Improving the Efficiency of Viewpoint Composition

Roberto Ranon and Tommaso Urli

Abstract—In this paper, we concentrate on the problem of finding the viewpoint that best satisfies a set of visual composition properties, often referred to as *Virtual Camera* or *Viewpoint Composition*. Previous approaches in the literature, which are based on general optimization solvers, are limited in their practical applicability because of unsuitable computation times and limited experimental analysis. To bring performances much closer to the needs of interactive applications, we introduce novel ways to define visual properties, evaluate their satisfaction, and initialize the search for optimal viewpoints, and test them in several problems under various time budgets, quantifying also, for the first time in the domain, the importance of tuning the parameters that control the behavior of the solving process. While our solver, as others in the literature, is based on Particle Swarm Optimization, our contributions could be applied to any stochastic search process that solves through many viewpoint evaluations, such as the genetic algorithms employed by other papers in the literature. The complete source code of our approach, together with the scenes and problems we have employed, can be downloaded from https://bitbucket.org/rranon/smart-viewpoint-computation-lib.

Index Terms—Virtual Camera Control, Viewpoint Computation, Virtual Camera Composition, Viewpoint Composition.

1 Introduction

The problem of finding an effective viewpoint within a 3D environment is pervasive in any 3D graphics application, such as modeling, video games, data visualization and virtual storytelling. When the viewpoint is not controlled by the user, an algorithm has to compute a good one according to some goals, e.g., in third person games, showing an unoccluded back view of the main character at some distance, also maintaining frame coherence. The algorithms employed in real-world applications are typically very specific and limited in the kind of goals they can achieve, for example not being able to frame more than one target, or to find suitable viewpoints in spatially complex scenarios. In those situations, programmers are forced to resort to ad-hoc camera scripting, often combined with manual, off-line viewpoints positioning.

In the last years, some researchers have proposed more general approaches, with the aim of requiring less manual intervention from the programmer and extending the expressive potential of automatic camera control. In this paper, we focus on approaches where the programmer defines camera control goals by arbitrarily combining basic

• *R. Ranon is with the Human-Computer Interaction Lab, Department of Math and Computer Science, University of Udine, Italy. E-mail: roberto.ranon@uniud.it*
• *T. Urli is with the Dept. of Electrical, Management and Mechanical Engineering, University of Udine, Italy.*

elements used in photographic composition, e.g. object(s) visibility, size or position in the resulting image, and a general purpose constraint or optimization solver finds the viewpoint or viewpoint path that best satisfy them. More specifically, this paper focuses on the basic problem of finding one optimal viewpoint, often referred to as *Virtual Camera (or Viewpoint) Composition* (hereinafter, *VC*) [1], [2], [3], [4], [5], [6], [7].

Interesting uses of VC have been proposed in 3D virtual visits creation [8] and virtual cinematography [9] (to automatically suggest potentially interesting viewpoints in the authoring process) and within sophisticated dynamic automatic camera control approaches [10], [11] (where VC computes viewpoints, e.g., that conform to cinematic conventions, and path planning is used to move the camera between them).

The approaches to VC proposed so far, although generally able to find viewpoints in complex situations (both spatially and in the number of targets and requirements), struggle to find practical application, for a number of reasons. First, performances reported in the literature are in most cases discouraging (e.g. in the order of several tenths or even seconds to solve not-so complex problems, or much varying in solving time). Second, papers typically lack an experimental evaluation on different and not trivial situations, because they use simple or very few experimental scenarios. Third, most approaches rely on solvers whose behavior is controlled by a number of parameters, and the optimal choice of parameter values, which is often critical, is rarely discussed and supported by experimental evaluation. Fourth, source code is typically not available, and therefore results are difficult to reproduce and compare.

Of the above mentioned limitations, efficiency is probably the most critical one. Being able to find a suitable new viewpoint in interactive times, e.g. in response to changes in the scene, would make it much more viable to adopt a VC-based approach in practical applications, and thus reduce the need for ad-hoc scripting in the camera control process.

In this paper, we propose a number of improvements to the state of the art, and we apply them to a Particle Swarm Optimization (PSO) solver [12], which has been also used by other papers in the literature [6], [7]. However, our contributions can be applied to any stochastic search process that finds a solution through many viewpoint evaluations, such as the genetic algorithms and local search methods employed by other papers in the literature. Specifically, we propose:

- a formulation of visual properties highlighting calculations that are common when evaluating more properties on the same object, and allow one to reason more easily about trade-offs between accuracy and time;
- a method to evaluate a viewpoint goodness, based on computing the screen representation of relevant objects first, and strategies to terminate earlier, when doing more computations would not benefit search, based on a satisfaction threshold;
- a computationally efficient solver initialization strategy, not based, as most in the literature, on computationally expensive geometric operators that prune the search space.

Perhaps more importantly, we validate our contributions through a considerable experimental evaluation, which involves 15 problems defined over three scenes, and includes, for the first time in the domain, solver parameters tuning, which significantly increases performances and is crucial for a fair comparison of different VC strategies. The experimental evaluation shows that our individual contributions are generally more efficient than corresponding typical choices in previous work, and that their combination results in performances that are quite suited to the typical requirements of interactive applications. While, for reasons of space, the paper reports overall results on the considered problems, the supplemental material contains detailed information on each problem and performances on it. Also, we provide, at https://bitbucket.org/rranon/smart-viewpoint-computation-lib, the complete source code of our approach and the employed problems, so that readers can reproduce our results, experiment with other problems, or make comparisons with their own solutions.

The paper is structured as follows. Section 2 discusses related work; Section 3 presents our approach. In Section 4, we discuss the experimental evaluation, and Section 5 concludes the paper and outlines future work.

2 Related Work

The process of computing the parameters of a virtual camera to render an image with a desired set of visual properties, e.g., to convey the appropriate information about the 3D scene or, more specifically, about some key objects in it, is composed of three main aspects: (i) how we define the desired set of visual properties, (ii) how this set of visual properties is turned into some measure of viewpoint goodness, and (iii) how we search for a viewpoint with a desired goodness.

In the following, we focus on approaches where visual properties are inspired by the basic elements used in photographic composition, such as object(s) size or position in the resulting image, and that do not limit the number of objects or properties that can be considered. These approaches assume that the geometry of the scene is known, and that its relevant objects are properly labeled. By contrast, other approaches, e.g., devoted to problems such as single object examination [13] or scene understanding where objects of interest are not known a priori [14], work mainly with low-level geometrical data and define measures of viewpoint goodness based mostly on that (e.g., see [15] for a review and comparison of different measures of goodness). Full 3D camera control is outside of our scope, and we refer the reader to the review of general methods by Christie et al [16].

2.1 Visual Properties

Visual properties to control an object's *inclusion*, *size*, *position / framing*, and degree of *visibility* (or *occlusion*) in the image, as well as *orientation* (sometimes called *angle*) with respect to the viewpoint, are common throughout most approaches, e.g. [1], [2], [3], [4], [5], [6]. Extensions to this core set include relative properties between two objects [2] (such as, object x must appear to the right of object y), and more high-level photographic properties, such as *visual balance* and *rule of thirds* [17], [7] that aim at enforcing well-established aesthetic criteria. To give precedence to a property over another (e.g., because a priori it is not known if both can be fulfilled), an additional numeric parameter that indicates the importance or *weight* of a property is adopted in most approaches.

2.2 Defining and Evaluating Viewpoint Goodness

While some early approaches [18], [19] treated properties as hard constraints, most approaches so far have chosen instead to treat properties as satisfaction functions that can have any value from no satisfaction (0.0) to full satisfaction (1.0). The goodness of a viewpoint is then typically defined as the (weighted) sum of each property satisfaction function.

When evaluating a property satisfaction, all approaches employ, for the sake of efficiency, some form of approximation, at least for properties that reason about the projection of objects, e.g., by computing the screen size of an object using its bounding sphere [5], [6] or axis-aligned bounding box [3]. Some approaches have also tried off-screen rendering to small resolution images to evaluate visual properties with more accuracy, but at the expense of computation times [7].

Occlusion properties are typically the most costly to evaluate, as they require to consider also other objects in the scene. Most approaches use ray casts [3], [5], [6] to evaluate the visibility of points belonging to an object or, more commonly, its bounding volume. Lino et al [10] process the scene to compute a 2D map of visibility of some key objects, but that is clearly efficient only if they do not vary much in time. Moreover, reasoning in 2D is not correct for all kinds of occluder. Christie et al. [20], drawing ideas from soft shadow techniques, use low-resolution rendering from a point (on a target object) to evaluate its visibility in a restricted region of space (e.g., around the current camera location). The approach is able to combine renderings to evaluate the visibility of multiple objects at the same time, and to increase accuracy as time passes by changing the points on the objects from which renderings are performed. However, the approach is efficient only when one needs to evaluate visibility in a limited region of space.

Using approximations introduces errors that may negatively impact both the user-perceived goodness of a viewpoint, as well as the search process, and an analysis of this issue is always understated or omitted in the literature. Ranon et al. [21] suggest that evaluating properties on off-screen rendered images of sufficient resolution could be taken as an ”accurate” evaluation (perceptual issues aside), and present a language to express most kind of properties in the literature and evaluate a viewpoint *accuracy* with respect to them. The approach, however, is meant as a way to compare the accuracy of different VC approaches, and not to evaluate viewpoints while solving a VC problem.

Another relevant aspect is what to return when a property actual value (e.g., the height of an object in the image) is close, but not equal to the desired one. Approaches that use satisfaction functions can naturally treat the situation as partial satisfaction (i.e., an intermediate value in the [0,1] interval), thus allowing the search process to reason about partially good viewpoints, and then find solutions even in over-constrained situations [16]. Moreover, slight deviations from a desired value might be acceptable, as they could not be noticed by the viewer. Various ways to model satisfaction decay from desired values have been proposed, e.g. linear [6] or Gaussian [7] functions, however with limited amount of external control and thus flexibility.

2.3 Finding Good Viewpoints

From the seminal work of Blinn [22], the research has evolved towards the use of general approaches for constraint-based reasoning [23] or optimization, such as genetic reasoning [4], [24], forms of local search [25], [26], or PSO [6], [7]. These approaches generally are not limited in the number of target objects, properties, or kind of scenes, and treat the space of viewpoint parameters as continuous (thus potentially exploring every possible solution), in contrast to early approaches that discretize the search space by considering only a subset of the possible viewpoints [3], a solution that moreover does not seem to provide efficiency advantages, as shown by some experimental tests in [6].

A general distinction can be made between approaches that, as the one presented in this paper, are designed to return viewpoints anywhere in the scene, e.g. [3], [23], [6], [7], [4], [5], and approaches that return viewpoints in a relatively small region of space around a point (e.g., around the previous camera position) [26], [25], [20]. The latter category of approaches generally needs to perform much less viewpoint evaluations, and is therefore able to return results in real-time; plus, as found viewpoints are close to previous ones, they quite naturally allow to move a camera in real-time. However, they can be easily stuck in local minima (e.g. not being able to resolve occlusions).

Approaches that can reason globally to the whole scene, instead, are much less prone to local minima, but their efficiency, at least from past results in the literature, is far from real-time, unless strong assumptions are introduced (e.g., Lino et al. [10] assume targets and scene geometry do not change much in time, and visibility can be correctly computed in 2D). Reported computation times are generally high (up to seconds) and/or very variable (also, in part, due to the stochastic nature of solvers) even on quite simple scenes and VC problems. A notable exception is [27], which uses fast algebraic techniques to find a viewpoint featuring exact on-screen positioning of two or three subjects. However, the approach is limited to situations where the desired on-screen position is known and, more importantly, does not handle occlusions.

One of the well-known issues is how to focus search on regions of space where solutions can be found. For example, approaches that initialize search randomly throughout the whole search space (e.g. [7], [24]) typically suffer from high variability in performances. Using a hybrid constrained optimization approach, where some properties are treated as geometric constraints to cut unfeasible parts of the search space, and optimization reasons with the remaining properties on the pruned search space, has been proposed as a solution to increase efficiency [3], [5], [6], [10], at the expense of losing some possible solutions in the case of over-constrained situations. However, even latest hybrid approaches such as [6] report an average solving time of half a second for a relatively simple VC problem which involves visibility of three objects on a small village scene. Moreover, the cost of cutting the search space through geometric operators can be considerable [5], [10].

Recent and promising attempts at combining global and local reasoning for real-time camera control avoiding the local minima problem are described in [10], [11]. In both cases, global reasoning is used to find viewpoints when locally found ones, for some reason (e.g. occlusions) are not suitable.

In general, papers in this area are quite lacking in the experimental evaluation of their proposed approach. Most papers demonstrate results on very few problems or use just one scene, which is often very simple (e.g., consisting of few more objects than the ones mentioned by the properties, or geometrically very simple). As a result, it is difficult to estimate performances in more complex and realistic situations. The lack of publicly available reference problems, moreover, causes each paper to use different scenes and VC problems, thus making it hard to reliably compare approaches. Comparisons of different solvers are presented in [6], [24], [28]. More specifically, [6] shows that a PSO solver obtains better performances than the heuristic solver by Bares et al. [3] in three VC problems in a village scene; [24] compares a custom-designed genetic search algorithm, octree-based best first and depth first search, and random search, using a single three-targets visibility problem in four different (interior and exterior) scenes and a set of time budgets between 15 and 250 milliseconds; and [28] compares a variant of the CMA-ES approach with other algorithms to evaluate how many different solution viewpoints they can find in three VC problems, each set in a different scene. However, in all these three papers (as well in any other experimental test published in this area) the parameters that control the behavior of solvers (e.g., number of initial candidates) are fixed and set to values that are not guaranteed to maximize solver performances. As it is generally known in the optimization field and as we experimentally show in this paper for VC, these parameters can heavily determine a solver efficacy in optimizing a given problem: using unsuitable values can then make it impossible to effectively assess a solver capabilities. Abdullah et al. [7] mention this problem and report that, with the PSO solver adopted in the paper, using more or less than 30 candidates was detrimental to PSO performances: however, as we experimentally show in Section 4, with PSO there are also other parameters which play a significant role.

A further, often understated issue is the variability in computation times that most approaches report when, as in many cases [3], [5], [6], [7], tests are run until a certain satisfaction value (or a maximum number of solver iterations) is reached. For example, average computation times for three different VC problems in [6] range from 150 milliseconds to about 2 seconds. This is clearly unacceptable for interactive applications, where typically the camera control process, as any other activity, has to provide results in a given time budget, which usually cannot be in the order of seconds. A better approach to evaluate real-world applicability would be to measure the results a solver can provide in a given time budget, as done in [24] and in this paper.

3 Our Approach

In this Section, we describe our approach to VC. We start by giving preliminary definitions, and then describe how we formulate a VC problem in terms of visual properties, and how we turn them into a measure of viewpoint goodness. We then discuss how we evaluate a viewpoint goodness, and how we search for a optimal viewpoint. Our approach features the most common kind of properties in the literature, but introduces improvements to allow for a faster evaluation of viewpoint goodness and greater efficiency of the search process.

3.1 Preliminary Definitions

In our approach, a viewpoint is defined by 8 real-valued components: three coordinates for the position (pos_x, pos_y, pos_z), three coordinates for the look-at point ($look_x$, $look_y$, $look_z$), a *roll* angle to define the horizon, and a field of view parameter γ. We consider aspect ratio AR as fixed (by the application or device) and thus not a parameter of the VC problem.

Most approaches in literature (e.g., [3], [5], [6]) use a more compact viewpoint representation with 7 real values, where the orientation is fully defined by three Euler angles (*yaw*, *pitch* and *roll*). A smaller representation corresponds to a smaller search space, which typically leads to better performance. However, our representation has the advantage of decoupling a viewpoint position and orientation, since - barring occlusions - the objects that are seen depend just on the look-at point, while, in case of Euler angles, they depend on both position and orientation. With solvers that optimize all viewpoint parameters at the same time, the theoretical advantage is that optimal values for viewpoint orientation (expressed as look-at point) can be determined almost independently from viewpoint position. In Section 4 we will experimentally compare the two alternatives.

Objects in the 3D scene mentioned in a VC problem are hereinafter called *targets*. Given a viewpoint v and a target t, we indicate with t_v some representation of t *as seen by* v. This representation could be obtained by rendering t from v, or by projecting the AABB of t to the viewport, with potentially ample variations in accuracy and computational cost.

3.2 Defining VC Problems

A *viewpoint specification* is constituted by a list $\{p_i\}$ of desired composition properties to satisfy, i.e., the specification is the logical conjunction of the properties. Allowing more complex expression, e.g. using logical disjunctions, requires minimal changes to the objective function presented later in this Section. Each property p_i is defined by a triple:

$$\langle eval(v, \text{args}), \text{sat}, \text{w} \rangle$$

where:

- *eval* depends on the kind of property, and its role is to compute an actual value from a viewpoint v and specific arguments $args$: For example, the function *Framing* (see Table 1) takes a viewpoint v, a target t and two viewport points defining a rectangle $Rect$, and returns the fraction of t_v area inside $Rect$;
- *sat* is a function taking the value computed by the corresponding $eval$ function, and returning a value in $[0, 1]$, where 0 means no satisfaction and 1 means full satisfaction. The *sat* function defines how satisfactory an actual value computed by *eval* is. In our approach, we define *sat* through a linear spline with an arbitrary number of control points.
- *w* (*weight*) is a real number used to specify the relative importance of a property.

Note that typically, in the literature, the *sat* function is embedded into the property definition by including a desired value as one of the property parameters. This means that the relation between the desired value and the actual computed value is

$eval$ **function**	**args**	**Semantics (**$eval$ **computes ..)**
Size	t, D	*Area, Width*, or *Height* (the possible values of D) of t_v in viewport-relative coordinates
Framing	t, $Rect$	the fraction of t_v which is inside $Rect$
RelativePosition	s, t, RL	the fraction of s_v which is *right, left, above*, or *below* (the possible values of RL) any point of t_v
Occlusion	s, t	the fraction of s_v which is occluded by t_v. t can be equal to *scene*, which means every object except s, when we want to take into account any source of occlusion
Angle	$t, \mathbf{u}$	the angle between vector $\mathbf{u}$ and the vector from t to the viewpoint; $\mathbf{u}$ can be also defined by the keywords *front, up* which respectively are t front and up vectors

TABLE 1
$eval$ functions of properties in our VC approach. In the table, s, t are *targets*; *Rect* is a 2D rectangle in viewport coordinates. In the third column, t_v indicates some representation of target t as seen by the viewpoint v.

fixed and cannot be changed. Our solution, while introducing a little complexity, gives much more flexibility in specifying exactly the meaning of satisfactory.

The *eval* functions available in our approach are presented in Table 1, and include the most common ones proposed in the literature. Note that the definitions we propose highlight the role of t_v when reasoning about a target t and do not constrain the way t_v is computed: this will be detailed in the next Section.

Having defined a viewpoint specification in terms of properties, the VC problem is then to find v that maximizes the objective function combining the satisfaction of all properties, i.e. v satisfaction:

$$\arg\max_{v \in V} \sum_i w_i sat_i(eval_i(v, args_i))$$

where V is the domain of possible viewpoints. In our approach, this is defined by one interval of allowed values for each viewpoint component.

3.3 Computing t_v for a target t

Following [21], an accurate representation of t_v can be obtained by rendering t from v at sufficiently large resolution (e.g., the user's application maximum resolution, as going beyond that would not make any practical difference) or, alternatively, by projecting and clipping all the polygons of t in screen space. Table 2 describes methods to compute the *eval* functions defined in Table 1 using off-screen renderings of targets, except for the *Angle* property, which can be accurately evaluated geometrically. All the described methods are based on rendering the target(s) with some flat color, transferring the resulting image from the GPU memory to RAM, and then processing its pixels.

Unfortunately, rendering-based methods are quite expensive in terms of time. For example, Table 4 reports the time needed to evaluate one *Size* or *Occlusion* property in one of the scenes of our experimental evaluation, called *City*, obtained by averaging 1000 property evaluations on random viewpoints (see Section 4 for details on the scene and our implementation). The cost for *Framing* and

Property	**computing t_v**	**Evaluation**
Size	render t (with flat color c) to texture	count c-colored pixels (for *Area*), count horizontal (vertical) c-colored pixels span for *Width* (*Height*)
Framing	as above	count c-colored pixels that are inside *Rect*, divide by number of c-colored pixels
RelativePosition	render s, t to the same texture (with flat colors c_s, c_t), with additive blending, depth test disabled	count c_s pixels that are left of (or right, above, below) $c_s + c_t$ or c_t colored pixels 1, divide by number of c-colored pixels
Occlusion	render s, t (s first) to the same texture (with flat colors c_s, c_t), with additive blending, depth test enabled (i.e. pixels belonging to c that are occluded will have color $c_s + c_t$)	count $c_s + c_t$-colored pixels, divide by number of c_s-colored pixels

TABLE 2

Rendering-based methods to evaluate properties in Table 1

RelativePosition properties would be similar, as they are based on the same operations. At a resolution of 1000 by 800 pixels, evaluating one property takes about 0.25 seconds. Reducing the resolution (and thus accuracy), the computational cost diminishes but, even at a resolution of 40 by 30 pixels, it still takes several milliseconds. As noted by other researchers [2], while rendering and pixel counting can be very fast, the problem here is the cost of transferring the image from GPU to CPU memory.

Table 3 illustrates our methods of choice for faster evaluation of properties, using bounding volumes and geometrical reasoning. With the exception of *Occlusion*, we compute t_v for a target t by taking the (oriented or axis-aligned) bounding box of t, finding the vertices of it that are visible from v, and projecting them, using the fast look-up table approach proposed by Schmalstieg and Tobler [29]. The resulting 2D hull polygon is then clipped by the viewport through a standard Cohen-Sutherland algorithm, and finally used for calculations as specified in the third column of the table. As the resulting polygon is convex, a contour integral approach can be used to quickly compute its area. Most approaches in the literature, instead, use bounding spheres or compute the 2D screen-space AABB of a target AABB projection, generally poorer approximations of t_v than our solution.

In the case of *Occlusion*, we do not compute s_v and t_v, but we sample a few points on the bounding box of s: we use the fairly standard solution of casting 9 rays from v to the center and each corner of s bounding box, and report the fraction of them which intersect t (or any other object in the case t is equal to the entire scene).

Note that, while some approaches compute t_v with respect to the viewplane [6], [3], our choice is to clip the projection of t to the viewport. This means that properties will reason just on the part of t which will end up in the image. In our opinion, this choice makes it easier to express a desired result in terms of properties, as one needs to reason on just what will end up in the image. On the other hand, this choice makes it harder for the search process to distinguish between situations

Property	**computing t_v**	**Evaluation**
Size	compute convex hull of projection of t bounding box, clipped by viewport	compute t_v *Area, Width* or *Height* by computing convex hull area, width or height, divide by viewport area, width or height
Framing	as above	clip t_v by *Rect* and compute its area, divide by t_v area
RelativePosition	as above	return area of s_v, clipped by relevant extreme coordinate of t_v, divided by area of s_v
Occlusion	compute 9 raycasts from v to s bounding box corners and center	return fraction of rays that intersect t

TABLE 3

Geometric methods to evaluate properties in Table 1, based on bounding boxes and ray casting.

Property	Mean Evaluation Cost (milliseconds)				
	rendering at 1000x750	rendering at 500x375	rendering at 80x60	rendering at 40x30	geom. eval. as in Table 3
Occlusion	261.68	63.10	9.61	8.96	0.1
Size	261.87	63.47	9.70	8.44	0.005

TABLE 4

Average cost to evaluate *Size* and *Occlusion* properties, through different methods, on one of the scenes of our experimental evaluation.

where a target is totally included in the image, and situations where it is partially included. For example, the same value for a *Size* property could be computed for a closer, partially included target, and for the same target, but farther and totally included, and these two situations clearly yield different visual compositions.

In order to tackle this problem, our approach always favors total inclusion of targets. To this purpose, before evaluating a property, we check how much its target(s) are included in the viewpoint view volume (by checking the bounding box corners coordinates in clip space), and, after property evaluation, we multiply the property satisfaction by the fraction of inclusion (with 0.0 being totally outside and 1.0 being totally inside), except for cases when total exclusion (e.g. *Size* equal to zero) is requested. In this way, compositions with partial inclusion of targets can still be partially satisfactory, i.e. not ruled out in the search process. The downside is that when one wants only a subpart of an object to be included, it has to be a distinct object in the scene, which the search process will try to include completely.

As the last column of Table 4 shows, evaluating a *Size* property with the proposed geometric method has an average cost of 0.005 milliseconds, while occlusion has a cost of 0.1 millisecond; this is respectively quicker than the equivalent rendering-based calculation at resolution 40x30 by four orders of magnitude (*Size*) and two orders of magnitude (*Occlusion*). This means that, accuracy considerations aside, a search process employing rendering-based methods is roughly expected to be slower by at least a couple of orders of magnitude than the same process employing the proposed geometric methods.

3.4 Evaluating a Viewpoint

The typical procedure for computing the satisfaction of a viewpoint is to evaluate each property, and then sum up the individual contributions. However, with many solvers one can safely interrupt a viewpoint evaluation when it is guaranteed that a certain satisfaction cannot be reached. For example, with PSO, a candidate viewpoint can influence the search process only if its satisfaction is greater than all previous evaluations of the same candidate - in cases when this is not true, by evaluating the candidate satisfaction we have just wasted time. Previous approaches (based also on other solvers than PSO) adopted therefore a lazy evaluation strategy, where properties are ordered by increasing cost of evaluation and evaluation of a candidate is aborted as soon as it is guaranteed that a certain satisfaction cannot be reached (where the threshold is dynamically established by the search process) [3], [6], thereby saving, when possible, the evaluation of costly properties (e.g., occlusions), without any effect on the search process. We build on that approach and extend it by:

- computing t_v for each target t mentioned by properties before actually evaluating any property. When a target is not seen by a viewpoint, the satisfaction of each property that mention it can be safely set to zero; moreover, as computing the same t_v can be required by more properties, its value is stored and then just retrieved when needed; to maximize the chance of stopping evaluation earlier, we order targets by decreasing contribution to the objective function, i.e. by $w_t = \sum w_i/|T_i|$, where T_i is the set of targets for each property p_i that mentions t;
- ordering properties evaluation by decreasing $w_i/cost_i$, where $cost_i$ is an estimate of the computational cost of evaluating property p_i (e.g., as in Table 4); in this way, properties that contribute most to the total satisfaction, and have similar cost, are evaluated first, with a greater chance of stopping evaluation earlier.

Our modifications to the lazy evaluation employed in [3], [6] are still conservative, i.e. a viewpoint evaluation is stopped only if there is a guarantee that the required threshold cannot be reached. In Algorithm 3.1, we describe our strategy. In the pseudo-code, P is the array of VC properties p_i, ordered by decreasing $w_i/cost_i$, T is the array of all properties' targets, ordered by decreasing contribution to the objective function, and *lazyLimit* is a satisfaction value which triggers the lazy condition: when we evaluate that the computed satisfaction cannot overcome *lazyLimit*, we stop and return -1, a failure flag, not considered as a satisfaction value.

Algorithm 3.1: EVALSAT($v, lazyLimit$)

```
$sat \leftarrow 0$
$satmax \leftarrow \sum w_i$
for each $t \in T$
  do
    project $t$ visible vertices
    if all vertices outside viewport
      then
        for each property $p_i$ mentioning $t$
          do
            $P \leftarrow P - p_i$
            $satmax \leftarrow satmax - w_i$
        remove each $p$ mentioning $t$ from $P$
        if $satmax < lazyLimit$
          then return $(-1)$
      else
        clip $t$ projected vertices
        compute $t_v$ area, and screen extents

for each $p \in P$
  do
    $s \leftarrow w_i * sat_p(eval_p(v, args_p))$
    $sat \leftarrow sat + s$
    $satmax \leftarrow satmax - s$
    if $satmax < lazyLimit$
      then return $(-1)$
return $(sat)$
```

The algorithm is structured in two phases. First, we compute t_v for all targets. Whenever a target is projected outside the viewport, we eliminate all properties that mention it from P, subtract their weights from the maximum achievable satisfaction, and check if we have triggered the lazy condition. In this way, all t_v computations that could be shared among properties are performed once, properties that refer to targets outside the viewport are not further considered, and, depending on the value of *lazyLimit*, we may stop evaluation earlier. In a second phase, we consider the remaining properties and evaluate them one by one, each time checking if the lazy condition has been triggered. For all properties but *Occlusion*, evaluation is computationally very simple, since we have already computed t_v for each target.

3.5 Solving VC problems

Our solver is based on the PSO approach [12], with a novel strategy for initializing candidates. PSO is a population based stochastic optimization technique, inspired by social behavior of flocking birds or schooling fishes. With standard PSO, the process is initialized with a population (swarm) of candidates (particles) and search for optimum is performed by flying the particles through the problem space "following" the current best one.

In our case, we use a swarm of N 8-dimensional particles, each one defining a viewpoint. Each particle p_i is described by:

- the particle's current *position* $\mathbf{x}_i = (x_{i_1}, x_{i_2}, \ldots, x_{i_8})$ (the eight values define a viewpoint's position, look-at point, roll angle, field of view);
- the particle's *velocity* $\mathbf{v}_i = (v_{i_1}, v_{i_2}, \ldots, v_{i_8})$;
- the particle's *best visited position* (as measured by the objective function *EvalSat*) $\mathbf{p}_i = (p_{i_1}, p_{i_2}, \ldots, p_{i_8})$, a memory of the best position ever visited during the search, together with the corresponding objective function value $best_i$.

The index of the particle that reached the global best visited position is denoted by g, that is, $g = \arg\max_{i=1,\ldots,N} best_i$.

At the beginning of the search, the particles position and velocities are initialized as described in the next Section. The search is performed as an iterative process which, at step n, and for each particle, evaluates $EvalSat$ on its position, and then modifies the velocity and position based on their values at step $n-1$, on each particle $\mathbf{p}$ value (its best visited position), and on the position of the best particle overall, i.e. $\mathbf{p_g}$, according to the following rules (superscripts denote the iteration number):

$$\mathbf{v}_i^n = \omega^{n-1}\mathbf{v}_i^{n-1} + c_1 r_1 \left(\mathbf{p}_i^{n-1} - \mathbf{x}_i^{n-1}\right) + c_2 r_2 \left(\mathbf{p}_g^{n-1} - \mathbf{x}_i^{n-1}\right)$$
$$\mathbf{x}_i^n = \mathbf{x}_i^{n-1} + \mathbf{v}_i^n \quad i = 1, 2, \ldots, N$$

Note that, according to these rules, $EvalSat$ can be computed for particle i setting $best_i$ as the *lazyLimit* argument. The values r_1 and r_2 are two uniformly distributed random numbers, whose purpose is to maintain population diversity. The constants c_1 and c_2 are called *cognitive* and *social* parameters, respectively describing a particle's trust in its own experience and in the best particle of the swarm. The value ω is an *inertial weight* and it establishes the influence of the search history on the current move; it linearly decreases, iteration after iteration, from an maximum value (ω_{init}) to a minimum one (ω_{end}). In general, it is known that the behavior of a PSO solver changes significantly depending on the values of these parameters, and finding suitable values for specific problems or classes of problems is the topic of several research papers. Particles can exit their domain during search: in this case, the objective function is set to a negative value without evaluating it.

Additionally, the PSO solver takes as input a maximum *time budget* T, and a *satisfaction threshold* $ST \in [0, 1]$. The process ends whenever: (i) any particle reports a satisfaction $sat > ST \cdot satmax$, where $satmax = \sum w_i$ is the maximum satisfaction ever obtainable by a particle, or (ii) the time elapsed since starting PSO is greater than the time budget. The stopping conditions are evaluated each time a new global optimum has been found (satisfaction threshold), and after each particle evaluation (time budget).

When the solver stops, the current best particle is returned as result.

3.5.1 Initializing Particles

Standard PSO initializes the particles with random position and velocities, since the algorithm has no knowledge about the search space before starting iterations. This strategy can have a negative impact on solving time, because random initialization is not guaranteed to produce particles that can quickly guide the search process towards an optimum. This problem is common to many stochastic optimization approaches.

As we discussed in Section 2, to tackle this problem, previous approaches have considered some kind of properties (typically, *Angle* and *Size*) as geometric operators that derive sub-regions of search space where a property can be satisfied [6], [5], [3]. By intersecting the derived sub-regions, one can derive a subset of search space where candidates are initialized and search can be somehow confined, however, with a considerable computational cost [5].

In an effort to find a less costly approach, instead of explicitly pruning the search space, we try to initialize each candidate in a position where at least one property is partially satisfied. We achieve this through an initialization process assigning a subset of the initial particles to each property (depending on its weight), and trying to initialize them in positions where the property can be better satisfied. Then, during search, we allow our candidates to explore all search space, and therefore do not limit a priori our solutions in the case of over-constrained problems.

We have determined methods that find viewpoint positions with a certain estimated satisfaction for *Size* and *Angle* properties. For example, for a *Size* property, one can roughly estimate the distance the viewpoint v should have from the target t for the *Size* of t_v to be a certain value by taking the bounding sphere of t and assuming it is centered on screen, and then using the well-known formula

$$\text{distance} = \frac{\text{target size}}{\text{target's projection size}} \cdot \frac{1}{\tan(\gamma/2)}$$

where the size is that of the bounding sphere.

For any distance, a viewpoint v on the sphere centered on t and with radius equal to the distance will (approximately) guarantee a certain *Size* of t_v. Considering a budget of particles to be assigned to a certain size property, since our *sat* function is a linear spline, we treat it as a probability density function, and use inverse transform sampling so that more particles will be initialized around the distances that give higher satisfaction. For an *Angle* property, it is straightforward to derive a similar procedure. The steps of the initialization process are the following:

1) we determine two sets of particles, *R* (random) and *NR* (non-random), such that: (i) $|R|$ + $|NR|$ = N (the total number of particles); (ii) $|R|$ = trunc(N * *r_part*), where *r_part* $\in [0, 1]$ controls the fraction of random particles we want to use;
2) for particles in *R*, the dimensions corresponding to the position of the viewpoint are randomly initialized in the search space;
3) for particles in *NR*, we consider the subset PS of properties for which we have an analytical method to find suitable viewpoint positions (currently, only *Size* and *Angle*). We partition *NR* into $|PS|$ subsets, where each subset is associated to a property, and the number of particles in the set depends on the weight of the property (properties with greater weight get more particles). For each subset and particle, the dimensions corresponding to the position of the viewpoint are set according to the strategy explained above;
4) for all particles, the dimensions corresponding to the look-at point of the viewpoint are randomly initialized inside the AABB of all problem targets;
5) for all particles, roll angle and FOV are randomly initialized in the search space.

The expected advantages with respect to purely random initialization are the following. First, setting the look-at point randomly inside the AABB of all targets ensures that (barring occlusions) initial candidates will frame at least some of the targets. Second, properties such as *Size* or *Angle* are very often used, as we typically want targets to be at least recognizable by the user; therefore, initializing candidates in areas where such properties are satisfied (at least to some degree) is likely much better than random initialization. Additionally, a percentage of particles can still be randomly initialized to promote exploration of the search space. In Section 4, we will experimentally determine an optimal value of *r_part* for subdividing particles into the *R* and *NR* subsets.

3.5.2 Implications of our viewpoint representation

With our viewpoint representation, a particle following the best one in the swarm is pushed to look towards the same look-at point of the best particle. With a representation using Euler angles, instead, a particle is pushed to look in the same direction as the best particle, which is not ideal if the two particles are far apart. This kind of decoupling helps the look-at points to drift towards the problem targets. However, in our case, when, for a particle, the look-at point gets very close to the viewpoint position, a small change in the position itself can correspond to a large change in how properties are satisfied. This is in theory a negative situation, since a smooth objective space is more favorable for solvers like PSO. However, due to the way we initialize particles, such situations are not likely to happen at the beginning of search; during search, even if this might happen for a particle, it would likely have little consequences on the overall search process, because PSO works with a population of candidates.

4 Experimental Evaluation

In this Section, we evaluate the effectiveness of our approach, and of its specific improvements over prior work. More specifically, we compare: (a) our approach, as described in this paper; (b) our approach, using a viewpoint representation with Euler angles (i.e., 7 components instead of 8); (c) our approach, using traditional lazy evaluation with properties ordered by decreasing cost instead of the strategy described in Section 3.4; (d) our approach, using random initialization of particles instead of the strategy described in Section 3.5.1; (e) a previous PSO-based approach by Burelli et al. [6].

For each variant, we have performed a specific parameters tuning phase to determine optimal values for the solver parameters. As described in the previous Section, the parameters are the number of particles (N), the fraction of random particles to be used in the initialization (*r_part*), and the other PSO parameters c_1, c_2, ω_{init}, and ω_{end}. Hereinafter, we use the term *setup* to denote some choice of values for the solver parameters. Parameters tuning allows us to make fair comparisons between different solving strategies, since any particular choice of parameters could result in poor performances for some strategy. For reasons of space and interest, we report full results about the parameters tuning phase only for our approach with all the proposed improvements.

4.1 Experimental setup

The approach described in this paper has been implemented as a C++ library which can interface with various rendering engines. The library uses the well-known Bullet physics engine for ray casting and thus occlusion assessment. As such, the entire scene is required to be processed by Bullet to build a so-called collision world. This can be done during scene loading and updates to the collision world - when there are changes to the scene - have a minimal cost. Bullet can use various shapes for ray-casting: to maximize accuracy at the expense of performance, we employed the original scene meshes, instead of simpler bounding volumes. In the experimental evaluation, our library is interfaced with the Ogre open-source rendering engine. All the experiments were run on an Apple MacPro equipped with two 2.8GHz Xeon processors, and 2 GBytes of RAM; however only a single core was dedicated to each solver run.

Scene	Triangles	Objects	Scene AABB
city	474083	324	300 x 100 x 300 (vol: 9 x 10^6)
house	324182	50	120 x 23 x 100 (vol: 276 x 10^3)
rooms	110474	240	13.9 x 3.0 x 21.8 (vol: 909.06)

TABLE 5

Complexity measures about the tested scenes.

4.1.1 Scenes and VC problems

We have employed three scenes, shown in Figure 1, respectively modeling a part of a city (*city*), a large house with several rooms and its exterior surroundings (*house*), and a building floor with 8 rooms, a corridor, and some humanoid characters (*rooms*). They differ by size (both in terms of polygons and scene volume, see Table 5), spatial layout, (all with several objects and potential occluders) and are meant to cover exterior and interior settings.

Over these scenes, we defined 15 VC problems to cover various typical situations, varying in the number of targets and properties. The simplest problem requires to frame one target at a specific size and angle, with no occluded parts (3 properties); more complex ones involve multiple targets (up to 4) with various desired sizes, angles, levels of occlusion and with additional framing or relative positioning requirements (the most complex problem involving properties on 4 different targets). For space reasons, problems and scenes are fully described in the paper supplemental material.

For all problems, we set a satisfaction threshold of 1.0 (i.e., stop only if a perfectly good solution is found). Out of the 15 problems, 7 were purposely built to be not satisfiable, i.e. it is not possible to find a viewpoint yielding full satisfaction, meaning that the search process will use all available time. Since in real applications one cannot always be sure that a VC problem will be satisfiable, it is important to evaluate the behavior of a VC approach also in these situations.

In every VC problem, the domain of possible viewpoints, i.e. the solver's search space, was set up as follows: position and look-at point ranges to the AABB of the scene (reported in table 5), roll angle set to zero (which constrains the solver to provide shots aligned with the natural horizon) and fixed field of view. The last two requirements are typical in interactive 3D applications, and practically remove 2 out of 8 dimensions from the search process; the position and look-at point domains, instead, are purposely set to a worst case, in which we don't give the solver any suggestion on where the solution viewpoint could be located in the scene. Finally, we used AABBs to compute targets properties.

Given that PSO is a randomized algorithm, we solved each problem 20 times to provide a more reliable estimate of the measured statistics. For each run, we logged the satisfaction of the computed viewpoint, the triggered PSO stopping condition, and the number of performed viewpoint evaluations.

4.1.2 Time budgets

To cover a range of 3D applications with different time constraints, we identified 6 time budgets (hereinafter, denoted by T): 5, 10, 20, 40, 100, and 200 milliseconds. For instance, a time budget of 5-20 ms allows one, roughly, to compute a new viewpoint for each frame, while a time budget between 40 and 200 ms could be suitable when a new viewpoint has to be computed from every few frames to a

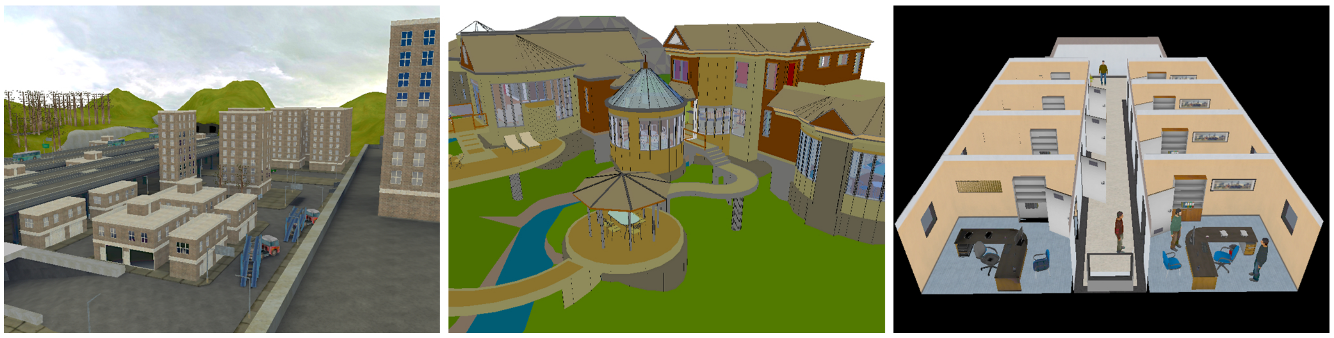


Fig. 1. Scenes used in the experimental evaluation: from left to right, *city*, *house*, and *rooms*.

few times per second.

4.1.3 Parameters Tuning

The approach we use for finding good solver parameter values is a variation of the popular parameters tuning algorithm *F-Race* [30], which is based on the Friedman rank sum test. F-Race starts with the so-called *full-factorial*, i.e. the Cartesian product of all the possible parameters domains and, at each iteration, tries each setup against a single *instance* (in our case, a $(problem, repetition)$ pair). After each iteration, the setups are ranked from the worst (high rank) to the best (low rank), and a Friedman post-hoc analysis is used to prune the setups that are inferior with statistical significance. The process stops when there are no more instances or one setup is left. Since our set of instances is not that large, we decided to apply the Friedman rank sum test directly on the whole set of experiments (the full-factorial against all instances), to obtain a more reliable parameter optimization process. Therefore, for each instance, the setups are ranked from best to worst satisfaction, then the rankings for each setup are summed up to derive a measure of how good the setup is across the set of problems.

To generate the full-factorial, one must first decide how to sample the parameters space. By looking at PSO-related literature and by running a preliminary set of experiments to identify suitable parameters ranges, we derived the possible values for each parameter shown in Table 6. These settings yield a full-factorial of 18720 setups, which, considering the 15 VC problems and 20 runs for each, translates to 5148000 runs per time budget, for a total of about 33 million runs of PSO for each tested solver variant.

4.2 Parameters Tuning Outcome

For each time budget, the Friedman test found a significant effect ($\alpha = 0.05$) of parameters setup on the computed satisfaction. Table 7 illustrates the main findings of the Friedman post-hoc analysis, per time budget, reporting (i) the number of *winning* setups, i.e. with significantly lower ranking than the others (i.e., higher satisfaction across the whole set of problems) and with no statistically significant differences between them; (ii) the best setup; (iii) parameter values used in the winning setups; (iv) median number of viewpoint evaluations performed, divided into full (i.e. when all problem properties were evaluated) and partial (i.e. when *lazyLimit* was hit during viewpoint evalua-

Name	Meaning	Values
N	number of particles in PSO	$[20, 30, 40, 60, 80, 100, 130, 160, 200, 240, 290, 340, 380]$
r_part	fraction of randomly initialized particles	$[0.0, 0.33, 0.66, 1.0]$
c_1	PSO cognitive parameter	$[0.0, 0.5, 1.0, 1.5, 2.0, 2.5]$
c_2	PSO social parameter	$[0.5, 1.0, 1.5, 2.0, 2.5]$
ω_{init}	PSO initial inertia weight	$[0.5, 1.0, 1.5, 2.0]$
ω_{end}	PSO final inertia weight	$[0.0, 0.5, 1.0]$

TABLE 6

Possible solver parameters values in the experimental evaluation.

tion). Overall, the winning configurations are about 0.008% of the full-factorial.

While the information reported in Table 7 is, of course, only guaranteed to be valid for the problems we tested, the diversity of scenes and problems in our experimental evaluation should make winning setups quite robust on other problems and scenes.

The number of particles in the winning setups (third column of Table 7) is towards low values in the considered range for $T = 5$ (up to 30 particles), $T = 10$ (up to 40 particles), and $T = 20, 40$ (up to 60 particles). As T increases, using any number of particles becomes effective; note, however, that the number of particles of the single best configuration for longest Ts is significantly lower than the maximum value of the allowed range, namely 130 for $T = 100$ and 200 for $T = 200$. Also, it is interesting to observe that, while we expected low-particles setups to be more successful on the shorter time budgets (where there is no time to handle a higher number of particles), winning setups with low number of particles are present also for longer time budgets.

Winning setups fully exploit our initialization strategy on shorter time budgets: all winning setups for $T = 5$ use $r_{part} = 0$ (i.e., all particles are initialized in a non-random fashion), while, for $T = 10$ and $T = 20$, it is also effective to randomly initialize up to one third of the particles, and up to two thirds for $T = 40$ and $T = 100$. For $T = 200$, there are winning setups that initialize randomly all the particles, probably trading initial higher satisfaction for a wider exploration of search space.

The social parameter c_2 is greater or equal to 1 for the winning configurations on most time budgets (except $T = 100$). There are no restrictions to the cognitive parameter c_1, which suggests that it has no particular influence on performances. The inertia weight parameters' values play an effect for $T \geq 10$ milliseconds: the effective range for ω_{end} is $[0.0, 0.5]$, while for ω_{init} values smaller or equal than 1.0 are preferable.

4.3 Performances Results

Figure 2 presents various box plots that show the distribution of normalized satisfaction values of the found viewpoints in the various time budgets, obtained by: our approach, with parameters tuned (violet color); our approach, before parameters tuning, i.e. using all parameter values in Table 6 (green color); our approach, using a viewpoint

T (ms)	How many	Best $(N, r\_part, c_1, c_2, \omega_{init}, \omega_{end})$	Restriction on parameters values	Median evaluations full	Median evaluations partial
5	102	20, 0, 2, 1.5, 1.5, 0	$N \leq 30$, $r_{part} = 0$, $c_2 \geq 1$, $\omega_{end} \leq 0.5$	60	13
10	167	30, 0, 2.5, 1.5, 0.5, 0	$N \leq 40$, $r_{part} \leq 0.33$, $c_2 \geq 1$, $\omega_{end} \leq 0.5$	108	43
20	117	30, 0, 2, 2, 0.5, 0	$N \leq 60$, $r_{part} \leq 0.33$, $c_2 \geq 1$, $\omega_{init} \leq 1.5$, $\omega_{end} \leq 0.5$	176	135
40	144	30, 0, 1.5, 2, 0.5, 0.5	$N \leq 60$, $r_{part} \leq 0.66$, $c_2 \geq 1$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	373	294
100	173	130, 0, 2, 2, 0.5, 0	$r_{part} \leq 0.66$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	707	625
200	218	200, 0, 1.5, 2.5, 0.5, 0.5	$c_2 \geq 1$, $\omega_{init} \leq 1$, $\omega_{end} \leq 0.5$	1204	1206

TABLE 7

Winning (statistically equivalently good) setups found by Friedman post-hoc analysis, per time budget.

representation with Euler angles, with parameters tuned (cyan color); our approach, using traditional lazy evaluation, with parameters tuned (blue color, labeled Non-smart evaluation in the figure); our approach, using random initialization of particles, with parameters tuned (yellow color); the PSO-based approach described in [6] (red color). Our supplementary material shows similar plots for each problem and scene. Note that the aim of tuning is to find setups that work better across all the problems. However, there is no guarantee (and typically it is not the case) that the winning configurations will work always better on every problem, and some variants might work better in some problem than the one that is better in general.

To compute normalized satisfaction, we divided the optimal viewpoint satisfaction by the highest one reachable in the problem at hand. For satisfiable problems, this is equal to the sum of the weights of the properties; for unsatisfiable problems, it was set to the highest satisfaction we could obtain by solving the problem 1000 times with $T = 500$ milliseconds. Note also that, for each problem and configuration, the data below the 5^{th} percentile and above the 95^{th} have been *winsorized*, in order to reduce the effect of spurious outliers.

It is important to point out that, since the normalized satisfaction is in the $[0.0, 1.0]$ range, small differences in the order of tenths can, for problems with several properties, result in one property being or not satisfied at all, with a noticeable visual difference in the final result. To make an example, figure 3 displays three different solutions to the same VC problem in the *rooms* scene, respectively obtained with $T = 10$, $T = 40$ and $T = 200$. The problem consists in framing three characters, such as the farthest one in all screenshots is seen from the front and with an eye level angle shot, unoccluded for at least 80%, and with height equal to 60% of the frame height; the other two should be unoccluded, with height equal to 60% of the frame height, and with an eye level or slightly high shot angle. Moreover, *Occlusion* properties have a double weight than others. The solutions we have displayed in Figure 3 are close to the median satisfaction we could obtain in the specific problem (which is unsatisfiable; the maximum satisfaction we could reach is 0.90 with $T = 500$). As we can see, with $T = 10$, the found viewpoint (with normalized satisfaction equal to 0.71) is not able

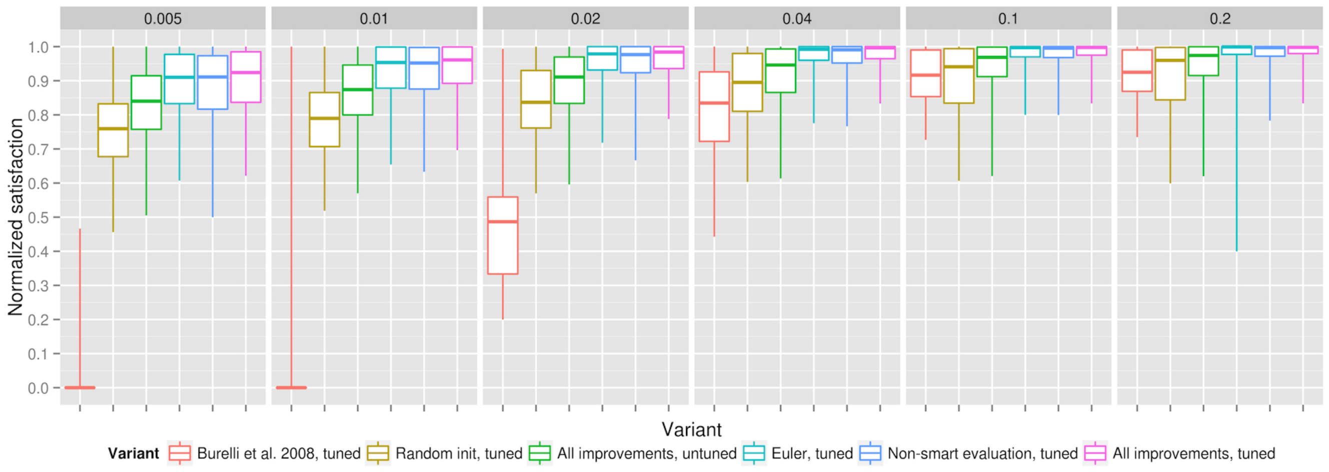


Fig. 2. Distribution of normalized satisfaction obtained in all problems for each tested variant.

to entirely frame the two closer characters (with the closest less than half framed). With $T = 40$, the found viewpoint (with normalized satisfaction equal to 0.83) better frames the three characters, but the farthest one is too occluded; With $T = 200$, the found viewpoint (with normalized satisfaction equal to 0.92) is finally able to frame all three characters, with occlusions that are minimally beyond the requirements.

4.3.1 *Parameters tuning effect*

How much does our approach gain by choosing a winning setup, instead of a random one in the full factorial? The answer is in the difference between the green (our approach, parameters untuned) and the violet (our approach, parameters tuned) box plots in Figure 2. By looking at them, it is clear that parameters tuning increases all the quartiles for every T. Also, the interquantile range, a measure of variance, is always smaller for winning setups, suggesting a more robust behavior. An interesting sweet spot is at $T = 40$ milliseconds, where the median satisfaction is very close to 1.0 despite the relatively short time budget. Interesting results, however, can be obtained also at $T = 20$, where the median satisfaction is about 0.98. Note that the green box plot includes data from the full factorial, i.e. all considered combinations of parameters, including winning ones, meaning that an unfortunate choice of parameters (e.g. choosing 100 particles with $T \leq 40$) could cause much worse performances than those shown in the box plot.

The statistical significance of this result is implied by the nature of the tuning process (the Friedman test).

4.3.2 *Initialization Strategy*

We isolated the experiments where $r_{part} = 1$ (i.e., random initialization of all particles), and performed parameter tuning to derive the winning setups in this particular case. The yellow (random initialization, parameters tuned) and violet (our approach, parameters tuned) box plots in Figure 2 show the advantage of our initialization strategy throughout all the time budgets. This was confirmed by a paired one-sided Wilcoxon's signed-

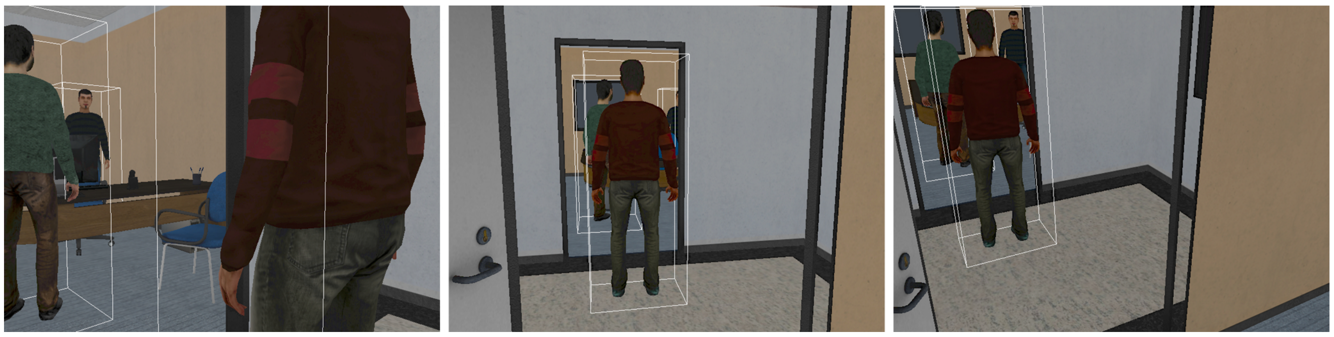


Fig. 3. Three solutions to the same problem with different time budgets: from left to right, 10, 40 and 200 milliseconds.

rank test, with $p \leq 0.05$ for all time budgets. Note that random initialization, in our tests, was not entirely random, as the look at point was still initialized in the AABB of the targets: without this, its results would have been considerably worse.

4.3.3 Evaluation Strategy

Comparing the performances of our approach with the version of it that uses a standard viewpoint lazy evaluation approach (both after parameter tuning: respectively, violet and blue box plots), we can see that in all time budgets, the quartiles for our approach are better. With more than $T = 100$, the two variants get closer in terms of satisfaction, however the first quartile and minimum for our approach is always better.

We ran a paired one-sided Wilcoxon's signed-rank test over the two distributions to assess the statistical significance of this result, which was confirmed with $p \leq 0.05$ for all time budgets.

4.3.4 Viewpoint Representation

The advantage of our 8-values viewpoint representation over a 7-values one using Euler angles can be seen by comparing the cyan (Euler variant) and violet plots (look-at variant) in Figure 2. From the plot it is clear that the 8-values representation is always slightly better. The Wilcoxon's test confirms that the result is statistically significant with $p \leq 0.05$ for all time budgets.

4.3.5 Comparison with Burelli et al. [6]

The PSO-based approach described in [6] is an hybrid constraint/optimization approach which, as discussed in Sections 2 and 3.5.1, uses geometric operators to derive a promising subset of search space where particles are confined. Moreover, the approach by Burelli et al. uses Euler angles to represent orientation, computes the screen representation using bounding spheres, and the same occlusion evaluation mechanism as the one described in this paper (for a fair comparison, both approaches used in the tests the same Bullet library and collision world). We slightly adapted our problems to Burelli et al. properties definition, so that both approaches could reach, with sufficient time at disposal, very close satisfaction values.

As one can see by looking at the red box plots, the approach by Burelli et al. is largely ineffective for $T = 5, 10, 20$, because its particles initialization mechanism takes almost the whole time budget, with the result that PSO cannot perform enough iterations to find good viewpoints. With longer time budgets, the approach by Burelli et al. gets closer to our approach, but remains significantly inferior, as confirmed by the Wilcoxon's signed-rank test with $p \leq 0.05$ for all time budgets.

5 Discussion and Conclusions

While the lack of reference problems still stands as common issue to all papers in this field that somehow limits the generality of our results, the ample experimental activity described in this paper allows one to draw some interesting conclusions and directions for future work.

First, as it is generally known in the field of optimization but was never quantified in VC papers, solver parameters tuning is crucial to performances, as it allows for significantly better results (both in terms of numerical satisfaction, and visually). A consequence is that, when testing different solvers, parameters tuning is necessary for a fair comparison. Of course, tuning on a set of problems does not guarantee that the derived parameters will be optimal also on other problems: however, in our case, the fact that the set of parameters we found could improve performances on each of the 15 problems (which differed in scene, properties, and number of targets) make us quite confident on their robustness to other situations. Plus, we found that some winning setups work well on almost all the range of considered time budgets. On the other hand, parameters tuning is a delicate operation as any change in the solver or viewpoint evaluation procedure (e.g., a less costly way to evaluate occlusions), or even in the specific machine computing capabilities, would probably entail a different set of winning setups. Moreover, considering the full factorial of setups makes parameters tuning a very time-consuming activity which combinatorially explodes as problems are added. Therefore, a direction for future work is to investigate quicker ways to derive optimal solver parameters. One possibility is to adopt the F-Race methodology of pruning inferior setups after each problem is considered. A more interesting direction would be to further investigate our experimental data (and possibly gather more) to see if there is a relation between a machine computing capabilities (in terms of viewpoint satisfaction evaluation) and a target time budget, with suitable values of solver parameters.

Our experimental evaluation shows also that proper candidates initialization is fundamental to find viewpoints as the allowed time decreases, and that a ”soft” strategy like ours, where we initialize candidates by considering one property at a time and do not prune search space, is more effective in time-constrained situations than other papers' solution of cutting search space through geometric operators, as the latter operation cost can substantially hinder search (in our tests, with time budgets equal or less to 40 milliseconds).

Our way of formalizing visual properties has two benefits. First, it highlights computations that are common between properties. This has allowed us to define a viewpoint evaluation strategy which was demonstrated to be experimentally better than traditional lazy evaluation in our tests. Second, our definition explicitly decouples the property semantic from the kind of abstractions that are used in computing a property value, which could be based on geometric calculations or off-screen rendering.

In particular, the rendering-based method, while not nearly adequately efficient for solving problems (at least for interactive applications), allows one to evaluate the accuracy of solutions found with faster, more approximate methods, which is another aspect in evaluating a VC approach. A detailed discussion on accuracy is beyond the scope of this paper, also because deviations from pixel-accurate measures do not take into account perceptual and more high level issues, such as recognizability of objects. Coming up with property evaluation methods that are cheap while not penalizing accuracy too much is another direction to pursue for making VC approaches more effective. For example, occlusion evaluations (which accounted for about 70% of the search time in our tests), could be improved by determining, given the size of occluders, the ideal number of ray casts in specific scenes.

Finally, by making our source code and problems publicly available, we hope to provide a step towards the construction of a VC benchmark for the evaluation and comparison of approaches, and ultimately lead to the adoption of more sophisticated camera control approaches in real world applications.

Acknowledgments

Our research has been partially supported by the PRIN 2010 project 2010BMCKBS_013.

References

[1] P. Olivier, N. Halper, J. H. Pickering, and P. Luna, "Visual Composition as Optimisation," in *Artificial Intelligence and Simulation of Behaviour*, 1999, pp. 22–30.

[2] N. Halper and P. Olivier, "CamPlan: A Camera Planning Agent," in *Smart Graphics 2000 AAAI Spring Symposium*. AAAI Press, 2000, pp. 92–100.

[3] W. H. Bares, S. McDermott, C. Boudreaux, and S. Thainimit, "Virtual 3D camera composition from frame constraints," in *Proceedings of the eighth ACM international conference on Multimedia*. ACM Press, 2000, pp. 177–186.

[4] J. H. Pickering and P. Olivier, "Declarative camera planning roles and requirements," in *Proceedings of the 3rd international conference on Smart graphics*, ser. SG'03. Springer-Verlag, 2003, pp. 182–191.

[5] M. Christie and J.-M. Normand, "A semantic space partitioning approach to virtual camera composition," *Computer Graphics Forum*, vol. 24, no. 3, pp. 247–256, 2005.

[6] P. Burelli, L. Di Gaspero, A. Ermetici, and R. Ranon, "Virtual Camera Composition with Particle Swarm Optimization," in *SG '08: Proceedings of the 8th International Symposium on Smart Graphics*, vol. Lecture No, no. 5166. Springer-Verlag, 2008, pp. 130–141.

[7] R. Abdullah, M. Christie, G. Schofield, C. Lino, and P. Olivier, "Advanced Composition in Virtual Camera Control," in *Smart Graphics*, 2011, pp. 13–24.

[8] L. Chittaro, L. Ieronutti, and R. Ranon, "VEX-CMS : A tool to design virtual exhibitions and walkthroughs that integrates automatic camera control capabilities," in *Proceedings of Smart Graphics 2010*, 2010.

[9] C. Lino, M. Christie, R. Ranon, and W. Bares, "The director's lens: an intelligent assistant for virtual cinematography," in *Proceedings of the 19th ACM international conference on Multimedia - MM '11*. New York, New York, USA: ACM Press, Nov. 2011, pp. 323–332.

[10] C. Lino, M. Christie, F. Lamarche, G. Schofield, and P. Olivier, "A Real-time Cinematography System for Interactive 3D Environments," in *2010 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2010.

[11] P. Burelli and G. N. Yannakakis, "Combining Local and Global Optimisation for Virtual Camera Control," in *IEEE Conference on Computational Intelligence and Games*, 2010.

[12] R. C. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks 1995*, vol. 4, 1995, pp. 1942–1948.

[13] P.-p. Vázquez, M. Feixas, M. Sbert, and W. Heidrich, "Automatic View Selection Using Viewpoint Entropy and its Application to Image-Based Modelling," *Computer Graphics Forum*, vol. 22, no. 4, pp. 689–700, 2003.

[14] C. Andujar, P. Vasquez, and M. Fairen, "Way-finder: guided tours through complex walkthrough models," *Computer Graphics Forum*, vol. 23, no. 3, pp. 499–508, 2004.

[15] A. Secord, J. Lu, A. Finkelstein, M. Singh, and A. Nealen, "Perceptual models of viewpoint preference," *ACM Trans-*

actions on Graphics, vol. 30, no. 5, Oct. 2011.

[16] M. Christie, P. Olivier, and J.-M. Normand, “Camera Control in Computer Graphics,” *Computer Graphics Forum*, vol. 27, no. 8, pp. 2197–2218, Dec. 2008.

[17] W. H. Bares, “A Photographic Composition Assistant for Intelligent Virtual 3D Camera Systems,” in *SG ’06: Proceedings of the 6th International Symposium on Smart Graphics*, ser. Lecture Notes in Computer Science, vol. 4073. Springer-Verlag, 2006, pp. 172–183.

[18] F. Jardillier and E. Languénou, “Screen-Space Constraints for Camera Movements: the Virtual Cameraman,” *Computer Graphics Forum*, vol. 17, no. 3, pp. 175–186, Aug. 1998.

[19] W. H. Bares and J. C. Lester, “Intelligent multi-shot visualization interfaces for dynamic 3D worlds,” in *Proceedings of the 4th international conference on Intelligent user interfaces (IUI 1999)*. ACM Press, 1999, pp. 119–126.

[20] M. Christie, J. Normand, and P. Olivier, “Occlusion-free Camera Control for Multiple Targets,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 2012, pp. 59–64.

[21] R. Ranon, M. Christie, and T. Urli, “Accurately Measuring the Satisfaction of Visual Properties in Virtual Camera Control,” in *Proceedings of Smart Graphics 2010*, ser. Lecture Notes in Computer Science, vol. 6133. Springer-Verlag, 2010, pp. 91–102.

[22] J. Blinn, “Where Am I? What Am I Looking At?” *Computer Graphics and Applications, IEEE*, vol. 8, no. 4, pp. 76–81, 1988.

[23] S. M. Drucker and D. Zeltzer, “Intelligent camera control in a virtual environment,” in *Proceedings of Graphics Interface 94*, 1994, pp. 190–199.

[24] P. Burelli and G. N. Yannakakis, “Global Search for Occlusion Minimisation in Virtual Camera Control,” in *IEEE Congress on Evolutionary Computation*. Barcelona: IEEE, 2010.

[25] O. Bourne, A. Sattar, and S. Goodwin, “A Constraint-Based Autonomous 3D Camera System,” *Constraints*, vol. 13, no. 1, pp. 180 – 205, 2008.

[26] P. Burelli and A. Jhala, “Dynamic Artificial Potential Fields for Autonomous Camera Control,” in *Proceedings of AIIDE’ 09: The Fifth Conference on Artificial Intelligence and Interactive Digital Entertainment*, C. J. Darken and G. M. Youngblood, Eds. Palo Alto, California, USA: AAAI Press, 2009.

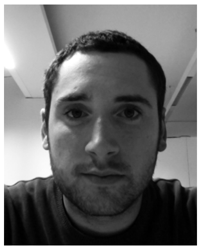
[27] C. Lino and M. Christie, “Efficient composition for virtual camera control,” in *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, ACM. ACM Press, 2012.

[28] M. Preuss, P. Burelli, and G. N. Yannakakis, “Diversified virtual camera composition,” in *Proceedings of the 2012t European conference on Applications of Evolutionary Computation*, ser. EvoApplications’12. Springer-Verlag, 2012, pp. 265–274.

[29] D. Schmalstieg and R. F. Tobler, “Real-time bounding box area computation,” Tech. Rep. TR-186-2-99-05, Jan. 1999.

[30] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp, “A racing algorithm for configuring metaheuristics,” in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, 2002, pp. 11–18.

Roberto Ranon is assistant professor in the Department of Mathematics and Computer Science of the University of Udine, where he is a member of the Human-Computer Interaction Laboratory (http://hcilab.uniud.it). His research interests are in Virtual Reality, 3D interfaces, and virtual camera control.



Tommaso Urli is a Ph.D. candidate at the Department of Electrical, Management and Mechanical Engineering of the University of Udine. His primary research field is the design, development and analysis of hybrid meta-heuristics for combinatorial optimization, with a bias towards timetabling and logistics problems. He is also interested in their application to computer graphics problems.