Virtual Cinematography Using Optimization and Temporal Smoothing

Alan Litteneker
University of California, Los Angeles
alitteneker@cs.ucla.edu

Demetri Terzopoulos
University of California, Los Angeles
dt@cs.ucla.edu

ABSTRACT

We propose an automatic virtual cinematography method that takes a continuous optimization approach. A suitable camera pose or path is determined automatically by computing the minima of an objective function to obtain some desired parameters, such as those common in live action photography or cinematography. Multiple objective functions can be combined into a single optimizable function, which can be extended to model the smoothness of the optimal camera path using an active contour model. Our virtual cinematography technique can be used to find camera paths in either scripted or unscripted scenes, both with and without smoothing, at a relatively low computational cost.

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; **Animation**; • **Mathematics of computing** → *Continuous optimization*;

KEYWORDS

virtual cinematography, optimization, camera motion

ACM Reference format:
Alan Litteneker and Demetri Terzopoulos. 2017. Virtual Cinematography Using Optimization and Temporal Smoothing. In *Proceedings of MiG '17, Barcelona, Spain, November 8–10, 2017,* 6 pages.
https://doi.org/10.1145/3136457.3136467

1 INTRODUCTION

The problem of virtual cinematography is one of the oldest in 3D computer graphics. The settings chosen for a virtual camera intrinsically affect both what objects are visible and how the viewer perceives them. In general, the goal of any declarative virtual cinematography system is to automatically select camera settings that match a user specified set of parameters for the final rendered frame.

However, the demands of even the simplest of photographic and cinematographic techniques require sophisticated parameters that may be difficult to quantify at best [Datta et al. 2006]. Given the inherent subjectivity of these art forms, users may frequently disagree both on how well a frame matches a particular aesthetic, and on the relative importance of aesthetic properties in a frame.

As such, virtual cinematography for scenes that are planned before delivery to the viewer, such as with 3D animated films made popular by companies such as Pixar, is generally solved manually by a human artist. However, when a scene is even partially unknown, such as when playing a video game or viewing a procedurally generated real-time animation, some sort of automatic virtual cinematography system must be used.

The type of system with which the present work is concerned uses continuous objective functions and numerical optimization solvers. As an optimization problem, virtual cinematography has some difficult properties: it is generally nonlinear, non-convex, and cannot be efficiently expressed in closed form. Furthermore, ensuring smoothness in camera paths can amplify the difficulty of this problem as the dimensionality of the search increases.

2 RELATED WORK

As previously mentioned, a diverse set of systems have been developed in this field [Christie et al. 2005]. The following are some of the projects most relevant to our work:

A number of systems have been developed that employ continuous optimization methods [Drucker and Zeltzer 1995; Gleicher and Witkin 1992]. Issues such as the sensitive nature of the shot weighting and difficulty with over-constraining led to the use of stochastic optimization [Abdullah et al. 2011; Burelli et al. 2008], as well as the development of systems based on boolean constraints [Bares et al. 2000] and logical reasoning [Funge et al. 1999]. Some researchers have even attempted to control in-game cameras, with varied success [Burelli 2012; Halper et al. 2001].

There have also been more lightweight methods, such as selecting shots from predefined libraries of geometry configurations [Elson and Riedl 2007], or by assuming preprocessed elements of environment geometry will remain constant over the motions of a scene [Lino et al. 2010].

Efficiently producing occlusion-free and collision-free camera poses or paths has proven problematic for several researchers. CamDroid [Drucker and Zeltzer 1995] supported the avoidance of collisions and occlusions through an expensive preprocessing of the scene geometry, which required the scene to remain static while the camera moved.

More recent work has taken a purely stochastic approach to occlusion satisfiability [Ranon and Urli 2014] or omitted support for collisions and occlusions altogether [Lino and Christie 2015], instead focusing on efficiency over a limited set of shot parameters. While much of this work has been restricted to static poses or simple interpolated paths, there has been some work on path smoothness thus far limited to specific types of splines or scene actions. [Assa et al. 2008; Galvane et al. 2015]

MiG '17, November 8–10, 2017, Barcelona, Spain

ACM ISBN 978-1-4503-5541-4/17/11…$15.00
https://doi.org/10.1145/3136457.3136467

3 VIRTUAL CINEMATOGRAPHY SYSTEM

Our prototype virtual cinematography system operates based on a continuous objective function that corresponds to a user-specified set of shot parameters. When operating on a moving scene, this function must be extended to handle variable parameterization as well as the modeling of smoothing. Once this objective function has been fully constructed, our system attempts to minimize the function using standard optimization techniques. The system is integrated with *Unreal Engine 4* for rendering and user input.

3.1 Objective Functions

To find a camera pose that matches the desired shot parameters, our system must be able to evaluate how well a particular camera pose matches the given shot parameters. This is done with an objective function $f : \mathbb{R}^N \mapsto \mathbb{R}$ for each shot parameter. To simplify the notation, these functions will be denoted as $f(\mathbf{x})$ despite requiring different numbers and types of parameters.

Multiple shot parameters are combined into a single objective function $f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k f_k(\mathbf{x})$ as a weighted sum of the their associated objective functions, a strategy that critically allows for compromises between shot parameters for which the minima do not precisely intersect. The influence of each parameter can be tuned by adjusting the values of α.

Evaluating the same functions over an entire camera path requires the parameterization of $\mathbf{x}$ over time to $\mathbf{x}(t)$, which can be evaluated as $\sum_i f(\mathbf{x}(t_i))$ over the set of discrete keyframes.

3.2 Temporal Smoothing

To model the inertia and momentum viewers expect from real-world cameras, we extend the user-specified temporal objective function with an active contour model [Kass et al. 1988]. In this augmented model, our system attempts to minimize the energy functional

$$E(\mathbf{x}(t)) = E_{\text{int}}(\mathbf{x}(t)) + E_{\text{ext}}(\mathbf{x}(t)) \tag{1}$$

comprising an internal energy

$$E_{\text{int}}(\mathbf{x}(t)) = \frac{1}{2} \int \alpha |\dot{\mathbf{x}}(t)|^2 + \beta |\ddot{\mathbf{x}}(t)|^2 \, dt, \tag{2}$$

where α and β are given constants and the overstruck dots denote differentiation with respect to t, and an external energy

$$E_{\text{ext}}(\mathbf{x}(t)) = f(\mathbf{x}(t)). \tag{3}$$

Determining suitable values for α and β generally requires experimental parameter tuning. If either is set too high, the resultant camera path may violate the desired parameters, while too low a setting may fail to alleviate the undesirable artificiality. It was found experimentally that $\alpha = 0.01$ and $\beta = 0.2$ produced satisfactory results.

3.3 Optimization Techniques

Given the nonlinear nature of many of our objective functions, finding a solution analytically is impossible for any moderately complex parameter set. As long as a given objective function is convex, a simple gradient descent (GD) algorithm can be used. However, this is generally inadequate as the underlying objective functions that must be optimized are likely to be non-convex and therefore cause our system to be caught in local minima. Simulated annealing (SA) provides a potential solution. While even quenched SA is less sensitive to local minima, the solutions it produces are far less precise than those found by GD when the latter does find the global minimum. By running an SA optimizer before a GD optimizer, the robustness of the former can be combined with the precision of the latter.

Optimizing for camera paths is a straightforward extension of optimizing for static poses. Without smoothing from the active contour model, a temporal objective function can be optimized by running the optimization algorithms locally on each frame. Optimizing with smoothing is somewhat more involved. Each time interval must first be initialized with a locally computed rough optimum that ignores the smoothing, then gradient descent can be performed such that every component of $\nabla E(\mathbf{x}(t))$ is computed for every time step t before any new values can be set.

3.4 Use Cases

While there are many different ways of utilizing the cinematographic tools outlined above, our system supports two primary categories of use cases, which are labeled as *scripted* and *unscripted* scenarios.

3.4.1 Scripted Scenarios. In a scripted scenario, our system attempts to find a satisfactory camera path for a preplanned scene before any part of it is to be displayed to the viewer. As this is an offline operation, our system can take as much time as is necessary to find a satisfactory solution. Furthermore, complete knowledge of all past, current, and future states of the scene are available at every instant of time. This is analogous to the traditional preproduction workflow of a live action or animated film.

Scripted scenarios can be solved in a straightforward manner by the optimization strategy described in Section 3.3.

3.4.2 Unscripted Scenarios. In an unscripted scenario, the goal of our system is to find a camera pose for the current instant in time given complete knowledge of the current and past states of the scene, but without any direct knowledge of the future. This is an online operation during the playback of the scene; therefore, our system must be able to find a satisfactory solution in real time given the viewer's desired frame rate. This is analogous to the shooting of an unscripted documentary or news material, or to video game camera control.

Without smoothing by an active contour model, the first frame camera pose is set either by the user or by running SA, then GD is run every frame starting from the values of the previous frame.

Unscripted scenarios with smoothing are significantly trickier, since in order for the active contour model to work effectively, our system must know something of the unknown future. As our current system is lacking contextual or behavioral data, as employed by [Halper et al. 2001] to form predictions more precisely, the prediction scheme currently supported is a Taylor series approximation where, for future keyframes $t_1, \ldots, t_n$,

$$\mathbf{x}(t_j) \approx \sum_{k=0}^{N} \frac{1}{k!} (t_j - t_0)^k \frac{d^k \mathbf{x}(t_0)}{dt^k}, \tag{4}$$

where $d^k \mathbf{x}(t)/dt^k$ is the k^{th}-order derivative of $\mathbf{x}(t)$. To alleviate instability stemming from discrete time steps, our system calculates

the necessary derivatives as local averages

$$\frac{d^k \mathbf{x}(t)}{dt^k} \approx \frac{1}{M\Delta_t} \sum_{i=0}^{M} \frac{d^{k-1}\mathbf{x}(t-\Delta_t i)}{dt^{k-1}} - \frac{d^{k-1}\mathbf{x}(t-\Delta_t(i+1))}{dt^{k-1}}, \quad (5)$$

where $\Delta_t = t_i - t_{i+1}$. Of course, this is a tradeoff as too large a value of M can create an undesirable sense of variable inertia. Values of $M = 10$ and $\Delta_t = 0.1$ sec were used.

Once these predictions are made, the same types of optimization techniques can be used to find a suitable camera path through the predicted period, and the first future keyframe $\mathbf{x}(t_1)$ can be used as the next camera state.

4 PREDEFINED SHOT PARAMETERS

While any valid objective function can be input by the user, our system supports common cinematic and photographic compositional aesthetics by providing a set of predefined shot parameter types, modeled as novel objective functions. These mimic the types of shot descriptions commonly used on "shot lists" in the preproduction of a film by live action directors and cinematographers.

While the components of our system integrated from the *Unreal Engine* rely on its particular coordinate system (left-handed, z-up, scale, etc.), the functions listed here are written to be as generic as possible. As several rules require the projection of a point onto the image plane, the function $\Pi(\mathbb{R}^3) \rightarrow \mathbb{R}^2$ is used to denote such a projection from world space to screen space, where any point $\mathbf{p}$ that will appear in frame will have $|v| \leq 1, \forall v \in \Pi(\mathbf{p})$. Note that this projection function must have knowledge of the current camera parameters (e.g., position, orientation, field of view, etc.), which are not explicitly listed as parameters for many functions.

All of the functions described below are point-based, meaning that all details of the scene are collected in the form of observing the position, orientation, etc., of specific points connected to objects of interest in the scene. This requires that complex elements in the scene be represented by multiple points. This point-based model varies from the explicit geometric models of [Bares et al. 2000; Jardillier and Languénou 1998], where each camera pose is evaluated based on observations of geometric primitives approximating complex objects, as well as from the rendering-based methods of [Abdullah et al. 2011; Burelli et al. 2008], where each camera pose is evaluated by analyzing a full rendering of the scene from that pose.

4.1 Visibility

An object the user desires to be visible must appear on-screen, but this apparently simple behavior requires two different types of parameters, denoted *frame bounds* and *occlusion*, both of which must be satisfied for an object to be visible.

4.1.1 Frame Bounds. The frame bounds rule is simply that any point that the user desires to be visible must appear inside the bounds of the frame. In many circumstances, some points are commonly required to be inset from the edge of the frame, a property commonly called headroom in live action cinematography. For human subjects, too much room between the top of the head and the edge of the frame causes the subject to appear small and insignificant, while too little causes the subject to appear restricted and boxed in [Brown 2013, p. 52], as is shown in Figure 1.

Figure 1: A frame that was specified to have too much headroom, as chosen by our system.

Given world space point $\mathbf{p}$ and constants $x_{\text{low}} < x_{\text{high}}$, where $-x_{\text{low}} = x_{\text{high}} = 1$ ensures a point is simply in frame and $0 < (|x_{\text{low}}|, x_{\text{high}}) < 1$ will produce headroom, the frame-bounds rule can be evaluated as

$$f(\mathbf{p}) = \sum_{x \in \Pi(\mathbf{p})} g(x), \quad (6)$$

where

$$g(x) = \begin{cases} (x - x_{\text{low}})^2 & x < x_{\text{low}}; \\ 0 & x_{\text{low}} \leq x \leq x_{\text{high}}; \\ (x - x_{\text{high}})^2 & x > x_{\text{high}}. \end{cases} \quad (7)$$

4.1.2 Occlusion and Collision. The occlusion rule is simply that there must be an unoccluded line of sight from the camera to an object that the user desires to be visible. A common distinction is made between clean shots where all of an object is unoccluded and dirty shots where only part of object is unoccluded.

Similarly, the collision rule simply states that the camera must not collide with scene elements. Just as it is generally undesirable for real world cameras to pass through walls and actors, users of virtual cameras generally expect their cameras not to pass through apparently solid virtual matter.

Unfortunately, building suitable objective functions for occlusion and collision is much more complicated. Unlike all of the other functions described, these parameters represent an observation about the geometry of the scene as a whole and therefore cannot be defined in the same concise analytic style used elsewhere. Furthermore, there will be a discontinuity in any direct observation of visibility at the boundary of occlusion and collision. To address these issues, our system provides an n-dimensional linear interpolator that, given camera position $\mathbf{c}$ and point position $\mathbf{p}$, can be expressed as

$$f(\mathbf{c}) = g_3(\mathbf{c}), \quad (8)$$

where

$$g_i(\mathbf{c}) = \frac{\mathbf{c}_i^a - \mathbf{c}_i}{\mathbf{c}_i^b - \mathbf{c}_i^a} g_{i-1}(\mathbf{c}^b) + \frac{1 - \mathbf{c}_i^a + \mathbf{c}_i}{\mathbf{c}_i^b - \mathbf{c}_i^a} g_{i-1}(\mathbf{c}^a). \quad (9)$$

Figure 2: Examples of a close-up shot (left) and a wide shot (right), both chosen by our system.

Figure 3: Examples of a shot from a low vertical angle (left) and a profile angle (right) chosen by our system.

Here, $\mathbf{c}^a$ and $\mathbf{c}^b$ are the closest sample locations on axis i such that $\mathbf{c}_i^a \leq \mathbf{c}_i$ and $\mathbf{c}_i^b \geq \mathbf{c}_i$ and $\mathbf{c}_j^a = \mathbf{c}_j^b = \mathbf{c}_j, \forall j \neq i$, and

$$g_0(\mathbf{c}) = B(\mathbf{c}, \mathbf{p}), \tag{10}$$

where

$$B(\mathbf{c}, \mathbf{p}) = \begin{cases} 1 & \text{if the path from } \mathbf{c} \text{ to } \mathbf{p} \text{ is occluded;} \\ 0 & \text{if the path from } \mathbf{c} \text{ to } \mathbf{p} \text{ is not occluded.} \end{cases} \tag{11}$$

As this function is not analytically differentiable, differentiation is computed numerically as $\frac{df(\mathbf{x})}{dx} \approx \frac{f(\mathbf{x}+\Delta_x)-f(\mathbf{x})}{\Delta_x}$ with $\Delta_x = 1\,\text{cm}$.

To improve smoothness, our system supports the convolution of the samples of the interpolator by a Gaussian kernel, which can be expressed mathematically as changing $g_0(\mathbf{c})$ to

$$g_0(\mathbf{c}) = \frac{1}{\sqrt{2\pi\sigma^2}} h_3(\mathbf{c}). \tag{12}$$

Here,

$$h_i(\mathbf{c}) = \sum_{j=-K/2}^{K/2} e^{-\frac{(j\Delta_i)^2}{2\sigma^2}} h_{i-1}(\mathbf{c}+j\Delta_i U_i), \tag{13}$$

where Δ is the set of convolution step sizes, U is the set of bases for $\mathbf{c}$, and $h_0(\mathbf{c}) = B(\mathbf{c}, \mathbf{p})$. Values $K = 7$, $\sigma = 1$, and $\Delta = 25\,\text{cm}$ are used. Of course, the convolution requires more samples than its unconvolved counterpart, decreasing efficiency.

Note that in all uses, $B(\mathbf{c}, \mathbf{p})$ is sampled in a uniform grid with steps that are specified at function construction. Samples in the current search neighborhood can be fetched on demand, then cached in a hashmap for efficient retrieval in later computations. This approach differs from [Drucker and Zeltzer 1995] in that it requires no preprocessing of the scene geometry, does not require the occluding or colliding geometry to remain static, and can scale to scenes of any size, given a utility for sampling $B(\mathbf{c}, \mathbf{p})$ that is equally efficient in scenes of any size.

4.2 Shot Size

The amount of space an object occupies in the frame reflects its importance in the scene. For human subjects, this is usually expressed in terms of how much of a particular person is in frame. For example, a view of a person's entire body, commonly called a wide or long shot, is frequently used to establish the relative geometry of objects in a scene, while a view of only their head and shoulders, commonly called a close-up shot, is often used to show dialog [Bowen and Thompson 2013, p. 8–11]. Examples of both are shown in Figure 2.

While there are several different ways of evaluating this rule using projective geometry, most tend to suffer from severe instability given even moderately suboptimal inputs. Given object points $\mathbf{a}$ and $\mathbf{b}$, camera position $\mathbf{c}$, and angle θ, the most stable objective function[1] we have found is

$$f(\mathbf{a}, \mathbf{b}, \mathbf{c}, \theta) = \left(\theta - \arccos\left(\theta - \frac{(\mathbf{a}-\mathbf{c})\cdot(\mathbf{b}-\mathbf{c})}{||\mathbf{a}-\mathbf{c}||\,||\mathbf{b}-\mathbf{c}||}\right)\right)^2. \tag{14}$$

As described above, the object points $\mathbf{a}$ and $\mathbf{b}$ are chosen to be the edges of the object to be in frame (e.g., top of head and mid chest for a close-up shot). For most purposes, the value of θ can be chosen to be the camera's vertical field of view. However, when headroom is desirable for the same points (see Section 4.1.1), this must be decreased somewhat so as to avoid unnecessary parameter conflict.

4.3 Relative Angles

The angle from which an audience are shown a particular object can dramatically change the way that an object or character is viewed. This can generally be divided into two categories, vertical and profile angles, examples of which are shown in Figure 3.

Objects, and people especially, appear more powerful, intimidating, or ominous when viewed from below, and conversely more weak, frightened, or small when viewed from above. This relative *vertical angle* is often used to demonstrate the changing relationships between characters in a story, making each relative rise or decline visually clear to the audience [Bowen and Thompson 2013, p. 34–39].

The way in which people are perceived is strongly affected by the angle of view relative to the direction in which they are looking, or *profile angle*. In general, characters that are viewed directly from the front are felt to be more connected to the viewer than characters viewed in profile, mimicking the way humans tend to face directly towards things upon which they are focused [Bowen and Thompson 2013, p. 40–43].

Given object point $\mathbf{p}$, camera position $\mathbf{c}$, unit object up direction $\mathbf{u}$, and desired angle θ relative to unit object look direction $\mathbf{d}$, the objective function for vertical angle is

$$f(\mathbf{p}, \mathbf{c}, \mathbf{u}, \theta) = \left(\theta - \arccos\left(\frac{\mathbf{p}-\mathbf{c}}{||\mathbf{p}-\mathbf{c}||}\cdot\mathbf{u}\right)\right)^2, \tag{15}$$

while the profile angle function is

$$f(\mathbf{p}, \mathbf{c}, \mathbf{u}, \mathbf{d}, \theta) = \left(\theta - \arccos\left(\frac{\mathbf{p}-\mathbf{u}((\mathbf{p}-\mathbf{c})\cdot\mathbf{u})-\mathbf{c}}{||\mathbf{p}-\mathbf{u}((\mathbf{p}-\mathbf{c})\cdot\mathbf{u})-\mathbf{c}||}\cdot\mathbf{d}\right)\right)^2. \tag{16}$$

[1] For example, the function $f(\mathbf{a}, \mathbf{b}) = (1 - |\Pi(\mathbf{a})_y - \Pi(\mathbf{b})_y|)^2$ is highly unstable, where the subscript denotes the y component of the vector.

Figure 4: An example of a frame specified for an actor not to have adequate look space, as chosen by our system.

4.4 Rule of Thirds

A standard rule of thumb in photographic composition is to place important objects near the third lines, both vertical and horizontal, in the frame. The most important objects in the scene are often framed nearest the intersections of these lines [Brown 2013, p. 51]. All the example frames shown in this section conform to this parameter.

Given a point $\mathbf{p}$ in world space as well as constants $0 \leq x_0 \leq 1$ and $0 < a \leq 1$, this parameter is modeled as

$$f(\mathbf{p}) = \sum_{x \in \Pi(\mathbf{p})} \frac{(x+b)^2}{(x+b)^2 + a} + \frac{(x-b)^2}{(x-b)^2 + a} \tag{17}$$

where

$$b = \sqrt{\frac{2\left(x_0^2 - a\right) + \sqrt{4\left(x_0^2 - a\right)^2 + 12\left(x_0^2 + a\right)^2}}{6}}. \tag{18}$$

To use the standard third lines, x_0 should be set to $1/3$. Selecting a value for a is a bit trickier, as too high a setting causes the function to affect only points in the immediate vicinity on-screen while too low a setting can cause the penalty for $|x| < x_0$ to decrease significantly. It was found experimentally that $a = 0.07$ produced satisfactory results.

4.5 Look Space

A common technique used to balance a shot is to give substantial space between a character and the frame bound in the direction they are looking, a property that has been given numerous names in the cinematographic literature, including "look space", "lead space", "nose room", "action room", etc. When this space is not provided, as is shown in Figure 4, viewers tend to perceive the character as psychologically on edge or boxed in, a property that can sometimes be utilized to interesting compositional effect [Brown 2013, p. 52].

Given object point $\mathbf{p}$, unit object look direction $\mathbf{d}$, camera position $\mathbf{c}$, and unit camera right direction $\mathbf{c}_R$, the look space parameter can be evaluated using the visibility frame bounds objective function $f_{\text{vis}}(\mathbf{p})$ from Section 4.1.1, as

$$f(\mathbf{p}) = f_{\text{vis}}\left(\Pi(\mathbf{p}) + \frac{\Pi(\mathbf{p}+\mathbf{d}) - \Pi(\mathbf{p})}{||\Pi(\mathbf{p}+\mathbf{c}_R) - \Pi(\mathbf{p})||}\right). \tag{19}$$

5 RESULTS

All of the described tools and techniques were implemented in a single-threaded prototype camera control system that was integrated with *Unreal Engine 4.9*, running online with the engine's standard game thread. Several different experimental scenes were then tested for both scripted and unscripted scenarios on a desktop computer with a 3.07 GHz processor and an Nvidia GTX 980 graphics card.

In all experiments, each camera pose required that our system optimize over 5 variables—position in $\mathbb{R}^3$, yaw, and pitch. This required that each shot had a static user-specified field of view and aspect ratio. In principle, our system is capable of optimizing over both static variables as well as camera roll, but we have not attempted to do so.

Note that for all the scripted scenes tested, a keyframe rate of 40 per second was subjectively found to produce a sufficiently smooth camera path. This is partially a side effect of the speed of action, which was often over 3 m/s. Scenes with slower action could use a lower keyframe density.

5.1 Speed

Measuring the speed of our system is difficult as it entirely depends on the tightness of the satisfaction boundaries. There is a tradeoff between processor time and solution quality, so a user willing to endure slower performance can set a higher bar for parameter satisfaction.

For most of the *scripted* scenes tested, a satisfactory unsmoothed camera path could be found using approximately 3,000 iterations of simulated annealing, followed by approximately 2,000 iterations of gradient descent, taking about 1.15 seconds per frame. With smoothing, the same two-phase optimization can be used to roughly locally optimize using about 50–75% of the iterations required without smoothing, followed by 1,000–1,200 iterations of gradient descent with smoothing at a total cost of about 1.75 seconds per frame.

In *unscripted* experiments, the unsmoothed algorithm was subjectively found to be capable of finding satisfactory solutions using 100 optimization iterations of gradient descent at an average of 60 frames per second. The smoothed algorithm was found to produce a subjectively satisfactory solution—given the predicition of 4 keyframes with $t = 0.25$ sec and with 10 iterations of gradient descent to initialize each keyframe, followed by another 10 iterations of active contour model gradient descent—at an average of 25 frames per second, which is more than fast enough to support real-time usage.

5.2 Camera Paths

A demonstrative scene involved a character running around an open area, where the parameter set was configured for the camera to follow the character from the front. At one point in the scene, the character passes close enough to a wall that requires deflecting the camera from the optimal path. The path of the camera, as shown in Figure 5 (top), is similar in all use cases tested.

An example of a scene in which our system met with more mixed success was also noted. In this scene, a character runs past a set of pillars while the camera follows him with a long lens in profile. As the optimal camera path for the relative angle parameter becomes occluded by the pillars, the system must compromise between the

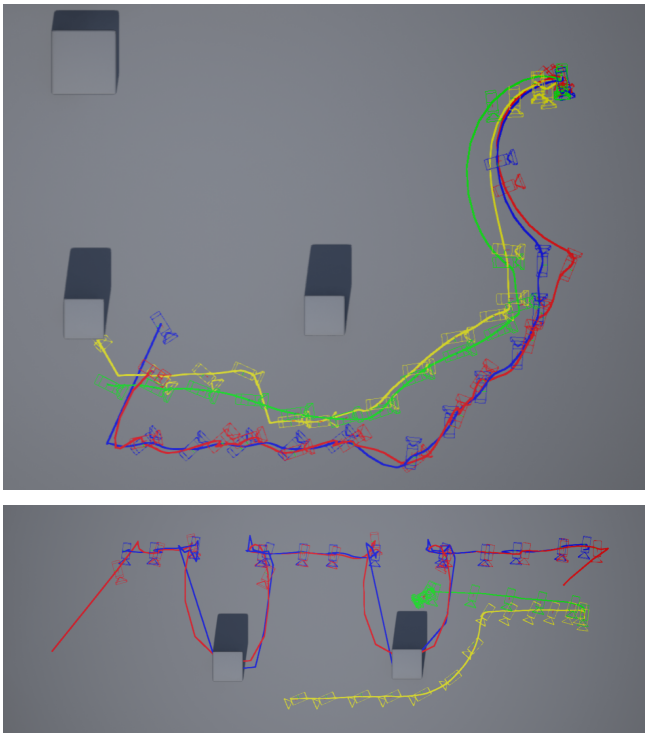

Figure 5: Overhead views of the camera paths computed for a simple moving scene (top) and a more complex scene where the camera's optimal path is blocked by a number of pillars (bottom). Color key: Green is unscripted-unsmoothed, yellow is unscripted-smoothed, red is scripted-unsmoothed, and blue is scripted-smoothed.

occlusion parameter and the relative angle parameter. As shown in Figure 5 (bottom), the paths of the various use cases vary much more significantly, and each suffers from its own particular issue.

The unscripted-unsmoothed strategy becomes stuck when it comes to the first occlusion boundary, while the unscripted-smoothed strategy manages to find its way around the first occlusion to trail the actor from a distance. While the latter solution seems intuitively better, the user has not explicitly specified which is preferable.

Meanwhile, both of the scripted strategies attempt to balance more evenly between all parameters, resorting to jagged solutions that momentarily collide with the scene geometry. However, these solutions represent a technically fairer compromise between the desired shot parameters.

6 FUTURE WORK

While we have received anecdotal feedback from users of the current prototype, we have not attempted to run a proper user study examining whether the rules and algorithms described are satisfactory for the rigors of professional use. Even the design of such a user study would be nontrivial [Lino et al. 2014]. The expressiveness of the system can likely be increased by including more common cinematographic parameters, such as frame balance, leading lines, separation, contrast, and depth of field, as would adding additional methods for combining multiple parameters.

Additionally, the integration of other optimization algorithms would likely improve overall performance and accuracy. Specializing the camera model for specific objective function types has been shown to increase efficiency [Lino and Christie 2015]. In unscripted scenarios, more accurate and stable predictions of future states could dramatically expand the use of multiple elements.

In real-world cinematography, actors are frequently positioned based on the physical constraints on the camera in the environment along with the content of the scene, a process that to our knowledge no existing automated cinematography system has attempted to emulate. The adjustment of character positioning may necessitate a significantly different optimization approach.

REFERENCES

Rafid Abdullah, Marc Christie, Guy Schofield, Christophe Lino, and Patrick Olivier. 2011. Advanced Composition in Virtual Camera Control. In *Smart Graphics*. Springer, Berlin, 13–24.

Jackie Assa, Daniel Cohen-Or, I-Cheng Yeh, Tong-Yee Lee, and others. 2008. Motion Overview of Human Actions. *ACM Transactions on Graphics (TOG)* 27, 5, Article 115 (2008), 10 pages.

William Bares, Scott McDermott, Christina Boudreaux, and Somying Thainimit. 2000. Virtual 3D Camera Composition From Frame Constraints. In *Proc. 8th ACM International Conference on Multimedia*. 177–186.

Christopher J. Bowen and Roy Thompson. 2013. *Grammar of the Shot*. Taylor & Francis.

Blain Brown. 2013. *Cinematography: Theory and Practice: Image Making for Cinematographers and Directors*. Taylor & Francis.

Paolo Burelli. 2012. *Interactive Virtual Cinematography*. Ph.D. Dissertation. IT University of Copenhagen.

Paolo Burelli, Luca Di Gaspero, Andrea Ermetici, and Roberto Ranon. 2008. Virtual Camera Composition with Particle Swarm Optimization. In *Smart Graphics*. Springer, Berlin, 130–141.

Marc Christie, Rumesh Machap, Jean-Marie Normand, Patrick Olivier, and Jonathan Pickering. 2005. Virtual Camera Planning: A Survey. In *Smart Graphics*. Springer, Berlin, 40–52.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. 2006. Studying Aesthetics in Photographic Images Using a Computational Approach. In *Computer Vision – ECCV 2006*. Springer, Berlin, 288–301.

Steven M. Drucker and David Zeltzer. 1995. CamDroid: A System for Implementing Intelligent Camera Control. In *Proc. 1995 ACM Symposium on Interactive 3D Graphics*. 139–144.

David K. Elson and Mark O. Riedl. 2007. A Lightweight Intelligent Virtual Cinematography System for Machinima Production. In *Proc. Artificial Intelligence and Interactive Digital Entertainment Conf.* 8–13.

John Funge, Xiaoyuan Tu, and Demetri Terzopoulos. 1999. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In *Proc. ACM SIGGRAPH*. 29–38.

Quentin Galvane, Marc Christie, Christophe Lino, and Rémi Ronfard. 2015. Camera-on-Rails: Automated Computation of Constrained Camera Paths. In *Proc. 8th ACM SIGGRAPH Conference on Motion in Games*. ACM, 151–157.

Michael Gleicher and Andrew Witkin. 1992. Through-the-Lens Camera Control. *Computer Graphics* 26, 2 (1992), 331–140. (Proc. ACM SIGGRAPH '92).

Nicolas Halper, Ralf Helbing, and Thomas Strothotte. 2001. A Camera Engine for Computer Games: Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence. *Computer Graphics Forum* 20, 3 (2001), 174–183.

Frank Jardillier and Eric Languénou. 1998. Screen-Space Constraints for Camera Movements: The Virtual Cameraman. *Computer Graphics Forum* 17, 3 (1998), 175–186.

Michael Kass, Andrew Witkin, and Demetri Terzopoulos. 1988. Snakes: Active Contour Models. *International Journal of Computer Vision* 1, 4 (1988), 321–331.

Christophe Lino and Marc Christie. 2015. Intuitive and Efficient Camera Control with the Toric Space. *ACM Trans. Graph.* 34, 4, Article 82 (July 2015), 12 pages.

Christophe Lino, Marc Christie, Fabrice Lamarche, Guy Schofield, and Patrick Olivier. 2010. A Real-Time Cinematography System for Interactive 3D Environments. In *Proc. 2010 ACM SIGGRAPH/EG Symposium on Computer Animation*. 139–148.

Christophe Lino, Rémi Ronfard, Quentin Galvane, and Michael Gleicher. 2014. How Do We Evaluate the Quality of Computational Editing Systems?. In *Proc. AAAI Workshop on Intelligent Cinematography And Editing*. 35–39.

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