Automated Staging for Virtual Cinematography

Amaury Louarn, Marc Christie, Fabrice Lamarche

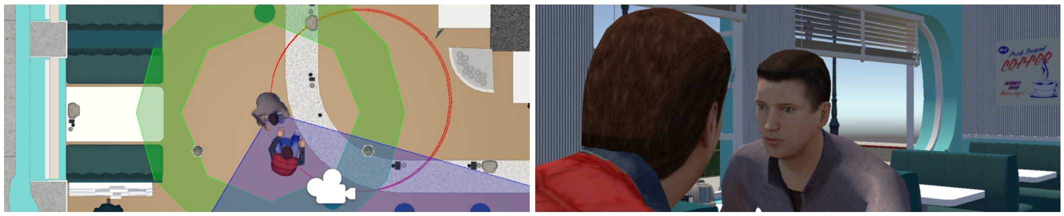
Automated Staging for Virtual Cinematography

Amaury Louarn
IRISA / INRIA
Rennes, France
amaury.louarn@inria.fr

Marc Christie
IRISA / INRIA
Rennes, France
marc.christie@irisa.fr

Fabrice Lamarche
IRISA / INRIA
Rennes, France
fabrice.lamarche@irisa.fr

Scene 1: Camera 1, CU on George front screencenter and Marty 3/4 backright screenleft. George and Marty are in chair and near bar.



(a) Scene specification **(b) Automated placement of camera and characters** **(c) Resulting shot enforcing the scene specification**

Figure 1: Automated staging for a simple scene, from a high-level language specification (a) to the resulting shot (c). Our system places both actors and camera in the scene. Three constraints are displayed in (b): *Close-up on George* (green), *George seen from the front* (blue), and *George screencenter, Marty screenleft* (red). The white camera is the location of the computed camera.

ABSTRACT

While the topic of virtual cinematography has essentially focused on the problem of computing the best viewpoint in a virtual environment given a number of objects placed beforehand, the question of how to place the objects in the environment with relation to the camera (referred to as *staging* in the film industry) has received little attention. This paper first proposes a staging language for both characters and cameras that extends existing cinematography languages with multiple cameras and character staging. Second, the paper proposes techniques to operationalize and solve staging specifications given a 3D virtual environment. The novelty holds in the idea of exploring how to position the characters and the cameras simultaneously while maintaining a number of spatial relationships specific to cinematography. We demonstrate the relevance of our approach through a number of simple and complex examples.

CCS CONCEPTS

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

KEYWORDS

Staging, Virtual Camera, Spatial reasoning, Constraint solving

ACM Reference format:
Amaury Louarn, Marc Christie, and Fabrice Lamarche. 2018. Automated Staging for Virtual Cinematography. In *Proceedings of MIG '18: Motion, Interaction and Games, Limassol, Cyprus, November 8–10, 2018 (MIG '18),* 10 pages.
https://doi.org/10.1145/3274247.3274500

MIG '18, November 8–10, 2018, Limassol, Cyprus

ACM ISBN 978-1-4503-6015-9/18/11…$15.00
https://doi.org/10.1145/3274247.3274500

1 INTRODUCTION

Cinematography is both an art and a technique; an *art* in that cinematography offers many ways of triggering emotions and playing with the spectators intellect without explicitly defining how; and a *technique* in that cinematography is constrained by physical possibilities such as locations, camera devices or lighting rigs as well as rules and conventions. The cinematographic art, in its general definition, encompasses the process of (i) staging, *i.e.* laying out entities in the scene to compose the image, (ii) lighting, *i.e.* how different light sources highlight the composition, and (iii) camerawork, *i.e.* how the camera is placed and moved according to the scene contents and author's intentions [Bordwell 2005].

With the advent of virtual 3D worlds, elements of this cinematographic art have been transposed to create more compelling and more immersive narrative experiences. Games, for example, have become over the last 15 years an increasingly cinematographic experience, reproducing distinctive film genres like war movies (*e.g.* the Battlefield game series by Electronic Arts). Games have also evolved, not just alternating between interactive play times and non interactive cutscenes, but merging both to create new interactive cinematographic experiences (*e.g.* Quantic Dream and Telltale games). However behind the scenes, the techniques employed in these games strongly rely on scripted approaches. All the different stagings, camerawork and lighting are pre-designed and played depending on the user interactions with marginal adaptations. In such interactive contexts, there is therefore a strong need for fully adaptive cinematography techniques, automatically deciding where to place characters, cameras and lights given the user interactions, alterations on the 3D scene and secondary characters involvement.

While techniques in computer graphics for placing, moving and editing cameras has received significant attention [Christie et al.

2008], a small number of contributions have addressed the problem of simultaneously planning the staging of characters (or more generally subjects) in the scene together with the staging of the cameras. Both are obviously strongly interdependent hence creating a complex problem. Most contributions consider the staging of the scene given as an input [Bares et al. 2000; Galvane et al. 2015a; Ranon and Urli 2014] for placing cameras, or enable limited changes in how the subjects are placed in the scene [He et al. 1996]. Some approaches based on mass-spring physical systems have been designed to stage subjects in theater plays [Talbot 2015] or for interactive stories [Kapadia et al. 2016], without considering the constraints specific to the camera placement and camera angles. A cinematography system was proposed in [Elson and Riedl 2007], that encompassed staging, blocking and filming, but only relied on a limited collection of combinations.

This paper proposes to address the problem of staging both the subjects and the cameras in a simultaneous process. To this end, we first propose the design of a film staging language that extends the Prose Storyboard Language (PSL) proposed in [Ronfard et al. 2015]. Our language expresses properties on the absolute or relative placement of subjects in a scene, together with film shooting properties on one or multiple cameras. We then propose a mean to operationalize elements of this language, and to perform a search for subjects and camera configurations satisfying the specifications. The underlying challenges stem from (i) the computational complexity of the process that requires solving elaborate spatial relations such as visibility or character framing and (ii) the relations that link cameras and entities making it difficult to place a camera without placing the subjects and conversely. Addressing the problem with classical techniques like numerical optimization would be computationally expensive, and would fail due to many local minima.

To address this problem, each property from the specification is expressed as a geometric operator that prunes the space of possible solutions (the search space) in a way similar to [Lino et al. 2010], but extending the approach to characters in addition to cameras. In particular, we propose a novel mean to prune the search space of possible cameras by using a 2D application of the Toric Space [Lino and Christie 2015], adapted to non-convex regions for the characters. The solving process mixes a *fixed-point* process that progressively reduces the search space using constraint propagation principles [Apt 1999], and an *elicitation* process that relies on sampling the domains of the variables and then re-applying the fixed-point process until a satisfactory assignment of values is done.

A range of experiments were conducted to demonstrate the relevance of our approach, computing results in under a minute. The approach can find multiple applications in the domain of game cutscenes, automatically staging the scene and the cameras given narrative constraints; for text-to-movie techniques; or as a foundation to an interactive staging tool for the film industry that more and more relies on 3D virtual environments to rehearse movies.

2 RELATED WORK

Camera placement in a virtual environment is a largely addressed problem [Christie et al. 2008]. Approaches are largely problem-dependent, while some cross-cutting issues remains (typically ensuring the visibility of a given target). This section will only address the placement of cameras following cinematographic properties.

First, different camera specification languages have been proposed [Bares et al. 2000; Ronfard et al. 2015], including the Prose Storyboard Language [Ronfard et al. 2015] (PSL). Designed in mind for both annotating movies and placing cameras, the language describes general absolute and relative cinematographic properties. Elements of shot composition (how characters are framed in the screen) can be specified: shot size (how large characters appear on the screen), character relative orientations (how the camera is placed in relation to the characters) and camera height relative to the scene (high, low or normal camera angle). This camera specification language has found some implementations [Galvane et al. 2015a] in the specific application to cinematic replays in games. Extensions of this language have also been proposed to characterize relations between different cameras over time, by defining Film Editing Patterns [Wu and Christie 2016]. Here we extend the PSL language and generalize it to multiple cameras and staging.

Besides the specification of such cinematographic properties, a large range of solving techniques has been explored to position a camera in a virtual 3D environment. A straightforward approach consists in expressing these properties as a composition of cost functions expressed on the camera parameters. General techniques such as constraint solving [Bares et al. 2000], numerical optimization [Drucker et al. 1992; Olivier et al. 1999; Ranon and Urli 2014], or hybrid numerical/geometric approaches have been proposed [Lino et al. 2010] to efficiently compute camera parameters.

In contrast, the problem of automatically staging actors has not been widely addressed. Talbot [Talbot 2015] has extensively explored such an issue in the context of theater plays, by relying on a dampened spring-mass system to express distance and angle properties between the characters composing the theater stage. This approach makes strong assumptions about stage layout and audience placement. Therefore, it cannot be easily generalized to complex movie scenes with multiple cameras.

More general schemes have also been proposed for object placement in scene layouts, based on spatial relations and spatial reasoning techniques [Clements and Battista 1992]. While we draw inspiration from these approaches, the specificity of camera and character placement requires dedicated techniques. Typically, camera placement often require to see characters at a specific position on the screen while character placement often needs to take into account the relations between characters, cameras, and props.

Finally, in the specific case of cinematographic systems, the machinima approach proposed in [Elson and Riedl 2007] provides a mechanism to express a film script (where there are camera and character specifications) into a placement of characters and cameras in a 3D environment. The approach relies on a library of camera placement and motions, coupled with a library of staging configurations. A combinatorial process using dynamic programming plans the optimal placement of characters and cameras over time by exploring possible combinations of cameras and character placement located in the 3D environment. The approach is however limited by the library, in which characters and cameras are placed at fixed locations, restraining the generalization of the method for arbitrary scenarios. Our approach is more general as it relies on a staging specification language, on the geometric analysis of the scene topology, and on a geometric solver to search for satisfactory assignments.

Figure 2: Specification of a shot from the movie "Tears of steel" (CC-BY 3.0 – Blender Foundation, 2012) using our language: "*Scene 1: Camera 1, MS on A 3/4 left screenright and B front screenleft. A is in doorway and on the left of B. B is close to cabinet and looks at A.*"

3 OVERVIEW

The input of our system is (i) a 3D model of a scene with a lightweight annotation that distinguishes the scene geometry (walls, ground, doors, creating a *floor map* of the environment) from the geometry of the props needed by characters (seats, bar, etc. creating a *subject map* of the environment) and (ii) a staging specification describing the layout of the cameras and characters composing the 3D scene. The output is a scene layout in which the characters and cameras are placed following the staging specification.

The process consists in first operationalizing the various elements of the specification into geometric pruning operators that remove the regions in which the properties are not satisfied. A solving process then alternates stages of pruning, using a propagation-based fixed-point algorithm, and elicitation, *i.e.* assigning values to some of the variables. The language is described in Section 4, its operationalization in Section 5, and the overall solving process in Section 6.

4 A STAGING SPECIFICATION LANGUAGE

Our specification language builds on the Prose Storyboard Language (PSL), and adds elements related to multiple cameras and staging of subjects (a subject being a character or an object). PSL is a context-free language that describes the main categories of screen events and subject actions, without describing any constraints related to their relative or absolute spatial positioning in scenes. This actually limits how the aspects of staging can be analyzed in movies. It also limits the capacity of the language to be used to perform actor staging, leading to ambiguities in how subjects should be placed in a virtual environment. In PSL, the camera staging process is composed, for each camera, of a set of *flat compositions* or *framings* described as:

```
<Composition> ::= <Size> on <Subject> [<Profile>] [<Screen>]
          {and <Subject> [<Profile>] [<Screen>]}*
```

The generic terminal *Size* is related to the frame size of the *Subject* in the scene (from extreme close ups to extreme long shots, see Fig. 3), *Screen* represents the position of the *Subject* on the screen and *Profile* his relative profile on the screen. The language is completed by transitions between framings. Terminals related to subjects are limited to the actions they perform, abstracted into *look at*, *move to*, *speak*, *use*, *cross*, *touch*, and *react*.

We propose to extend this language by proposing ways to specify character locations as well as spatial constraints between characters and objects, extending the expressiveness of the language and enhancing its applicability to problems of scene staging in virtual environments. In the following, we list the terms of our language and provide illustrations.

The main rules of our extended PSL language are as follow:

```
<Scene> ::= Scene <Scene number>: {<Shot> | <Specification>}*
          <Shot> ::= <Camera> <Composition>
        <Camera> ::= Camera <Camera number>,
 <Specification> ::= <Entity> {and <Entity>}* (is | are)
          <Constraint> {and <Constraint>}*.
```

Our extension of PSL adds the identification of scenes (<Scene> rule) and cameras when describing a composition (<Shot> rule). The identification of cameras enables the description of complex constraints involving cameras and actors. The scene identification provides a way to describe continuity constraints on character or camera placement and orientation. We also added three generic terminals to the language: *Entity*, *Object* and *Region*. An Entity is a generic term to represent either a subject as in the PSL language or a camera. The entities are the elements that will be computed by our process (*i.e.* the unknowns in our solving problem). An *Object* represents a piece of furniture or an object that is placed in the environment. A *Region* is a tagged zone in the environment used to specify location constraints for the entities. In addition, we add the term *element*, refering to either an entity, an object, or a region.

```
<Element> ::= <Entity> | <Object> | <Region>
```

The *Constraint* rule specifies the different spatial relations that can be expressed between elements (*i.e.* entities, objects or regions). Our language considers the following constraints:

4.0.1 Distance between entities. Distance is a key specification for staging and can be expressed in multiple ways. At first, relative distance can be specified using the `close` keyword as in:

```
close to <Element>
```

This constraint specifies that the distance between an entity and an element is lower than a given threshold. While *close* is a relative and scale-dependent notion, in the specific context of film staging it specifies "within a range of 1 meter". The property finds applications such as `Camera 1 is close to A`. Distance can also be expressed in an absolute way, specifying a distance between entities, being under or above a given value.

```
at [most | least] <value> from <Element>
```

4.0.2 Orientation. Orientation for staging defines how entities are placed in relation with one another (see Fig. 4). This constraint finds applications such as `A is facing B` or `A is 3/4 frontRight of B` for entities that have an implicit orientation.

4.0.3 Visibility. The visibility constraint is useful for identifying what should be shown or hidden, either to the camera or to an actor. More precisely, this property specifies that an element should be visible by an entity. It can also be expressed in its negative form when an element should not be visible. This constraint finds many applications such as defining a first framing where a subject is non-visible by the camera, followed by a second framing where the subject is visible.

```
[not] seeing <Element> {and <Element>}*
```

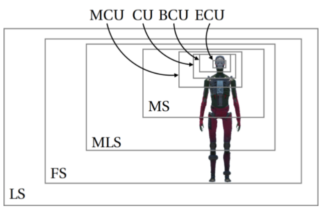


Figure 3: Several shot size.

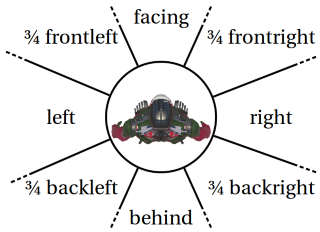


Figure 4: The 8 possible orientations, defining regions around entities.

4.0.4 In and Out relations. This constraint specifies that an entity should be inside or outside a given region. A region is a geometric area either defined by the user, or by the surface of an object. By default all the entities are placed in the free area of the scene topology, *i.e.* the area not in collision with any scene geometry. The constraint can be used as `A is in the shop`.

`[in | out of] <Region>`

4.0.5 Identity between scenes. This property ensures that the position or the orientation of a subject is the same as in a previously computed scene. This allows to have a continuity of staging for some subjects across multiple scenes. The Fig. 2 displays an example of annotation with our language.

`in same ((position [and orientation]) | orientation) as in scene <Scene number>`

5 OPERATIONALIZATION

In order to operationalize these properties, we first need to define an appropriate level of abstraction of the environment and of the elements. Our objective is then to search where to place our entities (an entity being a *Camera* or a *Subject*) in the abstracted representation of the environment in a way that satisfies a given specification.

5.1 Abstracting the problem

Given the complexity of simultaneously addressing staging and cinematography in 3D (*cf.* Section 2), we make the strong hypothesis of performing all computations in 2D abstractions of 3D scenes. In that we follow hypotheses followed by [Elson and Riedl 2007; Lino et al. 2010; Talbot 2015]. Typically, this relieves us from the problem of 3D visibility computation in arbitrary 3D environments, for which exact or approximate techniques remain computationally expensive [Durand et al. 2002]. Besides, reasoning in 2D still addresses a large range of staging and cinematography problems.

First we define the notion of a *Potential Location-Rotation Set* (PLRS). A PLRS is a set of disjoint 2D areas in which an entity can potentially be positioned, coupled with a set of possible ranges of orientations by which an entity can be turned (see Fig. 5). Initially, to each entity of the scene is associated a PLRS representing the search space of the entity (the entity being a variable, and the PLRS its domain). In our implementation, each 2D area in a PLRS is represented by a possibly non-convex 2D polygon in which the entity can be positioned. In the following, we shall refer to PLS (Potential Location Set) when dealing with positions only and PRS (Potential Rotation Set) when dealing with orientations only. All cameras and subjects are initialized beforehand to a PLRS that

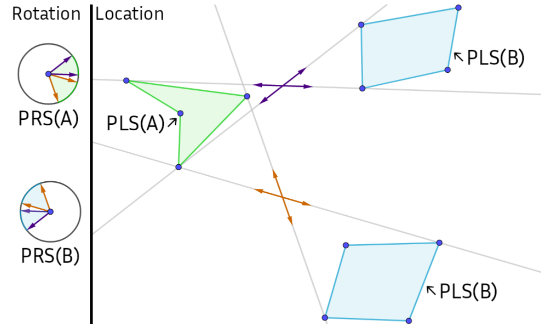


Figure 5: Potential Location-Rotation Set for two entities A (in green) and B (in blue). Note how the PLS of B is disjoint. Here the figure also illustrates how the `is facing` property reduces the range of orientations for entities A and B. The grey lines represent the cross-tangents between the PLS of A and each PLS of B. Their supporting vectors are color-coded in purple and orange, and are used to deduce the PRS of A and B thanks to the constraints `A is facing B` and `B is facing A`.

is equal to the bounding 2D region of the scene, together with a full range of potential orientations $[-\pi, \pi]$. Fig. 5 displays two different PLRS for two entities A and B. For illustration purposes, the orientations of entities A and B (the circles) have been pruned given the constraint that A is facing B and B is facing A.

The subjects and cameras have a fixed field of view respectively representing (i) the cone of vision set to the default value of 120 degrees and, (ii) the field of view of the camera set to the default value of 90 degrees. Each camera can however be associated a different field of view. The location and orientation of the subjects and cameras represent the unknowns of the problem. Computing a solution requires to compute an assignment of values to these unknowns that satisfy the specification. An assignment for an entity is a *Potential Location-Rotation Set* inside the initial PLRS reduced to a 2D point for the location and a scalar value for the orientation.

5.2 Treating the geometry

A prior stage consists in abstracting a 3D environment geometry (see example in Fig. 6). The provided environment geometry is split into two groups: the buildings (walls and anything that can obstruct visibility) and the objects (*e.g.* furniture).

Using only the buildings geometry, we compute its prismatic subdivision [Lamarche 2009] and its associated abstraction into a 2D topological map (a constrained 2D Delaunay triangulation). This map, that we call the *floor map*, identifies the possible locations where an entity (a subject or a camera) can be placed (C-free), locations where an entity cannot be placed (C-obstacle) as well as the constraints blocking camera visibility (the walls).

In a second step, we consider the objects (furniture, bar etc.). Each object may be informed to characterize which regions can the entities have relations with (*e.g.* an actor can *sit* on this part of the bench, an object can *stand* on that part of the table).

The floor map can then be exploited with information from the objects to create restrictions (*e.g.* the *front of the door* is the region inside the floor map that is located in front of the door).

Figure 6: An example 3D input mesh representing the environment, in which a distinction has been made between the buildings geometry (walls or geometry that obstructs the visibility) and the informed objects geometry (*e.g.* furniture like tables, stools and bar).

5.3 Geometric pruning operators

Given the specification language and the abstract representations, we designed geometric pruning operators. A pruning operator o_p relative to a constraint p of the language is an operator that prunes the locations and orientations inside a PLRS $\mathcal{X}$ by applying the operator o_p on each area $\mathbf{x} \in \mathcal{X}$, and then building the set union of these pruned locations and orientations in the following way:

$$o_p(\mathcal{X}) = \uplus_{\mathbf{x}\in\mathcal{X}} o_p^{\mathbf{x}}(\mathbf{x}) = \langle \cup_{\mathbf{x}\in\mathcal{X}} L(o_p^{\mathbf{x}}(\mathbf{x})), \cup_{\mathbf{x}\in\mathcal{X}} R(o_p^{\mathbf{x}}(\mathbf{x}))\rangle$$

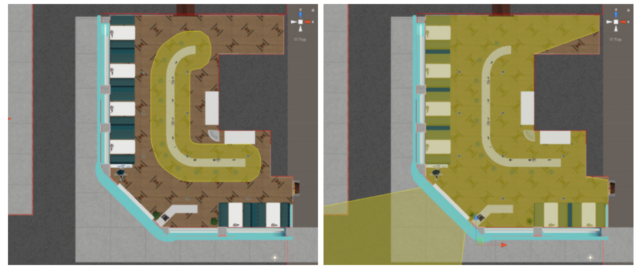
where $L(\mathbf{x})$ returns the potential Location region of $\mathbf{x}$ (respectively $R(\mathbf{x})$ the potential Rotation range), and $o_p^{\mathbf{x}}$ is the application of the pruning operator o_p on an area (rather than on a set of areas). In the following, we will detail how the o_p^x constraints are expressed as geometric operators over a single Location Rotation area $\mathbf{x}$.

5.3.1 Proximity between entities. The proximity operator $o_{prox}^{\mathbf{x}}$ between two entities e_1 and e_2 is a pruning operator that relies on computing the Minkowski sum [Wein et al. 2018] between a circle of radius r and the potential location set (PLS) of an entity (operation also known as offsetting by a radius r). There are two possibilities depending on the nature of the entities. The first case is when one entity is a fixed object and the other entity is an unknown (camera or a subject): the PLS of the unknown is intersected with the offsetting of the shape of the fixed entity by a radius d_m (d_m being the distance considered as near in this context, *i.e.* 1 meter). Fig. 7a displays the computed PLS of `Entity is close to bar`.

The second case is when both entities are unknowns (*e.g.* a camera and a subject or two subjects): the offsetting of the PLS of the second entity is computed and intersected with the PLS of the first entity. The proximity operator then performs the symmetrical operation, computing the offsetting of the PLS of the first entity and intersecting it with the PLS of the second one.

5.3.2 Distance between entities. The distance operator $o_{dist}^{\mathbf{x}}$ is based on the proximity operator $o_{prox}^{\mathbf{x}}$ but is generalized to prune the regions further than a given threshold value (property `at most`) or closer than a given threshold value (property `at least`).

5.3.3 Constraining an entity to a region. This corresponding pruning operator is straightforward and simply intersects the entity's PRLS with the specified region.



(a) Near Bar **(b) Bar visibility**

Figure 7: Example of regions illustrating the proximity property (`close to bar`) and visibility (`bar is visible`). The computed PLS areas are displayed in yellow.

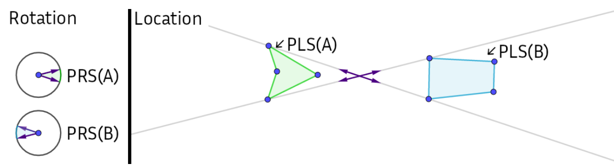


Figure 8: Pruning the PRS of two entities given their locations under the properties A faces B and B faces A.

5.3.4 An entity viewing an entity from an angle. The geometric operator $o_{ori}^{\mathbf{x}}$ computes a pruning of possible locations and orientations such that an entity A views the entity B from a given angle (facing, behind, ...). Orientations can be easily pruned by computing the partitioning tangents (see Fig. 8).

Given a region for an entity A, the locations of entity B such that B is (for example) facing A can be pruned by computing the wedge tangent to the region of A, with an angle equal to the range of orientations of entity A plus an offset orientation angle (corresponding to front, back, 34frontleft) and plus $[-\pi/8, \pi/8]$ to enable some variation around the offset angle. The resulting region for B is the intersected region of B with the wedge (see Fig. 9).

5.3.5 Visibility of an entity. The visibility operator $o_{vis}^{\mathbf{x}}$ reduces the PLRS of the camera with relation to an entity. The entity is either an object at a fixed location, or an entity with a PLRS. The computation relies on the triangular expansion technique [Bungiu et al. 2014] that, given a 2D point x, computes the areas such that any point in the computed area can see the point x. If the entity has a given area, then the area is uniformly sampled and the union of all areas is computed and becomes the new camera PLRS.

5.3.6 Visibility of an entity A from another entity B. If one of the two entities (here A) is an unknown (*i.e.* has a PLRS) and the other is an object, the PLRS can be pruned by (i) computing the area of visibility of the object using the technique for entity visibility, and (ii) intersecting it with the PLRS of A. If both entities are unknown, the only pruning that can be performed is on the potential orientations in the PLRS of B using the partitioning tangents.

5.3.7 Framing a subject on the screen. We here propose to focus on two types of framing properties: a camera framing one actor, and a camera framing two actors. These are two common framing settings for action and dialogues scenes. The constraint is expressed as a geometric operator $o_{frame}^{\mathbf{x}}$ over the PLRS.

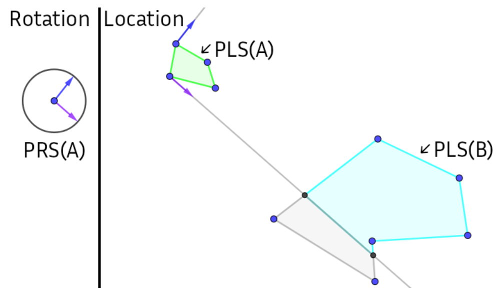


Figure 9: Pruning the PLS of B given the PLRS of A. In blue, the kept part, and in grey the pruned part.

Framing one actor is straightforward. In addition to other possible constraints, we first systematically use a `visibility` pruning operator $o^{\mathbf{x}}_{vis}$ to perform a geometric pruning of the environment and ensure the visibility of the actor by the camera. The orientation of the camera is then computed using the orientation operator $o^{\mathbf{x}}_{ori}$ to make the camera face the actor. If both the camera and the actors are restricted to a point in the environment, then the orientation ω of the camera is given by the orientation θ of the vector camera-actor and the onscreen position of the actor. More precisely,

$$\omega = \theta + tan^{-1}(\Delta_x tan(\Phi/2))$$

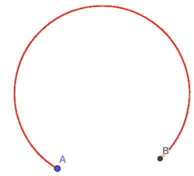
where Φ is the field of view of the camera and $\Delta_x \in [-1, 1]$ is the horizontal position of the actor on screen. Δ_x is arbitrarily mapped from the `<Screen>` constraint of the user input ; if no constraint is specified by the user, its default value is 0 (center of the screen).

If the camera and/or the actors domains are defined by a PLRS, we need to compute the two partitioning tangents that separate the two polygons (*i.e.* the tangents lines of both polygons where one is on one side of the line, and the other on the other side). The allowed orientation of the camera is then the interval of orientations between the support vectors of each tangent (see Fig. 8). When an entity is composed of several polygons, the allowed orientation is given by the union of each interval as illustrated in Fig. 5.

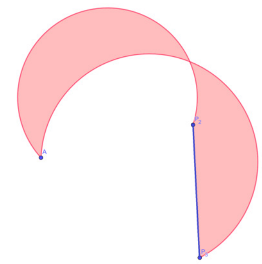
Framing two actors at the same time is more complex. We extend the toric space, proposed by Lino et al.[Lino and Christie 2012], to handle not only the case where two actors are fixed points, but also when they are defined by PLRS. We then propose a mean to prune the PLRS of the camera to remove every position that would not satisfy the framing constraints for at least one possible configuration of actors (bounded by their PLRS). In other words, for each possible configuration of actor, the resulting PLRS of the camera contains at least one camera configuration satisfyng the framing constraints.

When the PLRS is limited to a single point for both actors, the set of solutions is given by an arc-circle around the two actors as illustrated in Fig. 10a and demonstrated in [Lino and Christie 2012]. This means that, given the proper orientation, any camera position along this arc-circle satisfies user's constraints (specified through the *Screen* property and mapped to screen positions).

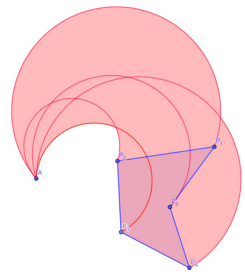
As illustrated in Fig. 10b, we first extend this solution to the case where one of the two actor's position is limited to a single fixed point while the other is bounded to a segment. The space of solutions is given by the area between the two solutions computed at each extremity of the segment. This method is further extended



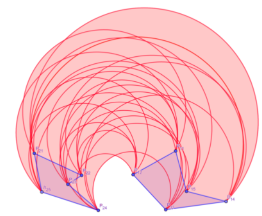
(a) Point to point



(b) Point to segment



(c) Point to polygon



(d) Polygon to polygon

Figure 10: Framing constraint for several topologies (point segment or polygon), for A and B. The solution set (in pink) is computed from the combination of all endpoints solutions (in red).

to the case PLRS by computing every combination of vertices on the polygons of the PLRS of each actor. The solution is given by the envelop computed from all endpoints solutions (see Fig. 10c and 10d). While this solution is intuitive, it was only experimentally tested – mathematical proof is left for future work.

6 SOLVING PROCESS

To compute a solution, we follow a two-step process. First, a pruning step using a fixed-point computation applies all the geometric operators by propagating the changes in order to reduce the PLRS of each entity. Then, a progressive sampling step is used to solve the staging configuration.

6.1 Fixed-point computation

This step consists in applying the sequence of *geometric* pruning operators from the constraints. These operators will act as pruning operators on the domain of the unknowns, *i.e.* removing inconsistent regions from $\mathbb{X}$ where $\mathbb{X}$ is a Cartesian product of PLRS (one PLRS per entity to solve). We rely on a fixed point computation by propagating the changes through the network of constraints [Apt 1999] (see Alg. 1), each time the domain of a variable is updated, the pruning operators that share this variable are re-applied. The remaining regions are such that there possibly exists a configuration in the reduced PLRS regions that satisfies the properties. The process stops when no more changes occur on the domains.

If the fixed-point computation results in an empty set, the problem is proven to be over-constrained, thus unsolvable. At this point, we report to the user the constraint that created the empty intersection, but we cannot report the exact set of problematic constraints.

Algorithm 1 Fixed point computation. Notations: C is a set of constraints, S is a set of fixed entities (entities which location/orientation is already solved) with their associated value, and $\mathbf{X}$ is the domain of the entities (a Cartesian product of PLRS).

function FIXEDPOINTCOMPUTATION($C, S, \mathbf{X}$)
    $C' \leftarrow \{c \in C \mid c.entity \notin S\}$
    while $C' \neq \emptyset$ **do**
        $c \leftarrow$ head(C')
        $\mathbf{X}' \leftarrow \mathbf{X}$
        $\mathbf{X} \leftarrow o_c(\mathbf{X}_{c.entity})$ ▷ pruning domain of $c.entity$
        if $\mathbf{X}' \neq \mathbf{X}$ **then** ▷ if pruning has occurred
            $C' \leftarrow C' \cup$ constraintsDependingOn($C, c.entity$)
        $C' \leftarrow C' \setminus c$
    return $\mathbf{X}$

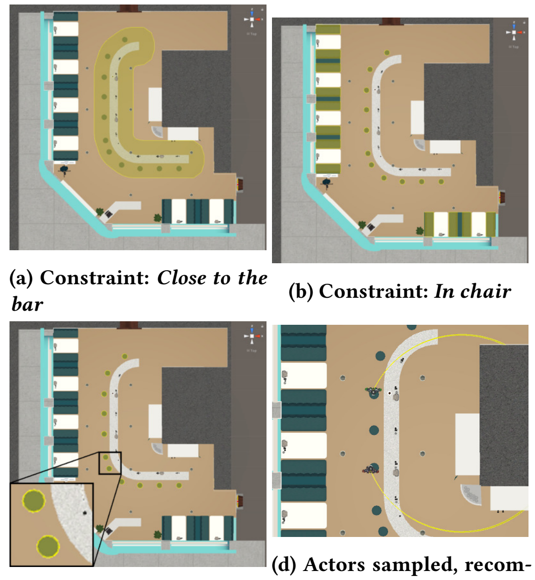
6.2 Progressive sampling

The fixed point computation reduces the domains of the entities, yet a sampling (or elicitation) process is required to compute exact solutions rather than areas of possible solutions. We rely on a progressive sampling technique where values are assigned to variables in a specific order and the constraints are propagated to reduce the search space of the other variables (see Alg. 2). By design, the presented algorithm is probabilistic complete, which means that if the number of trials increases without bound, the probability of not finding a solution if it exists approaches zero.

When discretizing the entities, the order in which they are sampled is important. Indeed, the first entity to be assigned a value will be decisive for the rest of the entities. We therefore propose the following heuristic. As a first step, we discretize the position

Algorithm 2 Solving algorithm. Notations: E is a set of entities, C is a set of constraints, i_{max} is the maximum number of trials, U is an ordered list of position or orientation variables (the unknowns) and $\mathbf{X}$ is the domain of the entities. S represents the set of entities for which location/orientation has been fixed.

function SOLVE($E, C, i_{max}, \mathbf{X}$)
    $U \leftarrow$ orderingHeuristic(E), $S \leftarrow \emptyset$, $i \leftarrow 0$
    while $i < i_{max}$ **do**
        $S \leftarrow$ SAMPLE($U, C, S, \mathcal{X}$)
        if $S \neq \emptyset$ **then**
            return S
        $i{+}{+}$
    return $\emptyset$
function SAMPLE($U, C, S, \mathbf{X}$)
    if $U = \emptyset$ **then**
        return S
    else
        $u \leftarrow$ head(U)
        $x_u \leftarrow$ samplePLRS($u, \mathbf{X}_u$)
        $S_{new} \leftarrow S \cup \{u \leftarrow x_u\}$
        $\mathbf{X}$ = FIXEDPOINTCOMPUTATION($C, S_{new}, \mathbf{X}$)
        if $\{\exists x \in \mathbf{X} | x = \emptyset\}$ **then**
            return $\emptyset$
        return SAMPLE($U \setminus u, C, S_{new}, \mathbf{X}$)



(a) Constraint: ***Close to the bar***

(b) Constraint: ***In chair***

(c) Intersection of ***Close to the bar*** **and** ***In chair***

(d) Actors sampled, recomputation of the camera constraints

Figure 11: A few computation steps for Scenario 1

of entities and then the orientation of entities. This order is based on the fact that a decision on an actor's position might strongly influence other entities (both in position and orientation), while a decision on an actor's orientation has less impact. Then, the entities are separated in two groups: actors first and cameras second. While actors can be placed without prior knowledge on the camera position, the inverse (placing cameras without knowing where are the actors) is much more difficult. Last, the entities are ordered based on the area of their PLRS (smallest area first). This is a classical strategy in constraint solving to treat the smallest domains first. In order to leverage the importance of the evaluation order during the sampling of one solution, each time an unsolvable situation is met, we discard intermediate results and restart a fresh sampling process.

7 RESULTS

In order to validate the proposed language as a staging description language usable for movie generation, we propose some scenarios based on real movie scenarios. First, we analyze a simple scenario, and unfold the stages of the solving process. We then show results obtained on more complex scenarios.

All computations were performed on a Dell precision 7520 laptop with a 3.10 GHz Intel® core™ i7-7920HQ and 32 Gb of RAM.

7.1 Stages of the solving process

The computation of a solution starts by expressing the geometrical operators used by the scenario description (see Scenario 1).

- A is close to the bar.
- B is close to the bar.
- A is in a chair.
- B is in a chair.
- B is at most 10m from A.
- A is facing B.
- B is facing A.
- The camera frames both A and B, A is on the left and B is on the right.

Then, the pruning operators are applied using the fixedPoint-Computation function (Alg. 1): first, *Close to the bar* (Fig. 11a); second, *In chair* (Fig. 11b). As A and B are targets of both of these properties, their PLRS is at this point the intersection of the two properties (Fig. 11c).

The remaining operators are also applied. The *Framing* operator will return the union of the manifolds for each point of each chair available for A and B. The *is facing* will also be computed, and will return the whole 360^o of orientation because the positions for A and B overlap. Finally, the *Distance* constraint will return the union of offsets of the chairs by a radius of 10m. But, as B is already constrained to the area of the chairs (which is a subset of the result of *Distance max*), its PLRS will not be modified. As this is the last constraint and no PLRS was modified, the fixed point is reached.

The sampling process starts. First, the PLS of A is sampled using an uniform distribution. Then, the constraints that depend on A are updated to reflect the newly fixed position of A. The *Framing* constraint will be updated as well as the *Distance* constraint. As A is placed at a precise position, the result will be a disc of radius 10m around A. The PLRS of B is then reduced to the intersection of *Close to the bar*, *In chair* and *Distance max*, *i.e.* the 5 chairs next to the bar around A. Since this constrained the PLRS of B, the *Framing* and *is facing* constraints are then updated once more.

Then, in a similar manner, the position of B is solved. This will update the *Framing* constraint (Fig. 11d) and the *is facing* constraints. As A and B have different positions, the *is facing* constraint results in each facing directly to the other.

Finally, the orientations of the entities are sampled. As all entities have their rotation constrained to a specific orientation (either by the *is facing* or the *Framing* constraints), the solving is trivial. Results are shown in Fig. 12 for three different runs of the algorithm.

7.2 Multiple scenes example

We propose a more complex scenario, describing a scene with 3 actors and 4 cameras (*cf.* Scenario 2). The scenario is composed of two static scenes which are computed one after the other. Results are shown in Fig. 13.

Scenario 1 Simple scenario

Scene 1: `Camera 1, MS on A screenleft and B screenright. A and B are close to the bar and in a chair. A is at most 10 meters from B and facing B.`

Scenario 2 Complex scenario with 3 actors and 4 cameras

Scene 1: `A faces B. B faces A. A and B are close to the bar and in chair. A and B and Camera 1 and Camera 2 are not seeing C. Camera 1, CU on B front screenright and BCU on A back screenleft. Camera 2, MCU on A right screenleft and B left screenright. Camera 3, FS on A and B and C. Camera 4, FS on C. Camera 4 is not seeing A and B.`

Scene 2: `A and B are in same position as in scene 1. C is at most 4 meters behind A and facing A. Camera 1, same position and orientation as in scene 1. Second camera MLS on B screenleft and A screencenter and C screenright. Camera 3, FS on A and B and C. Camera 4, MS on C. Camera 4 is not seeing A and B.`

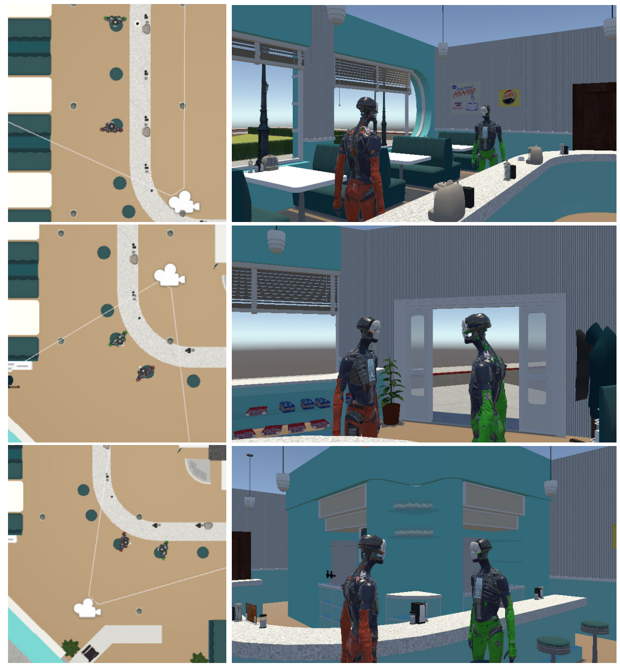


Figure 12: Three different results for the scenario 1

The resolution is done in two consecutive steps. We compute the first scene by solving the position and orientation for the 3 actors and the 4 cameras simultaneously. Then, the second scene is computed by building on the first one. Indeed, to ensure a coherent staging between the two scenes, we use the *same* constraint. The use of this constraint also reduces the computational cost of the second scene since some previously computed results are reused.

Computation took 71.4 seconds for the first scene and 16.6 seconds for the second scene.

7.3 Real case study: *Back to the future*

We then compared our results to an actual movie. We used the *Back to the future* dataset by Galvane et al.[Galvane et al. 2015b] which contains an artist's rendition of the *café* scene that is very close to the movie's sequence. We took three static scenes from the sequence, annotated them using the proposed language and solved the staging.

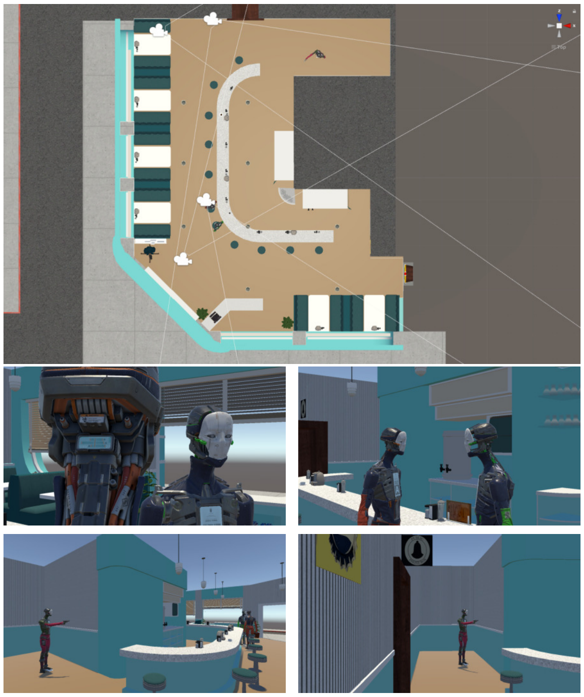
Results are shown in Fig. 14. We can see that our method produces results that are coherent with the scene specification, and that among the different computed configuration, one is very close to the director's staging for this scene.

The mean computation time is 36.7 seconds for the first scene, 9.8 seconds for the second scene, and 5.3 seconds for the third scene.

8 LIMITATIONS AND FUTURE WORK

While our system was successfully tested on a number of scenes and configurations, it does present some limitations that will need to be addressed in future work.

8.0.1 Constraints with multiple targets. With the notable exception of the *Framing* constraint (which handles a camera and two



(a) First scene

(b) Second scene

Figure 13: Scenario made of two consecutive static scenes. Position and orientation of actors A and B, and the first camera are kept between the two scenes.

entities), our system only handles constraints between two entities. While it is possible to specify constraints between multiple targets by strapping together several constraints, this does not allow the expression of common cinematographic rules such as the *line of action*. This constitutes an interesting lead for future work.

8.0.2 Extension to dynamic scenes. While our system currently handles static scenes, moving entities can be easily considered by performing computations at different time stamps and performing interpolations. A more principled approach consists in sampling regions and computing subject paths and camera paths rather than positions. This would however require the evaluation of constraints over time, hence computing the variations of the regions over time (spatio-temporal filtering).

CONCLUSION

This paper has first presented a language dedicated to the staging of subjects and cameras in virtual environments. From this language, we defined means to operationalize its tokens in terms of geometric pruning operators and sampling operators. We proposed a fixed point computation using these pruning and sampling operators to compute one or multiple solutions of a given specification. We have demonstrated the expressive power of our approach on a number of examples. In contexts such as narrative games or interactive storytelling, the approach enables a high degree of flexibility on how the characters and the cameras may be staged.

REFERENCES

K.R. Apt. 1999. The Essence of Constraint Propagation. *Theor. Comput. Sci.* 221, 1-2 (1999), 179–210.

W. Bares, S. McDermott, C. Boudreaux, and S. Thainimit. 2000. Virtual 3D Camera Composition from Frame Constraints. In *ACM International Conference on Multimedia*.

D. Bordwell. 2005. *Figures Traced in Light: On Cinematic Staging*. University of California Press.

F. Bungiu, M. Hemmer, J. Hershberger, K. Huang, and A. Kröller. 2014. Efficient computation of visibility polygons. *arXiv preprint arXiv:1403.3905* (2014).

M. Christie, P. Olivier, and J.-M. Normand. 2008. Camera Control in Computer Graphics. *Computer Graphics Forum* 27, 8 (December 2008), 2197–2218.

D.H. Clements and M.T. Battista. 1992. Geometry and spatial reasoning. *Handbook of research on mathematics teaching and learning* (1992), 420–464.

S.M. Drucker, T.A. Galyean, and D. Zeltzer. 1992. CINEMA: A System for Procedural Camera Movements. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*. ACM Press, New York, NY, USA, 67–70.

F. Durand, G. Drettakis, and C. Puech. 2002. The 3D visibility complex. *ACM Transactions on Graphics (TOG)* 21, 2 (2002), 176–206.

D.K. Elson and M.O. Riedl. 2007. A Lightweight Intelligent Virtual Cinematography System for Machinima Production. In *Proceedings of the Third AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'07)*. AAAI Press, 8–13.

Q. Galvane, M. Christie, C. Lino, and R. Ronfard. 2015a. Camera-on-rails: Automated Computation of Constrained Camera Paths. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (MIG '15)*. ACM, New York, NY, USA, 151–157.

Q. Galvane, R. Ronfard, C. Lino, and M. Christie. 2015b. Continuity editing for 3D animation. In *AAAI Conference on Artificial Intelligence*.

L. He, M. F. Cohen, and D. H. Salesin. 1996. The Virtual Cinematographer: A Paradigm for Automatic Real-Time Camera Control and Directing. In *SIGGRAPH 96 Conference Proceedings (Annual Conference Series)*, Holly Rushmeier (Ed.). ACM SIGGRAPH, Addison Wesley, 217–224. held in New Orleans, Louisiana, 04-09 August 1996.

M. Kapadia, S. Frey, A. Shoulson, R.W. Sumner, and M. Gross. 2016. CANVAS: Computer-assisted Narrative Animation Synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '16)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 199–209.

F. Lamarche. 2009. TopoPlan: a topological path planner for real time human navigation under floor and ceiling constraints. *Computer Graphics Forum* 28, 2 (March 2009).

C. Lino and M. Christie. 2012. Efficient Composition for Virtual Camera Control.

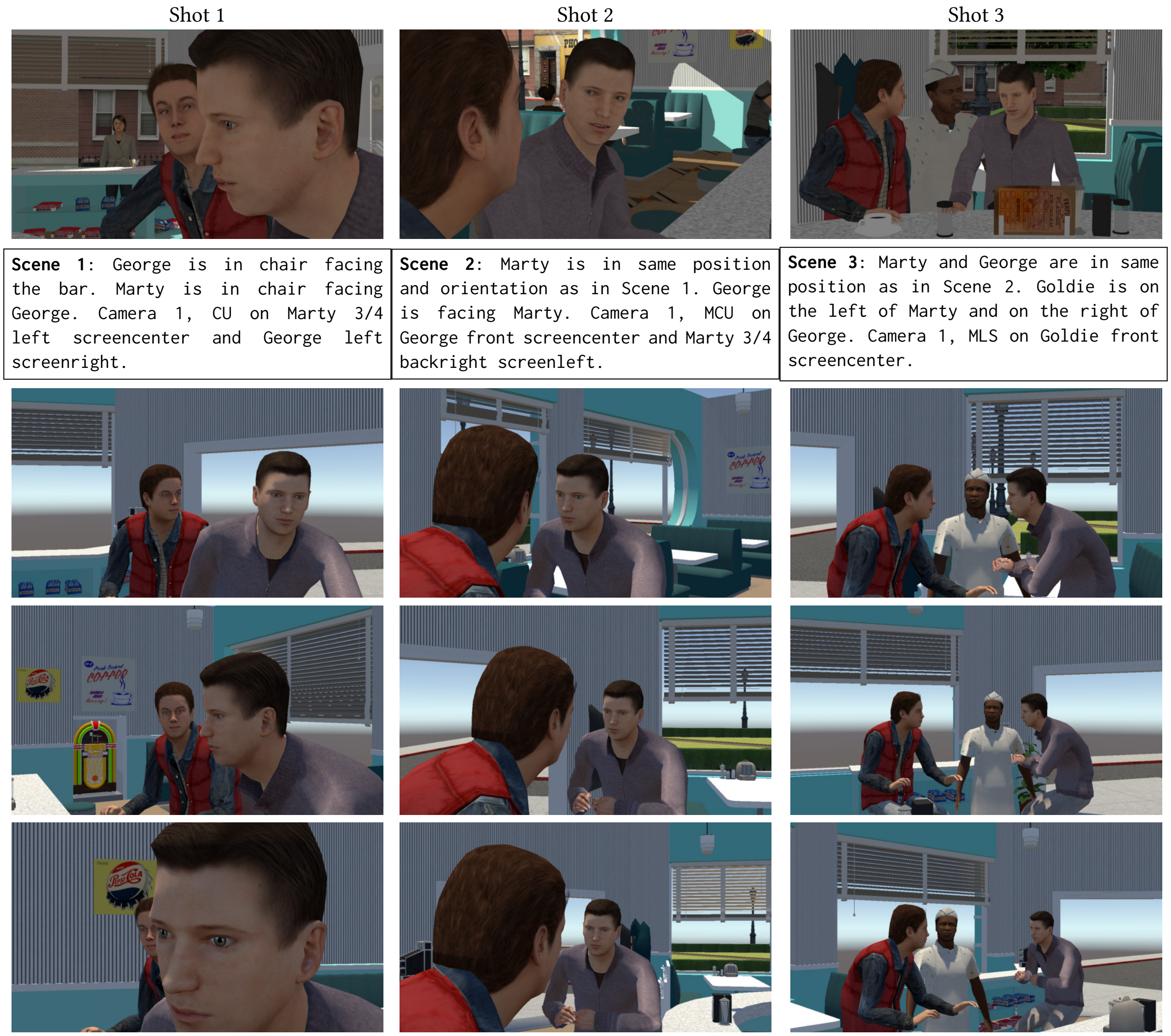


Figure 14: Comparison between an artist's rendition (first line) of 3 static shots taken from the cafe scene in *Back to the future* (1985) and our method (3 results shown, lines 3 to 5). The specification used to generate each shot is shown on line 2. The specification is written by the authors, following the original café scene from *Back to the future*, directed by Robert Zemeckis.

C. Lino and M. Christie. 2015. Intuitive and Efficient Camera Control with the Toric Space. *ACM Transactions on Graphics (TOG). Proceedings of ACM SIGGRAPH 2015* (2015).

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

P. Olivier, N. Halper, J. H. Pickering, and P. Luna. 1999. Visual Composition as Optimisation. In *AISB Symposium on on AI and Creativity in Entertainment and Virsual Art*. 22–30.

R. Ranon and T. Urli. 2014. Improving the Efficiency of Viewpoint Composition. *IEEE Transactions on Visualization and Computer Graphics* 20, 5 (May 2014), 795–807.

R. Ronfard, V. Gandhi, and L. Boiron. 2015. The Prose Storyboard Language: A Tool for Annotating and Directing Movies. *CoRR* abs/1508.07593 (2015).

C. Talbot. 2015. *Directing virtual humans using play-script spatiotemporal reasoning*. Ph.D. Dissertation. University of North Carolina at Charlotte.

R. Wein, A. Baram, E. Flato, E. Fogel, M. Hemmer, and S. Morr. 2018. 2D Minkowski Sums. In *CGAL User and Reference Manual* (4.11.1 ed.). CGAL Editorial Board. http://doc.cgal.org/4.11.1/Manual/packages.html

H-Y. Wu and M. Christie. 2016. Analysing Cinematography with Embedded Constrained Patterns. In *Eurographics Workshop on Intelligent Cinematography and Editing*, M. Christie, Q. Galvane, A. Jhala, and R. Ronfard (Eds.). The Eurographics Association.