

Applying Constraint Weighting to Autonomous Camera Control

Owen Bourne and Abdul Sattar
Institute for Integrated and Intelligent Systems,
Griffith University
PMB50 Gold Coast Mail Centre, QLD 9726
[o.bourne, a.sattar]@griffith.edu.au

Abstract

Automating camera control for third-person perspective computer games is a difficult and time-consuming task. One of the challenges games developers confront is how to manage the trade-off between implementation complexity and system usability. In this study, we investigate the application of constraint weighting techniques to the autonomous camera control problem. We demonstrate that this technique can significantly simplify autonomous camera control and reduce the gap between implementation and usability requirements. We describe the use of *weighting profiles* to control the behaviour of the camera and specialized heuristics for efficiently searching for the solution. We also describe a novel integrated visibility maintenance method. As part of the experimental study, we implemented a 3D game engine that supports dynamic environments; and demonstrate the effectiveness of the use of constraint solving techniques for autonomous camera control problems.

Introduction

Controlling an autonomous camera in third-person perspective computer games provides a unique set of challenges. The combination of high quality visual results, restricted computational power (for real-time applications) and unpredictable target movement often forces trade-offs between the capabilities of the camera system and efficient performance.

The primary function of a camera system is to maintain the visual coherency of the target. This requires the camera to maintain the position and alignment of the target in relation to the camera. Additional challenges such as maintaining smooth movement and acceleration, and visibility maintenance (occlusion avoidance) of the target provide areas of great difficulty when developing autonomous camera systems.

There are numerous proposed methods for addressing these challenges, originating from such diverse disciplines as robotics, medical imaging and virtual cinematography. Existing research has considered constraint satisfaction (Drucker & Zeltzer 1994; Bares & Lester 1999; Bares, Thainimit, & McDermott 2000; Halper & Olivier

2000; Halper, Helbing, & Strothotte 2001), potential fields (Beckhaus 2001), intelligent agents (Hornung 2003) and image-based visual servoing (Marchand & Courty 2000; Courty & Marchand 2001; Marchand & Courty 2002) for autonomous camera control. However, there is no generic solution to this problem.

The somewhat ad-hoc combinations of differing camera control methods and visibility maintenance systems has yet to produce an effective, unified methodology. Our research is directed towards achieving this goal by integrating constraint-weighted local search and ray-casting for visibility.

There are four major requirements of a successful autonomous camera system¹:

1. **Autonomy:** The camera must be able to move while maintaining the visual properties (e.g. size and orientation) and visibility of the target without intervention from the user (or level designer via triggers).
2. **Reactive:** The camera must be able to work reactively, without predictive information about future target positions.
3. **Real-time:** The camera must operate in real-time, without detriment to existing game engine components.
4. **Dynamic:** The camera must be able to deal with dynamic environments and multiple and dynamically changing targets.

In this paper we describe a detailed investigation of using constraint satisfaction techniques for representing and solving these camera control problems. Since the camera control problem involves a set of infeasible constraints, we use constraint weighting to give a preference order to these constraints. We then apply an incomplete but efficient constraint solving technique, known as *Stochastic Local Search*, for finding the best position of the camera for each frame. Our approach extends our existing work (Bourne & Sattar 2004a; 2004b) and takes into account a meta-level visibility constraint that influences the efficiency and structure of the underlying constraint solver.

The rest of this paper is organized as follows: the next section covers the necessary related work in the field. The

¹This is not a comprehensive list of the requirements of all autonomous camera systems.

representation used by our system is described in Problem Modelling, while the constraint solver is described in detail in Constraint Solving. A description of our system implementation and the experimental results is in Evaluation and Results, followed by the conclusion and direction for future research.

Background

A Constraint Satisfaction Problem (CSP) is defined as a triple $\langle V, D, R \rangle$, with a set of variables V , a domain D of values for each variable V and a set of relations R . A constraint is defined as a relation over a set of variables that is a subset of the cartesian product of the variables' domains.

The problem is reduced to efficiently determining an assignment to the variables such that all constraints are satisfied. If there is no consistent assignment (not all constraints can be satisfied), the problem is over-constrained (Freuder & Wallace 1996). These problems can be addressed by classifying constraints into *hard* (satisfaction is mandatory), and *soft* (satisfaction is preferable, but can be violated to satisfy other constraints).

The simplest constraint-based camera system uses a hard-constraint implementation based around polar or spherical co-ordinates (Stone 2004). The camera's position and orientation is solved directly in relation to the target. Smooth and controlled movements are generated by damping the constraint values. The damping ratios are very sensitive, and incorrect values can cause the camera to oscillate. It is common for these systems to have a large number of parameters (greater than 50) to control the camera.

Some recent work has attempted to address these damping issues by using pre-defined constraint relaxation percentages (Halper, Helbing, & Strothotte 2001)². In the general case, these relaxation percentages can alleviate some of the sensitivity of tuning the damping ratios. However, this still artificially locks the camera to pre-defined movement abilities (acceleration/deceleration, rotational freedom, maximum speed).

The work by (Bares *et al.* 2000) uses a weighting method for constraints (ranged between 0.0 and 1.0, where 1.0 makes a constraint *hard*). The constraints are satisfied using a recursive generate-and-test method, starting at a coarse resolution and refining over successive passes. The occlusion constraint is evaluated for each of the candidate solution that passes the generate-and-test solver.

A novel approach involves the use of pre-defined camera paths that are used in combination to provide the camera's movement (Christie, Languenou, & Granvilliers 2002). The interactive nature of computer games makes the commitment to a pre-defined path a non-optimal solution.

Various visibility evaluation methods have been investigated and applied to camera control. The most common are ray-casting methods (Giors 2004; Tomlinson, Blumberg, & Nain 2000) and shadow generation (Drucker & Zeltzer 1994; Halper, Helbing, & Strothotte 2001).

The time complexity of both methods increases as the environment becomes more complex. The ray-casting approach is often preferable as it consumes only minimal CPU time for evaluation. The shadow generation approach consumes CPU time to process the environments bounding volumes and the GPU fill-rate for creating the shadow information.

The integration of the visibility maintenance strategy and the camera control method is rarely attempted. The camera engine described in (Halper, Helbing, & Strothotte 2001) integrates the shadow approach with hard-constraint method described above.

The constraint satisfaction paradigm has been well established as an academic research area for over 40 years. The extensive study of these problems provides a wealth of information regarding the nature of the problems, as well as numerous representations and methods of searching for solutions. It is therefore desirable to utilize this information to addressing new problems (such as camera control). Our solution addresses inadequacies in existing works through the use of well-known constraint satisfaction representations and algorithms, which are detailed in the following sections.

Problem Modelling

The camera control problem is represented as a sequence of individual over-constrained problems. The visual properties of the target are achieved using soft constraints. No information about predicted target states are used by the constraint solver, thereby addressing requirement 2 (reactive).

In our representation, the variables V consist of each axis for the camera's position (X, Y and Z). The domain D for each variable V is a restricted set of the values in 3D space the camera can occupy. The relations R (constraints) define the visual properties of the camera.

Constraint Set

Some existing work uses different constraint representations, typically in terms of more specific visualization abilities (Halper, Helbing, & Strothotte 2001). Our constraint set can represent the same visual qualities (via reformulation), but uses a representation specifically designed for optimal evaluation performance.

The minimal set of constraints required to adequately represent the visual properties of the camera (requirement 1) and the real-time performance (requirement 3)³ are :

1. **Height:** represents the height relationship between the target height and the camera (positive or negative).
2. **Distance:** represents the distance relationship between the target position and the camera (must be positive).
3. **Orientation:** represents the angular alignment between the target facing vector and the camera's view vector (between 0° and 360°).
4. **Frame Coherence:** represents the minimal cost improvement before the camera moves (must be positive).

³Each additional constraint increases the computational complexity of the constraint solver. Simple representations are quicker for evaluation purposes.

²The spherical co-ordinate approach was used before it was formally published.

Height	$ DesiredHeight - (cam_y - obj_y) * HeightWeight$
Distance	$ DesiredDistance - \sqrt{((cam_x - obj_x)^2 + (cam_y - obj_y)^2 + (cam_z - obj_z)^2)} * DistanceWeight$
Orientation	$ Orientation_{desired} - (Orientation_{object} \bullet (cam - viewpoint) * \frac{180}{\pi}) * OrientationCost$

Table 1: Constraint evaluations and cost generation.

Each constraint requires 2 parameters: one for the value of the constraint; and the second for the constraint *weight*. The weight is used to indicate the preference of satisfaction for the constraint, where a higher weight = better satisfaction. The weight values have no preset limits, as in (Bares *et al.* 2000), and cannot be used to make the constraints *hard*.

The weights are unique for each constraint, and the set of weights is referred to as the *weighting profile*. The values of the weights are normalized in relation to the scale of values of the constraint (distance values are often much greater than height values), so the relative weighting of the constraints is maintained regardless of scale.

Multiple Dynamic Targets

Some implementations consider the camera’s viewpoint to be part of the CSP. Our implementation uses a weighted average strategy to select the camera viewpoint for multiple targets (addressing requirement 4). The average position between multiple points (targets) is evaluated mathematically in relation to the weights, rather than integrating this into the CSP.

This is done for two reasons: multiple-target situations constitute a minority of time (most of the time there is a single target); and the trade-off between computational complexity of solving it as a CSP and the frequency of multiple-target situations does not justify this representation for the single target case.

Constraint Solving

The camera’s position for each frame is determined by assigning values to variables such that the lowest cost solution is found. A series of potential solutions are processed to determine the lowest cost of solution, which is used for the camera’s position for each frame.

The cost of a constraint is determined by calculating the difference between the desired constraint value and the current potential solution value (Table 1 describes the cost calculations for our constraints). The total cost of a solution is the sum of all individual constraint costs.

The cost surface of the problem has many local minima (potentially-optimal solutions). Because strict adherence to the constraint values is not practical (requires damping or soft constraints), often the orientation constraint must be violated to some degree. This causes the problem to be over-constrained the majority of the time.

As there is often no perfect solution, the search is reduced to finding the best local minima. The large domain sizes required necessitate the use of a constraint solver that can

traverse large parts of the search space quickly. Stochastic local search algorithms are ideal for this purpose, and is the constraint solver used in our camera system.

```

generate random initial solution
for i = 0 to max flips
  if all constraints satisfied
    return assignment (solution)
  end if
  evaluate all possible moves from assignment
  select move according to heuristic
  enact move
end for

```

Figure 1: Local Search pseudo-code.

The generic form of the local search algorithm is shown in Figure 1. The constraint solver implemented in our system uses a modified form of this algorithm. Because the problem is over-constrained, a perfect solution can not be returned. In our method, the cost of the potential solution is calculated and the best solutions are kept. Once the search has made a pre-defined number of *moves* or attempts, the best solution found is returned as the optimal solution.

The solver operates on input about the target’s position and rotation (facing) values provided by the game. In order to address requirement 2 (reactive), the *height*, *distance* and *orientation* constraints are all single-frame constraints.

Due to the random nature of the search method, a mechanism must be put in place to avoid the camera randomly moving by small amounts when idle. To address this issue, we use a *frame coherence* constraint.

This constraint determines the distance the camera has moved in the previous frame, and attempts to maintain a level of coherency with this value for the distance to move in the current frame. The frame coherence constraint acts as a simple acceleration/deceleration control mechanism.

The use of the frame coherence constraint in combination with the constraint-weighted local search removes the necessity to pre-define the capabilities of the camera. The constraint setup and local search solver will always find the optimal trade-off between the constraints, so it is not necessary to set minimum/maximum limits for any physical camera movement properties.

Search Heuristics

The specialized nature of the problem creates the possibility of problem-specific search heuristics. These heuristics take

