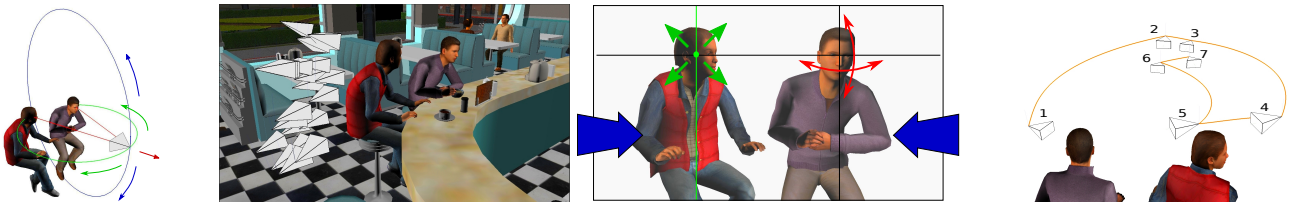
Intuitive and Efficient Camera Control with the Toric Space

Christophe Lino, Marc Christie

Intuitive and Efficient Camera Control with the Toric Space

Christophe Lino*
INRIA Rennes / IRISA, France

Marc Christie†
University of Rennes 1 / IRISA, France



The Toric space model | For computing viewpoints | For manipulating viewpoints | For interpolating viewpoints

Figure 1: *We present (a) the Toric space, a novel and compact representation for intuitive and efficient virtual camera control. We demonstrate the potential of this representation by proposing (b) an efficient automated viewpoint computation technique, (c) a novel and intuitive screen-space manipulation tool, and (d) an effective viewpoint interpolation technique.*

Abstract

A large range of computer graphics applications such as data visualization or virtual movie production require users to position and move viewpoints in 3D scenes to effectively convey visual information or tell stories. The desired viewpoints and camera paths are required to satisfy a number of visual properties (*e.g.* size, vantage angle, visibility, and on-screen position of targets). Yet, existing camera manipulation tools only provide limited interaction methods and automated techniques remain computationally expensive.

In this work, we introduce the *Toric space*, a novel and compact representation for intuitive and efficient virtual camera control. We first show how visual properties are expressed in this Toric space and propose an efficient interval-based search technique for automated viewpoint computation. We then derive a novel screen-space manipulation technique that provides intuitive and real-time control of visual properties. Finally, we propose an effective viewpoint interpolation technique which ensures the continuity of visual properties along the generated paths. The proposed approach (i) performs better than existing automated viewpoint computation techniques in terms of speed and precision, (ii) provides a screen-space manipulation tool that is more efficient than classical manipulators and easier to use for beginners, and (iii) enables the creation of complex camera motions such as long takes in a very short time and in a controllable way. As a result, the approach should quickly find its place in a number of applications that require interactive or automated camera control such as 3D modelers, navigation tools or 3D games.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques— [G.1.6]: Mathematics of Computing—Numerical Analysis[Optimization]

Keywords: Virtual Camera Composition, Interactive Camera Control, Virtual Camera Interpolation

1 Introduction

Virtual camera control is an essential component of many computer graphics applications. The virtual camera – as a window on 3D contents – conveys information, sense of aesthetics, and emotion. The proper selection of viewpoints and the proper design of camera paths are therefore of prime importance to precisely convey intended effects. Furthermore, the increased availability of realistic real-time rendering workstations as well as mobile devices and their growing usage in our everyday tasks, both call for interactive and automated techniques that would simplify the creation of effective viewpoints and speed up the overall design process.

To address this task, modeling and animation software propose different camera manipulation tools. However, most tools rely on the underlying mathematical representations of cameras and camera paths. A camera is therefore manipulated through a sequence of translation and rotation operations like any other node of the scene graph, and a camera path is constructed through a sequence of manually controlled spline-interpolated key-frames. While some visual widgets may assist the users, the precise control of a viewpoint remains a complex task, especially for beginners.

In the literature, a number of techniques have been proposed to ease the control of virtual cameras through contributions such as screen-space manipulations, automated viewpoint computation from visual properties, or automated path-planning techniques. See [Christie et al. 2008] for a detailed overview. However most contributions address a single aspect at a time (viewpoint computation, camera manipulation or path-planning). Furthermore, the problem of placing and moving virtual cameras has essentially been addressed through

*e-mail: christophe.lino@irisa.fr
†e-mail: marc.christie@irisa.fr

optimization techniques by expressing properties as cost functions over the degrees of freedom of the camera, generally resulting in significant computational costs. An ongoing challenge in the field is therefore to propose an approach that is expressive (in its capacity to model visual properties), computationally efficient and provides interactive control on the cameras.

Furthermore, as reported by [Sudarsanam et al. 2009], most camera control tools are based on the photographer's approach where the user or the system manipulates the camera as if it were in their hands. In contrast, artists are more interested in the qualitative features of the image, *i.e.* the visual layout of the elements composing the viewpoint including features such as position, size, vantage angle, or foreground and background contents.

In this work, we propose a novel and compact representation for virtual camera control called the *Toric space* (see Figure 1). We show how to use this representation to address three important challenges in the field: (i) the efficient computation of viewpoints from visual properties, (ii) the intuitive control of viewpoints through screen-space manipulation of visual properties and (iii) the effective computation of camera paths that preserve visual properties between viewpoints.

The *Toric space* is a generalization of the *Toric manifold* representation, introduced by Lino and Christie [2012], to a 3-dimensional search space $(\alpha, \theta, \varphi)$. This representation can actually be viewed as a generalization of the arcball camera principle [Shoemake 1992] to two targets. Where a normal arcball camera is defined in polar coordinates, locally to one target and always pointing to the center of the target, the Toric space defines three parameters such that every triplet $(\alpha, \theta, \varphi)$ represents a single camera position, with a camera orientation that automatically ensures a specified on-screen composition of its two targets.

A key benefit of this representation is that classical visual properties related to camera control (position, size and vantage angle of targets) can be easily expressed in the *Toric space*: whole regions that do not satisfy a visual property can be characterized and pruned. We rely on this pruning to propose a novel interval-based search algorithm that automatically computes the best viewpoint satisfying a user-defined set of properties, and compare our technique with recent results that use stochastic optimization (see Section 4). We then present two applications to camera control that rely on the *Toric space*: an intuitive screen-space manipulation technique (see Section 5) and a novel viewpoint interpolation technique (see Section 6).

The contributions of this work are:

- A novel representation for camera control that provides a compact search space in which viewpoint optimization problems can be efficiently addressed. This representation reduces a number of camera optimization problems from a 7-DOF search (position, orientation and field of view) to a 4-DOF search (position and field of view), for the class of problems that involve at least two targets on the screen. Our approach shows to be more efficient than recent stochastic-based techniques [Ranon and Urli 2014] and more precise.

- A novel interaction metaphor that offers intuitive screen-space manipulations through the interactive control of visual properties (such as size, vantage angle or location of targets), while maintaining constraints on others. As a result, the task of composing the layout of a viewpoint in a 3D environment is made simpler and more intuitive. This contribution is illustrated by a user evaluation comparing classical camera manipulation techniques available in 3D modelers with our technique.

- A novel interpolation technique between viewpoints which ensures the maintenance of visual properties along the generated path. The technique enables the rapid prototyping of effective and complex camera sequences such as long-takes, with very few operations.

Furthermore, all the contributions have been implemented as a plugin in the Autodesk MotionBuilder® tool [1] and this plugin has been used to run all the user evaluations and build the footage for the companion video.

2 Related Work

There is a large literature related to the control of cameras in virtual environments (see [Christie et al. 2008]). In the following section, we restrict the study to the approaches directly related with our work, namely automated viewpoint computation, screen-space manipulation for camera control, and viewpoint interpolation.

2.1 Automated viewpoint computation

The problem of automated viewpoint computation is expressed in a very elegant way by referring to the notion of *viewpoint entropy* [Vázquez et al. 2001]. Viewpoint entropy refers to the amount of information contained in a scene that is actually conveyed by a given viewpoint of the scene. Automated computation then searches for viewpoints maximizing this entropy. A measure of entropy can be defined through the aggregation of a number of visual descriptors such as an object's projected surface on the screen, it's saliency, curvature, or silhouette. Interestingly, descriptors may also include more aesthetic properties related to photographic composition such as the quality of the layout on the screen, the visual weights of the targets or diagonal dominance.

A related problem is the one of computing a viewpoint that needs to satisfy a number of visual features (size of a target, position, orientation, or visibility). The early work of Blinn [1988] proposes an iterative formulation to positioning two targets on the screen, one of them at a specified distance to the camera and the other with a given orientation. Lately the technique was improved using an efficient algebraic formulation [Lino and Christie 2012]. However, both methods target very specific problems relying on the assumption of an exact two-target onscreen positioning task, and lack solutions for more complex problems.

Our work generalizes [Lino and Christie 2012] in that it removes the assumption on exact on-screen positioning; in our model, targets can be constrained to regions of the screen and it supports interaction for one or two targets. It also offers more expressive properties by defining ranges of accepted values for vantage angle, target size and framing. It finally includes a novel and more general search process.

When more properties are involved, or those properties cannot be expressed as algebraic relations (*e.g.* visibility), it is necessary to switch to optimization techniques. There is an extensive body of literature on optimization approaches which spans from Drucker [1994] to Ranon [2014]. Languages have been specified to define visual properties [Olivier et al. 1999; Ranon et al. 2010], and to express these properties in general purpose solvers [Drucker and Zeltzer 1994; Olivier et al. 1999] or dedicated solvers [Bares et al. 2000; Christie and Normand 2005; Ranon and Urli 2014]. While computational costs have been for a long time a central issue, recent techniques based on stochastic approaches (particle swarm optimization [Ranon and Urli 2014]) offer close to real-time perfor-

[1] Autodesk MotionBuilder® 2014, `http://www.autodesk.com/products/motionbuilder/overview`

mance. However, they rely on sampling the space of locally satisfying regions for each property. The costs related to the satisfaction of visual properties are also generally aggregated into a single cost function using a linear weighted combination. Most solving techniques therefore fail at detecting inconsistencies in the properties and cannot distinguish a solution where all properties are half satisfied, from a solution where half of the properties only are satisfied.

In comparison, the technique we propose allows sampling the space of globally satisfying regions, by pruning non-satisfactory regions. It is consequently more efficient than existing techniques, is deterministic, and reports failure when the problem contains inconsistencies, or has no solution for a given precision.

2.2 Screen-space manipulation for camera control

The principle of screen-space manipulation for camera control is to offer indirect control of camera parameters by directly manipulating properties of the on-screen content.

Shoemake [1992] proposed the Arcball manipulation that enables intuitive rotations around the center of a target object by using a simple algebraic formulation. Through the manipulation, the camera moves on the surface of a sphere around the object while maintaining the look-at vector pointed at the target. This arcball manipulation has been generalized to perform navigation over the surface of objects through techniques such as Hovercam [Khan et al. 2005], Isocam [Marton et al. 2014] or Shellcam [Boubekeur 2014].

These techniques however do not address the specific problem of composing shots. A sophisticated screen-space manipulation technique was proposed by Gleicher and Witkin [1992] with their Through-The-Lens camera control approach. By directly manipulating the objects' locations on the screen (*i.e.* the projected positions of 3D points), the technique proposes to automatically recompute the camera parameters that satisfy the desired on-screen locations. The problem is expressed by minimizing an energy function between the actual and desired on-screen locations of objects. However when controlling two points on the screen, the problem is under-constrained; with three points, the problem only has two exact solutions – it is known as the P3P problem [Fischler and Bolles 1981]; and from four points, the problem is over-constrained and looks at minimizing the on-screen error. Though intuitive, the manipulation is difficult to use in practice for viewpoint manipulation tasks, since the problem is often over or under-constrained. Furthermore, the technique is limited to the manipulation of on-screen points only, an aspect later addressed by Courty and Marchand [2003] who expressed properties such as occlusion of targets, tracking of secondary objects or enforcement of textbook cinematographic trajectories.

Through the design of on-screen interactive widgets, multiple techniques have been proposed to ease the manipulation of the camera parameters. The IBar technique [Singh et al. 2004] proposes a perspective widget representation in which all the camera parameters can be controlled, and some of them in a simultaneous way. The interactions encompass camera-centric operations (pan, spin, zoom, dolly, rotate, center of projection) and object-centric operations that provides means for intuitive interactions such as changing the horizon, or changing the vanishing points. The approach has been further enhanced in the Cubecam interface [Sudarsanam et al. 2009]. While the level of control and possibilities of interaction are impressive, the control is only performed on the widget itself (not on the scene) and the interface appears complex.

Despite the range of tools to achieve manipulation tasks for camera control, few offer intuitive screen-space manipulation tools. Our contribution offers the advantage of through-the-lens manipulation [Gleicher and Witkin 1992], the simultaneous control of camera parameters, the expressiveness of Cubecam [Sudarsanam et al. 2009], and the possibility to manipulate visual properties [Courty et al. 2003] while constraining others.

2.3 Camera path planning

The planning of camera paths naturally owes a lot to contributions in robotics. For example, Nieuwenhuisen and Overmars [2004] rely on probabilistic roadmaps [Kavraki, LE Lydia and Svestka, Petr and Latombe, Jean-Claude C and Overmars, Mark H. 1996] to construct rough camera paths joining an initial camera position to a final camera position. Their technique then smooths out the computed path and adds anticipation in the camera's orientation during sharp turns. Probabilistic roadmaps have also been used to track virtual characters in 3D environments as in [Li and Cheng 2008]. Interestingly, the sampling process and the construction of the roadmap are performed in the local basis of the moving character, which ensures that the camera properly follows the target. The nodes and arcs in the roadmap are dynamically updated when collision with the environment occurs. Assa *et al.* [2008] propose an offline method based on the maximization, through a simulated annealing algorithm, of a viewpoint entropy (measuring how much of a single target's motion project onto the screen) along time while ensuring the camera path smoothness in terms of speed, acceleration and change in orientation. Yeh *et al.* [2011] improve the viewpoint entropy, and the search efficiency by using multiple A*-based searches and a backtracking mechanism.

In addressing tasks such as tracking objects and performing transitions between viewpoints, Oskam *et al.* [2009] rely on a static sampling of the free space in a 3D environment using fixed-sized spheres in order to (i) precompute a visibility graph between every pairs of spheres and (ii) compute a roadmap by creating edges between connecting spheres. At runtime, the algorithm relies on the roadmap to select the viewpoint which offers the best visibility on a moving target, or to compute transitions between two targets by maximizing the visibility of one of them along the computed path.

In the specific context of virtual cinematography (see [Yeh et al. 2012]), Lino *et al.* [2010] proposes a technique based on the dynamic computation of spatial partitions around moving targets. Spatial partitions are tagged with semantic information from film textbooks including shot size (*e.g.* medium close up, medium shot, long shot) and shot angle (*e.g.* internal, external, apex). A roadmap is then constructed by connecting the spatial partitions.

In contrast, we propose an efficient viewpoint interpolation technique that maintains multiple visual properties along the path, and offers an intuitive control on how to perform this transition (in [Oskam et al. 2009; Lino et al. 2010] the process is fully automated).

3 The Toric space

The *Toric space* is an generalization of the *2D manifold* representation [Lino and Christie 2012] to a 3-dimensional space represented as a triplet of Euler angles $(\alpha, \theta, \varphi)$. One manifold corresponds to a 3D surface, generated by a constant angle α between a pair of targets (A, B) and the camera; such a surface is also parametrized by a pair of Euler angles (θ, φ) defining horizontal and vertical angles around targets. A target represents any 3D object for which we know its center position in 3D. The manifold is defined in a way that every camera positioned on this surface can view the center of targets A and B at user-defined proofread(exact) on-screen locations. However, this representation abstracts target objects as points and cannot address more evolved visual properties.

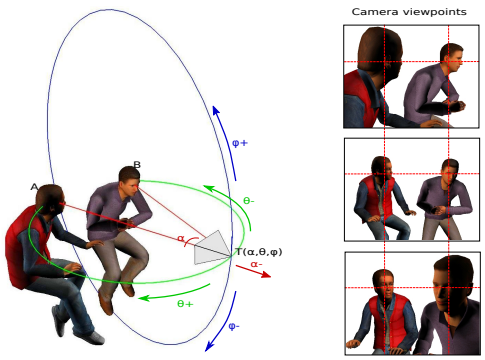


Figure 2: *In the Toric space representation, a viewpoint is parametrized with a triplet of Euler angles $(\alpha, \theta, \varphi)$ defined around a pair of targets; α defines the angle between the camera and both targets –it generates a manifold surface on which to position the camera–, θ defines an horizontal angle around the targets and φ defines a vertical angle around the targets.*

Our *Toric space* represents the whole set of manifolds that may be generated around a pair of targets (A, B) without knowing their exact on-screen locations. This space is locally defined with relation to these targets (see Figure 2). Using such a representation, the conversion from a camera in its Toric representation $T(\alpha, \theta, \varphi)$ to its Cartesian representation $C(x, y, z)$ is defined by the following algebraic relation:

$$C = A + (q_\varphi \cdot q_\theta \cdot AB).\sin(\alpha + \theta/2)$$

where q_φ is the quaternion representing the rotation angle φ around vector AB, q_θ is the quaternion representing the rotation angle $\theta/2$ around a vector t orthogonal to vector AB (t is defined as pointing up as much as possible so that viewpoints verifying $\varphi = 0$ are as close as possible to the eye-level of both targets). The last part of the equation represents the distance between the camera and the target A.

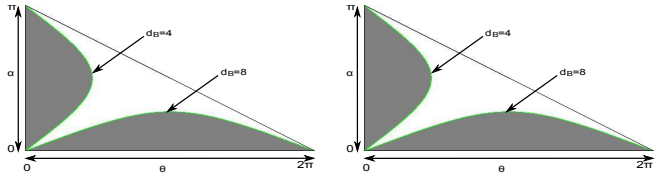
This representation can be used in tasks such as automated viewpoint computation; it reduces the search space from 7D (a standard camera is represented by its 3D position, 3D rotation and field of view) to 4D (3D vector in the Toric space and field of view) for all classes of problems that involve viewing at least two targets on the screen (considering that different parts of a single object can be considered as different targets).

More importantly, the key visual properties in camera control such as a target's on-screen positioning, on-screen size, vantage angle, or distance to camera can be expressed directly in the Toric space. We here provide an overview of how these properties are expressed in the Toric Space, as 1D or 2D solution sets. Note that more details on the computations can also be found in [Lino 2013].

3.1 On-screen positioning

One central problem is, given a field of view angle, to find the set of camera settings (position and orientation) satisfying a given on-screen positioning of targets. In contrast with [Lino and Christie 2012], we propose to use a soft definition of the framing of two targets (A and B), through their desired on-screen positions (p_A and p_B) and their accepted deviations from these positions (s_A and s_B) referred to as frames. In a practical way, each frame is defined as a 2D polygonal convex shape on the screen, within which the target should be projected. We first express the range of solution camera positions, then show how an appropriate camera orientation can be computed.

Figure 3: *Constraining the projection of targets A and B in on-screen convex shapes s_A and s_B reduces the domain of variable α in the Toric space. The set of cameras which satisfy the framing constraint (white area) is then given by a horizontal strip $\alpha \in [\alpha_{\min}, \alpha_{\max}]$ in the plane (θ, α).*



(a) Distance to A (b) Distance to B

Figure 4: *Solution sets (in white) corresponding to all camera positions within a range of distances to targets A and B. (a) Solution pairs (α, θ) for a distance to A within $[5, 10]$; each red curve corresponds to a bounding value of the interval of distance. (b) Solution pairs (α, θ) for a distance to B within $[4, 8]$; each green curve corresponds to a bounding value of the interval of distance.*

Camera positions. We build upon the solution of a simpler problem which considers two exact screen positions of A and B belonging to frames s_A and s_B respectively. This yields a specific angle between the camera and the targets A and B (*i.e.* a specific value α_i for variable α) which defines a specific 2D manifold surface in the Toric space. We then use the pairwise combination of all edges from s_A and s_B to compute a set of accepted values α_i. We finally prune the domain of variable α to a domain interval $r_\alpha = [\alpha_{\min}, \alpha_{\max}]$ representing the hull enclosing all computed values of α_i. The expression of this framing constraint then corresponds to a horizontal strip in the plane (θ, α), as shown in Figure 3. This solution interval on α does not depend on the values of parameters θ and φ.

Camera orientation. Given the range of accepted camera positions, we now compute a proper camera orientation for a given position. The method we propose enforces a user-defined camera roll angle ψ while maintaining as much as possible the desired on-screen positioning of targets. If the roll is left free, canted angle shots (shots for which the horizon is oblique) are generated, which cause unease and disorientation. Instead, the user can specify the angle he desires in order to enforce canted shots or keep the roll at zero. Our computation is a 3-step process. First, we define a "look-at" camera orientation as a quaternion q_{look} computed from the pair of targets. Similar to the classical look-at definition, q_{look} is computed as a mean direction l from the camera position C to each target: $l = \frac{1}{2}\left[\frac{CA}{\|CA\|} + \frac{CB}{\|CB\|}\right]$

Second, we compute a transformation rotation q_{trans} to apply to the camera; this aims at placing the screen projections of A and B as close as possible to their desired positions p_A and p_B, respectively at the center of the regions s_A and s_B. To do so, we define two points $p_O(0, 0)$ the origin of the screen and $p_M(x_M, y_M)$ the center point between p_A and p_B. We then build two 3D vectors p_O^3 and

p^3_M which represent the set of points projecting respectively at p_O and p_M on the screen. These vectors are expressed in the local basis of the camera

$$p^3_O(0,0,1) \text{ and } p^3_M\left(\frac{x_M}{S_x}, \frac{y_M}{S_y}, 1\right) \text{ scaled to unit length}$$

where $S_x = [\tan(\phi_x/2)]^{-1}$ and $S_y = [\tan(\phi_y/2)]^{-1}$. Values ϕ_x, ϕ_y are the horizontal and vertical camera field of view angles respectively. We then define q_{trans} as the rotation of angle (p^3_O, p^3_M) around the axis $m = p^3_M \times p^3_O$.

Third, we define a rotation q_ψ, of angle ψ around the camera's front direction; this represents the application of the desired roll angle ψ to the camera. The final camera orientation q is then expressed by

$$q = q_\psi \cdot q_{look} \cdot (q_{trans})^{-1} \tag{1}$$

As a result, our method can be viewed as a generalization of the standard look-at operator, which integrates the on-screen positioning of targets. Knowing a given camera position C in the Toric space, the computation of the corresponding orientation is (i) algebraic – hence fast and deterministic–, and (ii) allows the enforcement of an input camera roll angle ψ. This stands in contrast with previous techniques in automated viewpoint computation that consider searching over the camera orientation parameters when trying to minimize the on-screen errors [Olivier et al. 1999; Ranon and Urli 2014].

3.2 Distance

The distance property represents the distance to ensure between a target and the camera. We consider that this distance is specified using an interval of accepted values $[d_{min}; d_{max}]$. Our objective is thus to derive the subset of camera positions in the Toric space that satisfy this constraint. This problem can be expressed as a 2D solution set in the plane (θ, α). We detail the resolution for an exact distance to a target and we extend it to consider an interval of distances.

Exact distance to A. Assuming that the distance to the target A must be exactly d_A, a camera position satisfies this constraint iff it verifies the equation

$$\alpha = \text{ acos}\left(\frac{d_A - \|AB\| \cdot \cos(\theta/2)}{\sqrt{d_A^2 + \|AB\|^2 - 2 \cdot \|AB\| \cdot d_A \cdot \cos(\theta/2)}}\right) \tag{2}$$

The corresponding set of camera positions is displayed in red on Figure 4(a) for two distances d_A respectively defined at 5 and 10.

Exact distance to B. Assuming that the distance to the target B must be exactly d_B, there are here two possible cases: either $d_B \leq \|AB\|$ or $d_B > \|AB\|$. When $d_B \leq \|AB\|$, the camera position should verify $\theta \in \left]0, 2 \text{ asin}\left(\frac{d_B}{\|AB\|}\right)\right]$ (*NB:* in the particular case $d_B = \|AB\|$ the upper bound, *i.e.* π, is excluded). The camera position should also verify the equation

$$\alpha = \frac{\pi}{2} \pm \text{ acos}\left[\frac{\|AB\|}{d_B} \sin\left(\frac{\theta}{2}\right)\right] \tag{3}$$

Similarly, when $d_B > \|AB\|$, the camera position should verify the equation

$$\alpha = \frac{\pi}{2} - \text{ acos}\left[\frac{\|AB\|}{d_B} \sin\left(\frac{\theta}{2}\right)\right] \tag{4}$$

The corresponding set of camera positions is displayed in green on Figure 4(b) for two distances d_B respectively defined at 5 and 10.

Interval of accepted distance to A. To extend the distance specification to an interval $[d^A_{\min}, d^A_{\max}]$, we define the solution set as a function returning an interval on α for every value of θ:

$$\mathcal{I}^A_\alpha(\theta) = [\alpha^A_{\max}(\theta), \alpha^A_{\min}(\theta)] \backslash \{0, \pi\}$$

where $\alpha^A_{\min}(\theta)$ (respectively $\alpha^A_{\max}(\theta)$) is determined by replacing $d^A_{\min}$ (respectively $d^A_{\max}$) in Equation 2.

Interval of accepted distance to B. In a way similar to A, to consider an interval of distance $[d^B_{\min}, d^B_{\max}]$, we define the solution set as the function:

$$\mathcal{I}^B_\alpha(\theta) = \mathcal{I}^B_{\alpha,\max}(\theta) - \mathcal{I}^B_{\alpha,\min}(\theta)$$

where $\mathcal{I}^B_{\alpha,\min}(\theta)$ (respectively $\mathcal{I}^B_{\alpha,\max}(\theta)$) is determined by replacing $d^B_{\min}$ (respectively $d^B_{\max}$) in Equation 3 or 4.

Thus, when constraining the camera with intervals of distances to both A and B, the set of solution positions is computed as the intersection of both intervals on α, for every value of θ and φ $\left(\textit{i.e. } \mathcal{I}_\alpha(\theta) = \mathcal{I}^A_\alpha(\theta) \cap \mathcal{I}^B_\alpha(\theta)\right)$. Interestingly, this computation does not depend on the value of the variable φ.

3.3 Projected Size

The visual property related to projected size is defined as an interval $[s_{min}, s_{max}]$ of accepted screen areas (in terms of a percentage of the screen). The property can be easily expressed in the Toric space by rewriting the size constraint as a distance constraint. The drawback is that the target needs to be approximated by an enclosing sphere $\mathcal{S}$ of radius r (a solution used in a number of approaches [Christie and Languénou 2003; Olivier et al. 1999]).

Lets first assume we have pre-computed a camera position and orientation for which the target is centered at the origin of the screen. In such a case, the on-screen projection of the target's bounding sphere $\mathcal{S}$ is an ellipse which parameters a and b are computed as

$$a = \frac{r \cdot S_x}{d} \text{ and } b = \frac{r \cdot S_y}{d}$$

Here d represents the distance between the camera and the target. Given a target size s to reach, we can then determine the appropriate distance d to the camera:

$$d = r\sqrt{\frac{\pi \cdot S_x \cdot S_y}{4s}} \tag{5}$$

Given a set of target sizes in the range $[s_{min}, s_{max}]$ we can then compute the corresponding distances d_{min} and d_{max}, which correspond respectively to sizes s_{max} and s_{min}, and rely on Equation 5 to determine the solution set in the Toric space.

3.4 Vantage angle

A vantage property corresponds to a relative angle around a target. For instance, if we want to see a target from the front with a high angle, one can express it as a desired direction vector from the target. The solutions to this exact constraint is therefore a half-line whose origin is the target and whose direction is the vector v. By considering a possible deviation angle γ to this reference direction, the solution set corresponds to a vantage cone of directrix v and half-angle γ. Solution cameras then belong to this cone. The intersection of this cone with the Toric space is in fact complex to compute. We can however derive this computation by computing the intersection of the cone with a Toric manifold surface (*i.e.* a

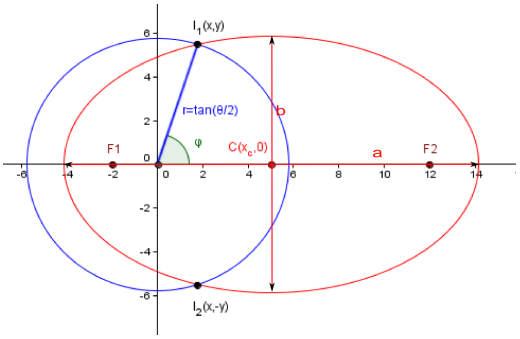
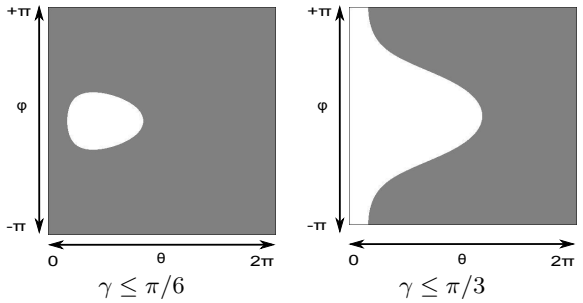


Figure 5: *Computation of the vantage function in the space (β, φ) in the case of an ellipse. The resolution is done through the intersection of the ellipse with a circle of radius $r = \tan(\beta)$. This resolution is similar in case of a parabola or a hyperbola.*



$\gamma \leq \pi/6$ $\gamma \leq \pi/3$

Figure 6: *Solution range of a vantage angle, for a given view direction (vantage vector) and an accepted angular deviation γ. In these examples, the angle between the line (AB) and the vantage vector is $\frac{\pi}{4}$. In each case, the white area represents the set of pairs (θ, φ) satisfying the vantage angle constraint.*

surface generated for a specific value of angle α). We use this computation to show that this vantage constraint can be expressed as a 2D solution set into the plane (θ, φ).

Let's first introduce two additional angles: $\beta, \beta' \in [0, \pi]$. β is the angle between the vector AB and the vector v_a whose origin is the target A and destination is the camera C. In the same way, β' is the angle between the vector BA and the vector v_b whose origin is the target B and destination is the camera C. Using the inscribed angle theorem, there is the following algebraic relationship between the Toric representation $(\alpha, \theta, \varphi)$ and these two angles:

$$\theta = 2\beta = 2(\pi - \alpha - \beta') \tag{6}$$

The solution set is computed in a 2-stage process. Assuming that the constraint is on the target A, we first cast the vantage problem as a resolution on the plane (β, φ). We then express the resulting solution set into the plane (θ, φ) by using the Equation 6. This process is the same for a constraint on target B, replacing β with β' in the formulas below.

Resolution method for target A. To determine the set of solutions for a vantage angle on A, we first introduce three planes $\mathcal{P}^{\beta<\pi/2}$, $\mathcal{P}^{\beta=\pi/2}$ and $\mathcal{P}^{\beta>\pi/2}$. Each of them is a plane whose normal is the vector AB, and for which the signed distance to A is respectively 1, 0, and -1. Each plane is built such that for a given vector v starting from the target A, the half-line of direction v will intersect only one of these planes: $\mathcal{P}^{\beta<\pi/2}$ if the angle $\beta = (AB, v)$ is lower than $\pi/2$, $\mathcal{P}^{\beta=\pi/2}$ if β equals exactly $\pi/2$, and $\mathcal{P}^{\beta>\pi/2}$ if β is greater than $\pi/2$.

We now tackle the problem of computing the intersection of a vantage cone of directrix v and the set of Toric manifolds. To do so, we build upon the intersection of the vantage cone and of the three planes we introduced. Remember that the intersection of a cone and a plane is a conic section $\mathcal{C}$. Consequently, we detect, then use, the appropriate equation of conic section to express the bounds of this intersection. Note that the intersection of the half-line of direction v with the plane can be expressed in polar coordinates as a point $p(\rho, \varphi)$ with $\rho = \tan(\beta)$. Next, for each possible value of the parameter φ, we compute the interval of solution values for parameter β (*i.e.* the set of points p which belong to the conic section). This resolution is illustrated for the case of an ellipse in Figure 5.

To compute the bounds of this interval, we first use the intersection(s) $I(x, y)$ (in Cartesian coordinates) of the conic section $\mathcal{C}$ (*e.g.* an ellipse) with a circle of radius $r = \tan(\beta)$. The upper and lower bounds of the solution interval $\mathcal{I}_\beta(\varphi)$ are calculated by using the formula $\beta = \text{atan}\left(\sqrt{x^2 + y^2}\right)$ We finally cast the computed interval $\mathcal{I}_\beta(\varphi)$ into an appropriate interval $\mathcal{I}_\theta(\varphi)$. In the case of a vantage angle property on A, we obtain $[\theta_{min}; \theta_{max}] = [2\beta_{min}; 2\beta_{max}]$. Examples of solutions, for various accepted deviations, are given in Figure 6.

4 Efficient viewpoint computation

Though no closed form solution has been found to date, the different visual properties we have defined present the benefit of efficiently pruning the Toric space. Relying on this feature, we propose a novel interval-based search algorithm which addresses the viewpoint computation problem (or virtual camera composition problem as defined in [Ranon and Urli 2014]). It consists in searching for the best viewpoint that satisfies a set of visual properties. Our algorithm incrementally prunes the domains of variables $(\alpha, \theta, \varphi)$ from the visual properties, and the resulting intersection determines the area of consistent solutions. We compare the efficiency of our technique in terms of precision and computational cost with recent stochastic techniques.

4.1 Combining Constraints

Our pruning algorithm is comprised of four consecutive steps. In the first three steps, we sample the ranges of possible values on successively the variables φ, θ, and α (to prune inconsistent viewpoints regarding the visual properties); these steps allow computing triplets $(\alpha_k, \theta_j, \varphi_i)$, all satisfying the specified visual properties. The last step consists in checking the targets' visibility (to prune viewpoints with insufficient visibility) for each computed triplet. This sampling-based technique therefore computes a representative set of solution viewpoints.

To define the sampling rate, we use a predefined number N of samples which we translate, at each step of the process, into a progressive sampling density computed to follow a uniform distribution of samples within the Toric space bounds (NB: the total volume of the Toric space is $2\pi^3$):

$$d_\varphi = 2\sqrt[3]{\frac{N}{2}},\ d_\theta = 4\sqrt[3]{\frac{N}{2}}^{-2} \text{ and } d_\alpha = N$$

These densities lead to a regular sampling distributed over all the intervals that are to be computed respectively in the first, second, and third step. The predefined number of samples can then be used to control the execution time of the solving process in situations where the time budget is limited (for instance, in Table 1(a), we

Time window (ms)	Method	Search time (ms)	Satisfaction mean	Satisfaction std. dev.
5	Ranon & Urli	5	85%	4.8
	Toric space	**3**	**89%**	**0**
10	Ranon & Urli	10	88%	2.5
	Toric space	**6**	**89%**	**0**
20	Ranon & Urli	18	90%	1.6
	Toric space	**13**	**90%**	**0**
40	Ranon & Urli	35	90%	1.2
	Toric space	**26**	**90%**	**0**
100	Ranon & Urli	86	91%	0.9
	Toric space	**51**	**90%**	**0**
200	Ranon & Urli	169	91%	0.5
	Toric space	**137**	**90%**	**0**

(a) Problems defined by Ranon and Urli [2014]

Time window (ms)	Method	Search time (ms)	Satisfaction mean	Satisfaction std. dev.
5	Ranon & Urli	5	9%	0
	Toric space	**3**	**75%**	**0**
10	Ranon & Urli	10	12%	11.1
	Toric space	**3**	**75%**	**0**
20	Ranon & Urli	20	10%	3.3
	Toric space	**19**	**93%**	**0**
40	Ranon & Urli	40	18%	18.2
	Toric space	**29**	**96%**	**0**
100	Ranon & Urli	99	24%	21.4
	Toric space	**77**	**100%**	**0**
200	Ranon & Urli	199	30%	25.5
	Toric space	**77**	**100%**	**0**

(b) Precise framing

Table 1: *Comparison of our technique with Ranon and Urli in measuring time and satisfaction of visual properties: (a) average values for five viewpoint computation problems defined by Ranon and Urli (b) average values for a single viewpoint computation problem with a precise framing property (see Section 4.2).*

used around 60 and 380 samples to solve problems within a 5ms and 40ms time window respectively).

In the first step, the range $\mathcal{I}_\varphi$ of possible values on the variable φ is computed as

$$\mathcal{I}_\varphi = \mathcal{I}_\varphi^A \cap \mathcal{I}_\varphi^B$$

Practically, $\mathcal{I}_\varphi^A$ and $\mathcal{I}_\varphi^B$ represent the intervals related to possible vantage properties respectively defined on targets A and B. By default, *i.e.* when no vantage property is formulated, these intervals are set to $[-\pi; +\pi]$. We regularly sample the interval $\mathcal{I}_\varphi$; this yields a number of values φ_i, used as inputs to the next step.

In the second step, the range $\mathcal{I}_\theta(\varphi_i)$ of possible values on the variable θ is computed as

$$\mathcal{I}_\theta(\varphi_i) = \mathcal{I}_\theta^A(\varphi_i) \cap \mathcal{I}_\theta^B(\varphi_i)$$

Practically, $\mathcal{I}_\theta(\varphi_i)$ is computed using the solution sets of the vantage angle properties, $\mathcal{I}_\theta^A(\varphi_i)$ and $\mathcal{I}_\theta^B(\varphi_i)$ respectively. We regularly sample the range $\mathcal{I}_\theta(\varphi_i)$; this yields a number of pairs (φ_i, θ_j), used as inputs to the next step.

In the third step, the range $\mathcal{I}_\alpha(\theta_j, \varphi_i)$ of possible values on the variable α is computed as follows. We first compute 3 ranges:

- $\mathcal{I}_\alpha^{OSP}$, which corresponds to the satisfaction of the on-screen positioning of both subjects (see Section 3.1);
- $\mathcal{I}_\alpha^{DIST}(\theta_j)$, which corresponds to the satisfaction of distance properties for both subjects (see Section 3.2);
- $\mathcal{I}_\alpha^{VANT}(\theta_j, \varphi_i)$, which corresponds to the satisfaction of vantage angles of both subjects (see Section 3.4).

From this, we compute the range $\mathcal{I}_\alpha(\theta_j, \varphi_i)$, which corresponds to the satisfaction of all the properties, as

$$\mathcal{I}_\alpha(\theta_j, \varphi_i) = \mathcal{I}_\alpha^{OSP} \cap \mathcal{I}_\alpha^{DIST}(\theta_j) \cap \mathcal{I}_\alpha^{VANT}(\varphi_i, \theta_j)$$

Then, we regularly sample the range $\mathcal{I}_\alpha(\theta_j, \varphi_i)$.

Therefore, the preceding steps provide a general way of computing solution triplets $(\alpha_k, \theta_j, \varphi_i)$, for which we can evaluate the visibility of targets. In addition, our algorithm provides a mean to check inconsistencies in each step of the application of the properties (*e.g.* unsolvable constraints, or conflicts between two or more constraints) that occur whenever the domain of a variable is empty (either through the pruning or the intersection process). We thus benefit from an effective way of providing feedback to the user when no satisfactory viewpoint can be computed, and providing a means of backtracking to address solvable sub-sets of constraints.

In the last step, for all triplets $(\alpha_k, \theta_j, \varphi_i)$, we compute the corresponding 7DOF camera configuration. Then, the visibility of each subject is evaluated by using a ray casting technique over an object-oriented bounding box representing the subject (we cast rays to the 8 corners and the center of the bounding box following recommendations in [Ranon and Urli 2014]). We finally either accept or discard the camera viewpoint depending on its satisfaction, which is defined with an interval of accepted visibility ratios on each subject (*e.g.* visibility set to $[80\%, 100\%]$). Note that in this step, one might use any other computation method, leaving the choice to use more accurate queries (though being expensive as demonstrated in [Ranon and Urli 2014]).

4.2 Comparison with existing techniques

We compare our technique with a recent contribution by Ranon and Urli [2014] that relies on Particle Swarm Optimization to compute viewpoints. Their paper provides a thorough comparison of different solving techniques as well as efficient heuristics, and represents to date the most advanced and general viewpoint computation technique. Our respective approaches are compared against three criteria: total computation time, degree of satisfaction of the visual properties, and variability in the satisfaction. This was greatly facilitated by the authors of [Ranon and Urli 2014] who made available their problem descriptions, software framework, and results.

To compare our methods in a thorough way, we considered the exact same descriptions of the problems (our viewpoint computation algorithm was integrated in their software framework). We then used the same time windows for both methods. In our case, we had to run a pre-computation step to determine the appropriate number N of samples to use during our resolution that would best fit the allowed time window. We also used the same satisfaction functions to evaluate and select the best camera viewpoints (through a direct call to their library).

Table 1(a) shows the results on the 5 problems the authors defined with one and two targets (RoomsA, RoomsC, RoomsD, PapersOffice and CityC). The table shows that, for composition problems, our method obtains satisfactions similar to theirs, but offers a fair improvement over their technique in terms of computation time – the difference in performance ranging from 20% to 40%. Table 1(b) shows the results obtained on a more precise framing problem we defined: the two targets Giovanni and Matteo were framed

in smaller screen regions and with smaller sizes (representing less than 10% of the screen) than all the other problems defined by Ranon and Urli [2014]. This case shows that as more and more precise on-screen positioning is requested, their approach fails to precisely satisfy the constraints. Our method outperforms their technique both in terms of satisfaction and search time. It also shows that our technique is not only efficient but also fairly robust since its output is deterministic in obvious contrast with stochastic techniques. The standard deviation in the satisfaction of properties obtained by Ranon and Urli [2014] is much higher when targeting precise compositions. This is due to the ability of our technique to precisely focus the search in areas that satisfy the constraints, rather than exploring larger regions of the search space where properties may not be satisfied.

As a result, our computation in the Toric space yields realtime results and, equally important, camera compositions of consistent quality.

5 Intuitive Image-space manipulation

Despite advances, the task of interactively manipulating viewpoints still faces a central challenge: operating the appropriate balance between freedom in control and user constraints. To address this challenge, we propose novel and intuitive camera manipulation tools, and demonstrate their benefits with a user study.

5.1 Screen-space manipulators

Our method takes advantage of our Toric space representation. We provide four screen-space manipulators: (i) *Position* (ii) *Size*, (iii) *Vantage*, and (iv) *Vertigo* manipulators. Starting from an initial camera location $(\alpha_i, \theta_i, \varphi_i)$ around a pair of targets, the camera is interactively repositioned at a new position $(\alpha'_i, \theta'_i, \varphi'_i)$ to reflect the user's on-screen manipulations (see Figure 7).

Position manipulator: the user manipulates one target's on-screen position while the other target's position is maintained (see Figure 7(a)). To re-position the camera, we first determine the new manifold surface on which to position the camera, *i.e.* the appropriate new parameter α'_i, from the new pair of targets' screen positions; we do so by using the original formula proposed in [Lino and Christie 2012]. We then search for a new position (θ'_i, φ'_i) on this manifold surface that satisfies both (i) the new targets' on-screen positions and (ii) a user specified roll angle ψ to generate horizontal or canted shots (see Equation 1). The search is performed over both 2D manifold parameters, by minimizing a cost related to the on-screen positioning error made on targets, and is expressed as:

$$\min_{(\theta,\varphi)} \left(p_A - p'_A\right)^2 + \left(p_B - p'_B\right)^2$$

where p_A, p_B are the desired targets' on-screen positions and p'_A, p'_B the on-screen positions obtained from position (θ'_i, φ'_i) on the new manifold.

Size manipulator: the user manipulates one target's on-screen size, while the on-screen positions of both targets are maintained (see Figure 7(b)). To re-position the camera, we first extract the current target screen size (see Section 3.3). This extracted value is increased or decreased depending on the dragging direction, and converted to an appropriate distance d between the target and the camera (see Section 3.2). We then search for a new camera position (θ'_i, φ'_i) on the same manifold surface (*i.e.* $\alpha'_i = \alpha_i$, since the on-screen positions do not changed). We compute the value of θ'_i by using the following formula:

$$\theta'_i = \pi \pm 2 \operatorname{acos}\left(d \frac{\sin \alpha'_i}{\|AB\|}\right)$$

There may exist two solution values for θ'_i (*cf.* [Lino and Christie 2012]); in that case, we select the value closest to the initial one θ_i (*i.e.* before the manipulation). Similar to the position manipulator, we search on the variable φ over the current manifold surface, to reach a camera position (θ'_i, φ'_i) which enforces the targets' on-screen positions.

Vantage angle manipulator: the user manipulates the vantage angle on a target, while both targets' on-screen positions are maintained as much as possible (see Figure 7(c)). To re-position the camera, we first compute the initial vantage direction v from the target to the camera. This direction is decomposed into two components: a horizontal view angle and a vertical view angle. The user can then change this vantage direction through screen movements which update the horizontal view angle and the vertical view angle. From the user's input manipulation, we recompute a new vantage direction v' by using the formula $v' = q_V \cdot q_H \cdot v$, where q_H and q_V are the two rotations corresponding to the horizontal and the vertical changes in angle respectively. We then search the new camera position on the same manifold (*i.e.* $\alpha'_i = \alpha_i$). We compute the new camera position (θ'_i, φ'_i) as the intersection of the direction v' with the current manifold surface. Note that the orientation satisfying the desired targets' on-screen positions does not always strictly satisfy the user-defined roll angle ψ. By using our orientation computation method (see Equation 1), we thus minimize the error made *w.r.t.* the desired targets' on-screen positions while applying the desired camera roll.

Vertigo manipulator: the user manipulates the camera field of view, while the targets' on-screen positions are maintained (see Figure 7(d)). To reposition the camera, in a way similar to the Position manipulator, we first determine the new manifold surface on which to position the camera, from the pair of targets' screen positions (that do not change) and the new field of view. We then compute a new camera position (θ'_i, φ'_i) on this new manifold in such a way that we get the smallest change between the pairs (θ_i, φ_i) and (θ'_i, φ'_i). To so, we maintain the value onthe variable φ (*i.e.* $\varphi'_i = \varphi_i$) and, given that the value for variable θ is enclosed within the interval $]0; 2(\pi - \alpha)[$, we update the corresponding value using the formula

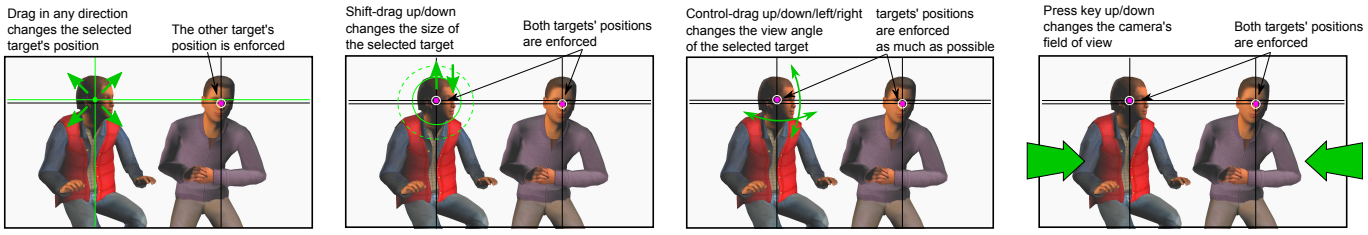
$$\theta'_i = \theta_i \left(\frac{\pi - \alpha'_i}{\pi - \alpha_i}\right)$$

As a result the camera maintains the on-screen positions of both targets while its field of view angle evolves. This classical camera motion is known as the *Vertigo* effect or *dolly-zoom*. While some approaches have proposed ways to compute such effects [Hawkins and Grimm 2007], the technique proposed here is straightforward with the Toric space representation.

Note that similar manipulators can also be derived from our Toric space representation to handle single-target cases, by carefully selecting two points on the target.

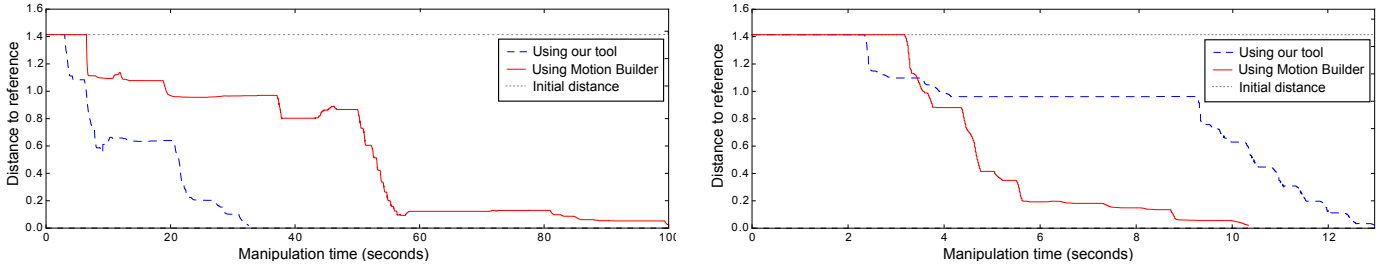
5.2 User evaluation

To evaluate the interest of our screen-space manipulators, we performed a subjective experimentation on the ease of use and the accuracy of our tool for reproducing viewpoints in a 3D context (which is an extremely classical task in 3D modelers), compared to the ease and accuracy when using a professional 3D modeler. We used MotionBuilder for comparison. Our target viewpoints involved composing two to five targets on the screen. After a training session (10 to 15 minutes distributed on both tools), subjects were asked to reproduce three reference viewpoints – (i) through the classical 3D interaction offered by Motion Builder and (ii) through



(a) Position manipulator (b) Size manipulator (c) Vantage manipulator (d) Vertigo manipulator

Figure 7: *Our screen-space manipulators. (a) the Position manipulator enables repositioning one target on the screen while the other target's on-screen position is maintained; (b) the Size manipulator enables resizing one target while both targets' on-screen positions are maintained; (c) the Vantage manipulator enables changing the view angle around one target, while targets' on-screen positions are maintained as much as possible; (d) the Vertigo manipulator enables changing the camera's field of view while both targets' on-screen positions are exactly maintained.*



(a) Novice user of 3D modelers (b) Expert user of 3D modelers

Figure 8: *Evolution of the distance from the user's manipulated viewpoint to a reference viewpoint, for a novice user and an expert user of MotionBuilder (both displayed in red). The distance obtained using our tool is displayed in blue. The manipulation time is in seconds.*

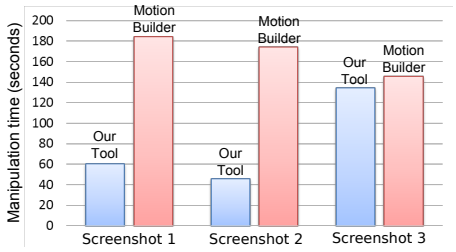


Figure 9: *Mean manipulation time (in seconds) required by the participants to reproduce a viewpoint using our manipulators (blue) compared to using the classical interaction of MotionBuilder (red).*

our screen-space manipulators. Each reference viewpoint was provided as a screenshot showing the desired camera view. The order of viewpoints and tools was randomly chosen for each participant. When the current user-manipulated viewpoint was sufficiently close to the reference viewpoint or when the user was satisfied enough, the manipulation was stopped and the history of the virtual camera was saved. 18 participants, with different experiences on 3D modeling tools, volunteered for this experiment. The distance of a user-manipulated viewpoint C to the reference viewpoint C_r was computed as a combination of the Euclidean distance on camera positions and a distance metric on camera rotations (represented as quaternions), using the formula

$$Dist\,(C) = \sqrt{\left(\frac{\|C_r - C\|}{\|C_r - C_i\|}\right)^2 + \left(\frac{|\arccos\left(2.\langle q_r, q\rangle^2 - 1\right)|}{\pi}\right)^2}$$

where $\langle q, q' \rangle$ is the dot product of q and q'. Here C_i denotes the initial camera position (*i.e.* at the beginning of the experiment); q and q_r denote the camera orientations of respectively C and C_r.

Figure 8 shows how the distance changes over time on the first viewpoint manipulation task (out of three). We report changes in distance for both a novice and an expert user of MotionBuilder. Our evaluations have shown that novices completed the tasks faster by using our manipulators rather than MotionBuilder camera manipulators (see Figure 9). They found our manipulators rather simple to understand and easy to use. In the case of the expert user, he required a bit more time during the first manipulation task with our manipulators (13 seconds with our tool *vs.* 11 seconds with MotionBuilder). However, on average he required approximately the same amount of time to complete all the tasks.

The comparison with more evolved screen-space manipulation techniques remains difficult. Typically, the interactions we propose such as pivoting around character or re-sizing cannot be easily expressed with through-the-lens [Gleicher and Witkin 1992] or visual-servoing techniques [Courty et al. 2003]. And power graphical widgets like IBar [Singh et al. 2004] and Cubecam [Sudarsanam et al. 2009] are rather complex to apprehend for beginners.

6 Effective Viewpoint Interpolation

A key challenge of viewpoint interpolation techniques is the effective control of the visual properties to be satisfied along the path. We believe that an effective viewpoint interpolation technique should (i) minimize the changes occurring in the image space and (ii) allow an easy control of both the camera motions and the framing along time. Taking advantage of our Toric space representation, we propose an effective and intuitive method to interpolate between key viewpoints, which takes inspiration from the classical

key-framing process.

In the following we show how we interpolate between two viewpoints: a key viewpoint k^0 (with position p^0), framing a pair of targets (A, B) at time t^0, and a key viewpoint k^1 (with position p^1), framing a pair of targets (A', B') at time t^1 – pairs can be the same, share one target or be completely different. Our process can be repeated over successive key viewpoints to build a more complex camera path. Figure 11 illustrates the overall interpolation pipeline.

Our method is founded on the idea that, to perform an effective interpolation between two viewpoints, we need to (i) compute a first trajectory which maximizes the visual properties over targets (A, B) between p^0 and p^1, (ii) compute a second trajectory which maximizes the visual properties over targets (A', B') between p^0 and p^1, and (iii) offer an effective way of controlling when and how to perform an interpolation between these two trajectories, by separating the process of interpolating camera positions and camera orientations. Intuitively, we want to control, in position and in orientation, how long we maintain the framing on the first pair of targets, how long we maintain the framing on the second pair of targets and how we interpolate in-between.

The first trajectory τ that links key-positions p^0 and p^1 is constructed by interpolating the respective visual properties related to the pair of targets (A, B) between times t^0 and t^1 (*i.e.* how these properties should evolve between both camera positions). To do this, we express key-positions p^0 and p^1 as two Toric triplets $(\alpha^0, \theta^0, \varphi^0)$ and $(\alpha^1, \theta^1, \varphi^1)$ defined around the pair of targets (A, B). For each target $i \in \{A, B\}$ and each position $j \in \{0, 1\}$, we extract a visual feature vector of the form (α^j, v_i^j, d_i^j); v_i^j is the unit vantage vector between the target i and camera position j, and d_i^j is the distance between the target i and camera position j. For a given ratio $x \in [0; 1]$, we interpolate each feature separately and linearly with x. We then define a function $F(x)$ providing an interpolated camera position around the pair of targets. Practically, F is computed as follows. For each target i, we compute the intersection point $T_i^x(\alpha^x, \theta_i^x, \varphi_i^x)$ of its interpolated vantage vector v_i^x with the manifold surface generated by the interpolated angle α^x; we also compute the distance $d_i^{\alpha,x}$ between this intersection point and the target i. We then define F as a trade-off on camera positions which reflects the interpolated visual features on both targets. Practically, the interpolated position on the path is computed as

$$F_{(A,B)}(x) = \frac{1}{2}\left[A + B + \sum_{i \in \{A,B\}} v_i^x \cdot \frac{d_i^x + d_i^{\alpha,x}\lambda_i^x}{1 + \lambda_i^x}\right]$$

where λ_i^x is a scaling factor that avoids giving too much importance to $d_i^{\alpha,x}$ when the enforcement of the screen positioning is not possible (typically when the camera crosses the line-of-interest (AB)); this factor is computed as a sinusoid taking as input the angle between the vantage vector v_i^x and the line (AB) separating the two targets. Figure 10 illustrates this first interpolation process.

The second trajectory τ' is computed in a similar way, linking key-positions p^0 and p^1 by interpolating the respective visual properties related to targets (A', B') between times t^0 and t^1.

Controlling the camera motion along time. We now have two camera paths τ and τ' between the two same camera positions. Each path minimizes the change over visual properties of a given pair of targets. We then combine these two paths along time, through a non-linear interpolation function $g_p(t)$ (with $t \in [t^0; t^1]$) defined over time and returning a value $x \in [0; 1]$ – g_p is defined so that $g_p(t^0) = 0$ and $g_p(t^1) = 1$. The final interpolated camera position $p(t)$ at time t is given by the formula

$$p(t) = F_{(A,B)}\left(g_p(t)\right).(1 - g_p(t)) + F_{(A',B')}\left(g_p(t)\right).g_p(t)$$

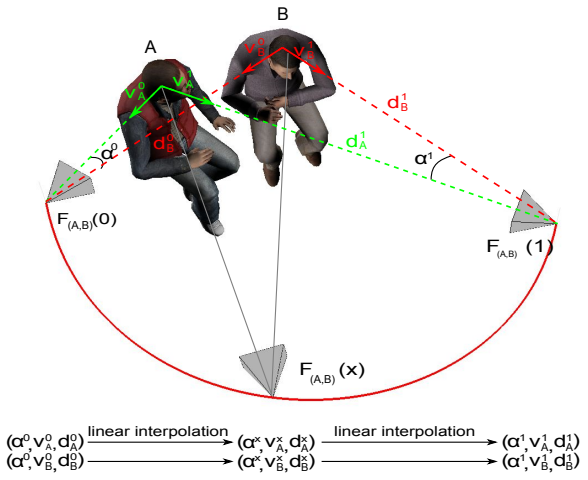


Figure 10: *Composition-based interpolation of the camera position around a pair of targets (A, B). For two key camera positions and a key framing to enforce on a pair of targets, we algebraically interpolate the camera position as a path which provides linear changes over their on-screen appearance. The path is defined through a function $F_{(A,B)}(x)$ such that any intermediate position* (i.e. *for $x \in]0; 1[$) is computed by relying on a linear interpolation of all visual properties of the pair of targets.*

This function aims at controlling when and how the camera moves between both viewpoints (see Figure 11(c)).

Controlling the framing along time. Unlike classical interpolation methods, we interpolate camera orientations along time not in terms of low-level rotations but in terms of how to frame targets. For a given camera position computed by the previous step, we compute the two camera orientations $q^t_{(A,B)}$ and $q^t_{(A',B')}$ which respectively enforce the framing of the first and second pair of targets at time t; these orientations are computed by using the formula given in Equation 1. In a way similar to the position, we combine these two orientations along time through a non-linear interpolation function $g_f(t)$ (defined similarly to $g_p(t)$). The final interpolated camera orientation $q(t)$ at time t is given by the formula

$$q(t) = q_{(A,B)}.(1 - g_f(t)) + q_{(A',B')}.g_f(t)$$

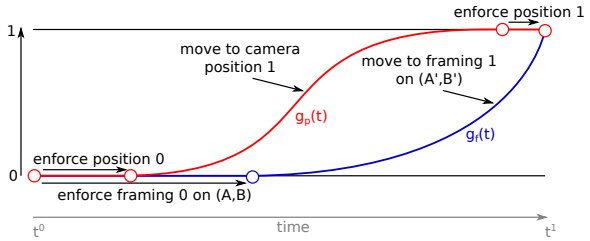
This function aims at controlling when and how the camera moves between both framings (see Figure 11(c))

Tuning the camera viewpoint along time. In our model, both functions g_p and g_f are parametrized by (i) the pair of key-viewpoints k^0 and k^1, (ii) the number of static frames following t^0, during which the properties of viewpoint k^0 (either in position or in framing) are enforced, (iii) the number of static frames preceding t^1, during which the properties of viewpoint k^1 are enforced. These functions also rely on default ease-in and ease-out functions, that allow to specify how much the camera will accelerate or decelerate between viewpoints k^0 and k^1. This model allows an intuitive and efficient control of the timing of the interpolation and the speed of the camera in terms of both position and framing. Figure 11 illustrates how, using our interpolation method, both the camera motion and framing can be parametrized and tuned through very few controllers.

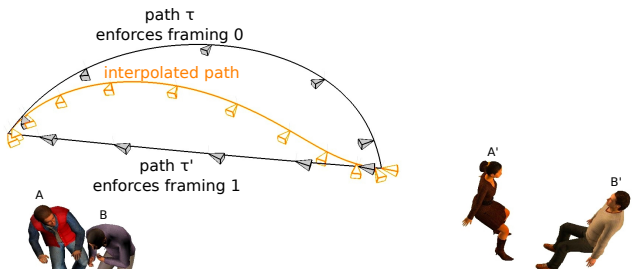
To illustrate the power of our interpolation method, we extracted 7 key viewpoints from the shots of a scene of the original movie *Back to the future (R. Zemeckis, 1985)*. We reproduced these shotd

(a) Framing 0 (t^0=0s)

(b) Framing 1 (t^1=18s)



(c) Controls over the camera position and framing



(d) Interpolation in position and framing

Figure 11: *Overview of the interpolation pipeline, between two key viewpoints. (a)(b) The user drafts two viewpoints at times t^0 and t^1. (c) (S)he controls interpolation curves over the camera motion and re-framing along time; (s)he is required to handle few controllers, encompassing the duration of enforcement, as well as ease-in/ease-out values controlling the speed of the camera. (d) For each key framing, we compute a camera path (τ and τ' respectively) that smoothly moves the camera between key positions while enforcing this framing. We finally interpolate both paths (in terms of the camera position and orientation) by relying on the interpolation curves.*

by using our interactive viewpoint manipulation tool. We then constructed a long take (or plan-sequence) of that scene, reproducing the viewpoints of the original shots, while enforcing smooth camera movements between viewpoints (see accompanying videos). The generated camera path is also displayed in Figure 1(d). A second and more complex motion was generated with 15 key viewpoints and without any other manipulation that controlling the time spent in each viewpoint. Given the low number of inputs, the generated result is a considerable improvement on standard interpolation techniques (see accompanying video).

In terms of computational cost, the technique spends an average value of 2ms per second of a movie at 30fps (91ms for a 7 key viewpoints sequence of 45 seconds, and 160ms for a 15 key viewpoints sequence of 80 seconds). This is due to the fact that the camera trajectories are all constructed using algebraic computations.

7 Discussion

An aspect that is only partially addressed in this contribution is the visibility of targets. While interpolating camera paths or during interactive manipulations, we do not check for visibility. In the tasks we address, this has not been pointed out as an issue by our users. Though, visibility could be handled directly in the Toric space by computing and intersecting a visibility map (computed with ray cast or hardware rendering techniques) with the set of Toric manifolds; expressing such a constraint in our Toric space in an efficient way is a challenging problem and is our next objective.

While expressive, our viewpoint manipulation technique is limited to handling two targets at a time. One can however interactively change the targets to easily create layouts involving three or more targets. In such case, the user selects two targets on which to perform manipulations, then changes one or two of the targets and continues the interaction in a seamless way. Furthermore, the user is not restricted to interacting with targets that fully appear on the screen; by using an extended frame that displays off-screen or partially off-screen targets, one can easily create viewpoints where targets are vertically cut by the frame (*e.g.* in over-the-shoulder shots).

In a similar way, our viewpoint computation method is limited to the case of two targets. In case of a single target, it simply reduces to choosing a pair of points on the target. Our viewpoint computation tool could also handle over-constrained cases considering three or more targets. In the case of three or more targets, our methods could also be adapted. By expressing such a problem as a set of two-target problems, one could derive a method that computes viewpoints which solves a first two-target sub-problem, then incrementally checks other constraints satisfaction. A more elegant solution would however require new research.

Finally, our viewpoint interpolation model enables the creation of camera motions that target minimizing changes on the on-screen layout (*e.g.* dolly-in, dolly-out, arcing around targets, or following targets). It enables the construction of complex cinematographic motions such as long-takes or trackings of characters which require many efforts when manually crafted. On the other hand, for some other, often linear-shaped, camera motions such as travellings, new results are required to express such paths in the Toric Space.

8 Conclusion

In this paper, we have introduced a novel camera viewpoint representation. The central benefit of our Toric space representation is its compact nature: any viewpoint computation problem that involves at least two targets can be expressed with a triplet of variables $(\alpha, \theta, \varphi)$. It casts simple camera optimization problems mostly conducted in a 7DOF space into a search inside a 4DOF space. Our Toric space representation thus provides as really powerful means on which to build high-level and efficient camera control techniques.

We have provided the algebraic expression, in the Toric space, of most classical visual properties employed in the literature. Coupled with an interval-based resolution algorithm, we have then proposed a very efficient viewpoint computation technique, that performs better than the existing state of the art techniques both in terms of computation time and consistent production of desirable viewpoints.

Building upon our Toric space representation, we have proposed intuitive viewpoint manipulators controlling the visual result directly in screen space. The combination of these manipulators shows strong benefits in the task of crafting viewpoints and is easier to use for beginners. We consequently believe this screen-space manipulation tool has the potential to be directly integrated in commercial 3D modelers.

Finally, we provided an effective camera motion planning algorithm which allows an intuitive and efficient interpolation between camera viewpoints, through very few operations. The benefit of our technique is that it enables an easy control of both the camera

motion and the camera framing along time, and that the generated paths preserve visual properties along time.

9 Acknowledgments

We would like to thank Emmanuel Badier who co-developed our MotionBuilder® plugin. We also thank Hui-Yin Wu, Billal Merabti, Quentin Galvane and Cunka Sanokho, who provided valuable help in running user studies and producing the technical video. This work was partly supported by the French National Research Agency (ANR), under grant agreement ANR-12-JS02-0008 and a *Carnot* funding granted to INRIA.

References

ASSA, J., COHEN-OR, D., YEH, I.-C., AND LEE, T.-Y. 2008. Motion Overview of Human Actions. *ACM Trans. Graph. 27*, 5.

BARES, W., MCDERMOTT, S., BOUDREAUX, C., AND THAINIMIT, S. 2000. Virtual 3D Camera Composition from Frame Constraints. In *ACM International Conference on Multimedia*.

BLINN, J. 1988. Where am I? What am I looking at? *IEEE Computer Graphics and Applications 8*, 4.

BOUBEKEUR, T. 2014. ShellCam: Interactive geometry-aware virtual camera control. In *IEEE International Conference on Image Processing*.

CHRISTIE, M., AND LANGUÉNOU, E. 2003. A Constraint-Based Approach to Camera Path Planning. In *Smart Graphics*, vol. 2733 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

CHRISTIE, M., AND NORMAND, J.-M. 2005. A Semantic Space Partitioning Approach to Virtual Camera Composition. *Computer Graphics Forum 24*, 3.

CHRISTIE, M., OLIVER, P., AND NORMAND, J.-M. 2008. Camera Control in Computer Graphics. *Computer Graphics Forum 27*, 8.

COURTY, N., LAMARCHE, F., DONIKIAN, S., AND MARCHAND, E. 2003. A Cinematography System for Virtual Storytelling. In *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling*, vol. 2897 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

DRUCKER, S. M., AND ZELTZER, D. 1994. Intelligent Camera Control in a Virtual Environment. In *Graphics Interface*.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM 24*, 6.

GLEICHER, M., AND WITKIN, A. 1992. Through-the-lens Camera Control. *Computer Graphics 26*, 2.

HAWKINS, A., AND GRIMM, C. 2007. Keyframing using Linear Interpolation of Matrices. *Journal of Graphics Tools 12*, 13.

KAVRAKI, LE LYDIA AND SVESTKA, PETR AND LATOMBE, JEAN-CLAUDE C AND OVERMARS, MARK H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4.

KHAN, A., KOMALO, B., STAM, J., FITZMAURICE, G., AND KURTENBACH, G. 2005. HoverCam: Interactive 3D Navigation for Proximal Object Inspection. In *ACM Symposium on Interactive 3D Graphics and Games*.

LI, T.-Y., AND CHENG, C.-C. 2008. Real-Time Camera Planning for Navigation in Virtual Environments. In *Smart Graphics*, vol. 5166 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

LINO, C., AND CHRISTIE, M. 2012. Efficient Camera Composition for Virtual Camera Control. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LINO, C., CHRISTIE, M., LAMARCHE, F., SCHOFIELD, G., AND OLIVIER, P. 2010. A Real-time Cinematography System for Interactive 3D Environments. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LINO, C. 2013. *Virtual Camera Control using Dynamic Spatial Partitions*. PhD thesis, University of Rennes 1, France.

MARTON, F., RODRIGUEZ, M. B., BETTIO, F., AGUS, M., VILLANUEVA, A. J., AND GOBBETTI, E. 2014. IsoCam: Interactive Visual Exploration of Massive Cultural Heritage Models on Large Projection Setups. *Journal of Computing and Cultural Heritage 7*, 2.

NIEUWENHUISEN, D., AND OVERMARS, M. H. 2004. Motion Planning for Camera Movements. In *IEEE International Conference on Robotics and Automation*.

OLIVIER, P., HALPER, N., PICKERING, J. H., AND LUNA, P. 1999. Visual Composition as Optimisation. In *Artificial Intelligence and Simulation of Behaviour*.

OSKAM, T., SUMNER, R. W., THUEREY, N., AND GROSS, M. 2009. Visibility Transition Planning for Dynamic Camera Control. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

RANON, R., AND URLI, T. 2014. Improving the Efficiency of Viewpoint Composition. *IEEE Transactions on Visualization and Computer Graphics 20*, 5.

RANON, R., CHRISTIE, M., AND URLI, T. 2010. Accurately Measuring the Satisfaction of Visual Properties in Virtual Camera Control. In *Smart Graphics*, vol. 6133 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

SHOEMAKE, K. 1992. ARCBALL: A User Interface for Specifying Three-dimensional Orientation Using a Mouse. In *Graphics Interface*.

SINGH, K., GRIMM, C., AND SUDARSANAM, N. 2004. The IBar: A Perspective-based Camera Widget. In *ACM Symposium on User Interface Software and Technology*.

SUDARSANAM, N., GRIMM, C., AND SINGH, K. 2009. CubeCam: A Screen-space Camera Manipulation Tool. In *Computational Aesthetics in Graphics, Visualization, and Imaging*.

VÁZQUEZ, P.-P., FEIXAS, M., SBERT, M., AND HEIDRICH, W. 2001. Viewpoint Selection Using Viewpoint Entropy. In *Vision Modeling and Visualization Conference*.

YEH, I.-C., LIN, C.-H., CHIEN, H.-J., AND LEE, T.-Y. 2011. Efficient Camera Path Planning Algorithm for Human Motion Overview. *Computer Animation and Virtual Worlds 22*, 2-3.

YEH, I.-C., LIN, W.-C., LEE, T.-Y., HAN, H.-J., LEE, J., AND KIM, M. 2012. Social-event-driven camera control for multicharacter animations. *IEEE Transactions on Visualization and Computer Graphics 18*, 9.