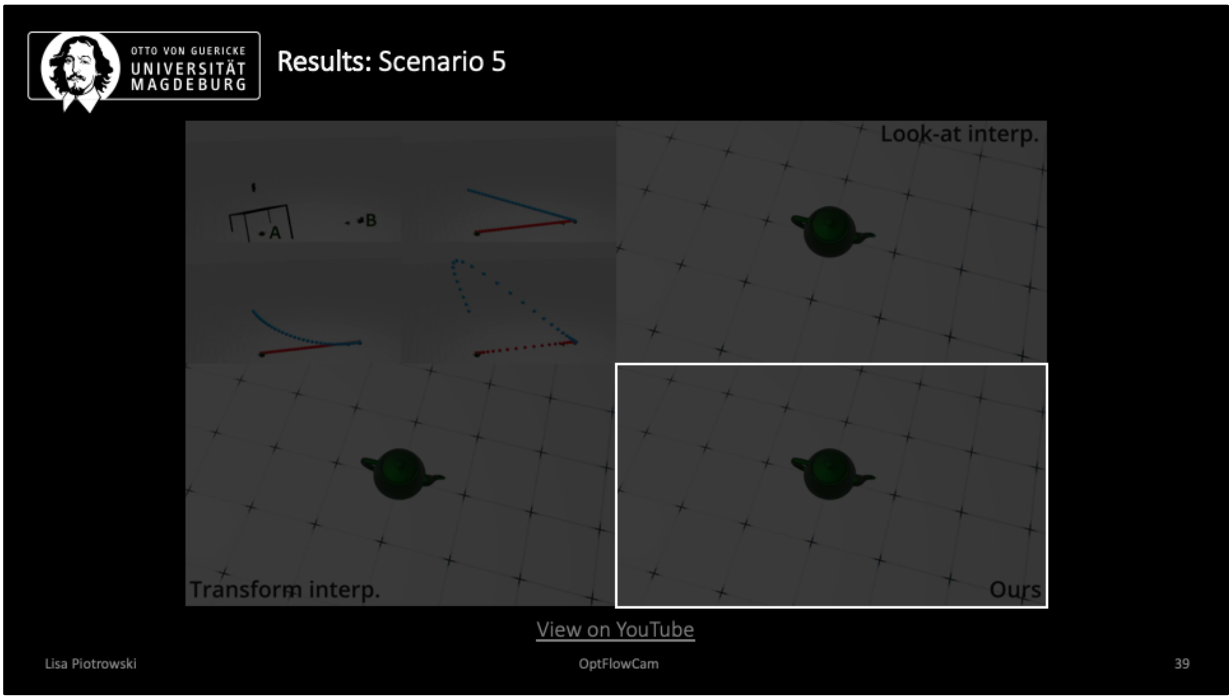


View on YouTube

Results: Scenario 4

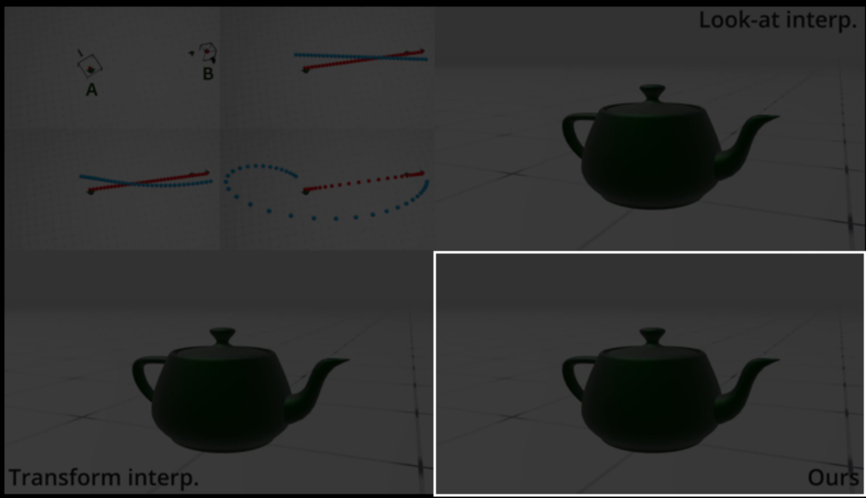


View on YouTube



Again nice zooming out/in for our method. We actually see the objects most of the time while the other methods just look at the floor.

Results: Scenario 6



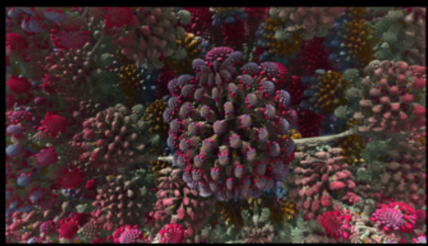
View on YouTube

Results: 3D Fractal

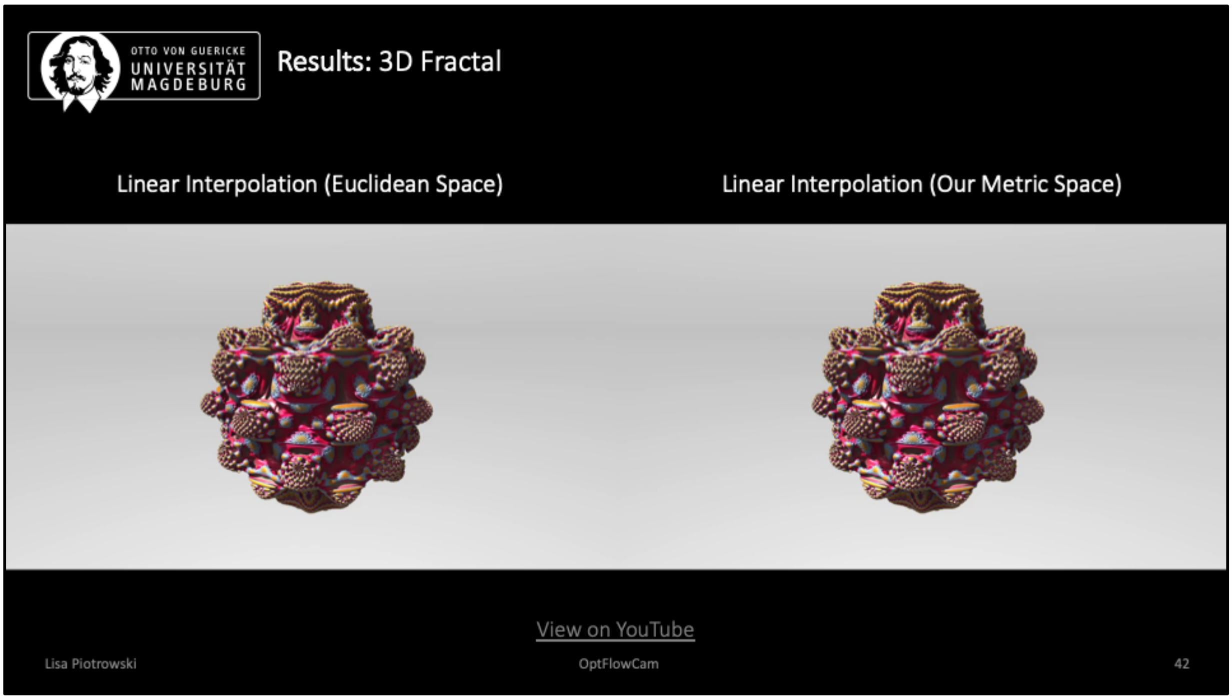
Keyframe Start

Camera distance to POI:
2 units

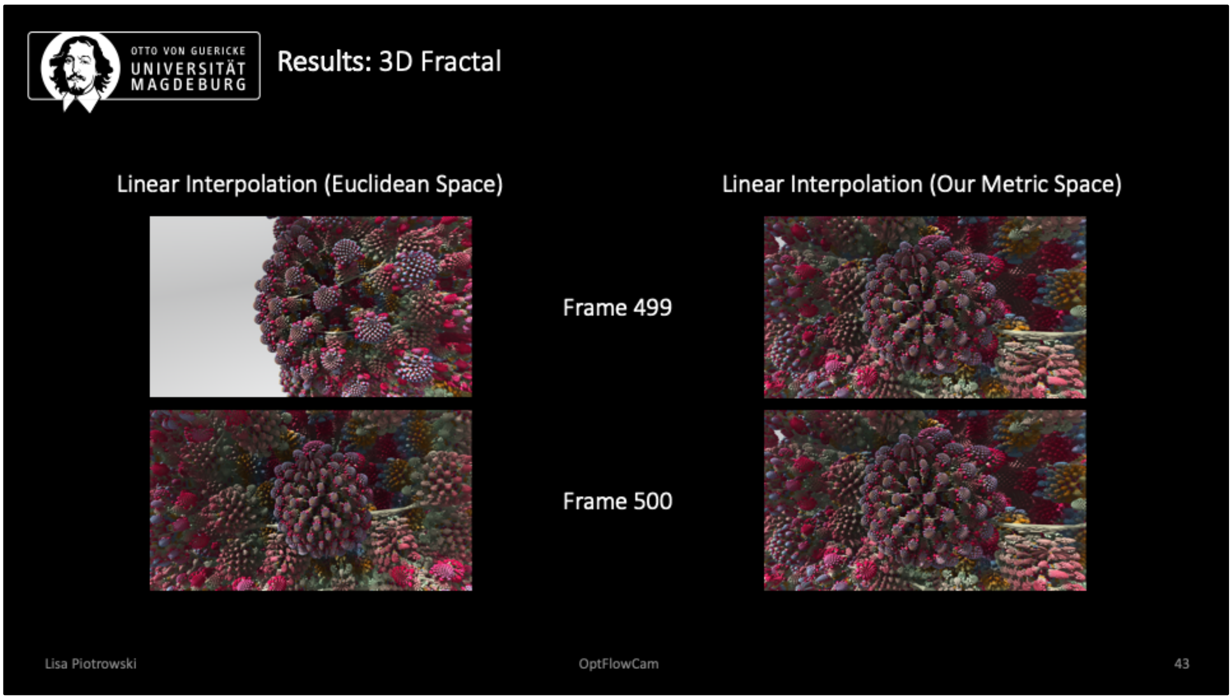
Keyframe End



Camera distance to POI:
1.4×10^{-12} units



Euclidean space interpolation looks like its not moving at all in the beginning and then “crashes” into the fractal. Our method gives an apparent constant speed impression

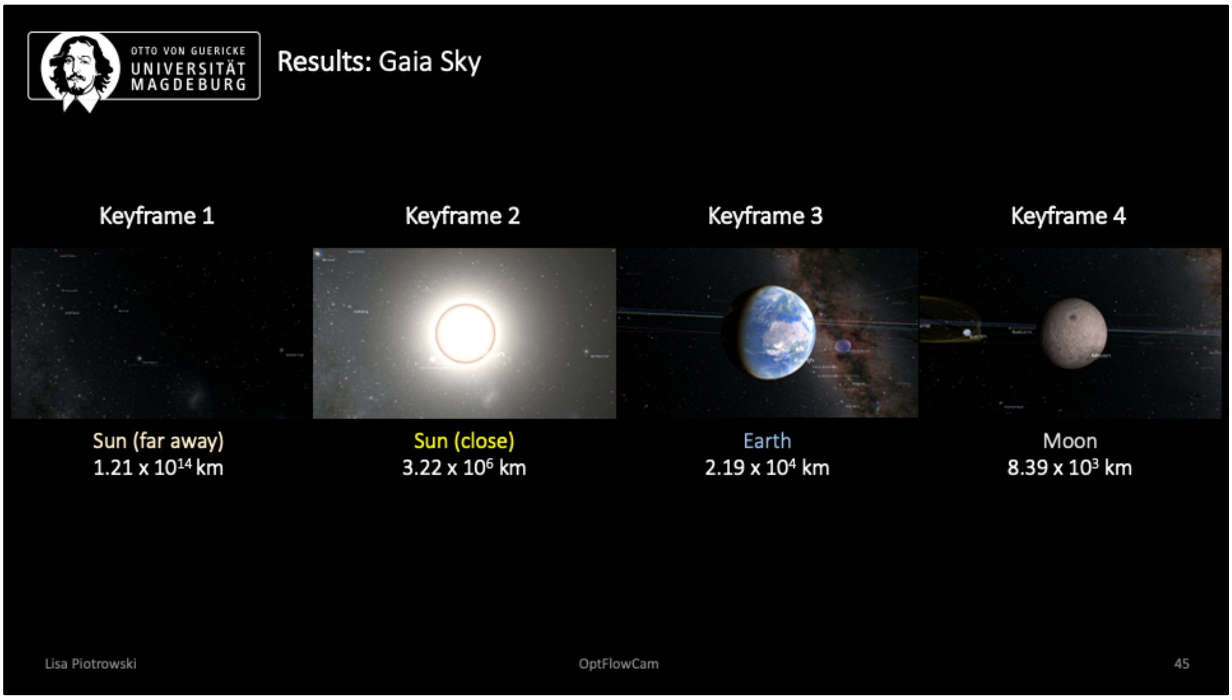


Comparison between the second to last and last frame of both methods. Euclidean method clearly has issues.

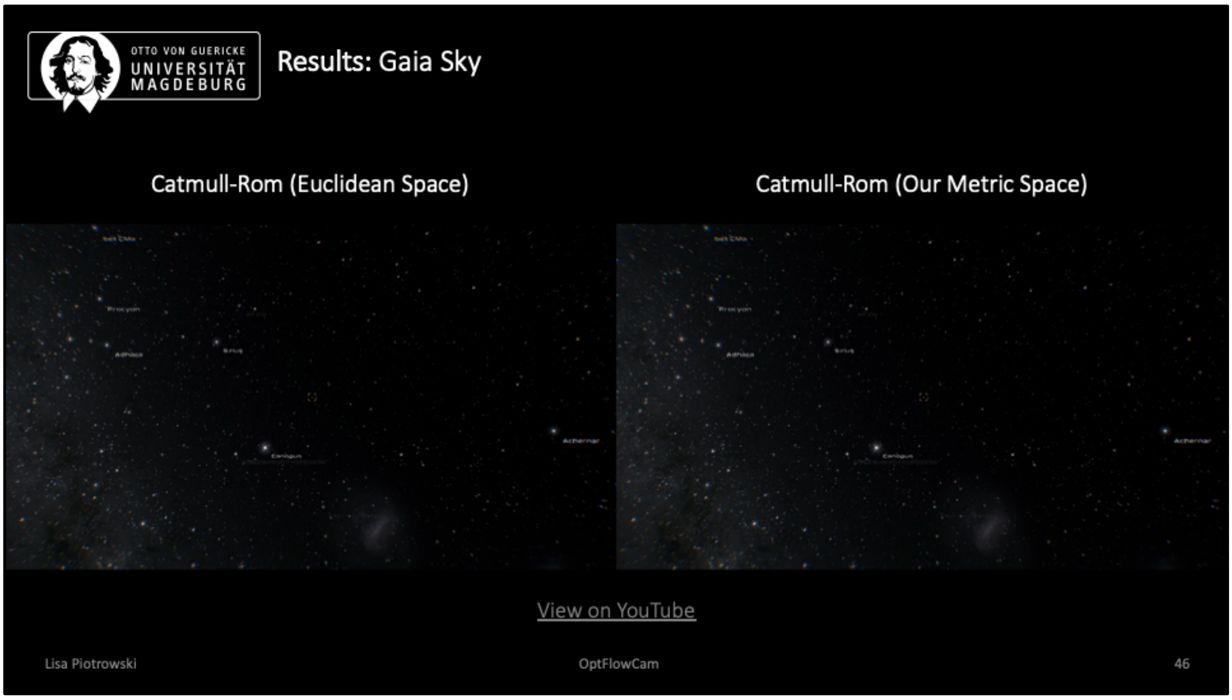
Method: Spline Interpolation

We can also chain multiple camera poses into one path.

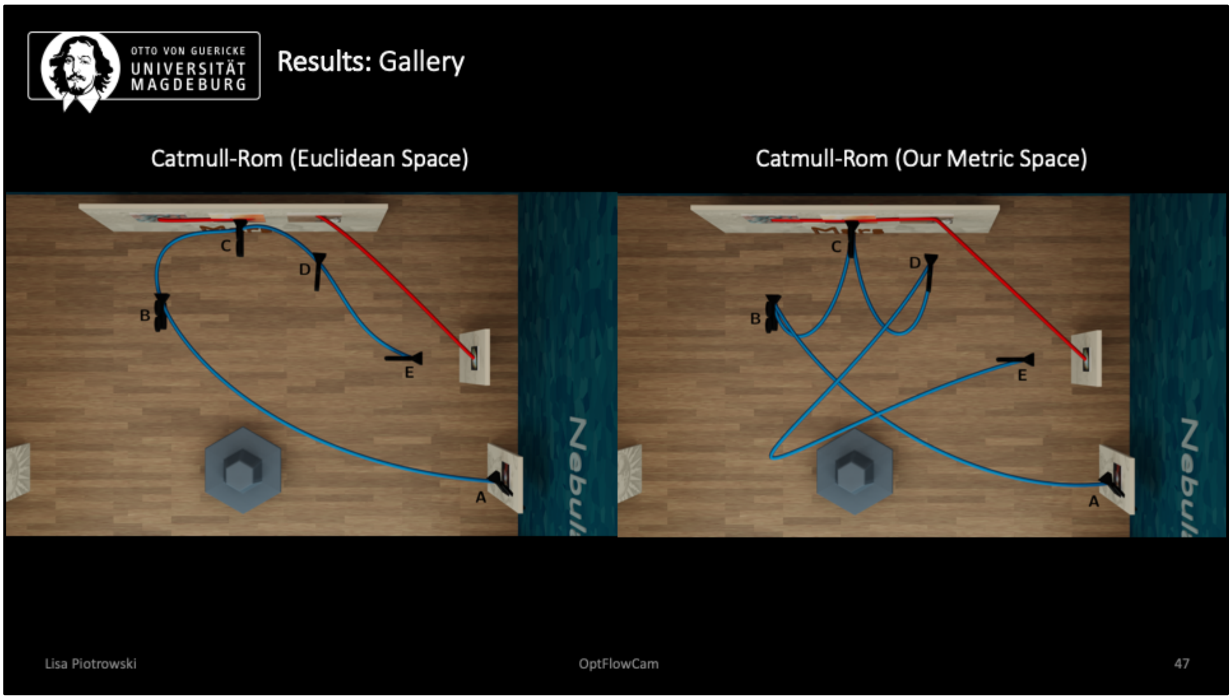
- In any spline evaluation scheme that is based on repeated linear interpolation we could substitute linear interpolation for our geodesics.
- We used Catmull-Rom curves in our examples.



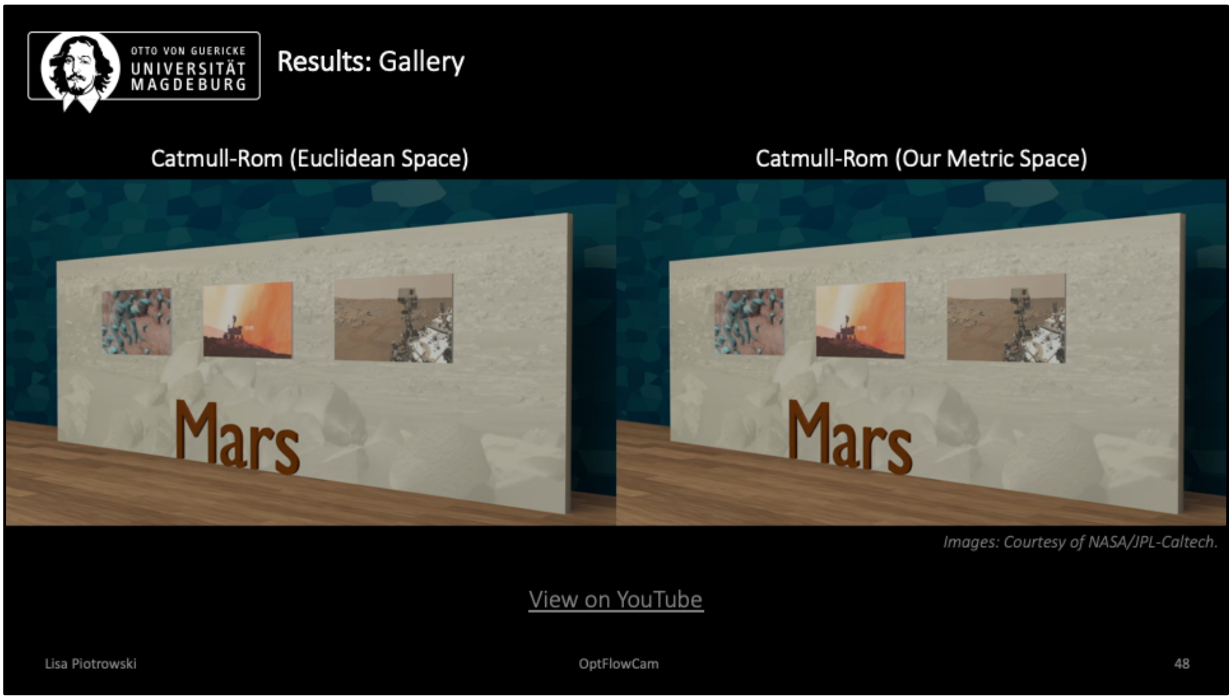
4 keyframe spline in Gaia Sky with varying distances



Euclidean method looks like it‘s not moving at the beginning then overshoots all the keyframe positions, so it‘s basically not possible to see anything of the interesting parts. Our method gives a nice path.



5 keyframe path in a gallery. Euclidean space path slides along the pictures in the gallery so it’s basically impossible to see anything. Our path looks like it would not be smooth but because the camera moves very slowly at the keyframe positions, it still looks smooth


OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG
Results: Gallery
Catmull-Rom (Euclidean Space)
Catmull-Rom (Our Metric Space)
Mars
Mars
Images: Courtesy of NASA/JPL-Caltech.
View on YouTube
Lisa Piotrowski
OptFlowCam
48

Results: Mars Rover

View on YouTube

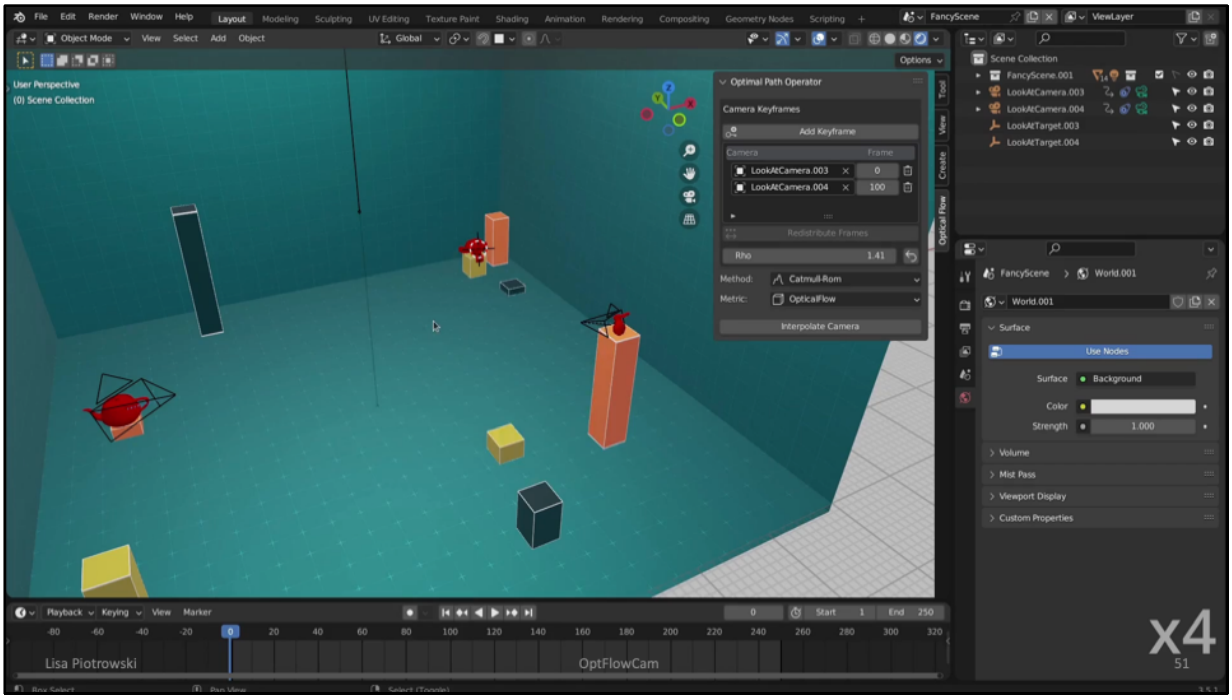
Conclusion

We introduced a new camera metric space that…

- … is **simple** to use
- … gives us **closed-form geodesics**
- … can handle **extreme scale variations** in scenes
- … can be used in **interactive contexts**

Lisa Piotrowski OptFlowCam 50

Of course we also have limitations which we discuss in the paper

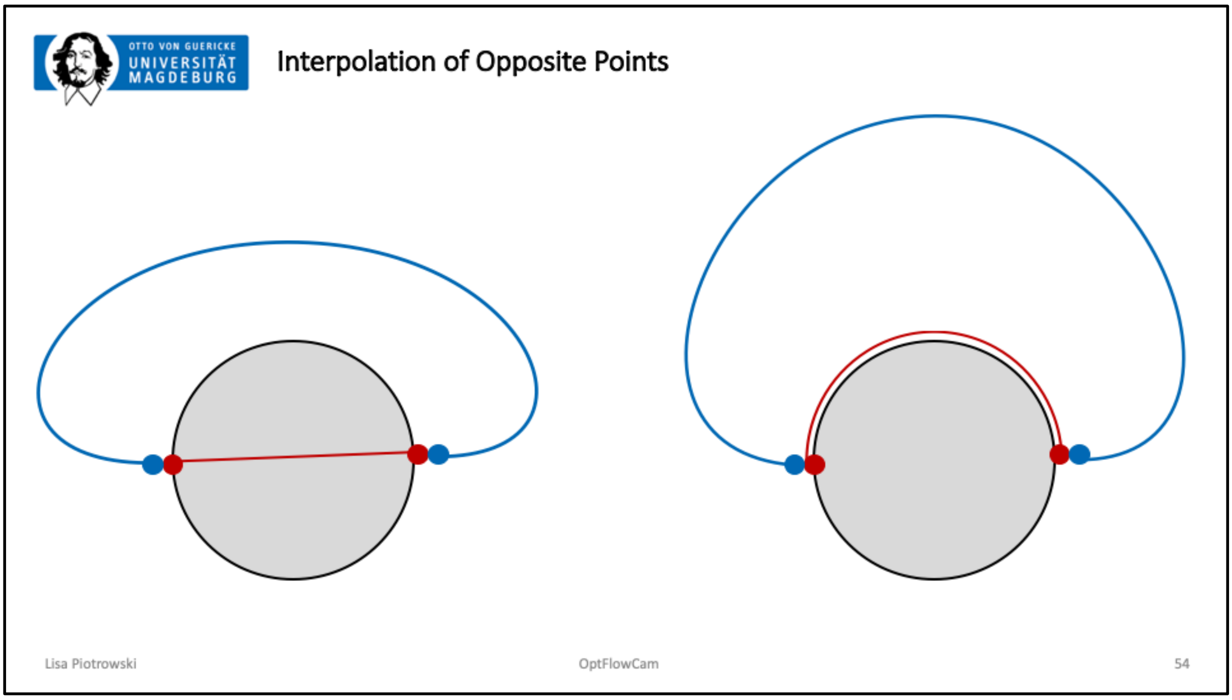


Blender plugin demonstration


EUROGRAPHICS
CONFERENCE 2024
OptFlowCam
A 3D-Image-Flow-Based Metric
in Camera Space
for Camera Paths in Scenes
with Extreme Scale Variations
https://livelyliz.github.io/OptFlowCam
OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG
Lisa Piotrowski, Michael Motejat
Christian Rössl, and Holger Theisel

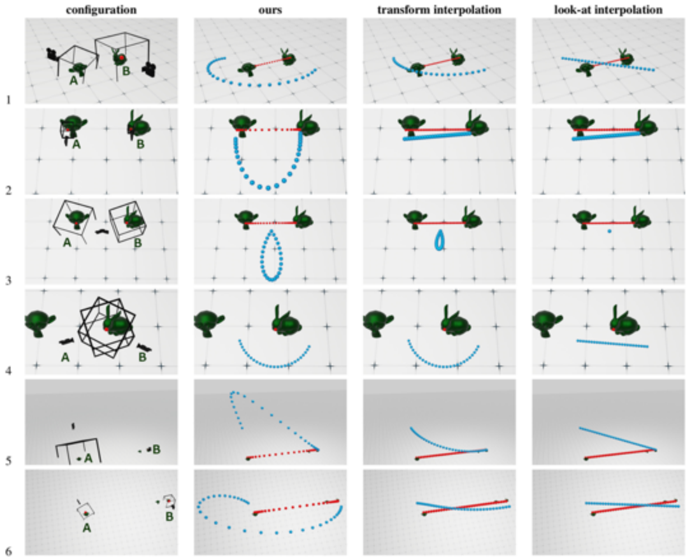
Method: Length of Geodesic Path

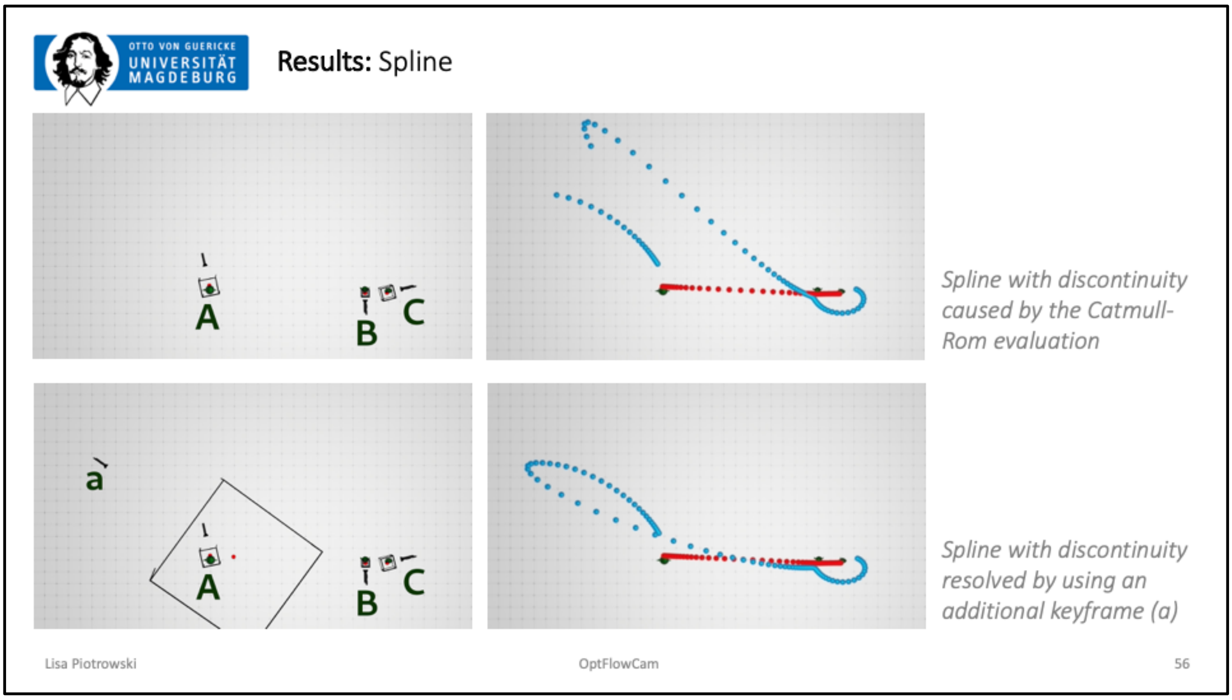
$$\mathrm{dist}(\overline{\mathbf{c}}_0, \overline{\mathbf{c}}_1) = \int_0^1 \sqrt{\dot{\overline{\mathbf{c}}}(t)^T \, \overline{\mathbf{M}}(\overline{\mathbf{c}}(t)) \, \dot{\overline{\mathbf{c}}}(t)} \quad dt$$
$$= \sqrt{\frac{(r_1 - r_0)^2}{4} + \frac{\xi^2}{6}}.$$



One limitation is that we do not consider any shape of an object. So if we have opposite points of the sphere, the path might not ”zoom out” enough to be pleasant. We would rather want something as on the right

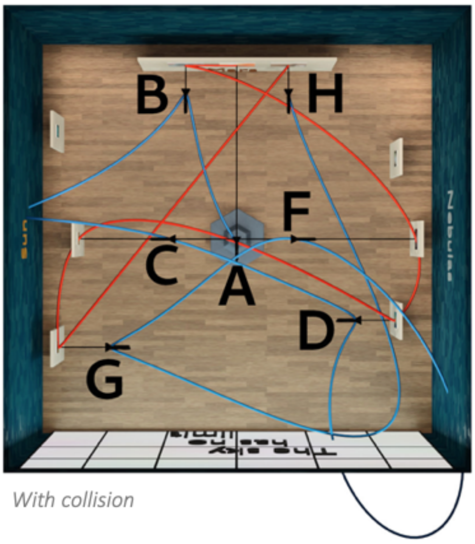
Results: Simple Scenes



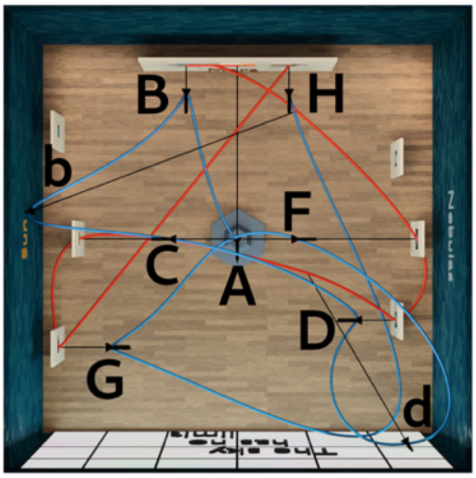


Another limitation with Catmull-Rom curves creating gaps

Results: Gallery

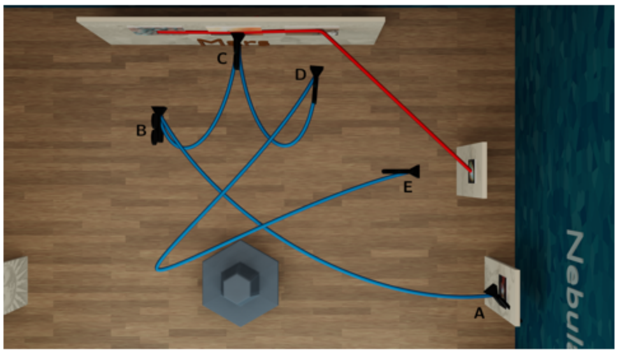


With collision

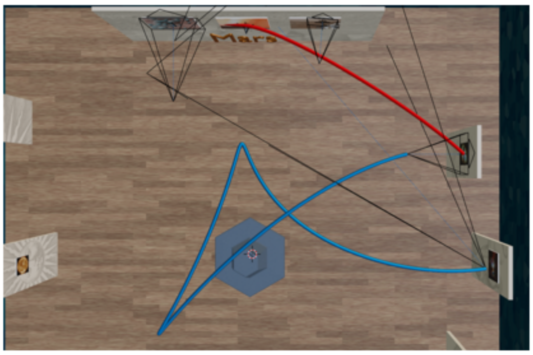


Without collision (and additional keyframes b + d)

Results: Different Spline Curves

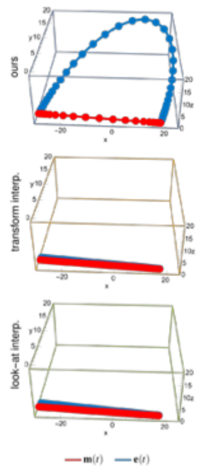


Catmull-Rom

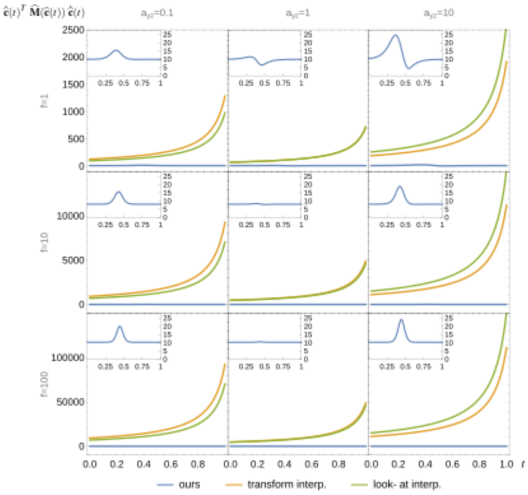


Bézier Curve

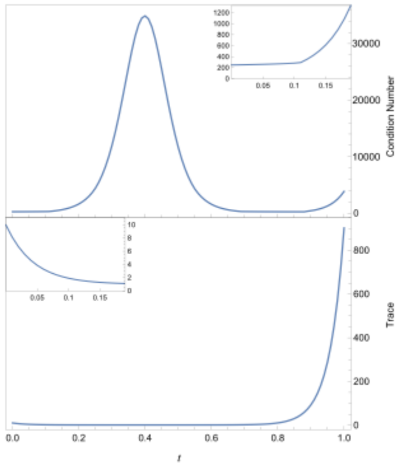
Results: Metric and Velocity Along Path



Paths



Average Squared Flow Magnitude



Condition Number and Trace

Results: Scenes

View on YouTube

Results: Interactive Blender Session

[View on YouTube]