DOI: 10.1111/cgf.15056
EUROGRAPHICS 2024 / A. Bermano and E. Kalogerakis
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 2

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

Otto von Guericke University Magdeburg, Germany
https://livelyliz.github.io/OptFlowCam/

Abstract

Interpolation between camera positions is a standard problem in computer graphics and can be considered the foundation of camera path planning. As the basis for a new interpolation method, we introduce a new Riemannian metric in camera space, which measures the 3D image flow under a small movement of the camera. Building on this, we define a linear interpolation between two cameras as shortest geodesic in camera space, for which we provide a closed-form solution after a mild simplification of the metric. Furthermore, we propose a geodesic Catmull-Rom interpolant for keyframe camera animation. We compare our approach with several standard camera interpolation methods and obtain consistently better camera paths especially for cameras with extremely varying scales.

CCS Concepts
• ***Computing methodologies*** → ***Computer graphics; Perception;*** • ***Human-centered computing*** → *Interaction techniques;*

1. Introduction

Camera paths are used to convey information about a scene or object in both 2D and 3D scenes. This task is a common one, be it in scientific and commercial visualization or computer games, static or interactive environments. Especially when connecting multiple viewpoints, it is vital not to disorient the viewer and to provide context about where the camera is and where it is going so that the viewer can build a mental map of the scene. This can be done, e.g., by zooming out on a detail, showing an overview, and then zooming in on another detail. For 2D images and documents, this has been done by van Wijk and Nuij [vN03], but there is currently no equivalent method for the common case of 3D scenes.

We consider camera path construction as a mixed automatic and interactive process: the user specifies a sequence of camera positions (key frames) from which a "good" camera path is automatically constructed. Several criteria for the quality of a camera path have been proposed, such as smoothness, or the minimization of arc length, total curvature, bending energy, acceleration, or other measures. In addition, further criteria from cinematography can be considered to incorporate aesthetic aspects of a path. This makes camera path construction a multi-objective optimization problem where parts of the objectives cannot be described in a purely mathematical way.

A common approach is to consider piecewise polynomial spline curves as camera paths. They are naturally smooth, minimize certain energies like combinations of arc length and simplified bending energy, and come with a simple representation of intuitive control points [Far02]. Polynomial spline curves can be described by repeated linear interpolation of control camera points (e.g., by applying De Casteljau's algorithm for Bézier curves or De Boor's algorithm for B-Spline curves). Since the linear interpolation requires a metric to be computed, spline curves in the camera space depend on an underlying metric in that space. Usually, this metric is assumed to be the Euclidean metric.

In this paper, we introduce a new Riemannian metric in camera space that measures the 3D image flow in the 3D image space induced by the movement of the camera (Section 4). This measure is related to, but not identical with, optical flow. Based on this, we consider the linear interpolation between two cameras as the camera path with minimal 3D image flow, i.e. the shortest geodesic in the new metric. Since the computation of geodesics as boundary value problem is numerically challenging, we introduce a mild simplification of the metric (Section 4.3) which allows the computation of geodesics in a closed form (Section 4.4). With this, we propose an interpolating Catmull-Rom spline where linear interpolation is replaced by geodesics computation (Section 4.5).

Optical flow is known to be related to the perception of self-motion [WH88] and a factor in causing cybersickness in virtual reality [WR22]. So, the motivation of our approach is the assump-

© 2024 The Authors. Computer Graphics Forum published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.
This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

tion that by minimizing the flow in the 3D image, we achieve a perceptually favorable camera path.

As we will show in our results, our metric also accomplishes that the camera provides context by zooming. This is especially critical for camera paths involving extreme change of scale, e.g., when the camera moves from a strong close-up in one part of the scene to either a close-up in another part or a global overview of the scene. We show that our approach gives a significantly better camera path than in an Euclidean camera space, in particular for paths with extreme scaling.

2. Background

Before we go into detail about our method, we want to give a short review of metric spaces and geodesic paths. For more information and a more in depth introduction, we refer the reader to literature on the topic of Riemannian geometry [DC92,Pet06] and differential geometry [Str50].

In the context of this work, a metric space is a differentiable manifold that is equipped with a Riemannian metric tensor $\mathbf{M}(\mathbf{x})$ which depends on the point in the manifold $\mathbf{x}$. In other words, it is a *locally* Euclidean space with an object that allows us to measure lengths and angles in that space. One example is the standard Euclidean space $\mathbb{R}^n$ with its metric tensor the identity matrix $\mathbf{M} = \mathbf{Id}$. A less trivial example is the sphere.

If we have a parametrized curve $\mathbf{x}(t)$ on the manifold that is defined on the parameter interval $t \in [0,1]$, then the length of the curve given the metric tensor $\mathbf{M}$ is

$$\int_0^1 \sqrt{\dot{\mathbf{x}}(t)^T\, \mathbf{M}(\mathbf{x}(t))\, \dot{\mathbf{x}}(t)} \quad dt, \tag{1}$$

where $\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt}$. This is the well known formula for arc length of parametric curves in Euclidean space. Now, to connect two points on the manifold, we often want to do so by using the “straightest” path with (locally) minimal length, often referred to as a geodesic path. While a geodesic does not necessarily has to be of minimal length [Str50, p. 140], geodesics are minimizers of a variational problem involving the squared arc length energy functional. The variational problem we need to solve for is as follows: Given are two points on the manifold $\mathbf{x}_0$ and $\mathbf{x}_1$. We need to find a curve $\mathbf{x}(t)$ such that $\mathbf{x}(0) = \mathbf{x}_0$, $\mathbf{x}(1) = \mathbf{x}_1$ and the energy

$$\int_0^1 \dot{\mathbf{x}}(t)^T\, \mathbf{M}(\mathbf{x}(t))\, \dot{\mathbf{x}}(t) \quad dt, \tag{2}$$

is minimal. The solution does not only give a geodesic [Pet06, p. 116] but a geodesic in equal-speed (or arc length proportional) parametrization [DC92, p. 194], i.e. $\dot{\mathbf{x}}(t)^T\, \mathbf{M}(\mathbf{x}(t))\, \dot{\mathbf{x}}(t)$ is constant for all t. This would in fact not be the case if we use Eq. (1) instead of Eq. (2).

The solution then satisfies the geodesic equation [Pet06], here given in matrix notation as

$$\ddot{\mathbf{x}} = \frac{1}{2}\,\mathbf{M}^{-1}\left(\nabla(\dot{\mathbf{x}}^T \mathbf{M}\, \dot{\mathbf{x}}) - 2\,(\nabla \mathbf{M}\, \dot{\mathbf{x}})\, \dot{\mathbf{x}}\right). \tag{3}$$

In general, there is no closed-form solution to this boundary value problem. Even finding a numerical solution can be hard depending on the behavior of the metric. Because of this, simplifications to the problem space can vastly improve the chances to find a solution which we leverage in our work.

3. Related Work

Keyframes are established in 3D modelling and animation software for defining a camera path as a sequence of frames with fixed attributes such as position or orientation of the camera. Interpolation techniques, e.g., Catmull-Rom or Bézier splines, are then used to compute the attributes in the intervals between the keyframes. Designing camera paths using this technique is typically intuitive because the control polygon is a very coarse version of the path. It also offers a great deal of creative freedom within certain constraints to ensure smoothness. However, creating a good camera path requires a skillful animator. The challenges lie in interpolating rotations and selecting an appropriate velocity that prevents viewer discomfort. This is particularly relevant when the keyframes exhibit vastly different scales, for instance when zooming in or out on features. As our concept can also allows keyframe interpolation, it can be used as a drop-in alternative for traditional keyframe approaches. It can be used similarly to standard keyframing and is fast enough to allow for interactive editing of the path.

Keyframe interpolation can be accomplished through various methods. The simplest approach is linear interpolation of each attribute. Although this may work well for the camera position, it often results in undesired outcomes for the camera orientation. Due to the fact that interpolating rotation angles does not typically equate to a rotation around a fixed axis, the result can be unintuitive.
A better option would be, i.e. to interpolate the positions the camera should look at and the position of the camera itself linearly. This way, the orientation of the camera is determined by the two positions and an additional, globally chosen vector that defines the upright position. However, this can also lead to awkward camera poses when the path of the view target and the camera position pass close to each other.
Another method is similar to the work of Alexa [Ale02]. It involves interpolating the *transformations* linearly opposed to linear interpolation of the *attributes* defining these transformations. This has the advantage that, e.g., the rotation is performed around a fixed axis. We will address the specific transformations when we compare our method to alternatives in Section 6.

Other automatic path generation techniques go beyond simple interpolation. They often optimize or generate paths based on different requirements such as cinematography [MC00, CL03, NAD*17, GH21], saliency [AVF04, XYH*18], or physical properties of the path [JRT*15, GHN*16, XYH*18]. Some algorithms even try to optimize a given camera path so that the optical flow in the resulting camera image is reduced [AA10, HLH*16]. These techniques often utilize numerical optimization algorithms and depend on the visible scene. This means that they have to be recalculated when the scene changes even if the keyframes did not.
In contrast, our approach is only partially dependent on the scene by the use of a look-at target which can even be defined in the absence of scene geometry. Moreover, its closed-form solution is more akin to an interpolation scheme although it still originates from an optimization problem.

Dim.	Space	Notation
1D	scalar	s, t
3D	Euclidean space	$\mathbf{x}, \mathbf{v}, \mathbf{M}$
4D	Euclidean homogenous coord.	$\tilde{\mathbf{x}}, \tilde{\mathbf{v}}, \widetilde{\mathbf{M}}$
9D	camera space	$\hat{\mathbf{x}}, \hat{\mathbf{v}}, \widehat{\mathbf{M}}$
7D	simplified camera space	$\bar{\mathbf{x}}, \bar{\mathbf{v}}, \overline{\mathbf{M}}$

Table 1: *Notation. Lower case letters denote scalars or vectors while upper case letters denote matrices.*

One approach that achieves smooth paths without an optimization routine but still incorporates scene and composition information is that of Lino and Christie [LC15] which works in a dimensionally reduced camera space, the toric space. The interpolation of multiple viewpoints is a non-linearly interpolated blend of two linearly interpolated curves The idea of the toric space was also used in the work of Galvane et al. [GCLR15] to put the camera on a "rail" to smoothly track a target object in a scene.
Similar to our approach, the user needs to specify what will be seen on screen even though our method is less specific in regard of what object will be seen where in the viewport. While the viewpoint interpolation in [LC15] can be customized to increase and decrease speed at the start and end of the curve, our interpolation technique does this inherently without needing further adjustments. This is an advantage especially in scenes with large change of scale.

The most significant work that is related to ours is that of van Wijk and Nuij [vN03]. They propose a solution for a geodesic path for zooming and panning in large documents or images, which can be thought of as a 2D camera space. In summary, the resulting motion can be described by the relation of zooming level and horizontal movement velocity. The further away the imaginary camera is from the image plane, the faster it can pan. It also reveals that the optimal zooming velocity is exponential to the current zoom level (or distance to the image plane).
Our method shares the same principle idea and extends it by camera rotation, which makes it suitable for 3D scenes and more general camera paths.

Geodesic paths also play a role in other areas of computer graphics. A notable one is relativistic ray tracing, where the assumption that light travels in a straight line is no longer true. This can, for instance, be used to render light phenomena around black holes [DKC*22, HDL22]. There exist several different metrics to describe the relativistic properties which all result in different geodesics in these areas and therefore different renderings.
However, finding solutions for geodesics in those contexts is simpler than in our case because it is an initial value problem (point and direction) instead of a boundary value problem (point to point). Thus, we cannot apply the same methods to calculate our geodesic paths.

4. Method

In this section, we will first explain the camera model as a 9D vector describing the position, size, and orientation of the camera frustum. Then we define the camera space metric as a measure of the flow inside the 3D image induced by the camera motion. Since the original camera space is too complicated to solve for geodesic paths, we simplify the problem by replacing the frustum with the best-fitting cube, which gives us a simpler metric. The simplified problem then allows us to find a closed-form solution for geodesic paths. Finally, we explain how we can interpolate more than two points in camera space using our results and a recursive scheme for Catmull-Rom splines. The notation used in this section is summarized in Table 1.

4.1. Camera Model

In computer graphics, several approaches and naming conventions exist to define a camera and its transformations. In the viewing pipeline, a view frustum undergoes a projective transformation, converting it into a canonical view volume - a 3D cube aligned with the camera axes. This canonical view volume is then orthographically projected to a 2D image. While different rendering pipelines assign various names and spatial extents to the canonical view volume (e.g., "clipping space" in OpenGL with $[-1,1]^3$), we refer to it as the "3D image" with an extension of $[-1/2,1/2]^3$. Note that the extent does not influence the shape of the final geodesics. Here we are only interested in the transformation from the view frustum to the 3D image, as this sets the camera parameters.

We also want to clarify the differences and connections between optical flow and 3D image flow. While optical flow is a flow field defined in the *2D image space* of the final projected image, we use the concept of a *3D image space* for 3D image flow. As already mentioned, 3D image is just another name for the 3D canonical view volume. This means, conceptually it is the step of the viewing pipeline right before the projection to the 2D image. Simply put, optical flow is the projection of the 3D image flow that occurs on visible surfaces. Choosing the 3D flow field in the whole volume over the optical flow has the advantage that we can make simplifications to the transformations we use and are not subject to changes in surface visibility or scene geometry.

The camera model we are using to define our camera space is based on a standard pinhole camera model and is illustrated in Fig. 1. Usually, such a camera model is defined in terms of position of the camera center (or eye point), the orientation of camera and an internal camera parameter like focal length or opening angle. We will use a 9-dimensional camera model

$$\hat{\mathbf{c}} = (mx, my, mz, sx, sy, sz, \phi, \theta, \psi)^T \tag{4}$$

that consists of three 3D vectors which define the external camera parameters:

- $\mathbf{m} = (mx, my, mz)^T$: position of the camera frustum's mid point
- $\mathbf{s} = (sx, sy, sz)^T$: positive x, y and z scaling factor of the camera frustum, that define the distance of the eye point to $\mathbf{m}$
- $\mathbf{o} = (\phi, \theta, \psi)^T$: Euler angles defining the rotation around the x-, y- and z-axis respectively

Additionally, we need one internal parameter that defines the opening angle, which is constant and not subject of interpolation. We choose f as the value defining the distance of the eye point to the frustum center (relative to sz) which is related to the focal length.

Based in the camera parameters, we define the transformation

$\mathbf{p}$ that maps the 3D image (unit cube centered around the origin) to the view frustum. Each camera parameter is associated with a homogenous transformation matrix

$$\widetilde{\mathbf{T}} = \begin{pmatrix} \mathbf{Id} & \mathbf{m} \\ \mathbf{0} & 1 \end{pmatrix}, \quad \widetilde{\mathbf{S}} = \begin{pmatrix} \operatorname{diag}(\mathbf{s}) & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}, \quad \widetilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \tag{5}$$

$$\widetilde{\mathbf{F}} = \begin{pmatrix} 1-\frac{1}{4f^2} & 0 & 0 & 0 \\ 0 & 1-\frac{1}{4f^2} & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{4f} \\ 0 & 0 & -\frac{1}{f} & 1 \end{pmatrix} \tag{6}$$

where

$$\mathbf{R} = (\mathbf{r}\;\mathbf{u}\;\mathbf{a}) \tag{7}$$

$$= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \tag{8}$$

is the matrix that defines the orientation of the camera and $\mathbf{r}, \mathbf{u}, \mathbf{a}$ define its column vectors. They represent the unit right, up and view vector of the camera in world coordinates, respectively. The eye point $\mathbf{e}$ can be calculated as

$$\mathbf{e} = \mathbf{m} - f\, s\, \mathbf{a} \tag{9}$$

where $s = (sx \cdot sy \cdot sz)^{\frac{1}{3}}$ and $\mathbf{a}$ is defined as in Eq. (7). With this in place, we can define a projective transformation matrix in homogenous coordinates

$$\widetilde{\mathbf{P}}(\widehat{\mathbf{c}}) = \widetilde{\mathbf{T}}\,\widetilde{\mathbf{R}}\,\widetilde{\mathbf{S}}\,\widetilde{\mathbf{F}} \tag{10}$$

that transforms a point $\mathbf{x}$ with homogenous coordinates $\widetilde{\mathbf{x}} = (\mathbf{x}, 1)^T$ from the 3D image space $\Omega = [-\frac{1}{2}, \frac{1}{2}]^3$ to the view frustum in world space with

$$\widetilde{\mathbf{p}}(\widehat{\mathbf{c}}, \widetilde{\mathbf{x}}) = \widetilde{\mathbf{P}}(\widehat{\mathbf{c}})\; \widetilde{\mathbf{x}}. \tag{11}$$

The corresponding 3D transformation is then

$$\mathbf{p}(\widehat{\mathbf{c}}, \mathbf{x}) = \frac{1}{\widetilde{\mathbf{p}}(\widehat{\mathbf{c}}, \widetilde{\mathbf{x}})[4]} \begin{pmatrix} \widetilde{\mathbf{p}}(\widehat{\mathbf{c}}, \widetilde{\mathbf{x}})[1] \\ \widetilde{\mathbf{p}}(\widehat{\mathbf{c}}, \widetilde{\mathbf{x}})[2] \\ \widetilde{\mathbf{p}}(\widehat{\mathbf{c}}, \widetilde{\mathbf{x}})[3] \end{pmatrix}. \tag{12}$$

where $[i]$ refers to the i-th vector component.

4.2. Metric of the 9D Camera Space

As already mentioned in Section 2, our metric is supposed to measure the flow in the 3D image induced by the motion of the camera. For that we need to pull back the flow induced in the frustum to the 3D image space and then measure its average squared length by the energy functional. Based on that, we define the metric tensor which will turn out to be an integral over the 3D image space.

An infinitesimal movement of the camera is defined as $\widehat{\mathbf{c}}' = \widehat{\mathbf{c}} + \lambda\widehat{\mathbf{v}}$, where $\widehat{\mathbf{v}}$ is a change in the camera parameters (i.e. a vector in camera space) and $\lambda \to 0$. When we now use the displaced camera to transform $\mathbf{p}(\widehat{\mathbf{c}}, \mathbf{x})$ back to the image space, we get a new point

$$\mathbf{x}' = \mathbf{p}^{-1}\left(\widehat{\mathbf{c}}',\; \mathbf{p}(\widehat{\mathbf{c}}, \mathbf{x})\right) = \mathbf{p}^{-1}\left(\widehat{\mathbf{c}} + \lambda\widehat{\mathbf{v}},\; \mathbf{p}(\widehat{\mathbf{c}}, \mathbf{x})\right). \tag{13}$$

We consider the limit of the difference between $\mathbf{x}$ and $\mathbf{x}'$ as the 3D image flow $\mathbf{f} = \lim_{\lambda \to 0} \frac{\mathbf{x}' - \mathbf{x}}{\lambda}$ that the camera movement $\widehat{\mathbf{v}}$ induces at every point $\mathbf{x}$ in the 3D image space. This way, $\mathbf{f}$ is the directional derivative of $\mathbf{p}^{-1}$ in the direction $\widehat{\mathbf{v}}$, mapped from the world space to the image space. It can be written as

$$\mathbf{f}(\widehat{\mathbf{c}}, \widehat{\mathbf{v}}, \mathbf{x}) = (\nabla_{\mathbf{x}}\, \mathbf{p})^{-1} (\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p})\; \widehat{\mathbf{v}} \tag{14}$$

where

$$\nabla_{\mathbf{x}}\, \mathbf{p} = (\mathbf{p}_x \;\; \mathbf{p}_y \;\; \mathbf{p}_z) \tag{15}$$

$$\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p} = (\mathbf{p}_{mx} \;\; \mathbf{p}_{my} \;\; \mathbf{p}_{mz} \;\; \mathbf{p}_{sx} \;\; \mathbf{p}_{sy} \;\; \mathbf{p}_{sz} \;\; \mathbf{p}_{\phi} \;\; \mathbf{p}_{\theta} \;\; \mathbf{p}_{\psi}) \tag{16}$$

are the spatial Jacobian and the Jacobian in camera space, respectively. Here, subscripts denote partial derivatives. Fig. 2 shows an illustration of this concept. As we can see, the smaller the frustum is, the larger is the overall magnitude of the vectors in the field in 3D image space for the same movement $\lambda\widehat{\mathbf{v}}$.

Our objective is now to measure the average squared length of the vectors in the 3D image space volume. We calculate this by the triple integral over the unit cube $\Omega = [-\frac{1}{2}, \frac{1}{2}]^3$

$$\begin{aligned} &\int_\Omega \mathbf{f}^T \mathbf{f} \quad d\mathbf{x} \\ &= \int_\Omega \widehat{\mathbf{v}}^T (\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p})^T (\nabla_{\mathbf{x}}\, \mathbf{p})^{-T} (\nabla_{\mathbf{x}}\, \mathbf{p})^{-1} (\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p})\; \widehat{\mathbf{v}}\; d\mathbf{x} \\ &= \widehat{\mathbf{v}}^T\; \widehat{\mathbf{M}}(\widehat{\mathbf{c}})\; \widehat{\mathbf{v}} \end{aligned} \tag{17}$$

where

$$\widehat{\mathbf{M}}(\widehat{\mathbf{c}}) = \int_\Omega (\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p})^T (\nabla_{\mathbf{x}}\, \mathbf{p})^{-T} (\nabla_{\mathbf{x}}\, \mathbf{p})^{-1} (\nabla_{\widehat{\mathbf{c}}}\, \mathbf{p})\; d\mathbf{x} \tag{18}$$

is a symmetric, positive definite matrix we use as the Riemannian metric in the 9D camera space. $\widehat{\mathbf{M}}$ has a closed form for which we will provide a derivation with Maple in the additional materials.

In theory we can use this metric to find geodesics, however, as mentioned at the beginning, not every metric is feasible for numerically computing geodesics and simplifications can greatly increase the chance to do so. The reasons why we choose to simplify the problem are twofold. First, the size of the metric varies strongly in different regions of the camera space. Second, the condition number can become very large. Both properties pose considerable numerical challenges and make a fast numerical computation of geodesics impossible for our metric. We show an example for this when we compare the simplified metric to the 9D metric in Appendix B.

4.3. Metric of the 7D Simplified Camera Space

The general idea of our simplification is to replace the pyramid shape of the actual camera frustum with the best fitting cube, i.e. a cube with the same center, volume and orientation. We can see an illustration in Fig. 3. Because of the cubical shape of the simplified view frustum, we can reduce the dimensionality of the camera space to 7D by setting $sx = sy = sz = s$. So the new camera is

$$\bar{\mathbf{c}} = (mx, my, mz, s, \phi, \theta, \psi)^T. \tag{19}$$

With this camera model, we can compute the transformation $\mathbf{p}(\bar{\mathbf{c}}, \mathbf{x})$ from 3D image space to 3D world space by applying the equations (10)–(12) under the replacement $\widetilde{\mathbf{F}} = \mathbf{Id}$. Similar to the equations (14)–(18), we get

$$\mathbf{f}(\bar{\mathbf{c}}, \bar{\mathbf{v}}, \mathbf{x}) = (\nabla_{\mathbf{x}}\, \mathbf{p})^{-1} (\nabla_{\bar{\mathbf{c}}}\, \mathbf{p})\; \bar{\mathbf{v}} \tag{20}$$

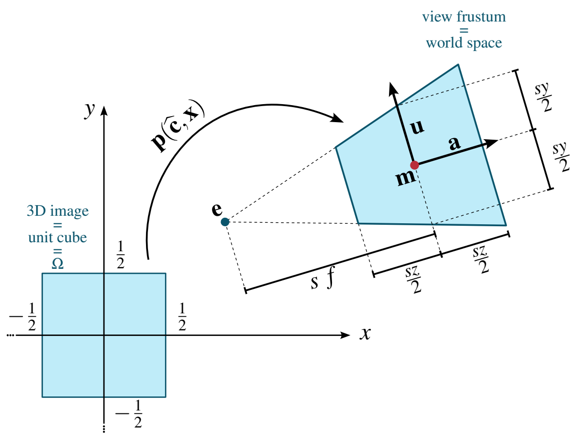

Figure 1: *9D camera model $\widehat{\mathbf{c}}$. The transformation $\mathbf{p}$ maps a unit cube in 3D image space (left) to the view frustum in 3D world space (right).*

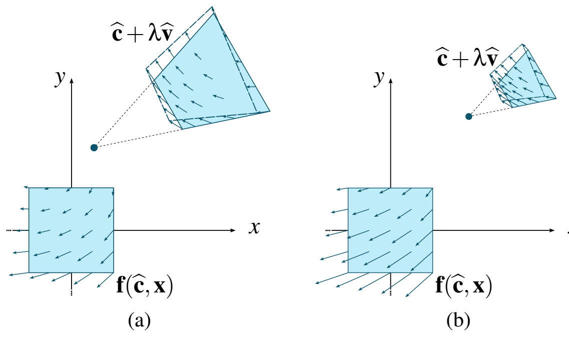
$\widehat{\mathbf{c}}+\lambda\widehat{\mathbf{v}}$
y
x
$\mathbf{f}(\widehat{\mathbf{c}},\mathbf{x})$
(a)
(b)

Figure 2: *Moving the camera from $\widehat{\mathbf{c}}$ to $\widehat{\mathbf{c}}+\lambda\widehat{\mathbf{v}}$ induces a flow field $\mathbf{f}$ in the 3D image space. The average flow magnitude is larger for the same change in the camera parameters if the frustum has a smaller scale (compare (b) to (a)). This means to minimize the flow magnitude of $\mathbf{f}$, the camera needs to move slower if its eye point is closer to the point of interest $\mathbf{m}$.*

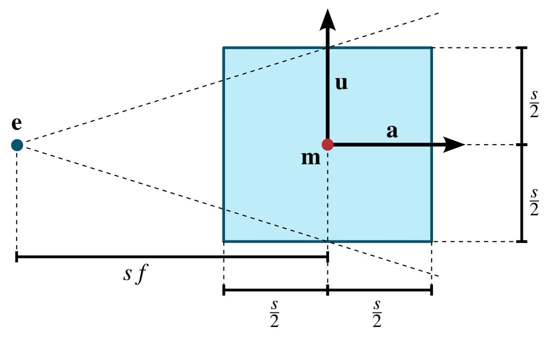

Figure 3: *Simplified 7D camera model $\overline{\mathbf{c}}$. We replace the frustum by the best-fitting cube.*

and

$$\overline{\mathbf{M}}(\overline{\mathbf{c}}) = \int_\Omega (\nabla_{\overline{\mathbf{c}}}\,\mathbf{p})^T (\nabla_{\mathbf{x}}\,\mathbf{p})^{-T} (\nabla_{\mathbf{x}}\,\mathbf{p})^{-1} (\nabla_{\overline{\mathbf{c}}}\,\mathbf{p})\; d\mathbf{x}. \tag{21}$$

Fortunately, $\overline{\mathbf{M}}(\overline{\mathbf{c}})$ has a simple closed-form solution

$$\overline{\mathbf{M}}(\overline{\mathbf{c}}) = \begin{pmatrix} \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{s^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4s^2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & 0 & -\frac{\sin\theta}{6} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & -\frac{\sin\theta}{6} & 0 & \frac{1}{6} \end{pmatrix} \tag{22}$$

that is even that simple to allow closed-form solutions for geodesics. We provide a Maple sheet in the additional materials to prove the form of $\overline{\mathbf{M}}$.

4.4. Geodesics of the Simplified Camera Space

In Section 2 we established that geodesics are minimizers of the energy functional. Thankfully, our simplified metric allows us to derive a closed-form solution for this variational problem which we prove in Appendix A. In the following, we will give this solution deconstructed into three separate parts $\mathbf{m}(t)$, $s(t)$ and $\mathbf{R}(t)$, where $\mathbf{R}(t)$ is the time varying orientation matrix for the camera. We choose to use $\mathbf{R}(t)$ instead of working directly with the Euler angles $\mathbf{o}(t)$ because converting to $\mathbf{R}$ makes it easier to express the desired rotation. It is possible to extract the angles from the orientation matrix but one needs to take special care because the same orientation can be expressed with different angles.

As we can see in the structure of $\overline{\mathbf{M}}$, the upper left 4×4 block that is connected to the position and scale of the frustum is independent of the lower right 3×3 block connected to the rotation. This means that we can handle both parts separately. For this section, we still assume a parameter interval of $t\in[0,1]$.

4.4.1. Defining $\mathbf{R}(t)$

First, we need to find the axis of rotation to transform the orientation of $\overline{\mathbf{c}}_0$ to $\overline{\mathbf{c}}_1$. We can do this by utilizing the matrices $\mathbf{R}_0 = (\mathbf{r}_0\ \mathbf{u}_0\ \mathbf{a}_0)$ and $\mathbf{R}_1 = (\mathbf{r}_1\ \mathbf{u}_1\ \mathbf{a}_1)$.

Using the two matrices, the axis of rotation is then the normalized eigenvector $\mathbf{k}$ corresponding to the zero eigenvalue of $(\mathbf{R}_1-\mathbf{R}_0)^T$, i.e., we search the solution for

$$(\mathbf{R}_1-\mathbf{R}_0)^T\,\mathbf{k} = 0 \;\Leftrightarrow\; \mathbf{R}_0\mathbf{R}_1^T\mathbf{k} = \mathbf{k}, \quad \mathbf{k}^T\mathbf{k} = 1. \tag{23}$$

This solution always exists because the matrix $\mathbf{R}_0\mathbf{R}_1^T$ is again a 3D rotation matrix and, therefore, needs to map one vector (the axis of rotation $\mathbf{k}$) to itself. In the case that every eigenvalue of $(\mathbf{R}_1-\mathbf{R}_0)^T$ is zero (when the two matrices are identical), we can simply choose any unit vector as axis of rotation because the angle of rotation is zero and therefore the choice of the axis does not have any influence.

We can now define a change of basis matrix $\mathbf{R}_\mathbf{k} = (\mathbf{k}_1\ \mathbf{k}_2\ \mathbf{k})$. The other two vectors are chosen as

$$\mathbf{k}_2 = \frac{\mathbf{k}\times\mathbf{a_0}}{\|\mathbf{k}\times\mathbf{a_0}\|} \quad , \quad \mathbf{k}_1 = \mathbf{k_2}\times\mathbf{k}. \tag{24}$$

This ensures that we can determine the angle of rotation ξ with

$$\xi = \operatorname{arctan2}\left((\mathbf{R_k}^T \mathbf{a}_1)[2],\ (\mathbf{R_k}^T \mathbf{a}_1)[1]\right). \tag{25}$$

We chose arctan2 over the simple arctan because it allows us to recover an angle in the range of $(-\pi, \pi]$ for ξ. Using

$$\mathbf{R}_\xi(t) = \begin{pmatrix} \cos(\xi\, t) & -\sin(\xi\, t) & 0 \\ \sin(\xi\, t) & \cos(\xi\, t) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

the final orientation matrix for the camera at time t is

$$\mathbf{R}(t) = \mathbf{R_k}\, \mathbf{R}_\xi(t)\, \mathbf{R_k}^T\, \mathbf{R}_0. \tag{26}$$

In other words, the geodesic for the rotation is a rotation with constant speed by ξ around a fixed axis of rotation $\mathbf{k}$.

4.4.2. Defining $\mathbf{m}(t)$ and $s(t)$

First, we want to define some notation for convenience. We choose

$$d = \|\mathbf{m}_1 - \mathbf{m}_0\|, \quad \mathbf{d} = \frac{\mathbf{m}_1 - \mathbf{m}_0}{d} \tag{27}$$

to denote the distance and normalized direction vector from $\mathbf{m}_0$ to $\mathbf{m}_1$ respectively. Additionally,

$$b_i = \frac{s_1^2 - s_0^2 + (-1)^i\, 4d^2}{4\, s_i\, d} \quad (28) \qquad r_i = \ln\left(-b_i + \sqrt{b_i^2 + 1}\right) \quad (29)$$

for $i = 0, 1$. With those, we can define

$$s(t) = \frac{s_0 \cosh(r_0)}{\cosh((r_1 - r_0)t + r_0)} \tag{30}$$

as well as

$$m(t) = \frac{s_0}{2}\cosh(r_0)\tanh((r_1 - r_0)t + r_0) - \frac{s_0}{2}\sinh(r_0) \tag{31}$$

$$\mathbf{m}(t) = \mathbf{m}_0 + m(t)\, \mathbf{d}. \tag{32}$$

In the special case $d = 0$, this converges to

$$s(t) = s_0\, \exp\left(\operatorname{sign}(s_1 - s_0)\, |\ln(s_1) - \ln(s_0)|\, t\right) \tag{33}$$

$$\mathbf{m}(t) = \mathbf{m}_0. \tag{34}$$

Note that the equations for $s(t)$ and $\mathbf{m}(t)$ are similar to zooming and panning form in the work of van Wijk and Nuij [vN03]. To summarize, the following theorem is the main theoretical basis of our approach:

Theorem 1 The camera path $\bar{\mathbf{c}}(t)$ given by $\mathbf{m}(t)$ (32), $s(t)$ (30) and $\mathbf{R}(t)$ (26) is a geodesic in the metric space given by $\overline{\mathbf{M}}$ (22). Further, $\bar{\mathbf{c}}(t)$ is equal-speed (or arc length proportional) parametrization with interpolation conditions $\bar{\mathbf{c}}(0) = \bar{\mathbf{c}}_0$, $\bar{\mathbf{c}}(1) = \bar{\mathbf{c}}_1$.

We prove the theorem in Appendix A.

4.4.3. Length of the Geodesic Path

With the definition of the geodesic in place, we can determine the geodesic distance (or the length of the geodesic path $\bar{\mathbf{c}}(t)$) of two points in the camera space as

$$\operatorname{dist}(\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1) = \int_0^1 \sqrt{\dot{\bar{\mathbf{c}}}(t)^T\, \overline{\mathbf{M}}(\bar{\mathbf{c}}(t))\, \dot{\bar{\mathbf{c}}}(t)}\quad dt \tag{35}$$

$$= \sqrt{\frac{(r_1 - r_0)^2}{4} + \frac{\xi^2}{6}}. \tag{36}$$

where $\dot{\bar{\mathbf{c}}}(t) = \frac{d\bar{\mathbf{c}}}{dt}$. For $d = 0$, this converges to

$$\operatorname{dist}(\bar{\mathbf{c}}_0, \bar{\mathbf{c}}_1) = \sqrt{\frac{(\ln(s_1) - \ln(s_0))^2}{4} + \frac{\xi^2}{6}}. \tag{37}$$

4.5. Catmull-Rom Spline Interpolation

Until now we only discussed how we can join two points in camera space. Usually, we want to connect more when we design a camera path. In the spirit of the work of Nava-Yazdani and Polthier [NYP13], we want to join a sequence of camera positions with repeated linear interpolation in the metric camera space.

Given is a sequence of $n+1$ camera positions $(\bar{\mathbf{c}}_0, \ldots, \bar{\mathbf{c}}_i, \bar{\mathbf{c}}_j, \ldots \bar{\mathbf{c}}_n)$ at time values (or knots) $(t_0, \ldots, t_i, t_j, \ldots, t_n)$ where $t_i < t_j,\ i < j$ that should be interpolated and joined to one continuous, smooth path. As one option for interpolation, we could try to construct a C^1/C^2 continuous Bézier spline curve. However, constructing the necessary control points is difficult because the camera space is not Euclidean. Keep in mind that the construction of an interpolating B-Spline curve in Euclidean space requires the solution of a global linear system [Far02]. In our metric this linear system translates to a system of nonlinear equations for which no closed form solution is available. For this reason, we chose Catmull-Rom splines instead. They show C^1 continuity, and we can use a recursive algorithm similar to De Boor's algorithm [BG88, YSK09] to evaluate them without the need of solving a global system of equations.

Catmull-Rom splines are also ideal for keyframes because they allow the user to determine the time where a certain point should be reached instead of just determining the shape of the curve and fixing the parametrization. However, the parametrization affects the shape and quality of the camera path. Following Yuksel et al. [YSK09], we propose a centripetal parametrization based on geodesic distances in camera space.

While Catmull-Rom splines are interpolating splines, they do not interpolate the first and last control points because these only determine the tangent at the beginning of the first and end of the last segment. For that reason, we need two additional points $\bar{\mathbf{c}}_{-1}$ and $\bar{\mathbf{c}}_{n+1}$ as well as their knot values t_{-1} and t_{n+1}. We achieved reasonable results with

$$\bar{\mathbf{c}}_{-1} = \bar{\mathbf{c}}_0 \qquad t_{-1} = -t_1 + 2t_0 \tag{38}$$

$$\bar{\mathbf{c}}_{n+1} = \bar{\mathbf{c}}_n \qquad t_{n+1} = -t_{n-1} + 2t_n. \tag{39}$$

We remark that there are some difficulties when using Catmull-Rom splines. For one, the recursive evaluation scheme is computationally more expensive than the equivalent of just joining the keyframes with geodesic segments because we do not have a closed-form solution anymore. We will discuss this in the implementation section.

Another problem we noticed was due to the extrapolation that is part of the Catmull-Rom evaluation scheme. The recursive computation requires us to evaluate the geodesic outside of the $[0,1]$ range. This means that, if a rotational part is present, the angle with which we rotate around the rotational axis $\mathbf{k}$ is outside of the $[0,\xi]$ range. If ξ is already close to π in any direction, extrapolation can push the angle of rotation to $> \pi$. In consequence, the next step of

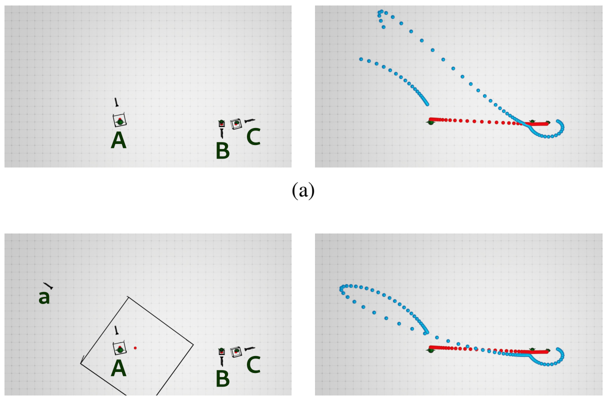

Figure 4: *Catmull-Rom spline composed of 3 keyframes. The path starts at **A** (frame 0), continues to **B** (frame 80) and ends in **C** (frame 100). (a) The unmodified spline exhibits a gap because of the extrapolation in the Catmull-Rom scheme and the discontinuity present in the* arctan2 *function. (b) This behavior can be corrected by inserting an additional keyframe **a** at frame 21, right before the discontinuity.*

the recursive evaluation scheme can flip the direction of rotation because arctan2 can only recover angles in the range of $\pm\pi$. This leads to a discontinuity in the evaluation scheme from one sample point to the next which then leads to gaps in the path. We can see this happening in Fig. 4a. We were able to resolve this problem by inserting a new keyframe right before the discontinuity as seen in Fig. 4b. Sometimes the issue could also be resolved by choosing another parametrization for the keyframes. Both chordal and centripetal Catmull-Rom curves were working well in practice when we used our distance measure Eq. (36) to compute them.

5. Implementation

We implemented our camera path as a Python add-on script for the 3D modeling software Blender. The implementation allows us to interactively add keyframes and see the resulting camera path as a geometric object. Additionally, it allows us to inspect the path from the view point of the camera. All examples, with the exception of Gaia Sky and the Mandelbulb fractal, were generated with that add-on. It is available at https://livelyliz.github.io/OptFlowCam/.

As noted above, the Catmull-Rom splines need a lot longer to compute than the simple geodesic path. For example, a path that was sampled at 200 equidistant samples of the interval $[0,1]$, the single core implementation for the Catmull-Rom spline was about 100x slower than the simple geodesic path (1.1 seconds compared to 0.01 seconds). We were able to leverage the slow computation time by parallelizing it with Pythons multiprocessing package but the computation time was still 10x slower (about 0.1 seconds). We should also note that our implementation was not optimized for speed, so there might be a more efficient implementation that reduces the difference in computation time even further.

Because of the computation time, we opted to not have automatic updates of the path while editing. However, it is still possible to interact with the path in a reasonable manner. At this point, we want to refer the reader to the *video* in the additional materials that shows an interactive session in Blender.

For the examples in Gaia Sky [SJMS19] and Mandelbulber [Man], we used another Python script that generates files for both softwares that allow us to import our camera paths.

We already mentioned that the Catmull-Rom interpolation can create gaps in the path and we can prevent that with additional keyframes. To avoid that the user has to add these manually, we implemented a method that automatically inserts new keyframes at path discontinuities. For that, we detect angle and length discontinuities of the Euclidean tangent vectors of the path. We found that this works well in practice for most situations. However, regions of high curvature or an insufficient number of keyframes are still challenging. The downside of this approach is that it additionally increases the computation time because the spline curve has to be computed every time a new keyframe is inserted and the process may need several iterations.

We also noticed some numerical issues with our current implementation. If b_i (Eq. (28)) becomes large ($> 10^6$), floating point precision is insufficient to accurately compute r_i. This is why for large values we opt to replace Eq. (29) with

$$r_i = -\operatorname{sign}(b_i)\ \ln(2\,|b_i|), \tag{40}$$

which has the same limit for $b_i \to \infty$ as Eq. (29) but is more robust.

Generally, it can happen that keyframes are not interpolated exactly due to numerical errors. However, while the differences are noticeable when precision is required, the frame is still close to the original (see frame 500 of Fig. 8).

6. Results

In this section, we want to show our results and compare them to standard methods:

1. linear interpolation of the look-at point (or view target) and linear interpolation of the camera eye point positions, i.e.

$$\mathbf{m}(t) = (1-t)\ \mathbf{m}_0 + t\ \mathbf{m}_1, \qquad \mathbf{e}(t) = (1-t)\ \mathbf{e}_0 + t\ \mathbf{e}_1$$

2. linear interpolation of the transformations involved in the camera space (which is similar to the work of Alexa [Ale02]), i.e., the use of $\mathbf{R}(t)$ as defined in Eq. (26) and

$$\mathbf{m}(t) = (1-t)\ \mathbf{m}_0 + t\ \mathbf{m}_1, \qquad s(t) = (1-t)\ s_0 + t\ s_1.$$

First, we want to demonstrate the simple geodesic paths and their parametrization on multiple examples for a very simple scene. The scene consists of a plane and three objects (a teapot, a monkey head and a bunny). In Fig. 5, each row represents a different test scenario. In the first column we can see the start configuration, i.e. the two endpoints of the path as well as the best fitting cube for the frustum. The resulting paths for each technique are displayed in the 3 remaining columns. We show both the paths for $\mathbf{e}(t)$ (in blue) and $\mathbf{m}(t)$ (in red). Our method usually produces paths that "spread" more, i.e. in Euclidean space they are longer, form larger arcs, and deviate more from the simple linear connection between

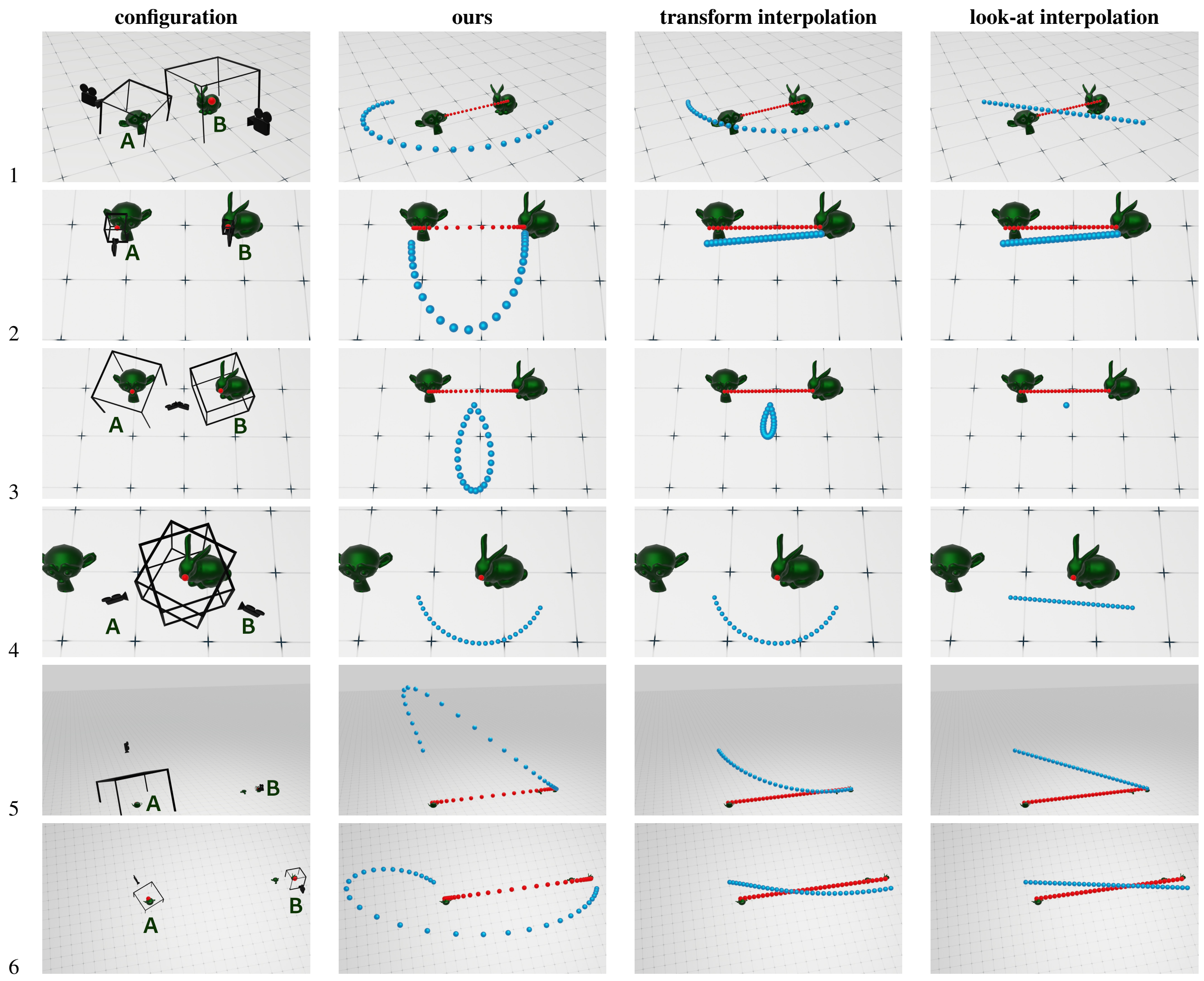

Figure 5: *Simple test scenes. The camera moves from point **A** to point **B**. The blue points depict the path of the eye point ● $\mathbf{e}(t)$ while the red points represent the path of the look-at point ● $\mathbf{m}(t)$. In the first column we can see the two endpoint configurations with the simplified camera frustum. The second, third and fourth columns show the results for different interpolation methods. From left to right those are: our method, a linear interpolation of the transforms based on [Ale02] and a linear interpolation of the eye point and look-at point.*

the keyframes. Also, especially the second, fifth and sixth row show that $\mathbf{m}(t)$ of our path has a very different parametrization than the other two methods. It is denser at the beginning and end of the path and sparser in the middle, which means that the camera moves faster in the section between the interesting points (keyframes). This showcases exactly the nature of the camera space metric. Just as in the work covering the 2D space [vN03], it is more "expensive" for the camera to move sideways as long as the frustum is small. So the camera will move away from the target point first. Because of this, the camera reveals more of the scene context that connects the two target points which makes it easier to follow where the camera is moving and how the two points are located in relation to each other.

Fig. 6 shows an example for the Catmull-Rom spline where we want to explore the mars rover *Perseverance*. Again, the blue curve represents the path of the camera eye point and the red path the path of the view target point. The path switches between different zoom levels. At the beginning (keyframe **A**) the camera is very close to the rover but then describes a wide arc to display it in its entirety. This example demonstrates that with only a few keyframes, we can explore the whole rover and switch between interesting areas of different size and orientation and still produce a smooth path. The scene features a lot of open space and contains only a few objects. Still, we had to insert a keyframe (**D**) to resolve a collision with a rock in the upper left corner of the image.

With Fig. 7 we want to illustrate that our method can also be

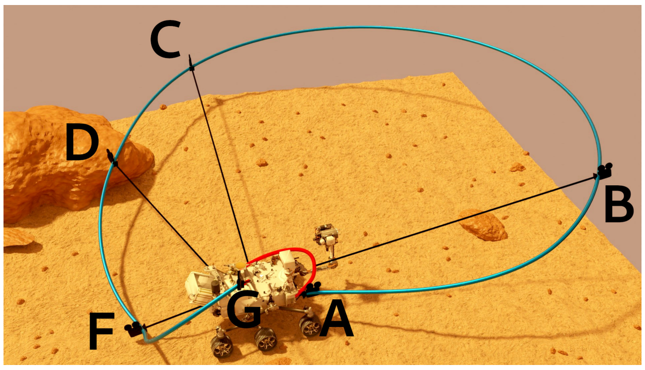

Figure 6: *Exploring the Mars rover* Perseverance*. The path consists of 6 keyframes (**A**-**G**) which are joined by a Catmull-Rom spline. While keyframes **A** and **F** focus more on special details of the rover, keyframes **B**, **C** and **G** show a larger area. Keyframe **D** ensures that the path does not collide with the rock.*

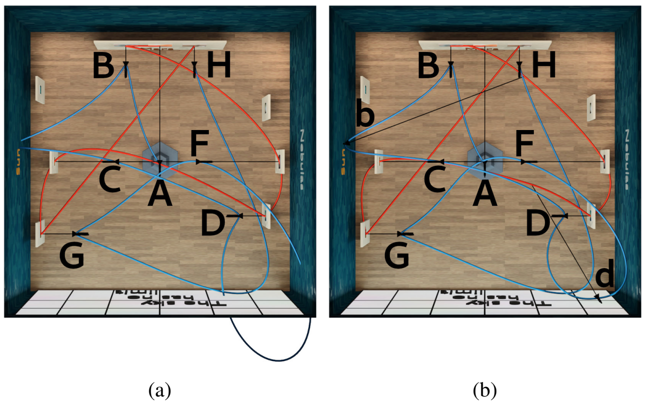

Figure 7: *Resolving collision in a gallery scene. (a) The original Catmull-Rom spline path through a gallery. On the left and bottom wall, the blue eyepoint path ● $\mathbf{e}(t)$ penetrates the geometry of the wall. (b) The collision was resolved by inserting the lower case letter keyframes **b** and **d**. The red look-at target path ● $\mathbf{m}(t)$ changes only slightly by this addition.*

used for larger scenes and enclosed spaces. This examples features a gallery with three images each at the west, north and east wall. The path switches between keyframes that show an overview of the whole wall and the detailed view of one specific image, so we vary all parameters of the camera space quite a bit. A particular difficulty in this scene compared to Fig. 6 is the enclosed space where we need to avoid collisions. Our method does not naturally avoid collisions because it has no scene information so the designer of the path has to make sure to resolve them. This is in no way different to the usual methods used for keyframing, so it is not a disadvantage of our method. Fig. 7 compares the path before and after the collision was resolved.

The greatest strength of our method lies in the extreme change of scale of the view frustum. We demonstrate that by zooming in on the mandelbulb (a 3D fractal) that was realized by two cameras: one at a certain distance and one very close to the object. The scene was rendered with the fractal software Mandelbulber [Man] which allowed us to import and render the externally computed path. Fig. 8 presents a few frames rendered from the camera's point of view. We can see that the standard (Euclidean linearly interpolated) path the software produces does not slowly zoom into the fractal like our path does. So, while our camera path gives the impression of constant speed, the other path seems to speed up towards the end.

Another scene we used is the space exploration software Gaia Sky [SJMS19]. The use case is similar to that of the fractal because it needs to cover distances that differ by multiple orders of magnitude (e.g., 1pc (3.0857×10^{16}m) compared to 1AU (1.495×10^{11}m)). A section of the resulting path is pictured in Fig. 9 and the accompanying video.

7. Discussion

Even though the spline interpolation method currently does not have real-time performance, it is still fast enough for interactive editing and designing paths by hand. The camera model we use seems a bit unusual at first but in practice, a designer can still work with it as they normally would, which we can see in the interactive editing session provided as a *video* on the project webpage. The only difference is that (either explicitly or implicitly) the current point of interest (POI) needs be defined for each keyframe.

It should be noted that our paths may not be suitable for all situations. In scenes with narrow passages and high object density, our paths can cause issues due to their zooming behavior. This can create large arcs and increase the likelihood of collisions between the eye point path and geometry. Although it is possible to resolve collisions by adding more keyframes or adjusting the initial ones, predicting how the path will react to changes remains challenging. Related to this is the behavior of the path of the look-at point $\mathbf{m}(t)$. If the keyframes are positioned on opposite sides of an object, the path may pass through the object. Additionally, if the object is sufficiently large, the camera may not move far enough away from its surface to ensure a pleasant path. For example, this could be an issue with a model of a planet where the camera keyframes are located at the poles. The effect may be reduced by adding more keyframes or utilizing a predefined $\mathbf{m}(t)$. Exploring the latter option in the future would be beneficial for the overall method. The very particular behavior can also interfere with the artistic choices of the designer of the camera path, so if absolute control is needed, our paths should not be used. Nonetheless, they can provide a good starting point and are viable to use alongside traditional keyframing.

Unlike standard interpolation techniques, our method requires the user to specify a distance from the camera where the POI for the current view is located. The only change for the user is that the view direction from the eye point is now a general 3D vector instead of a normalized 3D vector. Taking the POI into account is only a minimal overhead in the interactive camera setting (as shown in a video of the interactive session available on the project website). Manually specifying interesting objects is also part of other camera interaction tools [LC15] and would be required for tracking tasks anyway. However, we do not consider the object's shape

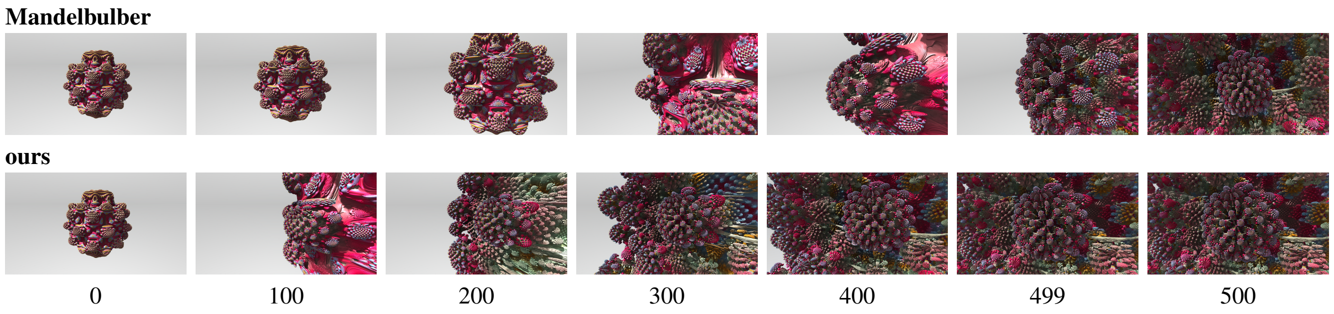

Figure 8: *Mandelbulb fractal scene from the fractal generation software "Mandelbulber" [Man]. The frame numbers are written below the images. The camera moves from a distance of* 2 *units to a distance of* 1.4×10^{-12} *units to the surface. The perceived speed of the camera increases for the standard linearly interpolated path (top). The difference is especially apparent in frame 499 to 500, where it "jumps" from an overview to a close up. Our path (bottom) approaches the final position more gradually, as we can see when we compare frame 400 to 500.*

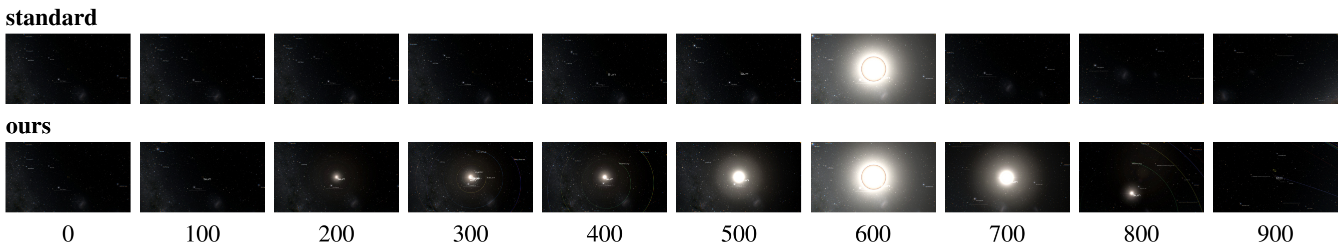

Figure 9: *Selected frames from the "Gaia Sky" [SJMS19] test scene. Frame numbers are written below the images. Frames 0 and 600 are consecutive keyframes from a path consisting of a total of 4 keyframes. The camera's distance to the sun (point of interest) in the first keyframe is about 3.93 parsec (*$\approx 1.212 \times 10^{14}$ km*) and in the second keyframe it is about* 3.22×10^{6} km. *While with our approach the initial empty space was traversed quickly and the camera gradually slows down as it approaches the sun (second keyframe), the standard Catmull-Rom spline shows the sun in the keyframe only. In the example video of this scene it becomes even more apparent that a standard interpolation approach produces unsatisfactory results.*

In addition, the POI can be obtained automatically by conventional positioning of the camera using ray casting from the eye point in the view direction or by more sophisticated techniques that can identify an interesting object in a scene.

Another limitation of the current approach are dynamic scenes and tracking tasks. As introduced in this paper, the view direction is determined by our method as the curve $\mathbf{m}(t)$ which makes it difficult to accurately track moving parts of the scene. However, if would be possible to replace $\mathbf{m}(t)$ in the calculation of $\mathbf{e}(t)$ (Eq. (9)) with a user defined curve.

We want to emphasize that, to our knowledge, there are no other techniques that can handle sclae changes as large as our method does. The techniques mentioned in Section 3 mostly incorporate cinematographic goals where a change from overview to detailed view (or the other way around) is usually a change in the distance to the POI of maybe a factor 10 to 100. While that suffices in most camera planning tasks, we want to point out that for applications such as space visualization but also general data visualization, there might be more extreme cases that need to be handled. In our Gaia Sky and Mandelbulb examples, our method can handle a change in distance of a factor of up to 10^{12}.

8. Conclusion

To summarize, this paper introduced a new interpolation scheme for camera paths which was designed to be the geodesic of a metric camera space. The metric we used is based on the average magnitude of the flow in the 3D image. This flow is induced by the movement of the camera and is independent of the scene that is actually visible. However, our camera model still incorporates what is important to the scene by explicitly using the camera look-at point. By simplifying the problem space, we were then able to calculate the geodesic as a parametric curve. To go one step further, we also showed that multiple camera poses can be connected by an interpolating spline.

The examples demonstrate that the resulting paths are distinct from the paths produced by other simple interpolation schemes and we believe them to be better suited for applications that need automatic viewpoint changes. Due to the automatic speed adjustment that happens when the camera focus switches from an overview to detailed view, we are able to generate smooth paths that create the impression of a near-constant speed which is often not the case for other methods. This is especially important when the scene in questions needs to handle multiple magnitudes of difference in scale, e.g., when navigating in space exploration software like Gaia Sky.

We want to note that our approach does not solve the general problem of camera path planning. It does not automatically calculate good viewpoints of the scene or a dynamic object or resolve collisions with scene geometry. It could, however, be used conjunction with existing techniques if those techniques involve some kind of viewpoint interpolation. Exploring this would be an interesting direction for future work.

Acknowledgements

We want to thank NASA/JPL-Caltech for making 3D models of their rovers and images of space available to the general public. Open Access funding enabled and organized by Projekt DEAL.

References

[AA10] ARGELAGUET F., ANDUJAR C.: Automatic speed graph generation for predefined camera paths. In *Smart Graphics*. Springer Berlin Heidelberg, 2010, pp. 115–126. doi:10.1007/978-3-642-13544-6_11. 2

[Ale02] ALEXA M.: Linear combination of transformations. *ACM Transactions on Graphics 21*, 3 (Jul 2002), 380–387. doi:10.1145/566654.566592. 2, 7, 8

[AVF04] ANDUJAR C., VAZQUEZ P., FAIREN M.: Way-finder: guided tours through complex walkthrough models. *Computer Graphics Forum 23*, 3 (2004), 499–508. doi:10.1111/j.1467-8659.2004.00781.x. 2

[BG88] BARRY P. J., GOLDMAN R. N.: A recursive evaluation algorithm for a class of catmull-rom splines. *SIGGRAPH Comput. Graph. 22*, 4 (Jun 1988), 199–204. doi:10.1145/378456.378511. 6

[CL03] CHRISTIE M., LANGUÉNOU E.: A constraint-based approach to camera path planning. In *Smart Graphics*. Springer Berlin Heidelberg, 2003, pp. 172–181. doi:10.1007/3-540-37620-8_17. 2

[DC92] DO CARMO M. P.: *Riemannian Geometry*. Birkhauser Verlag AG, 1992. 2

[DKC*22] DEMIRALP A. C., KRÜGER M., CHAO C., KUHLEN T. W., GERRITS T.: Astray: A Performance-Portable Geodesic Ray Tracer. In *Vision, Modeling, and Visualization* (2022), Bender J., Botsch M., Keim D. A., (Eds.), The Eurographics Association. doi:10.2312/vmv.20221208. 3

[Far02] FARIN G.: *Curves and Surfaces for CAGD*. Elsevier, 2002. doi:10.1016/b978-1-55860-737-8.x5000-5. 1, 6

[GCLR15] GALVANE Q., CHRISTIE M., LINO C., RONFARD R.: Camera-on-rails: automated computation of constrained camera paths. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games* (New York, NY, USA, Nov. 2015), MIG '15, Association for Computing Machinery, p. 151–157. URL: https://doi.org/10.1145/2822013.2822025, doi:10.1145/2822013.2822025. 3

[GH21] GEBHARDT C., HILLIGES O.: Optimization-based user support for cinematographic quadrotor camera target framing. In *Proceedings of the 2021 ACM Conference on Human Factors in Computing Systems, CHI'21* (2021). doi:10.1145/3411764.3445568. 2

[GHN*16] GEBHARDT C., HEPP B., NÄGELI T., STEVŠIĆ S., HILLIGES O.: Airways: Optimization-based planning of quadrotor trajectories according to high-level user goals. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (May 2016), CHI '16, ACMAssociation for Computing Machinery, pp. 2508–2519. doi:10.1145/2858036.2858353. 2

[HDL22] HISSBACH A.-M., DICK C., LAWONN K.: An Overview of Techniques for Egocentric Black Hole Visualization and Their Suitability for Planetarium Applications. In *Vision, Modeling, and Visualization* (2022), Bender J., Botsch M., Keim D. A., (Eds.), The Eurographics Association. doi:10.2312/vmv.20221207. 3

[HLH*16] HUANG H., LISCHINSKI D., HAO Z., GONG M., CHRISTIE M., COHEN-OR D.: Trip synopsis: 60km in 60sec. *Computer Graphics Forum 35* (2016), 107–116. doi:https://doi.org/10.1111/cgf.13008. 2

[JRT*15] JOUBERT N., ROBERTS M., TRUONG A., BERTHOUZOZ F., HANRAHAN P.: An interactive tool for designing quadrotor camera shots. *ACM Transactions on Graphics (SIGGRAPH Asia 2015) 34*, 6 (2015). doi:10.1145/2816795.2818106. 2

[LC15] LINO C., CHRISTIE M.: Intuitive and efficient camera control with the toric space. 1–12. doi:10.1145/2766965. 3, 9

[Man] Mandelbulber v2 2.30. URL: https://github.com/buddhi1980/mandelbulber2. 7, 9, 10

[MC00] MARCHAND E., COURTY N.: Image-based virtual camera motion strategies. In *Proceedings of the Graphics Interface 2000 Conference* (May 2000), pp. 69–76. doi:10.20380/GI2000.11. 2

[NAD*17] NÄGELI T., ALONSO-MORA J., DOMAHIDI A., RUS D., HILLIGES O.: Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization. *IEEE Robotics and Automation Letters 2*, 3 (2017), 1696–1703. doi:10.1109/LRA.2017.2665693. 2

[NYP13] NAVA-YAZDANI E., POLTHIER K.: De casteljau's algorithm on manifolds. *Computer Aided Geometric Design 30*, 7 (2013), 722–732. doi:10.1016/j.cagd.2013.06.002. 6

[Pet06] PETERSEN P.: *Riemannian geometry*. Springer, 2006. 2, 12

[SJMS19] SAGRISTÀ A., JORDAN S., MÜLLER T., SADLO F.: Gaia sky: Navigating the gaia catalog. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (Jan 2019), 1070–1079. doi:10.1109/TVCG.2018.2864508. 7, 9, 10

[Str50] STRUIK D. J.: *Lectures on Classical Differential Geometry*. Addison Wesley Publishing Company Inc., 1950. 2

[vN03] VAN WIJK J. J., NUIJ W. A. A.: Smooth and efficient zooming and panning. In *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No.03TH8714)* (Oct 2003), IEEE Computer Society, pp. 15–23. doi:10.1109/INFVIS.2003.1249004. 1, 3, 6, 8

[WH88] WARREN W. H., HANNON D. J.: Direction of self-motion is perceived from optical flow. *Nature 336*, 6195 (Nov. 1988), 162–163. doi:10.1038/336162a0. 1

[WR22] WU F., ROSENBERG E. S.: Adaptive field-of-view restriction: Limiting optical flow to mitigate cybersickness in virtual reality. In *28th ACM Symposium on Virtual Reality Software and Technology* (Nov. 2022), ACM. doi:10.1145/3562939.3565611. 1

[XYH*18] XIE K., YANG H., HUANG S., LISCHINSKI D., CHRISTIE M., XU K., GONG M., COHEN-OR D., HUANG H.: Creating and chaining camera moves for quadrotor videography. *ACM Transactions on Graphics (Proc. SIGGRAPH) 37*, 4 (2018), 1–13. doi:10.1145/3197517.3201284. 2

[YSK09] YUKSEL C., SCHAEFER S., KEYSER J.: On the parameterization of catmull-rom curves. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling* (Oct 2009), ACM. doi:10.1145/1629255.1629262. 6

Appendix A: Derivation of the Equations of the Geodesic Path

In addition to Table 1, we use the following notation:

- $\check{\mathbf{x}}, \check{\mathbf{v}}, \check{\mathbf{M}}$ for the 4D space describing the position and scale subspace of the 7D simplified camera space
- $\widehat{\mathbf{x}}, \widehat{\mathbf{v}}, \widehat{\mathbf{M}}$ for the 3D space describing the rotation subspace of the 7D simplified camera space

To prove Theorem 1, we realize that $\overline{\mathbf{M}}$ has the form

$$\overline{\mathbf{M}} = \mathrm{diag}(\check{\mathbf{M}}(mx, my, mz, s),\ \ \widehat{\mathbf{M}}(\phi, \theta, \psi)) \tag{41}$$

© 2024 The Authors.
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.

with

$$\check{\mathbf{M}} = \operatorname{diag}\left(\frac{1}{s^2},\frac{1}{s^2},\frac{1}{s^2},\frac{1}{4s^2}\right), \quad \widehat{\mathbf{M}} = \begin{pmatrix} \frac{1}{6} & 0 & -\frac{\sin\theta}{6} \\ 0 & \frac{1}{6} & 0 \\ -\frac{\sin\theta}{6} & 0 & \frac{1}{6} \end{pmatrix}. \tag{42}$$

This shows that we can treat the (mx,my,mz,s) subspace and the (ϕ,θ,ψ) subspace independently when searching for geodesics. A geodesic in (mx,my,mz,s) subspace is a curve $\check{\mathbf{c}}(t)$ fulfilling the geodesic equation [Pet06]

$$\ddot{\check{\mathbf{c}}} = \frac{1}{2}\,\check{\mathbf{M}}^{-1}\left(\nabla(\dot{\check{\mathbf{c}}}^T\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}}) - 2\,(\nabla\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}})\,\dot{\check{\mathbf{c}}}\right) \tag{43}$$

with

$$\check{\mathbf{M}} = \check{\mathbf{M}}(\check{\mathbf{c}}(t)) = \operatorname{diag}\left(\frac{1}{s(t)^2},\frac{1}{s(t)^2},\frac{1}{s(t)^2},\frac{1}{4s(t)^2}\right) \tag{44}$$

$$\nabla(\dot{\check{\mathbf{c}}}^T\,\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}}) = \left(\dot{\check{\mathbf{c}}}^T\,\check{\mathbf{M}}_{mx}\,\dot{\check{\mathbf{c}}},\,\cdots\,,\dot{\check{\mathbf{c}}}^T\,\check{\mathbf{M}}_s\,\dot{\check{\mathbf{c}}}\right)^T \tag{45}$$

$$\nabla\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}} = \left(\check{\mathbf{M}}_{mx},\check{\mathbf{M}}_{my},\check{\mathbf{M}}_{mz},\check{\mathbf{M}}_s\right)\dot{\check{\mathbf{c}}} \tag{46}$$

and $\dot{\check{\mathbf{c}}}$, $\ddot{\check{\mathbf{c}}}$ are the first and second derivatives of $\check{\mathbf{c}}(t)$, and $\check{\mathbf{M}}_{mx} = \check{\mathbf{M}}_{mx}(\check{\mathbf{c}}(t))$, $\check{\mathbf{M}}_{my} = \check{\mathbf{M}}_{my}(\check{\mathbf{c}}(t))$, $\check{\mathbf{M}}_{mz} = \check{\mathbf{M}}_{mz}(\check{\mathbf{c}}(t))$, $\check{\mathbf{M}}_s = \check{\mathbf{M}}_s(\check{\mathbf{c}}(t))$ are directional derivatives of $\check{\mathbf{M}}(mx,my,mz,s)$ at $\check{\mathbf{c}}(t)$. Here, we are using a matrix notation to avoid introducing Christoffel symbols. Setting

$$\check{\mathbf{c}}(t) = \begin{pmatrix}\mathbf{m}(t)\\ s(t)\end{pmatrix} = \begin{pmatrix}\mathbf{m}_0 + m(t)\,\mathbf{d}\\ s(t)\end{pmatrix}, \tag{47}$$

(43) is a system of ODEs for the unknown function $m(t)$, $s(t)$. Fortunately, it has a closed-form solution

$$m(t) = -\frac{1}{2}\,p_1\,\tanh(p_2\,t + p_3) + p_4 \tag{48}$$

$$s(t) = \frac{p_1}{\cosh(p_2\,t + p_3)} \tag{49}$$

and p_1,p_2,p_3,p_4 are free parameters.

To show that this indeed fulfills (43), we apply elementary differentiation rules to (47)–(49), and get

$$\dot{\check{\mathbf{c}}} = \begin{pmatrix}\dot{m}(t)\,\mathbf{d}\\ \dot{s}(t)\end{pmatrix}, \quad \ddot{\check{\mathbf{c}}} = \begin{pmatrix}\ddot{m}(t)\,\mathbf{d}\\ \ddot{s}(t)\end{pmatrix} \tag{50}$$

with

$$\dot{m}(t) = -\frac{1}{2}\,\frac{p_1\,p_2}{(\cosh(p_2\,t+p_3))^2} \tag{51}$$

$$\ddot{m}(t) = \frac{p_1\,{p_2}^2\sinh(p_2\,t+p_3)}{(\cosh(p_2\,t+p_3))^3} \tag{52}$$

$$\dot{s}(t) = -\frac{p_1\,p_2\,\sinh(p_2\,t+p_3)}{(\cosh(p_2\,t+p_3))^2} \tag{53}$$

$$\ddot{s}(t) = \frac{\left((\cosh(p_2\,t+p_3))^2 - 2\right)p_1\,{p_2}^2}{(\cosh(p_2\,t+p_3))^3}. \tag{54}$$

Further, the directional derivatives of $\check{\mathbf{M}}$ at $\check{\mathbf{c}}(t)$ are

$$\check{\mathbf{M}}_{mx}(\check{\mathbf{c}}(t)) = \check{\mathbf{M}}_{my}(\check{\mathbf{c}}(t)) = \check{\mathbf{M}}_{mz}(\check{\mathbf{c}}(t)) = \mathbf{0} \tag{55}$$

$$\check{\mathbf{M}}_s(\check{\mathbf{c}}(t)) = -\operatorname{diag}\left(\frac{2}{s(t)^3},\frac{2}{s(t)^3},\frac{2}{s(t)^3},\frac{1}{2s(t)^3}\right). \tag{56}$$

This gives

$$\nabla(\dot{\check{\mathbf{c}}}^T\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}}) = \begin{pmatrix}\mathbf{0}\\ -\frac{1}{2}\,\frac{{p_2}^2\cosh(p_2\,t+p_3)}{p_1}\end{pmatrix} \tag{57}$$

$$(\nabla\check{\mathbf{M}}\,\dot{\check{\mathbf{c}}})\,\dot{\check{\mathbf{c}}} = \begin{pmatrix}-\frac{{p_2}^2\sinh(p_2\,t+p_3)}{p_1\,\cosh(p_2\,t+p_3)}\,\mathbf{d}\\ -\frac{1}{2}\,\frac{{p_2}^2(\sinh(p_2\,t+p_3))^2}{p_1\,\cosh(p_2\,t+p_3)}\end{pmatrix}. \tag{58}$$

Further,

$$\check{\mathbf{M}}^{-1} = (\check{\mathbf{M}}(\check{\mathbf{c}}(t)))^{-1} = \operatorname{diag}\left(s(t)^2,s(t)^2,s(t)^2,4s(t)^2\right). \tag{59}$$

Then (57), (58), (59) together with (50), (51), (53) proof (43). To show the equal-speed parameterization in (mx,my,mz,s) space, we have to prove that $\dot{\check{\mathbf{c}}}(t)^T\,\check{\mathbf{M}}(\check{\mathbf{c}}(t))\,\dot{\check{\mathbf{c}}}(t)$ is constant over time, i.e. independent of t. In fact, from (50), (51),(53),(46), and keeping in mind that $\mathbf{d}$ is a unit vector, we get

$$\dot{\check{\mathbf{c}}}^T(t)\,\check{\mathbf{M}}(\check{\mathbf{c}}(t))\,\dot{\check{\mathbf{c}}}(t) = \left(\frac{p_2}{2}\right)^2. \tag{60}$$

To ensure end point interpolation of of the geodesic $\check{\mathbf{c}}(t)$, we have to set the free parameters p_1,p_2,p_3 and p_4 to fulfill

$$s(0) = s_0 = \frac{p_1}{\cosh(p_3)}, \quad s(1) = s_1 = \frac{p_1}{\cosh(p_2+p_3)} \tag{61}$$

$$m(0) = 0 \;= -\frac{1}{2}\,p_1\,\tanh(p_3) + p_4 \tag{62}$$

$$m(1) = d \;= -\frac{1}{2}\,p_1\,\tanh(p_2+p_3) + p_4. \tag{63}$$

The solution of (61)–(63) is

$$p_1 = s_0\cosh(r_0)\,, \qquad p_2 = r_0 - r_1$$

$$p_3 = -r_0\,, \qquad p_4 = -\frac{s_0\sinh(r_0)}{2}$$

with r_0,r_1 as defined in Eq. (29). In order to compute geodesics in (ϕ,θ,ψ) subspace with metric $\widehat{\mathbf{M}}$, we consider the rotation matrix $\mathbf{R}(t)$ from the Euler angles $\mathbf{o}(t) = (\phi(t),\theta(t),\psi(t))$ by applying Eq. (26). This gives a time derivative of $\mathbf{R}(t)$ as

$$\dot{\mathbf{R}}(t) = \mathbf{R}(t)\begin{pmatrix}0 & k_3 & -k_2\\ -k_3 & 0 & k_1\\ k_2 & -k_1 & 0\end{pmatrix} \tag{64}$$

with

$$\mathbf{k} = \begin{pmatrix}k_1\\k_2\\k_3\end{pmatrix} = \begin{pmatrix}\sin(\theta)\,\dot{\psi} - \dot{\phi}\\ -\sin(\phi)\cos(\theta)\,\dot{\psi} - \cos(\phi)\,\dot{\theta}\\ -\cos(\phi)\cos(\theta)\,\dot{\psi} + \sin(\phi)\,\dot{\theta}\end{pmatrix} \tag{65}$$

that describes a differential rotation around the axis $\mathbf{k}$ with the angular speed

$$\|\mathbf{k}\| = \sqrt{-2\,\sin(\theta)\,\dot{\phi}\,\dot{\psi} + \dot{\phi}^2 + \dot{\theta}^2 + \dot{\psi}^2}. \tag{66}$$

At the same time, the differential distance in the direction $\dot{\mathbf{o}}T$ in the metric space $\widehat{\mathbf{M}}$ is

$$\sqrt{\dot{\mathbf{o}}^T\,\widehat{\mathbf{M}}\,\dot{\mathbf{o}}} = \frac{1}{\sqrt{6}}\|\mathbf{k}\|. \tag{67}$$

This means that the distance in $\mathbf{o}$ is proportional to the angular speed of $\mathbf{R}(t)$. Because of this, geodesics between two orientation matrices $\mathbf{R}_0$, $\mathbf{R}_1$ are computed in the following way: First, find

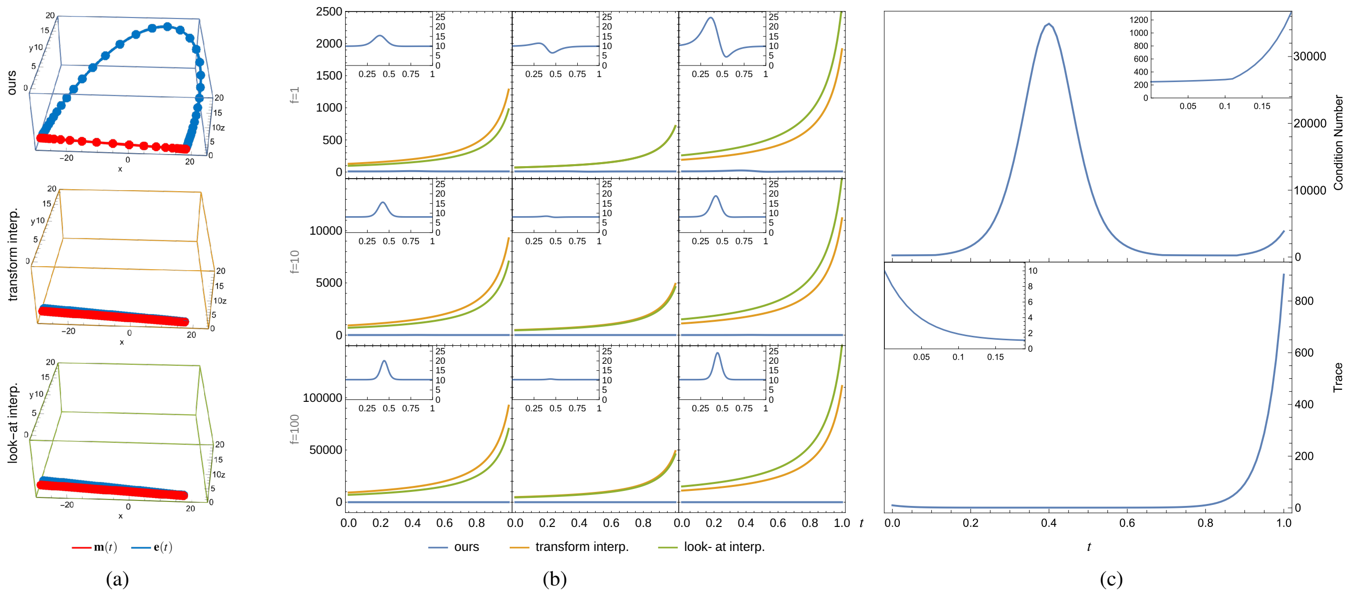

Figure 10: *(a) The curves we use for comparison in this figure. The curve direction is from left to right. While the curve for the transformation interpolation and the look-at interpolation are similar, their eye point path is not identical. (b) The velocity plot for each curve and different f and a_{yz}. Note the scale of the vertical axis for each row. The insets show the plot for the velocity of our curve as a close-up. (c) The trace and condition number of the 9D metric $\widehat{\mathbf{M}}(\widehat{\mathbf{c}}(t))$ for our path with $f = 1$ and $a_{yz} = 1$. The insets show a close-up for $0 \leq t \leq 0.2$.*

the constant rotation axis $\mathbf{k}$ and rotation angle ξ such that rotation around $\mathbf{k}$ with angle ξ transforms $\mathbf{R}_0$ to $\mathbf{R}_1$. Then let $\mathbf{R}(t)$ the rotation around $\mathbf{k}$ with angle $t\xi$ for $t \in [0,1]$. This is done in Section 4.4.

Appendix B: Justification of the Simplification from 9D to 7D

To give evidence that the simplification of the camera space metric is a sensible choice to make, we show that geodesics produced with the simplified metric are still closer to being a geodesic in the original metric, in particular in comparison to the alternatives we chose in Section 6. For that, we need to recall Section 2 and Theorem 1, which stated that the geodesic is a curve of constant velocity in the metric space. This means that

$$\dot{\bar{\mathbf{c}}}(t)^T\, \overline{\mathbf{M}}(\bar{\mathbf{c}}(t))\, \dot{\bar{\mathbf{c}}}(t) = \text{const.} \quad \forall\, t \in [0,1]. \tag{68}$$

What we want to show is that the velocity calculated with the 9D metric instead of the 7D metric is still close to being constant. We are converting the 7D camera path $\bar{\mathbf{c}}(t)$ to its 9D equivalent $\widehat{\mathbf{c}}(t)$ by using the same $\mathbf{m}(t)$ and $\mathbf{o}(t)$ and converting the cube to a camera frustum with aspect ratios

$$a_{xy} = \frac{sx}{sy}, \quad a_{yz} = \frac{sy}{sz}. \tag{69}$$

Additionally, sx, sy, sz need to satisfy $s = (sx \cdot sy \cdot sz)^{\frac{1}{3}}$.

As an example we choose the path depicted in Fig. 10a because it features different scales for $\|\mathbf{m}(t) - \mathbf{e}(t)\|$ at the start and end point without being too extreme. Specifically, $\|\mathbf{m}_0 - \mathbf{e}_0\| = 0.651126$ and $\|\mathbf{m}_1 - \mathbf{e}_1\| = 0.0651126$ while the Euclidean distance between the look at points is $\|\mathbf{m}_0 - \mathbf{m}_1\| = 47.3825$.

Fig. 10b shows the result of the velocity evaluation with the 9D metric for all three paths and different values for f and a_{yz}. The aspect ratio $a_{xz} = 1.78 = (16:9)$ is held constant. As we can see, the variation in speed is much larger for the look-at interpolation and the transformation interpolation than it is for our method. It is especially noticeable at the end of the path when the camera moves closer to the look-at point. This indicates, that even in the 9D metric space, our camera path is much closer to a geodesic than alternative paths. However, the inset in each plot shows that our path is not an actual geodesic in the 9D space because the velocity is not constant. For $f = 100$ and $a_{yz} = 1$, the real frustum is close to the best-fitting cube from the simplified camera space and consequently, the velocity plot for our path is the closest to being constant. We conclude that the simplification from 9D to 7D camera space does not give significantly different geodesics but simplifies the problem of computing geodesics significantly.

Fig. 10c presents the condition number and the trace of the 9D metric along our path as it is depicted in Fig. 10a. While the condition number is largest when the eye point is the farthest away from the look-at point and then rises again at the end of the path, the trace is larger when the camera is close to the look-at point. Our path is already close to being geodesic, so we can assume that the behavior for the real 9D geodesic path will be similar. In consequence, a numerical method would need to be able to deal with the variations in both condition number and trace to converge to a solution.

© 2024 The Authors.
Computer Graphics Forum published by Eurographics and John Wiley & Sons Ltd.