Proceedings of the First
Artificial Intelligence and Interactive
Digital Entertainment Conference
(AIIDE-2005)

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 system[1]:

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

[1] This 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 ratio's 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).

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

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).

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

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

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

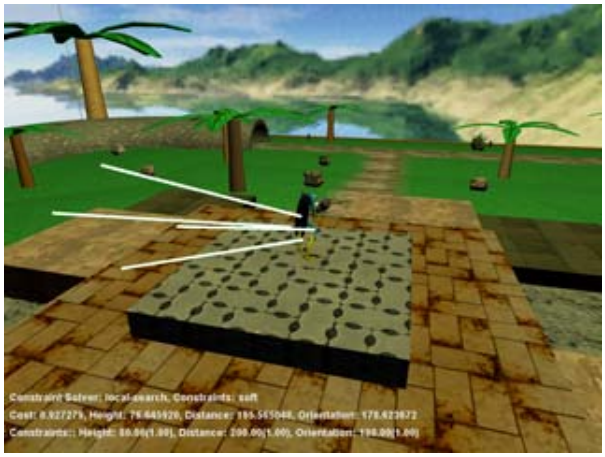

Figure 2: Occlusion avoidance rays.

advantage of the nature of the problem (ordered domains, geometric nature) to optimize the constraint solver.

The direction of the search (per variable) can be directed by the constraints during evaluation. This process is straightforward when dealing with a single variable that is affected by a single constraint (Y variable for height), but becomes more difficult for combined variable/constraint cases.

For example, if the current potential solution has a Y value lower than the desired height, the constraint can notify the solver that values lower than the current value will produce higher-cost solutions, and therefore do not need to be searched. The solver can then prune the domain of these values to avoid searching unnecessary parts of the search-space.

The distance and orientation constraints relate to both the X and Z variables. There is often a conflict between the direction to prune the search space for each constraint (orientation may want to increase, distance may want to decrease a variable). To address this issue, the distance and orientation constraints can work *co-operatively* or *competitively*.

The co-operative strategy prunes the domain only when both constraints agree on the pruning (i.e. both constraints desire larger X values). During contention (each constraint desires a different prune), the domain is not pruned and the solver is free to continue making moves.

The competitive strategy prunes the domain based on the dominant constraint (the constraint with the highest cost). This results in the constraint which is causing the highest cost contribution getting temporary control of the search to more quickly find a better (lower cost) solution.

Camera Behaviours

The setting of the weighting profiles can create different types of camera behaviours. These behaviours are achieved by manipulating the weight profile so that a constraint (or combination of constraints) is more likely to be satisfied.

The frame coherence constraint is particularly useful for controlling the 'aggressiveness' of the camera. A high frame coherence weight keeps the camera's movements smooth

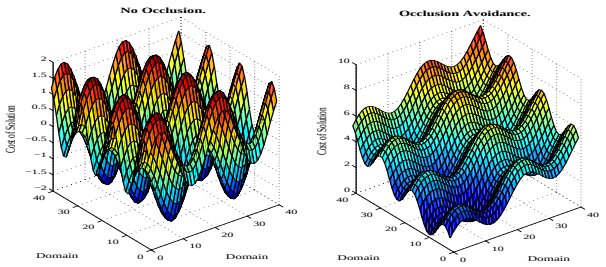

Figure 3: Visibility manipulated cost structure.

and controlled. The weighting keeps the camera stationary (or at motion) until there is substantial change in the target's position. A low frame coherence weight produces the opposite behaviour.

The weight profiles (and constraint values) can be arbitrarily switched during run-time. The constraint solver implicitly interpolates between the old and new values. Therefore, changes to the camera can be scripted into the game, with the camera smoothly interpolating between the values without explicitly performing this step.

Visibility Maintenance

Target visibility is evaluated by 4 rays cast from the target position to the camera's domain extents (top/bottom/left/right), shown in Figure 2. A fifth ray is used as the 'ideal' position and is used only when all four rays are occluded and the camera needs to move towards a 'safe' position. This is typically closer to the target than the distance constraint value (approximately half), but can be anywhere depending on the situation. For efficiency purposes, ray intersection tests are performed on the environment's bounding volumes in most cases. Enclosed objects (tunnels) require either sub-division of the bounding volumes or polygon-level intersection tests.

The occlusion value of a ray is used to artificially modify the cost of solutions closer to unoccluded positions. An example of this modification is shown in Figure 3. This figure is for illustrative purposes only, and does not accurately represent the search space.

The graph on the left shows a normal cost surface with no occlusion issues. There are a number of local minima which represent possible solutions to the problem. Each extent of the surface is not modified for visibility and the search takes place normally.

The graph on the right shows an example where three of the rays (corners) are occluded. The surface is now manipulated so solutions closer to the unoccluded value have lower costs than those in occluded areas. This gradually draws the camera away from the occluded positions, which often results in the constraints being increasingly violated.

Once the occlusion is resolved, the cost surface is returned to normal and the regular operation is resumed. The solver automatically interpolates the camera's position from the occlusion avoidance position towards the desired constraint values defining the visual properties of the target.

Figure 4: Screenshot of test environment.

Visibility maintenance is assigned a weight (like all constraints), however it is not considered a constraint. The weighting does not have any explicit meaning until there are occlusion issues and is simply used to guide how vigorously the camera moves away from occluded positions (higher weights produce a steeper slope) towards unoccluded positions.

A significant advantage of this method is that it is rare that explicit collision detection and response is required between the camera and the environment. The weighting of the visibility information causes the camera to naturally avoid contact with scene geometry, therefore reducing the risk of the camera getting 'stuck' to the environment.

The integrated nature of the visibility maintenance methods enables the ability to operate in arbitrary environments and domains. No modification is required to the constraint solver itself in order to add more sophisticated behaviours or occlusion avoidance strategies.

Evaluation and Results

To evaluate the performance of our camera system, we implemented it within a 3D game engine. The engine supports dynamic environments, and changing/multiple target objects. The experiments were run on an AMD Athlon 2800+ processor with 512Mb of RAM running Windows XP service pack 2. Our test environment (Figure 4) consisted of approximately 225,000 triangles and 350 bounding volumes.

The camera system was tested using a combination of scripted and interactive control of the target object. The scripted data allowed for fine-tuning of the constraint weights. The interactive control was used to prove the real-time performance and smooth movement without prediction.

Our experiments have shown that our camera system satisfies the real-time requirement adequately. The camera system averaged 0.00094 seconds per frame to perform all visibility (ray-casting) and constraint solving.

The major results for this type of work lie in the smoothness of movement and competency of the camera. This information is not quantifiable, and must be evaluated on animation information generated by the camera system. This evaluation is best done by experienced artists familiar with game design. Demonstrations of the competence of this implementation can be demonstrated during oral presentation.

The setup of our constraint solver for experimental purposes was as follows: *moves*=3000, *domain size*=10.0, using a uniform weighting profile of 1.0 for all constraints (11 parameters in total). These values represent a conservative approach to the camera system, and can be further fine-tuned to optimize the computational load of the camera implementation for specific game implementations.

Experimental studies have shown that the co-operative strategy outperforms the competitive strategy in most cases in terms of visual quality. This method is used in our implementation.

Performance Enhancements

The performance of the implementation is heavily dependent on the math libraries used. Our implementation uses modified versions of the math libraries described in (Van Verth & Bishop 2004). More efficient methods (or approximations) exist, but fast math functions are outside the scope of our research.

Further performance gains can be obtained by running the visibility evaluations (ray-casting) less often. Rather than evaluating for each frame, the visibility evaluations can be done every n frames, where n produces suitable results in the domain. Analysis of the constraint solver in the environment (how many moves to best solution) provides an indication of how many moves are required by the camera system to generate suitable visual results.

Dynamically choosing domain values that are close approximations of where the camera is likely to find an ideal position is also useful. This restricts the size of the domain producing fewer potential positions to search, thereby reducing the overall search time.

Comparisons to existing work is difficult due to differences in representations, constraint solving methods and test environments. Our simplified representation performs all of the visualization requirements of existing works (e.g. (Bares *et al.* 2000) and (Halper, Helbing, & Strothotte 2001)), and initial tests demonstrate the time benefits of our approach. However, a thorough empirical study is need to compare the results in an unbiased manner.

Conclusions & Future Work

This research has evolved from an initial desire to provide a simple autonomous camera to the development of a complex and flexible camera which can help define mood based on weighting profiles. The use of constraint-weighted local search has allowed us to exploit the expressive power of constraints while maintaining real-time speeds. This led to the development of a sophisticated and competent camera system with minimal computational overhead.

The camera system described in this paper provides a modular approach, which can be used in arbitrary systems without modification. Control of the camera is obtained

through values for the constraints, which eliminates the need to modify the constraint representation or constraint solver.

The modelling methodology and minimal constraint set, along with the weighted average targeting system successfully addressed all four requirements defined in the introduction. The use of soft constraints provided all of the benefits of constraint damping (including some new ones), while eliminating the possibility of oscillation.

We have shown an integrated and effective modelling of the camera control problem which provides significant advantages over existing representations. This reduced the problem of camera development to selecting an efficient and effective constraint solver. The representation allows for extensions to the camera's capabilities through new constraints, without the need to modify the representation or constraint solver.

There are a number of directions this research can take in future. Continued research into more efficient heuristics for the local search algorithm shall be investigated. The extension of the viewing properties of the camera through the introduction of new constraints can be investigated.

The work in this paper used hand-tuned constraint weights to generate the camera movements. The use of a genetic algorithm to generate constraint weights which cause the interactive camera to match a scripted camera movement is certainly worthwhile to be investigated.

The application of this camera system into a more sophisticated autonomous cinematography system will be investigated. By delegating the camera control to the constraint solver, an autonomous agent can be left to the more specific task of shot selection and mood determination. Both of these are easily achieved at the low-level by our camera system.

References

Bares, W. H., and Lester, J. C. 1999. Intelligent multi-shot visualization interfaces for dynamic 3D worlds. In *1999 International Conference on Intelligent User Interfaces (IUI'99)*, 119–126.

Bares, W.; McDermott, S.; Boudreaux, C.; and Thainimit, S. 2000. Virtual 3D camera composition from frame constraints. In *Eight ACM International Conference on Multimedia*, 177–186.

Bares, W. H.; Thainimit, S.; and McDermott, S. 2000. A model for constraint-based camera planning. In *AAAI 2000 Spring Symposium Series on Smart Graphics*, 84–91.

Beckhaus, S. 2001. Guided exploration with dynamic potential fields: The CubicalPath system. *Computer Graphics Forum* 20(4):201–210.

Bourne, O., and Sattar, A. 2004a. Applying constraint satisfaction techniques to 3D camera control. In Wallace, M., ed., *Principles and Practice of Constraint Programming (CP2004)*, 811. Toronto, Canada: Springer.

Bourne, O., and Sattar, A. 2004b. Applying constraint satisfaction techniques to 3D camera control. In Webb, G. I., and Yu, X., eds., *17th Australian Joint Conference on Artificial Intelligence*, 658–669. Cairns, Australia: Springer.

Christie, M.; Languenou, E.; and Granvilliers, L. 2002. Modeling camera control with constrained hypertubes. In Hentenryck, P. V., ed., *Principles and Practice of Constraint Programming (CP2002)*, 618–632. Ithaca, New York, USA: Springer.

Courty, N., and Marchand, E. 2001. Computer animation: A new application for image-based visual servoing. In *IEEE International Conference on Robotics and Automation (ICRA'2001)*, 223–228. Seoul, Korea: IEEE Press.

Drucker, S. M., and Zeltzer, D. 1994. Intelligent camera control in a virtual environment. In *Graphics Interface '94*, 190–199.

Freuder, E. C., and Wallace, R. J. 1996. Partial constraint satisfaction. *Over-Constrained Systems* 63–110.

Giors, J. 2004. The Full Spectrum Warrior camera system. In *Game Developers Conference*.

Halper, N., and Olivier, P. 2000. Camplan: A camera planning agent. In *AAAI 2000 Spring Symposium on Smart Graphics*, 92–100.

Halper, N.; Helbing, R.; and Strothotte, T. 2001. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum* 20(3).

Hornung, A. 2003. Autonomous real-time camera agents in interactive narratives and games. Master's thesis, Rheinisch-Westfälishce Technische Hochschule Aachen.

Marchand, E., and Courty, N. 2000. Image-based virtual camera motion strategies. In *Graphics Interface Conference (GI'2000)*, 69–76.

Marchand, E., and Courty, N. 2002. Controlling a camera in a virtual environment. *The Visual Computer* 18(1):1–19.

Stone, J. 2004. Third-person camera navigation. In Kirmse, A., ed., *Game Programming Gems 4*. Charles River Media. 303–314.

Tomlinson, B.; Blumberg, B.; and Nain, D. 2000. Expressive autonomous cinematography for interactive virtual environments. In *4th International Conference on Autonomous Agents (AGENTS-00)*, 317–324. Catalonia, Spain: ACM Press.

Van Verth, J. M., and Bishop, L. M. 2004. *Essential Mathematics for Games & Interactive Applications*. Morgan Kaufmann.