DOI: 10.1111/cgf.13949
EUROGRAPHICS 2020 / U. Assarsson and D. Panozzo
(Guest Editors)

Volume 39 (2020), Number 2

Real-time Anticipation of Occlusions for Automated Camera Control in Toric Space

L. Burg[1], C. Lino[2], M. Christie[1,3†]

[1]Univ Rennes, Inria, CNRS, IRISA, M2S [2]LTCI / Télécom Paris, Institut Polytechnique de Paris, France [3]Beijing Film Academy, China

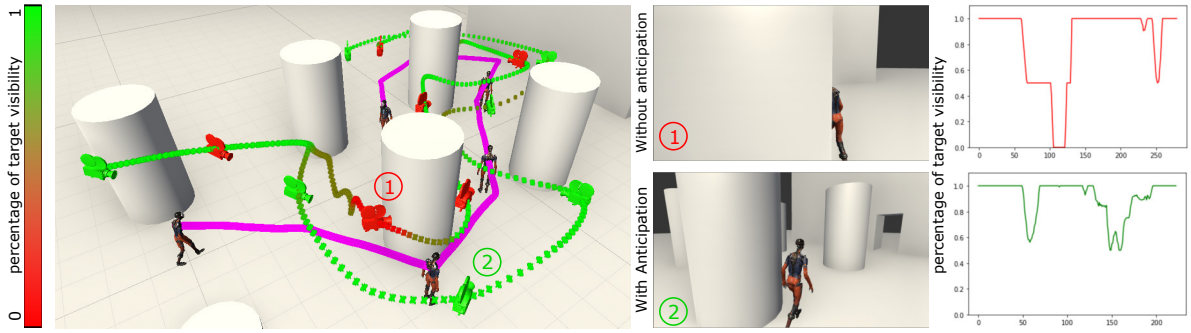


Figure 1: *Our real-time camera control system automatically anticipates and avoids static and dynamic occluders. Starting from the same frame, our system without anticipation activated (red camera) is unable to anticipate, hence remains occluded until the character moves away from the column (middle image 1); using our anticipation technique (green camera), our system enables the camera to predict the occlusion, and move to an unoccluded view of the character (middle image 2).*

Abstract

Efficient visibility computation is a prominent requirement when designing automated camera control techniques for dynamic 3D environments; computer games, interactive storytelling or 3D media applications all need to track 3D entities while ensuring their visibility and delivering a smooth cinematic experience. Addressing this problem requires to sample a large set of potential camera positions and estimate visibility for each of them, which in practice is intractable despite the efficiency of ray-casting techniques on recent platforms. In this work, we introduce a novel GPU-rendering technique to efficiently compute occlusions of tracked targets in Toric Space coordinates – a parametric space designed for cinematic camera control. We then rely on this occlusion evaluation to derive an anticipation map predicting occlusions for a continuous set of cameras over a user-defined time window. We finally design a camera motion strategy exploiting this anticipation map to minimize the occlusions of tracked entities over time. The key features of our approach are demonstrated through comparison with traditionally used ray-casting on benchmark scenes, and through an integration in multiple game-like 3D scenes with heavy, sparse and dense occluders.

CCS Concepts

•***Computing methodologies*** → ***Rasterization;*** *Procedural animation;* •***Applied computing*** → *Media arts;*

1. Introduction

Controlling cameras is a core aspect of nearly all interactive computer graphics applications (*e.g.* computer-aided design, cultural heritage exploration or entertainment). Indeed, one key component is to make the user able to automatically track target objects in the scene (most often one or two objects of interest) through the lens of a virtual camera. Hence, controlled cameras should at any time provide interesting viewpoints, with as much visibility as possible on these target objects, while providing smooth camera motions.

In most interactive 3D applications, cameras are traditionally controlled through low level interaction metaphors (*e.g.* trackball) which offer a limited degree of control and still remain un-intuitive for non-expert users. Furthermore, no convincing solutions are of-

† marc.christie@irisa.fr

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

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

fered for camera collisions or occlusion avoidance. Hence, there is a clear requirement to provide a higher level of automation in the control of virtual cameras. On the one hand, in the last two decades, researchers have proposed solutions to account for the geometry of 3D scenes and/or to incorporate more cinematographic knowledge in camera systems. However, few works have focused on the issue of real-time occlusion avoidance and occlusion anticipation when dealing with fully dynamic scenes, and most of which require heavy pre-computations. In the other hand, recent efforts have been made in the industry to propose more evolved, yet lightweight, tools; for instance, Unity's *Cinemachine* tool provides ways to manually craft camera rails along which the camera will be moved at run-time to follow objects. Yet, it is still required that users manually handle cinematic camera placement, avoid collisions, and setup triggers for camera-switching operations (*i.e.* cuts) as soon as the current camera lacks visibility on filmed objects. Providing a lightweight yet evolved camera system which maximizes visibility of targets requires to address three main challenges: first efficiently estimating the visibility of moving targets remains (a complex and computationally expensive process in a real-time context), second anticipating changes in the environment to reduce the occlusion of targets by the camera, finally maintaining the quality of camera motions despite necessary displacements to anticipate occlusion and improve visibility.

In this paper we address these challenges by proposing a real-time cinematic camera control system for fully dynamic 3D scenes, which maximises visibility of one or two targets with no pre-computation. To do so, we leverage previous contributions of [LC12, LC15] on the Toric Space, which offer an efficient way to address cinematic camera placement, yet without addressing the issues of visibility computation/maximization nor quality of camera motions. Our contributions are:

- an efficient shadow-mapping algorithm to compute visibility information into a Toric space coordinate system;
- a customizable anisotropic blurring algorithm to generate, in Toric space, an *occlusion anticipation map*;
- a texture-based style model to encode viewpoint preferences (*i.e.* directorial styles in viewpoints can be expressed as *textures*);
- a physically-plausible camera control model enabling, from these information, to create smooth camera motions in real-time. We supplement this model with a set of strategies (*e.g.* introduce cuts) to handle symptomatic cases (*e.g.* when no un-occluded viewpoint can be found locally).

2. Related Work

Positioning cameras in 3D scenes is highly connected to the notion of viewpoint entropy [VFSL02], *i.e.* measuring the quality of a viewpoint with regard to the amount of information it conveys for a given scene. When addressing the problem of moving cameras that should follow moving targets in a cinematographic way, three criteria are crucial: (i) maintain enough visibility on these targets, (ii) provide viewpoints in which the visual arrangement of targets follow common aesthetic rules used by cinematographers [RT09b] and (iii) provide smooth camera motions.

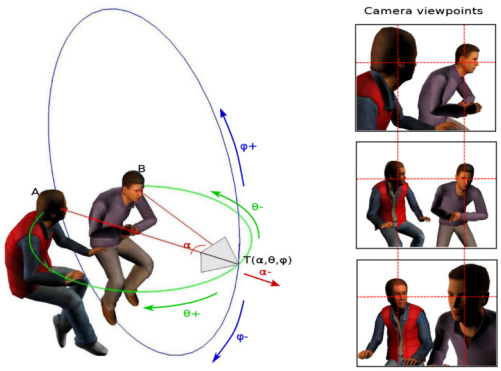


Figure 2: *Toric space representation proposed by [LC12, LC15] to algebraically solve combinations of on-screen constraints. A viewpoint is parameterized with a triplet of Euler angles (α, θ, φ) defined around a pair of targets (A and B); α (red) defines the angle between the camera and both targets – it generates a spindle torus on which to position the camera –, θ (green) defines the horizontal angle and φ (blue) the vertical angle around the targets.*

2.1. Virtual Camera Control

General problem Most approaches have tackled this problem as a search or a constraint-solving problem in a 7D camera space (*i.e.* optimizing a 3D position, 3D orientation, and field of view for each camera, and each time step). One key issue is the strongly non-linear relation between the low-level camera parameters and the visual arrangement constraints (*e.g.* targets on-screen positions, size, or viewing angles) to satisfy. Hence, for efficiency, it is often required to reduce the dimension of the problem, by either optimizing the camera position and orientation in two separate steps or by constraining the possible camera positions. For instance, Burtnyk [BKF*02] proposed an interactive control of cameras through 2D inputs, where the authors constrain the cameras to move on hand-crafted 3D surfaces aimed at providing interesting views around targets. Recently, Lino and Christie [LC12, LC15] proposed the Toric space, a 3D compact space in which some visual constraints can be directly encoded (in terms of both camera position and orientation) or solved algebraically. This space is defined as a continuous set of spindle torii, around two targets. A 7D camera configuration is fully determined by 3 parameters: an angle (α) algebraically computed from the desired targets on-screen positions – any camera position (and its associated orientation) on this α-Toric surface satisfies this key constraint –, as well as a horizontal and a vertical angle (θ and φ) on the torus (see figure 2). Furthermore, visual constraints can be robustly and interactively enforced or manipulated, while this is a difficult problem in the 7D camera space. Yet, visibility information remains not straightforward to compute in Toric space, which is a key motivation of this work.

Physically-based motions Contributions have also recently investigated camera physics in their cinematic control, both for virtual cameras (*a.k.a.* virtual cinematography) or for real cameras held onto unmanned aerial vehicles (*a.k.a.* aerial cinematography).

The primary focus in virtual cinematography is to output more plausible paths, *i.e.* visual constraints are of higher im-

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

portance compared to strictly obeying physics laws. Galvane *et al.* [GCLR15] and Litteneker *et al.* [LT17] proposed quite comparable path smoothing techniques. Both take as input the animation of one or two characters and a set of desired on-screen layouts along time, and output a smooth camera path. The satisfaction of on-screen constraints along this path is firstly maximized either using a costly optimization in 7D camera space [LT17] (with simulated annealing, plus a gradient descent) or algebraic solutions in Toric space [GCLR15]. Both would provide a noisy camera path. They hence secondly smooth the computed path: [GCLR15] proposed to approximate it as a cubic Bezier curve, then locally optimize the camera velocity and acceleration (in position and orientation) by using constrained optimization. In a way similar, [LT17] proposed to locally optimize the path curvature and stretch by using an active contour model. However, [GCLR15] is an offline method, and does not account for visibility or collisions. Litteneker's method is similar to ours in term of path smoothing. One major difference is their use of a sparse visibility/occlusion evaluation (through ray-casting from a grid of neighbor camera positions) together with an inaccurate visibility prediction (through an isotropic blur kernel). In contrast, we use a smooth visibility evaluation (shadow maps), together with an anisotropic occlusion prediction following a predicted evolution of the scene. In parallel, as in [GCLR15], we can enforce on-screen constraints by projecting our camera paths in Toric space.

In parallel, the aerial cinematography community has focused on computing feasible and collision-free paths (this is a high-order requirement), while maximizing visual constraints' satisfaction. Nageli *et al.* [NAMD*17] formulated this real-time camera control process as a receding-horizon optimal control in the low-level aerial vehicle parameters, while optimizing a set of visual constraints and collision constraints. They further extended their technique to following handcrafted paths and avoiding collision between multiple aerial vehicles [NMD*17]. Galvane *et al.* [GLC*18] have built upon an extended version of the Toric Space (a C^2 surface, named Drone Toric Space) which additionally enforces a collision-avoidance distance around targets. They then proposed to rely on graph-oriented motion planning algorithms (on camera positions) to dynamically re-compute feasible (C^4) paths, while optimizing visual constraints in the Drone Toric Space to follow hand-drawn paths or to coordinate multiple vehicles. Both Nageli *et al.* and Galvane *et al.* consider a rather static environment, with a small number of dynamic occluders (the targets) and colliders (targets and other vehicles) declared by the user and approximated as ellipsoids. In contrast, our method works for 3D scenes with unknown dynamic objects of any complexity.

2.2. Visibility for computer graphics

Visibility (and by extension shadow) computation in 3D scenes has received much attention in computer graphics, mainly for rendering purposes. An extensive overview of general visibility and shadow computation is beyond the scope of this paper, though a good insight in visibility problems classification can be found in [COCSD03], and overviews of modern shadow computation techniques can be found in [WP12, ESAW16]. Hence, we here only review techniques that area relevant to our domain of interest.

Ray-casting techniques heavily rely on direct ray-primitive intersections [Rot82] to check if some geometry exist between two 3D points. This is mostly used in path-tracing techniques, as it allows computing point-point visibility very efficiently. By casting many rays, one can obtain a rough approximation of the from-point visibility of a complex geometry. It however remains more expensive than rasterization, as it requires to cast many rays, and to dynamically maintain a bounding-volume hierarchy. One reason is that it still heavily relies on CPU computations (at least until hardware ray-tracing GPUs become the norm).

Rasterization-based techniques heavily rely on the hardware graphics pipeline, which offers a very powerful tool (called the depth map or z-buffer). It enables approximating *shadow maps* [Wil78] by computing a depth map from a light source, or even *soft shadows* [HH97] by using multiple shadow maps computed from sampled points on an area light source. [RGK*08] reversed this concept by computing many low-cost (128x128 pixel) shadow maps from sparse 3D scene points. They approximate the scene as a set of sample points, split them into subsets, and render each subset into a shadow map, with a point-based rendering method. As a result, they obtain inaccurate shadow maps. These maps provide a very poor approximation of direct visibility, but are good enough to efficiently approximate indirect visibility (illumination and shadows) from punctual or area light sources. Our technique is inspired by the shadow mapping concepts, which remain cheap (*i.e.* the z-buffer is encoded in the hardware of most GPUs) and provide a precise-enough approximation of the from-point visibility in complex and dynamic scenes. More generally, we design our computation pipeline to be massively parallel (*i.e.* shader-oriented).

2.3. Visibility for camera control

Visibility is also a central issue when controlling virtual cameras, though its has been paradoxically under-addressed in this domain. Different concerns must be reconciled: (i) the level of precision (from rough to precise approximation), (ii) whether dynamic occluders or complex geometries are accounted for, and (iii) whether the camera can anticipate on future occlusions. In most existing works, the visibility is computed by casting rays toward a rough geometric abstraction (often a bounding box or sphere) of targets, with no or few anticipation. We hereafter focus on techniques handling visibility for camera motions. For a more general overview of camera control problems, see [CON08a].

Global visibility planning To reduce the cost of computing visibility from a camera path, some approaches proposed to pre-process a visibility data structure. Oskam *et al.* [OSTG09] create a visibility-aware roadmap, where nodes represent a set of spheres sampled so that together they cover the whole scene, and an edge represents the connectivity (*i.e.* intersection) between two spheres. They pre-compute a huge table of approximated sphere-sphere visibility by launching rays between each pair of spheres. At run time, for a single target positioned in one sphere, they can plan camera paths in the roadmap. They account for the travelled distance and the target visibility along the path, by simply fetching the visibility table for any travelled sphere. They also propose a simple anticipation process. They find the sphere closest to the camera, while not

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

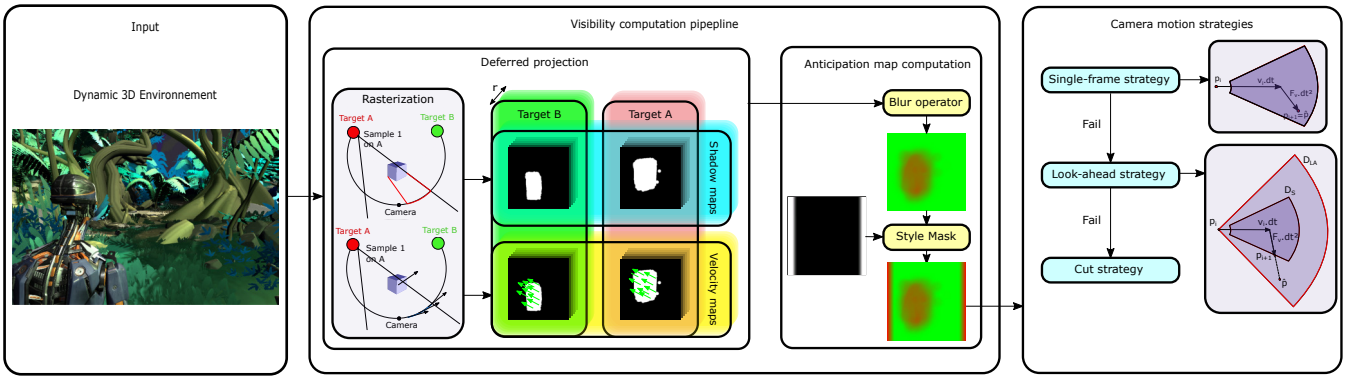


Figure 3: *Overview of the system: (left) a dynamic 3D scene serves as input, (middle) our visibility anticipation process consists in computing shadow maps together with occluder velocity maps for each target, and combining them to construct an occlusion anticipation map (right) we use this anticipation map together with a physical camera model to both maximize visibility and camera motion smoothness.*

visible from it, and compute its occlusion risk. They finally slightly move the camera around the target to maximize its visibility. Lino *et al.* [LCL*10] rely on a real-time global visibility computation. They first pre-compute a 2.5D cell-and-portal [TS91] of an indoor scene. They can, in real-time, re-compute the visibility of moving targets, into 2.5D volumes (*visibility volumes*) storing their visibility degree (from fully visible to fully occluded). They further combine this visibility information with volumes storing the viewpoint semantics (*i.e.* how filmed targets can be arranged on the screen), to form *director volumes*. They finally connect director volumes into a roadmap. This enables to dynamically re-plan camera paths accounting for the travelled distance, visibility and intermediate viewpoints semantics along the path. Their approach can also make cuts (*i.e.* teleport the camera), by filtering director volumes according to film editing conventions. Global approaches are quite efficient, but require heavy pre-computation on the scene geometry and are not adapted to compute visibility or plan paths in fully dynamic scenes or with complex geometries (*e.g.* trees or furniture).

Local visibility planning To plan paths in scenes with dynamic occluders, researchers have also focused on reactive planning techniques. Inspired by occlusion culling methods used in rendering, Halper *et al.* [HHS01] proposed an occlusion avoidance relying on *Potential Visibility Regions* (PVR). They firstly define preferred viewpoints as a set of bounding spheres. Each sphere is shaded regarding to its preference level (the higher, the brighter the color). They secondly use a depth buffer from the target position. They render all volumes from most to least desirable, while considering the depth of occluders. As output, they generate an image buffer whereby the brightest color (*i.e.* most desirable camera position) that first pass the depth test is visible. To anticipate occlusions, they also consider the past occluders trajectories and accelerations, and solve the camera for that predicted state. In turn, the camera path is adapted so that the camera can be at this predicted position at the prediction time. Christie *et al.* [CNO12] extend this concept to two targets. They first compute low-resolution depth buffers from a large sample of points on each target, in the direction of the camera. They combine these depth buffers to create visibility volumes around the camera. They also extend it to three or more targets [CON08b] by combining pair-wise computations, and aggregate visibility in a temporal window to avoid over-reactive camera behaviors. Our technique relies on similar concepts, while we project results in a configuration space (the Toric space) allowing to also solve for the viewpoint semantics. Local approaches can be efficient too. They are however best suited for planning paths step-by-step. Hence, if no solution exists in the rendering, they will fail to find an unoccluded position (even if one exists). In turn they cannot plan a global path to this position. In our approach, we overcome this problem by considering more global visibility information, to make cuts, as an alternative when no local solution is found.

Visibility in Toric Space remains an under-addressed problem. [LC15] proposed to compute a set of camera positions in Toric space, satisfying other visual constraints. They convert them back to Cartesian space, evaluate targets visibility with ray-casting, and discard those with low visibility. The main drawbacks is that their visibility computation remains imprecise and costly, while they target placing cameras in static scenes only. To our knowledge, there has been no other attempt to compute visibility, and none to anticipate occlusions in Toric space.

3. Overview

The core of our method consists in solving an optimization problem in a 6D camera space (position and orientation) in real-time that accounts for 3 main criteria: (i) computing a sequence of viewpoints that satisfy a user-specified visual arrangement of targets (*e.g.* desired on-screen positions), while (ii) maintaining as much visibility on the filmed targets as possible, and (iii) ensuring that camera motions remain as smooth as possible. First, by using the Toric space coordinate system [LC15], given two desired on-screen positions for two distinct targets, the camera orientation is fully determined by a given position in the Toric space. We leverage this property to re-write the overall problem as a constrained-optimization on the camera 3D position only (rather than a 6D).

The Toric space being defined by two targets (abstracted as two 3D points), we propose a tracking with only one target but rely on the classical gaze-room filming convention [RT09b] by considering another 3D point placed in the forward direction of the target that represents the target's anticipated 3D location. By composing on

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

Table 1: *Notations*

$(\mathbf{u},\mathbf{v})$	angle between two vectors
$\mathbf{u}\cdot\mathbf{v}$	dot product
o	target object
s	sample point index
r	number of sampled points on target object o (*i.e.* $s \in [1\,..\,r]$)
i	camera motion iteration
(x,y)	pixel/texture coordinates
$T(x,y)$	value in a 2D buffer/map T
S_s^o	map computed for sample point s on object o

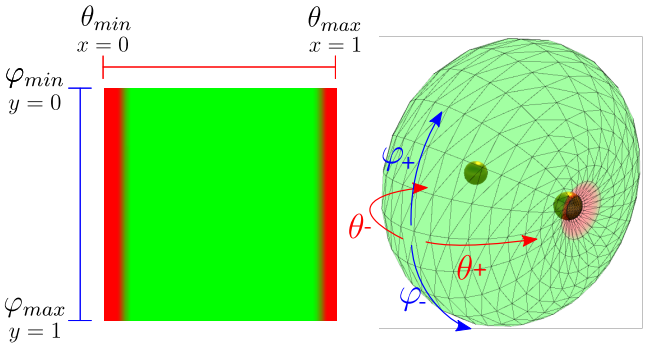


Figure 4: (x,y) *texture mapping expressed in the Toric space* (θ,φ).

the screen, the target together with its anticipated position, we provide enough visual space ahead of the character, hence satisfying the gaze-room convention. The desired on-screen positions of these two targets is enforced by maintaining the camera on a given Toric surface within the Toric space [LC12]. This addresses criterion (i).

To address criterion (ii) (visibility), a key requirement is the ability to efficiently evaluate the visibility of our two targets for a large number of camera configurations. To further maximize this visibility over time, we need to anticipate the motions of the camera, targets and occluders, *i.e.* computing the visibility ahead of time.

To do so, we propose a 2-stage computation pipeline (see figure 3). In the first stage (deferred projection), we draw inspiration from the shadow mapping technique to compute the visibility of both target objects. We render occluders from the viewpoint of each target and project this information onto the current Toric surface (see figure 2). We therefore obtain a 2D scalar field on this surface which we refer to as a *target shadow map* (or *S-map*). The field encodes the composed visibility of target objects at the current frame. Then, in addition, we compute the velocity of our occluders and generate a corresponding 2D vector field in the Toric space which we refer to as the *occluder velocity map* (or *V-map*). In the second stage, we combine the *S-map* and the *V-map* to compute a new 2D scalar field corresponding to an anisotropic blur of occlusions along the occluders' velocity directions given by *V-map*. In a word, we compute the predicted visibility of the occluders knowing their velocity into a scalar field to which we refer as *anticipation map* (*occlusion anticipation map* or *A-map*). This encodes the probability of future occlusion of the targets, expressed in Toric space, within a given time window.

To address criterion (iii) (smooth camera motions) we propose the design of a physically-driven camera motion controller in which our camera is guided by a number of external forces. We start by defining a search area which approximates the locus of future camera positions, given a time window and our physical camera model. We then sample points in the search area and project them on the Toric surface. For each projected point, we extract the anticipation information stored in *occlusion anticipation map*, and then decide where to move the camera. To handle specific cases, different camera motion strategies have been defined (see Section 5. This addresses the combination of criteria (i) and (iii).

4. Visibility computation pipeline

We detail the two stages of our visibility computation: (i) computing pairs of *target shadow map* and *occluder velocity map* for the two targets; (ii) combining both maps to generate an *occlusion anticipation map*. Both stages rely on a mapping between a Toric surface and a texture.

The texture is mapped to through angles θ (horizontal angle) and φ (vertical angle) of the Toric surface. Before performing the projection, we build a mesh representation of this surface on which every vertex is supplied with two additional channels of information: (a) its texture coordinates, that will linearly map the Toric angles (θ,φ) illustrated in figure 4, (b) the normal and tangent to the surface at this vertex that compose the local surface basis matrix TBN which is later required in the camera motion strategy.

We rely on these additional channels to (a) transform any local pixel of a projected map into its corresponding Toric texture coordinates (as illustrated in figure 5), and (b) transform an occluder velocity expressed in the Cartesian space, into a plane tangent to the Toric surface (*i.e.* a local Toric-space velocity vector). With this in mind, we can detail our visibility computation pipeline.

4.1. Deferred projection of shadows and velocities

In order to obtain a good approximation of the visibility of potentially complex 3D targets, we propose to perform visibility computation for a number (r) of random sample points on the surface of each target object (inspired by [CNO12]). Let's consider one such point s, picked on target object o. For this point, we will render the *S-map* (S_s^o) as well as the *V-map* (V_s^o) using deferred projections.

G-Buffers To compute these two projections, we first need to render required information into a set of screen-space textures. We perform two render passes, both using the same camera projection (looking from s to the current camera position on the Toric surface): the first render pass only renders the mesh of the Toric surface, and the second renders the occluders (*i.e.* the rest of the scene without the object o to avoid self-occlusions. In the first pass, we store the depth map in texture Z_{toric}, the mesh 3D positions in P_{toric}, and the surface texture coordinates provided in the extra information channels on the mesh in texture UV together with its normal and tangent vectors (in two textures N and T). The P_{toric} map is later used to compute the velocity of the Toric by finite differences at the

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

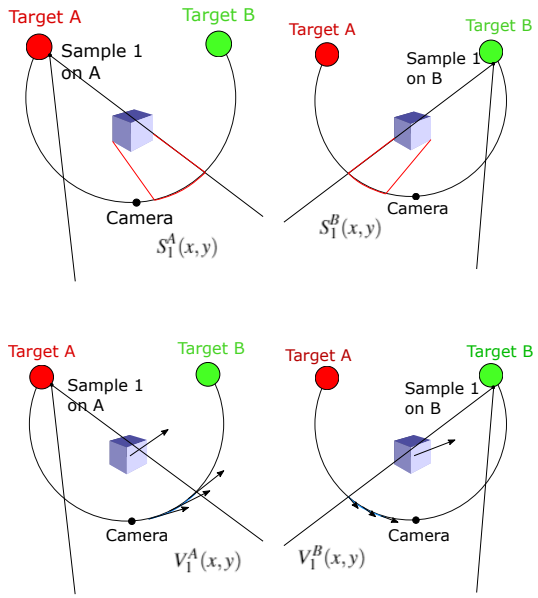


Figure 5: *Deferred projection performed from one sample on each target, with at the top the shadow maps, and at the bottom the velocity maps.*

current location. In the second render pass, we store the depth map (in texture Z_{occ}) and the occluders 3D positions (in texture P_{occ}). The information stored in the map $UV(x,y)$ is required to express a local position (x,y) of a map into the corresponding global position in Toric coordinates on the *S-map*s and *V-map*s.

4.1.1. Target shadow maps

For each point s on the target object o, we fill the 2D *target shadow map* by simply comparing depths:

$$S_s^o(UV(x,y)) = \begin{cases} 1 & \text{if } Z_{occ}(x,y) < Z_{toric}(x,y) \\ 0 & \text{otherwise} \end{cases}$$

4.1.2. Occluder velocity maps

We now want to predict locations from which this sample point will be occluded in the next frames. To do so, we will rely on the current occluders velocity. What we propose is to first render this velocity into a velocity map (V_s^o). By deriving our position buffer P_{occ} using finite difference between two frames, we determine such a world-space velocity $\mathbf{v}_{occ}(x,y) = dP_{occ}(x,y)/dt$. In the same way, we determine the world-space velocity $v_{toric}(x,y)$ of the Toric surface. We finally express the occluders velocity in the Toric space:

$$V_s^o(UV(x,y)) = TBN^t.(\mathbf{v}_{occ}(x,y) - \mathbf{v}_{toric}(x,y))$$

where the matrix TBN is computed from the normal and tangent vectors $N(x,y)$ and $T(x,y)$. Do note that by removing the torus velocity, we get the relative velocity with respect to the torus (moving with targets). Then, by projecting it in the tangent space, we can

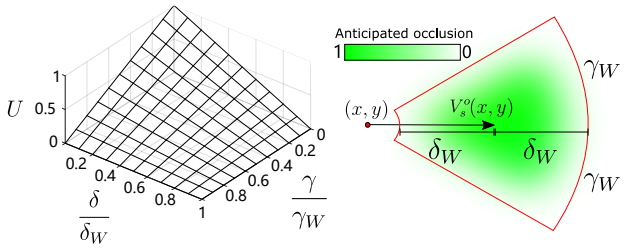


Figure 6: *Uncertainty function U. (left) graphical representation of U accounting for the distance δ and angle γ to the velocity vector V. (right) blur kernel in texture space for a pixel at position (x,y); the length and angular cutoffs are marked as red lines.*

in fact remove its 3rd component (orthogonal with the normal) to obtain a 2D vector expressed in its tangent plane, which we store in the map.

4.2. Computing the Occlusion Anticipation Map

We now have generated $2r$ pairs of shadow+velocity maps (one for every sample point). We combine all maps to generate a single *occlusion anticipation map* $A(\theta,\varphi)$ encoding the area of probabilities of our target objects being occluded or not (from 0 – none will be occluded – to 1 – both will always be occluded).Do note that shadow+velocity maps contain information only for a region of the torus, typically where the information has been projected around the current camera location. So at this stage, we will only update the *A-map* in these regions.

Now, in order to account for uncertainty in the future location of occluders, we propose an *occlusion prediction model* which relies on classical image processing operators for efficiency.

Given a point s, sampled on a target o, we can easily check whether it is (or not) occluded from a location (θ,φ) by reading the associated value $S_s^o(\theta,\varphi)$. Knowing the occluder's velocity (*i.e.* $V_s^o(\theta,\varphi)$), we propose to estimate the amount of future occlusion through an uncertainty function U, which in practice is an anisotropic blur operator, directed along the occluder's velocity. The operator takes as input a pair of 2D vectors (expressed in Toric space):

$$U(\mathbf{v}_1,\mathbf{v}_2) = U_l(\delta).U_a(\gamma) \quad (1)$$

where $\delta = \|\mathbf{v}_1\| - \|\mathbf{v}_2\|$ and $\gamma = (\mathbf{v}_1,\mathbf{v}_2)$.

In other words, for a neighbor location (θ',φ'), we compare the length and angular differences of the extracted velocity $\mathbf{v}_1 = V_s^o(\theta,\varphi)$ and vector $\mathbf{v}_2 = (\theta',\varphi') - (\theta,\varphi)$.In practice, we cast both comparisons into falloff functions (as illustrated in figure 6):

$$U_l(\delta) = max\left(1 - \frac{|\delta|}{\delta_W}, 0\right), U_a(\gamma) = max\left(1 - \frac{|\gamma|}{\gamma_W}, 0\right)$$

where δ_W and γ_W are custom parameters that represent the length and angular cutoff values of our uncertainty function (in our tests, we used values given in table 2).

Applying the prediction model Given our prediction model for one point s on one object o, we can now focus on computing

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

the probability of occlusion from any possible camera location, *i.e.* $A_s^o(\theta,\varphi)$. In theory, this can be computed by integrating function U over the couple S_s^o and V_s^o:

$$\iint_{-\infty}^{+\infty} U\left(V_s^o(\theta+x,\varphi+y),(-x,-y)\right)S_s^o(\theta+x,\varphi+y)\ dx\ dy$$

But in practice, we rely on textures. Hence, we compute its value at a pixel-wise level, through a double sum over pixels of S_s^o.

Aggregating predictions for all samples Our $2r$ partial occlusion predictions (each for a single point s) can now be aggregated into an overall *A-map*:

$$A(\theta,\varphi) = \frac{1}{2r}\sum_{o\in\{A,B\}}\left[\sum_{s=1}^{r} A_s^o(\theta,\varphi)\right]$$

5. Camera motion

We have all required information to design a camera motion controller that can enforce on-screen constraints (this is the purpose of our supporting Toric surface) and avoid future occlusions of target objects. Put altogether, we provide a low-dimensional camera space (the 2D Toric surface), supplied with a scalar field encoding the risk of our targets being occluded in the next frames.

In the following, we propose to mimic a physical model [Rey99, GCR*13] where our camera will behave as a particle on which external forces are applied steering it toward unoccluded camera locations. The motion of the camera can therefore be formulated as a function of its previous position ($\mathbf{p_i}$), velocity ($\mathbf{v_i}$) and k external forces ($\mathbf{F}$):

$$\mathbf{v}_{i+1} = \mathbf{v_i} + \left[\frac{1}{m}\sum_{j=1}^{k}\mathbf{F}_{i+1}^{j}\right]dt \quad \text{then} \quad \mathbf{p}_{i+1} = \mathbf{p_i} + \mathbf{v}_{i+1}.dt$$

with dt the time step, and m the particle mass. Note that we also need to ensure the camera remains on the surface of our moving torus.

Physically plausible camera motion To enforce visibility at a low computational cost, we propose a ad-hoc position update, which is nonetheless explained by a physical model with two external forces:

- a visibility force ($\mathbf{F}_v$) that steers the camera towards a location with a lower occlusion risk;
- and a damping force ($-c\mathbf{v_i}$) controlling the camera's reactivity (the higher the damping factor c, the lower the reactivity).

We firstly observe that, in a physical model

$$p_{i+1} = p_i + \left[v_i\left(1-\frac{c}{m}dt\right)dt\right] + \frac{1}{m}\mathbf{F}_v.dt^2 \qquad (2)$$

the middle part is a fixed translation, function of the velocity, damping, and elapsed time. The right part represents a bounded surrounding area, function of the two external forces and the elapsed time, containing all possible camera locations at the next frame (or if we increase dt, after a few frames).Starting from this observation, our general strategy is to search within this surrounding area for a less occluded camera location. This should provide a physically-plausible camera motion, near-optimal in the sense of the three constraints we stated earlier (smooth motion, on-screen composition, and occlusion avoidance).

In order to move the camera to a potentially less occluded location (if one exists), we propose three local sampling strategies, applied in this order:

1. we use a *single frame* search D_S for a less occluded camera location in the next frame $(i+1)$. We therefore restrict the search to the surrounding area which the camera can reach at the next frame (see figure 9a).
2. when D_S finds no satisfying solution (*i.e.* a configuration less occluded), we use a *look-ahead* search D_{LA} in a wider surrounding area that the camera can reach after a given fixed number of frames. We then steer the camera, by selecting the closest position inside area D_S (see figure 9b).
3. when D_{LA} still finds no satisfying solution, we search a *cut* to a further, less occluded, location. We cast this into a stochastic search in area D_C (in practice the whole surface) and then instantly teleport the camera to this position; this is a common strategy in video games as well as movies (camera "cut" policy [RT09a]).

Note that our *A-map* will here act as a regularization operator in these local searches, by casting high-frequencies contained in the *S-map*s into a low-frequency prediction valid for a few frames ahead.

To perform the search process we evaluate the anticipated occlusion at position $\mathbf{p}(\mathbf{v}_i)$ (*i.e.* if no external force applies), and then randomly sample locations in a given surrounding area D_X. The selected location $\hat{\mathbf{p}}$ must provide an improvement in the predicted occlusion:

$$\hat{\mathbf{p}} = \underset{(\theta,\varphi)\in D_X}{\arg\min}\ A(\theta,\varphi),\ \text{subject to}\ A\left(\hat{\mathbf{p}}\right) \leq A\left(\mathbf{p}(\mathbf{v}_i)\right)$$

Moving cameras in Toric space A purely physical model of camera motion is defined in world space, while our *A-map* is expressed in Toric space along the coordinates (θ,φ). An easy way to define our camera motion model would be to compute the motion directly in Toric space coordinates. However since the mapping introduces distortions, *i.e.* the same amount of motion in Toric coordinates does not correspond to the same amount of motion in Cartesian space, depending on where the computation is performed on the surface and on the size of the Toric surface. The size is dependent on the Toric α value (angle between the camera and the targets) and the distance 3D between the targets. We instead propose to define the camera motions in Cartesian world coordinates using a plane tangent to the Toric space. We first define the search area on the tangent plane of the torus at the current camera position; we then uniformly sample points in the search area, which we project onto the Toric surface, *i.e.* to now obtain plausible 2D locations in Toric coordinates (θ,φ) so to exploit the information held in the *occlusion anticipation map*. This process is illustrated in figure 7.

5.1. Physically-plausible search areas

In the following, we define the shape of the search areas D_S and D_{LA}, as well as explain how to evaluate occlusions in area D_C.

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

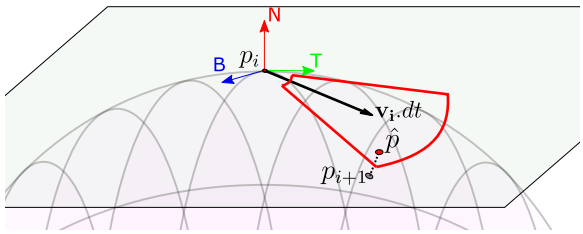
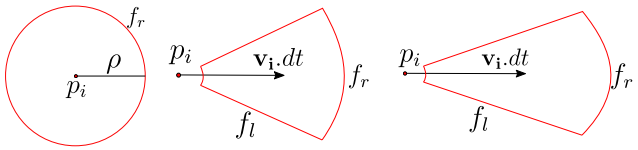
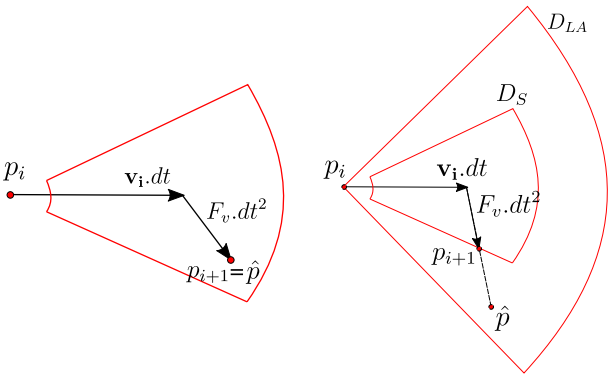


Figure 7: *3D view of our local search process for one frame.*



(a) *Low velocity* **(b)** *Medium velocity* **(c)** *High velocity*

Figure 8: *Impact of camera velocity on the single-frame search area.*



(a) *Single-frame search strategy.* **(b)** *Look-ahead search strategy (over several frames).*

Figure 9: *Local strategies*

Single-frame search The locations which our camera can physically reach within a single frame define an area bounded by the range of possible forces $\mathbf{F}_v$ which can be applied to the camera. Here we propose to approximate this range of possibilities using a linear threshold and a rotational threshold (f_l and f_r respectively, see figure 8), *i.e.* enabling the camera to perform, at each frame, a range of changes in velocity and in direction. Obviously the range of changes are dependant on camera physical parameters such as previous velocity ($\mathbf{v}_i$) and damping factor (c). In an empirical way, we propose to express both changes as:

$$f_r = \frac{2\pi - \chi}{e^{g.\|\mathbf{v_i}\|dt}} + \chi \text{ , and } f_l = 2h.\|\mathbf{v_i}\|dt \qquad (3)$$

where f_r represents the angle threshold dependant on the speed, which is used to define the range of possibilities in angle $[-f_r/2, f_r/2]$. Constant χ is a constant (in radian) representing the angular span at high speed, and g is a constant defining how the angles evolves as a function of the camera velocity. f_l represents a relative distance threshold, also dependent on the speed, used to define the range of possibilities in positions as $\mathbf{p}_i + \mathbf{v}_i.dt + [-f_l/2, f_l/2]$. h is a constant controlling the linear freedom in speed. By playing with these parameters, one can easily control the camera reactivity to match specific requirements of an application.

With the current formulation, at low camera speeds (*e.g.* $\|\mathbf{v_i}\| = 0$), the threshold f_l is null. To handle such situations, we provide another constant ρ representing the radius in a way that $f_l \geq \rho$ in all cases. Thanks to this, we will keep searching at low camera speeds (see figure 8a) and provide a sufficient impulse as soon as a local solution is found. After this local search, if we have successfully found a less occluded location, we can update the camera position, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}}$.

Look-ahead strategy In case there is no better solution in D_S, we search a larger area D_{LA} by bounding locations which the camera can physically reach within a few frames, rather than just the next frame. We simply need to replace dt by Ndt in equation 3 ($N > 1$ is a user-defined value). After this new local search, if a solution is found, we can update the camera position, moving it in the direction of the best solution, while staying in D_S, *i.e.* $\mathbf{p}_{i+1}$ is the intersection of the edge of D_S and the segment $[\mathbf{p}(\mathbf{v}_i), \hat{\mathbf{p}}]$ (see Fig. 9b). Assuming that a solution has been found, we need to update the velocity $\mathbf{v}_{i+1}$, which could be done by simply deriving the camera's position. In practice, this velocity is dampened so that the camera stabilizes when there are no occlusions:

$$\mathbf{v_{i+1}} = \left(\frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{dt}\right)\left(1 - \frac{c}{m}dt\right)$$

Cut strategy The cut strategy is applied when no better solution can be found in D_S and D_{LA} and enables the camera to jump to another location. In such case we search in the area D_C, which is much larger than the current camera neighborhood. Indeed, as illustrated in figure 5, the *A-map* computation is only performed in a local area around the current camera position. In fact, recomputing the entire map (Toric surface) would be too inaccurate because of high distortion in the rendered buffers, and low precision around the camera position. Furthermore the camera performs a continuous motion in most cases, so computing the whole map is unnecessarily expensive. This D_C area is additionally pruned for camera locations with a 30° angle rule of the current camera location. This a very common film editing convention [RT09a] preventing jump-cuts.

To perform this search in D_C we propose to rely on ray-casting. We cast rays to the same sample points s as before, and only rely on these tests to compute an occlusion ratio (*i.e.* there is no occlusion prediction). As soon as a less occluded position is found we perform a cut by teleporting the camera, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}}$. Further, in this case, we reset the camera's velocity, *i.e.* $\mathbf{v}_{i+1} = \mathbf{0}$.

When no better solution has been found in these 3 searches, then we consider that the best option is to leave our camera particle follow its current path, *i.e.* $\mathbf{p}_{i+1} = \hat{\mathbf{p}} = \mathbf{p}(\mathbf{v}_i)$.

5.2. Style masks

While the camera moves autonomously to minimize occlusion, one might also want to avoid some viewpoints, depending on the

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

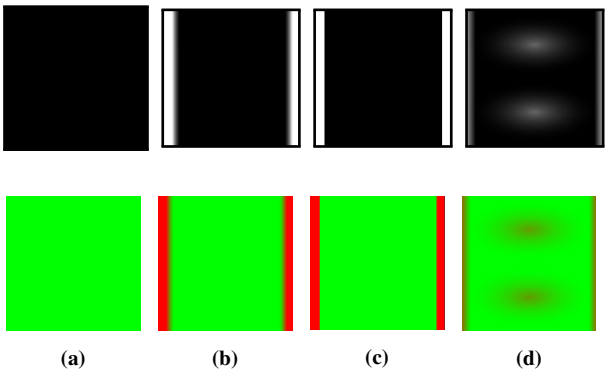
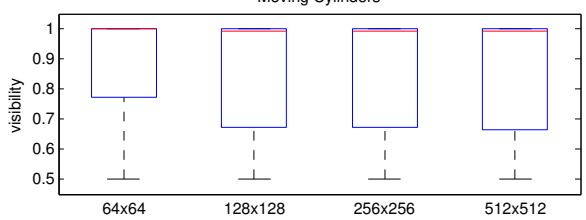


Figure 10: *Examples of masks (top) and the impact on the* A-map *(bottom). (a) Empty mask, (b) flexible exclusion of viewpoints behind targets (with a gradient), (c) hard exclusion of viewpoints behind targets, (d) smooth repulsive mask avoiding regions behind the targets, top views and bottom views. Mask images (top) are surrounded by a box so as to highlight edge gradients.*

Table 2: *The set of default values we used to run our tests*

parameter	r	δ_w	γ_w	χ	g	h	ρ
value	5	0.6	0.5	10°	5	0.5	0.1



(a) *Visibility performance for different maps resolutions (the higher the better).*

Size	64x64	128x128	256x256	512x512
FPS	197.09	130.04	88.80	18.96

(b) *Framerate for different map resolutions.*

Figure 11: *Comparison of performances for different map sizes. Red: median value; Blue: 1st to 3rd quartile. The resolution impacts strongly the computational cost (b) but not the capacity to maintain visibility (a).*

targeted application. For example, in interactive applications with character control, one might prefer to not see the scene from bellow or from above angles. For this purpose, we propose the notion of style masks on the *A-map* to influence the camera motion, or avoid specific areas (see examples in figure 10). We update the *A-map* using the following formulation:

$$A(x,y) = 1 - (1 - A(x,y) * Mask(x,y))$$

6. Results

Implementation We implemented our camera system within the Unity3D 2018 game engine. We compute our G-buffers through the Unity's integrated pipeline, while we perform our image processing stages (sections 4.1.1, 4.1.2, 4.2 and 5.2) through Unity

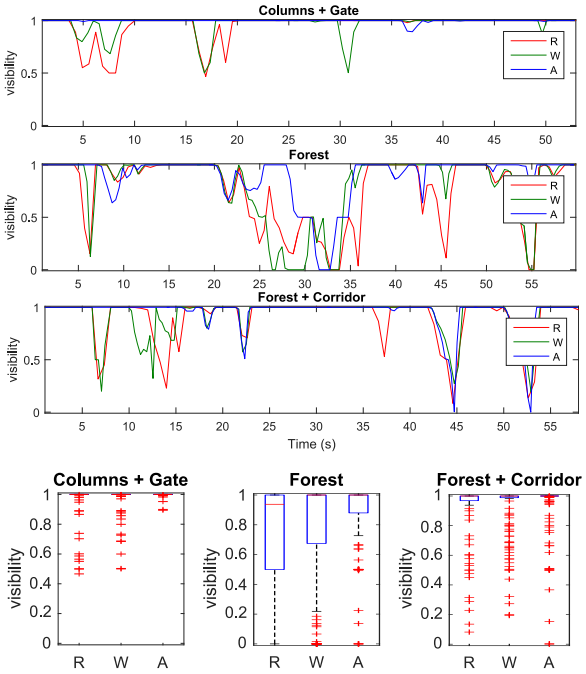


Figure 12: *Visibility comparison on 3 complex scenes, using the ray-cast method (R), our method with anticipation (A) or our method without anticipation (W). (top) Visibility along time (in s), (bottom) side-by-side comparison of performances. Red line: median value; Blue: 1st to 3rd quartile; Red crosses: outliers.*

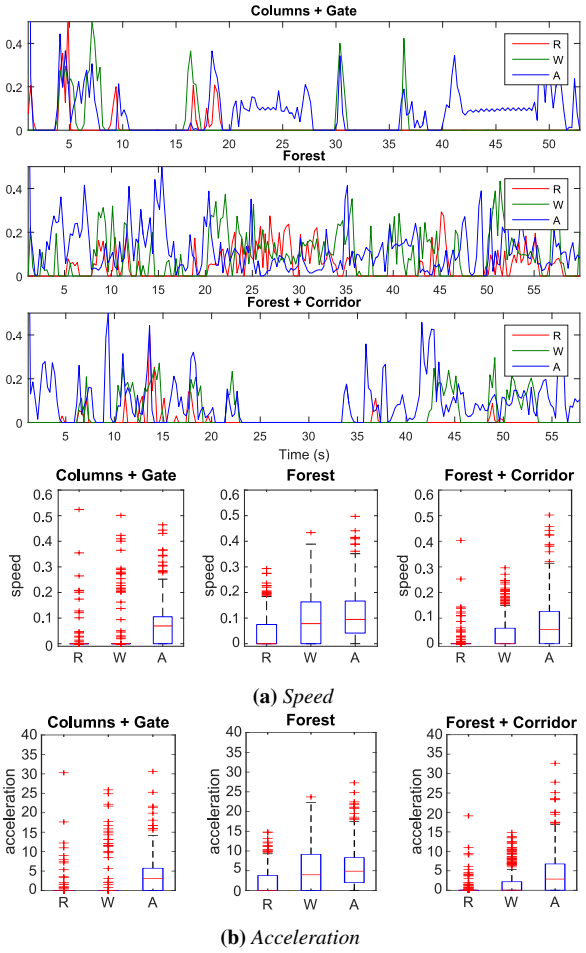
Compute Shaders. All our results (detailed in section 6) have been processed on a desktop computer with a Intel Core i7-7820X CPU @ 3.60GHz and a NVidia Titan Xp.

We evaluate our camera system along two main criteria: how much the targets are visible (1 meaning that both targets are visible, 0 both targets are occluded), and how smooth the camera motions are. We compare the performance of our system with different sets of parameters and features *i.e.* changing the size of computed maps, and using (or not) our occlusion anticipation *vs.* a commonly used occlusion computation. We perform comparisons on 4 scene configurations (illustrated in the accompanying video) made of a simple scene and three complex scenes: (ii) a scene with a set of columns and a gate (**Columns+Gate**) which the target avatar goes through (iii) a scene with the avatar travelling a forest (**Forest**) and (iv) a scene with the avatar travelling a mix of forest and corridors with big walls (**Forrest+Corridor**).

Comparisons are performed as a post-process to not influence the performance of the system. To provide a fair comparison between techniques, we measure the visibility degree on both targets by casting a large number of rays (1 ray per vertex on each target) . In a way similar, we evaluate the quality of camera motions by computing their derivatives, *i.e.* speed and acceleration, which provide a good indication of camera smoothness.

Impact of map resolution As a stress test, we ran our system on our simple scene configuration, where a pair of targets (spheres) is moving in a scene which we progressively fill with cylinders moving in random directions. We add up to 70 cylinders (4 per

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.



(a) *Speed*

(b) *Acceleration*

Figure 13: *Motion features on complex scenes, if using the ray-cast method (R), or our method with (A) or without (W) anticipation. (top) reported camera speed for 3 benchmarks, (bottom) speed (a) and (acceleration) distribution.*

frame) until 47s of simulation; the whole simulation lasts about 60s. We ran this test with medium-size cylinders, or large cylinders. We have evaluated the ability of our system to compute and enforce visibility by comparing the mean (actual) visibility along time, and the average frame rate. As shown in figure 11, decreasing the map resolution does not yield any noticeable loss in visibility enforcement, even for 64x64 maps. Conversely, there is a noticeable computational speedup. In all our following tests we rely on 64x64 maps.

As a second test, we ran our system on the three complex scenes and focused on the system's performances when using a brute-force ray-casting to evaluate visibility (R), or using our computed maps while enabling (A) or disabling (W) the occlusion anticipation stage. When disabled (W), our *A-map* was computed as the averaged value in each *S-map*, *i.e.* we remove the use of our uncertainty function. Brute-force raycasting is very common in game engines, but its cost prevents the computation of our anticipation map with real-time performances; hence, in this case (R) we directly cast rays (to the $2r$ sample points) at the search stage (*i.e.* we compute no anticipation map). For all three techniques, we compare their cost, and ability to enforce visibility (figure 12) and to provide smooth camera motions (figure 13). We then provide a breakdown of the computational cost of our method (A) (table 3).

	Each map update		Each frame	
Steps	Projections (*S-map* + *V-map*)	*A-map*	Search	Camera update
mean	9.8	11.9	4	2.8
(st. dev)	(3.16)	(2.25)	(0.95)	(1.34)

Table 3: *Computation time in ms. We use 64x64 maps, 3 sample points per target (at each update, we recompute the maps for one sample on each target), and 50 sampled points in the search area.*

Impact of the visibility computation technique From our results, it appears clearly that with (A) or without (W) anticipation methods always improve visibility on targets compared to the ray-cast based method (R). Further, enabling the anticipation stage (A) provides an improvement compared to disabling it (W). Typically, in the *Forest* scene which is a scene specifically designed with a high density of occluders, the method with anticipation (A) shows best performance (see figure 12). In the *Columns + Gate* scene, their is a clear benefit in using our anticipation step, especially at moments where the camera has to follow the target through the gate.

Motion smoothness Our motion strategies lead to smaller variations of velocity and acceleration (figure 13), while outliers can be due to either cuts, or our strategy at low camera speed, providing some impulse to the camera. Furthermore, when anticipation is not used ((R) and (W)), the acceleration remain lower, but at the cost of reducing the visibility.

Cost of the visibility evaluation As expected, casting rays (on the CPU) is much more expensive than computing our maps (on the GPU). In our tests, we experimentally chose $r = 5$ to enable a fair comparisons of the system's performances (*i.e.* a fast-enough frame rate for the ray-cast based method (R)). As expected intuitively, the cost of computing our maps is linear in the number r of samples, while computing and fetching the anticipation map is made at a fixed cost, as we perform the search on the GPU (hence in parallel).

Computational costs breakdown As shown in table 3, the most expensive stage per frame is the computation of the *A-map*, then the projections of all *S-maps* and *V-maps*. The search and camera update are conversely inexpensive stages. To improve computational costs, we notice that, by tweaking parameters δ_W and γ_W, we can predict occlusion for a long-enough time window W. Doing so, we propose an optimization: to not update all $2r$ partial *A-map* at every frame, but 2 of them only (one per target). In other words, each map A_s^o would be re-computed every r frames. Moreover, we propose to not make an update at every frame, but instead to use a refresh rate matching our time window W. It will dictate when to make an update, *i.e.* every W/r seconds. In our tests, we make an update every 0.1 second (*e.g.* if $r = 5$, any map is updated every 0.5 second). We experimentally noticed that, for moderate values of r, the impact of this optimization on our system's visibility criteria is not noticeable.

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

7. Discussion and limitations

We have highlighted the key features of our approach: low computational cost and improved visibility through anticipation. There however remains some limitations. First, there is an intrinsic limitation in the underlying Toric space representation, *i.e.* the camera must move at the surface of a moving torus, while there is no such restriction in the 7D camera space. We would like to investigate the provision of slight changes in the desired screen locations of targets to expand the range of motions; for instance we could also compute a slightly smaller and a slightly wider torus, leading to the computation of a 3D *A-map*. Typically, this would enable to move the camera closer to the avatar, *e.g.* to avoid some occlusion. In our future work, we would like to better exploit the depth information in the *S-map* to derive a camera motion control over all Toric-space parameters (φ, θ and α). Second, our approach is primarily designed to avoid occlusion, while simple secondary constraints can be added with style masks. More complex (and ideally dynamic) secondary constraints could be added by providing better representations. Finally, taking a step further, the Toric space we rely on is one possible model, allowing to enforce on-screen constraints; in the future, we would like to adapt our rendering+anticipation framework to perform our anticipation for 3D primitives defined around one or more targets, on which we could define more complex motions with physical constraints, such as cranes or drones.

8. Conclusion

In this paper, we have proposed a real-time occlusion avoidance approach for virtual camera control. The system first computes an occlusion map by projecting occluders on a camera control surface (the Toric surface), and then exploits information on the velocity of the occluders vertices, to derive an anticipation map. The anticipation map is then exploited by a physical camera motion control to compute a new camera position minimizing occlusions. We compared our system with an elaborate ray-casting approach, and with our system in which anticipation was disabled. Results reported better performances both in terms of computational cost (compared to ray-casting), overall visibility as well as smooth motions both in terms of camera velocity and acceleration.

References

[BKF*02] BURTNYK N., KHAN A., FITZMAURICE G., BALAKRISHNAN R., KURTENBACH G.: Stylecam: interactive stylized 3d navigation using integrated spatial & temporal controls. In *ACM symposium on User interface software and technology* (2002). 2

[CNO12] CHRISTIE M., NORMAND J.-M., OLIVIER P.: Occlusion-free camera control for multiple targets. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012). 4, 5

[COCSD03] COHEN-OR D., CHRYSANTHOU Y. L., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003). 3

[CON08a] CHRISTIE M., OLIVIER P., NORMAND J.-M.: Camera control in computer graphics. *Computer Graphics Forum 27*, 8 (2008). 3

[CON08b] CHRISTIE M., OLIVIER P., NORMAND J.-M.: *Occlusion-free Camera Control*. Research Report RR-6640, INRIA, 2008. 4

[ESAW16] EISEMANN E., SCHWARZ M., ASSARSSON U., WIMMER M.: *Real-time shadows*. AK Peters/CRC Press, 2016. 3

[GCLR15] GALVANE Q., CHRISTIE M., LINO C., RONFARD R.: Camera-on-rails: automated computation of constrained camera paths. In *ACM SIGGRAPH Conference on Motion in Games* (2015). 3

[GCR*13] GALVANE Q., CHRISTIE M., RONFARD R., LIM C.-K., CANI M.-P.: Steering behaviors for autonomous cameras. In *Motion in Games* (2013). 7

[GLC*18] GALVANE Q., LINO C., CHRISTIE M., FLEUREAU J., SERVANT F., TARIOLLE F., GUILLOTEL P.: Directing cinematographic drones. *ACM Transactions on Graphics 37*, 3 (2018). 3

[HH97] HECKBERT P. S., HERF M.: *Simulating soft shadows with graphics hardware*. Tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1997. 3

[HHS01] HALPER N., HELBING R., STROTHOTTE T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum 20*, 3 (2001). 4

[LC12] LINO C., CHRISTIE M.: Efficient composition for virtual camera control. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012). 2, 5

[LC15] LINO C., CHRISTIE M.: Intuitive and efficient camera control with the toric space. *ACM Transactions on Graphics 34*, 4 (2015). 2, 4

[LCL*10] LINO C., CHRISTIE M., LAMARCHE F., SCHOFIELD G., OLIVIER P.: A real-time cinematography system for interactive 3d environments. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2010). 4

[LT17] LITTENEKER A., TERZOPOULOS D.: Virtual cinematography using optimization and temporal smoothing. In *International Conference on Motion in Games* (2017). 3

[NAMD*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). 3

[NMD*17] NÄGELI T., MEIER L., DOMAHIDI A., ALONSO-MORA J., HILLIGES O.: Real-time planning for automated multi-view drone cinematography. *ACM Transactions on Graphics 36*, 4 (2017). 3

[OSTG09] OSKAM T., SUMNER R. W., THUEREY N., GROSS M.: Visibility transition planning for dynamic camera control. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009). 3

[Rey99] REYNOLDS C. W.: Steering behaviors for autonomous characters. In *Game developers conference* (1999), vol. 1999. 7

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics 27*, 5 (2008). 3

[Rot82] ROTH S. D.: Ray casting for modeling solids. *Computer graphics and image processing 18*, 2 (1982). 3

[RT09a] ROY THOMPSON C. J. B.: *Grammar of the edit*, second edition ed. Focal Press, Boston, 2009. 7, 8

[RT09b] ROY THOMPSON C. J. B.: *Grammar of the shot*, second edition ed. Focal Press, Boston, 2009. 2, 4

[TS91] TELLER S. J., SÉQUIN C. H.: Visibility preprocessing for interactive walkthroughs. *ACM SIGGRAPH Computer Graphics 25*, 4 (1991). 4

[VFSL02] VÁZQUEZ P.-P., FEIXAS M., SBERT M., LLOBET A.: Viewpoint entropy: a new tool for obtaining good views of molecules. In *ACM International Conference Proceeding Series* (2002), vol. 22. 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *ACM Siggraph Computer Graphics 12*, 3 (1978). 3

[WP12] WOO A., POULIN P.: *Shadow algorithms data miner*. AK Peters/CRC Press, 2012. 3

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.