A Photographic Composition Assistant for Intelligent Virtual 3D Camera Systems

William Bares[1]

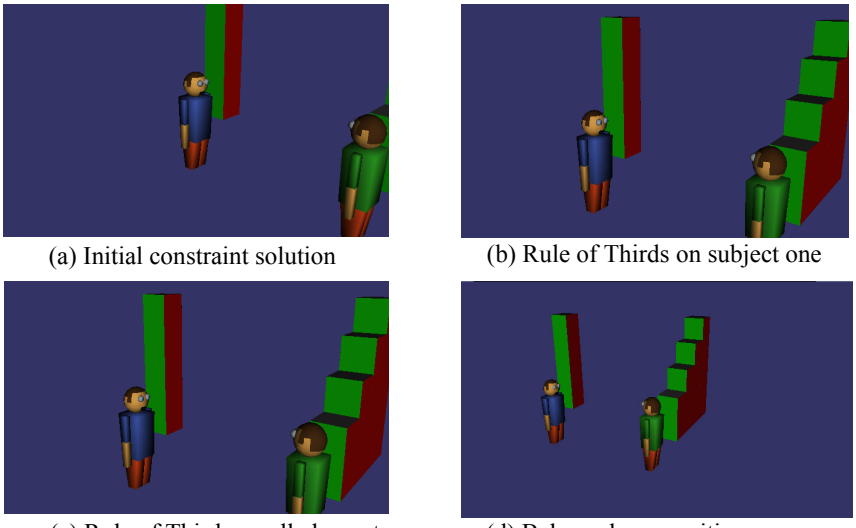
[1] Millsaps College, Department of Computer Science,
Jackson MS 39210, USA
{bareswh}@millsaps.edu
http://home.millsaps.edu/bareswh/

Abstract. A human photographer can frame an image and enhance its composition by visualizing how elements in the frame could be better sized or positioned. The photographer resizes elements in the frame by changing the zoom lens or by varying his or her distance to the subject. The photographer moves elements by panning. An intelligent virtual photographer can apply a similar process. Given an initial 3D camera view, a user or application specifies high-level composition goals such as Rule of Thirds or balance. Each objective defines either a One-D interval for image scaling or a Two-D interval for translation. Two-D projections of objects are translated and scaled in the frame according to computed optima. These Two-D scales and translates are mapped to matching changes in the 3D field of view (zoom), dolly-in or out varying subject distance, and rotating the aim direction to improve the composition.

1 Introduction

Virtual camera systems automatically compute camera shots using proven cinematography conventions to visualize computer-generated narratives, replay events, or dynamically update camera views so users may interact with a simulation. They analyze scenes to position the camera to view subjects with minimal occlusion from a desirable angle and distance. Consequently, much of the prior work has focused on geometric and temporal properties such as view angle, shot distance, minimizing occlusions, and finding collision- and occlusion-free camera motion paths. Composition is significant since a poorly composed shot detracts from its visual message. For example, a background object may not occlude the subject, but its color, size and location in the frame may carry greater visual weight inadvertently making it the apparent subject. Many automated virtual camera systems are directed only about how a specified set of subjects are to appear in the shot. Other than being instructed to avoid occlusion, non-subject objects are typically ignored. This method augments an existing application's virtual camera system with a composition assistant that analyzes the frame and proposes incremental changes to improve composition properties such as balance or the Rule of Thirds. For example, a typical 3D game camera system positions the camera at a relative view angle and distance, rotates the view angle to avoid occlusion, and aims at the center of the subject(s). This composition assistant adds a

"post processing" step to improve the composition by adjusting the action game's camera aim direction and distance-to-subject and/or field of view lens angle. For example, our constraint-based implementation of a typical 3D game camera is directed to compute a front-right angle view of a subject (at center of frame) whose height should span half that of the frame. Two unconstrained non-subjects are visible at the right edge (Figure 1a). Considering only the single constrained subject, use the Rule of Thirds to align it on lines that split the frame into thirds (Figure 1b). Considering all visible elements, the assistant can apply the Rule of Thirds for the whole composition (Figure 1c). The assistant evenly balances the "visual weight" of all elements about the vertical line through the frame center (Figure 1d). Larger elements have greater weight as do elements farther from center [14, 17].



(a) Initial constraint solution (b) Rule of Thirds on subject one

(c) Rule of Thirds on all elements (d) Balanced composition

Fig. 1. Examples of enhancing the composition of an initial solution

2 Related Work

Automatic camera assistants adjust camera position and aim to maintain an occlusion-free view of subjects [16] or project the center of a subject to a desired point on screen [8]. A method similar to this work analyzes a subject's silhouette to optimize aim direction and shot distance to obtain a satisfactory Rule of Thirds layout for a single subject [9]. The Virtual Cinematographer uses Blinn's equations [3] to project an actor to a given point in the frame. It improves a composition by moving virtual actors so they stand on their anticipated staging marks [13].

One method based on visual servos adjusts camera properties to track a moving target [6]. An approach optimized for computer games computes incremental changes to camera properties to maintain subject height, view angle, and visibility [10]. The Virtual Cameraman uses interval calculations to find camera paths whose interpolated positions satisfy desired image properties at specified points in time [12]. A variation on this method uses hypertubes to parameterize camera movements [5]. In another work a digital camera automatically detects the main subject and shifts the image to place it on the Rule of Thirds lines and blurs the background [1].

Constraint-based solvers are told how subjects should appear by view angle, size and location in the frame, and avoidance of occlusion. CAMDROID finds solutions using sequential quadratic programming [7]. CAMPLAN relies on genetic algorithms to satisfy specified visual properties including size, position, relative distance of subjects to the camera, and occlusion [11]. Bounding spheres, wedges, and planes can be used to limit the camera position search space for constraints of shot distance, view angle, and relative position of subjects, respectively [2]. Bounding volumes can also be used to semantically partition space into regions that yield compositionally distinct shots by shot distance, view angle, occlusion, and relative position of subjects [4]. Since constraint specifications are provided prior to computing a shot, it is impractical to specify composition constraints on non-subject objects since one does not know which ones will be visible in the shot.

3 Photographic Composition

Composition is the arrangement of visual elements in the picture frame. Texts suggest guidelines, but caution that experts sometimes break these "rules." [14, 17].

Emphasis: Simplify and reduce distracting clutter around the subject(s).

Placement or layout: The Rule of Thirds places subject(s) so they lie along the two horizontal and two vertical lines that divide the frame into nine equal-sized zones. More space should be left in front of a subject in motion.

Balance: *Balance* refers to distribution of the visual weight of the elements. Objects that are larger, brighter, closer to the edges, higher, and on the left side have more weight. Several subjects gazing towards one subject enhances its weight.

4 Composing in Two-Dimensions

The projections of all visible objects in the frame of an initial camera solution are approximated by a list of bounding rectangles. Rectangle coordinates are normalized with the bottom-left corner of the frame at (-*aspect*, -1) and the top-right corner at (+*aspect*, +1), where *aspect* is the aspect ratio of frame width divided by height.

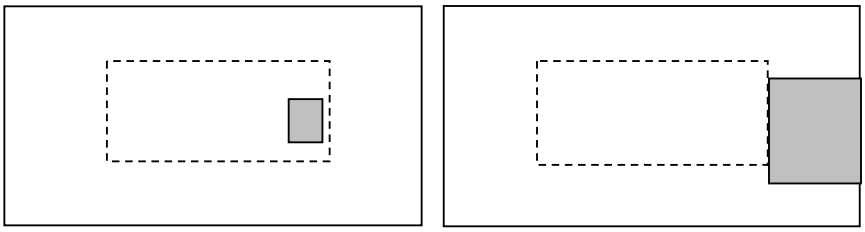
Given one or more user- or application-specified composition goal(s), such as balance or Rule of Thirds, the assistant applies a re-size or shift transformation to all elements. A re-size corresponds to a dolly or zoom of the virtual 3D camera and a shift corresponds to a pan. Each composition goal is expressed as one or more re-size and/or shift transforms of a particular element.

4.1 Resizing Composition Elements

Resize an element by multiplying its endpoints by a positive constant. This scale transform applies about an origin at the center of the frame. For example, scaling an element that lies to the right of center by a factor of 2 will double its size in the frame and also cause it to shift to the right. For each primitive re-size composition operator, find a `ScalarInterval` of scale factors [minimum, optimum, maximum] that achieves that objective. The following primitive scaling operators are implemented:

Scale to Fill Rectangle: Find `ScalarInterval` that scales an element so that it fills as much of the interior of a specified rectangle as possible.

Scale to Outside of Rectangle: Find `ScalarInterval` that scales an element so that it lies outside of the specified rectangle. For example, the interval of scale factors [minimum = 1.5, optimum = 1.5, maximum = +infinity] will scale the solid rectangle so that it lies outside of the dashed-line rectangle (Figure 2). This operator can be used to scale up to simulate a dolly-in or zoom-in to crop out a non-subject.



(a) Before scale to outside rectangle (b) After scale to outside rectangle

Fig. 2. Scale solid rectangle to lie entirely outside of dashed rectangle

Scale to Fit Inside Rectangle: Find `ScalarInterval` that scales an element so that it fits inside the specified rectangle.

Scale to Line: Find `ScalarInterval` that scales an element so that it lies on the specified line. The optimal scale places the element's center on the line.

Scale to Side of Line: Find `ScalarInterval` that scales an element so that it lies on the designated side of the specified line.

Scale to Area: Find `ScalarInterval` that scales an element so that its area lies within the specified [minimum, optimum, maximum] range.

Scale to Width: Find `ScalarInterval` that scales an element so that its width lies within the specified [minimum, optimum, maximum] range.

Scale to Height: Find `ScalarInterval` that scales an element so that its height lies within the specified [minimum, optimum, maximum] range.

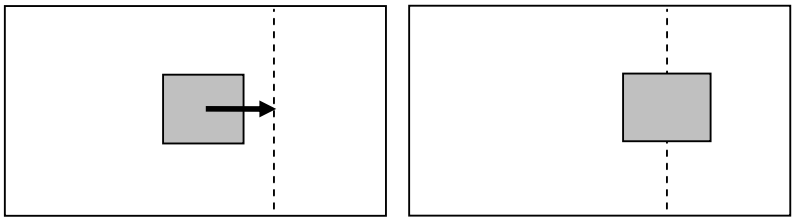
Scale to inside rectangle, scale to outside rectangle, scale to line, and scale to side of line can also be applied to transform points.

4.2 Shifting Composition Elements

Shift an element by adding a translation vector (dX, dY) to its endpoints. For each primitive shift composition operator, find a `RectangleInterval` [minimumDx, optimumDx, maximumDx] x [minimumDy, optimumDy, maximumDy] that bounds all translation vectors that achieve that objective applied to a rectangle or point element. For transforming a point, a threshold distance is also specified so the interval has sufficient width. The following primitive translation operators are implemented:

Move to Point: Find `RectangleInterval` that translates an element so that it lies on a specified point. The optimal (dX, dY) moves the element's center to the point.

Move to Line: Find `RectangleInterval` to translate an element so its center optimaly lies on a specified vertical or horizontal line. For example, the arrow is the optimum translation of the solid rectangle (its width is 0.8 units) so that its center lies on the vertical line. The horizontal dimension of the `RectangleInterval` is [0.15, 0.55, 0.95] meaning shift right by no less than 0.15, ideally by 0.55, and no more than 0.95. The vertical dimension of the `RectangleInterval` is [-infinity, 0, +infinity] (Figure 3).



(a) Before move to line (b) After move to line

Fig. 3. Translate rectangle so its center lies on a vertical Rule of Thirds line

Move to Side of Line: Find `RectangleInterval` that translates an element so that it lies on the designated side of a specified vertical or horizontal line.

Move to Inside Rectangle: Find `RectangleInterval` that translates an element so that it lies inside a specified rectangle. This transform can determine the range of translations that keep an element entirely inside the bounds of the frame.

Move to Outside Rectangle: Find `RectangleInterval` that translates an element so that it lies outside a specified rectangle.

4.3 Measuring Visual Weight

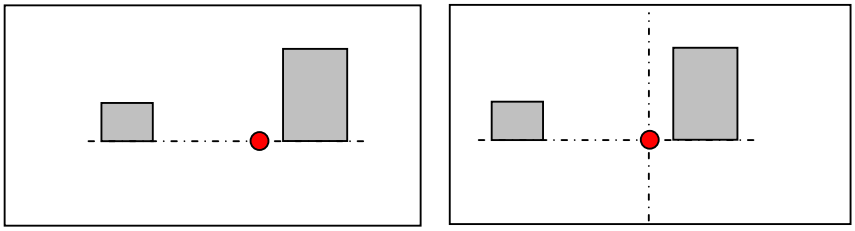
Compute the weight of each visible element in the frame by accumulating the following components, each of which is evaluated as a normalized value between 0 and 1.0.

brightnessWeight = maximum intensity(red, green, blue of element's color)
The application specifies the predominate color of each subject
horizontalWeight = AbsoluteValue(element.CenterX()) / (0.5 * frame.width())
Increase by 10% if element is left of frame center
verticalWeight = (element.CenterY() - frame.minY()) / frame.height()
sizeWeight = element.diagonalLength() / frame.diagonal()
gazeWeight = number of elements pointing to this element / (numElements-1)

For gaze, compute the angle between element E's projected heading vector and a vector directed from the center of element E to the center of subject being weighed.

4.4 Balancing the Composition

Translate elements so the Center of Visual Weight (CoVW) coincides with a specified horizontal U-axis coordinate H. In Figure 4 achieve formal balance by translating Center of Visual Weight to the vertical line through H = 0 at the frame center. The assistant can be instructed to move the Center of Visual Weight to lie on a specific vertical line to place greater weight off center for informal or dynamic balance.



(a) Find Center of Visual Weight (b) Pan to shift CoVW to mid line

Fig. 4. Achieve formal balance by shifting elements so CoVW is at horizontal center

Given that the visual weight of each element is w_i and the horizontal u-coordinate of each element is u_i. To compute the u-coordinate of the Center of Visual Weight:

$$centerU = \sum_{i=1}^{num\_elements} w_i * u_i$$

The v-coordinate of the Center of Visual Weight is found in a similar fashion.

4.5 Compound Composition Goals

For example, suppose we want leading space (60% of frame width) ahead of the subject in the center of Figure 1a and exclusion of non-subjects. To optimize the virtual camera's aim direction, construct an AND/OR Tree for the desired translate transforms. To leave more space to the right of our subject, find the Move to Side of Line `RectangleInterval` that moves the subject's bounding rectangle to lie on the left side of a vertical line left of frame center. Keep the subject entirely visible in the frame by a Move to Inside Rectangle transform. In this example, the user chooses to exclude all non-subjects. Exclude each non-subject (second character and boxes) with a Move to Outside of Rectangle transform which is implemented by a disjunction of four Move to Side of Line transforms, one per edge of the frame. Figure 5 depicts the AND/OR Tree to find a single translation to pan the camera. Internal nodes represent AND or OR operations. An AND node may be "weak" meaning that satisfying a subset of its children is acceptable. Leaf nodes contain `RectangleInterval` transforms. Only one of the five OR nodes to exclude one of the four stacked boxes on the right is depicted. Intervals are marked with benefit scores, which propagate upwards. Sort child nodes by descending score to facilitate finding a partial solution with the greatest benefit when unable to satisfy all objectives.

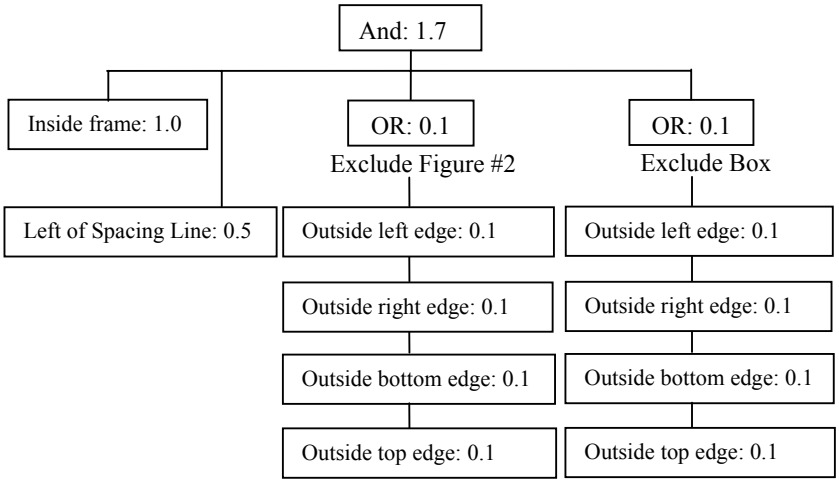


Fig. 5. AND/OR Tree to pan to give leading space and exclude non-subjects

In this example, the pan can only satisfy leading space ahead of the subject. Update the aim direction and re-project the bounding rectangles of all elements. Construct a similar separate AND/OR Tree to optimize the dolly distance and field of view/zoom using `ScalarInterval` leaf nodes. This second pass finds a scale transform that successfully zooms-in to exclude all but one of the non-subjects (Figure 6).

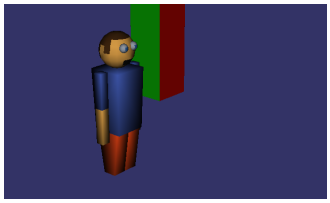


Fig. 6. Refined composition for leading space and exclusion of non-subjects

5 Optimizing Composition Transforms

The same optimization algorithm is used to compute the optimal interval representing either a range of possible scale factors to resize the composition elements or a range of possible translation factors to pan the composition elements. Each OR-node is treated as a variable whose value represents the selection of one of its child nodes. The backtracking constraint-satisfaction algorithm finds a consistent assignment of values to each OR-node. Figure 5 represents a problem with two variables, one per OR node. Each OR node has four possible values or directions in which to pan the camera to exclude an element. Initially no variables are assigned values.

Let `maxSolution` be the empty interval with a score of 0.
Let the initial `solution` be an empty interval with a score of 0.

```
while ( is unassigned variable ) {
    do {
      For current variable, assign next untested value
      Evaluate AND/OR Tree to find solution interval
    } while(solution is inconsistent and
            untested values remain for current variable )

    if( consistent value is found for current variable )
    {
       current variable = next unassigned variable
       if(solution.score() > maxSolution.score())
         maxSolution = solution
    }
    else backtrack to prior variable with untested value
}
return maxSolution
```

The do-while loop tries OR-node values until it finds a consistent solution. A solution is consistent if the intersection of all AND-node child intervals along with the intervals corresponding to the current assignment of OR-node variables is non-empty. If an AND node is specified as being weak, the algorithm excludes the lowest priority non-intersecting intervals to account for partial solutions in cases of conflicts. If a consistent solution is found, then compute the benefit score for the successfully intersected intervals. The optimum value of the result interval is a weighted average of the optimum values of the two intervals being intersected. If either input interval has an identity optimum, 1.0 for scale or (0, 0) for translation, then the result optimum is taken from the other input interval. The importance of the result is taken as the sum of the importance of the input intervals. An overall importance is propagated up the tree to rank solutions. Record the non-empty solution interval having the maximum benefit score. The `maxSolution` interval is returned and is used to find an equivalent camera dolly, zoom, or pan to improve the composition.

6 Mapping Solution Intervals to Camera Parameters

6.1 Camera Field of View Angle for Scale

Given a positive *scale* factor to re-size all elements in the frame, find the vertical field of view angle in radians spanning the full field of view that yields a zoom in or out.

Let `pHeight` be the height of the front face of the perspective frustum. Let `pDistance` be the distance from the camera point to the front face of the frustum.

```
FOV = 2.0 * atan( 0.5 * pHeight / (scale * pDistance) )
```

6.2 Camera Dolly Distance for Scale

Given a *scale* factor to re-size all elements in the frame, find the signed dolly distance to translate the camera along its unit-length aim vector. Given reference point `RefXyz` at the center of the 3D subject geometry. Find `RefUvn` by converting into UVN Camera Coordinates with the v-coordinate adjusted to be non-zero if necessary.

Let `pDistance` be the distance of the projection plane to the camera position.

```
v = pDistance * refUvn.v() / refUvn.n()
scaledV = scale * v
```

Using similar triangles, `scaledV / pDistance and refUvn.v() / (refUvn.n() + dolly)`, find the dolly distance:

```
dolly = (pDistance*refUvn.v() – v*scale*refUvn.n()) /
        (v * scale)
```

6.3 Camera Aim Direction for Translation

Given a translation (dU, dV), expressed in normalized frame coordinates, find the new camera aim direction to pan the frame. A reference point RefXyz at the center of the subject geometry is given. Compute the new aim direction N-axis as follows:
`v = dV * 0.5 * pHeight`

Find length h of hypotenuse vector H of the right triangle formed by camera position, translated point (dU, dV) and yet-to-be-determined N-axis.
`h = sqrt( v*v + pDistance*pDistance )`
Let `camPosToObj = refXyx – cameraPosition`
`H = h * camPosToObj`
Set local camera system U-axis vector ("right hand of virtual camera person").
`U = H x UP`, where UP is the world's global "up" vector.

Find angle in radians between vectors H and yet-to-be-determined aim vector N.
`angle = asin( v / h )`

Let Q be the quaternion representing the counter-clockwise rotation about U by angle. Convert the quaternion into an equivalent 4x4 rotation matrix.
`rotMatrix = Q.convertTo4x4Matrix()`
Rotate H counterclockwise by angle about axis U to form updated N vector.
`N = rotMatrix.transformVector(H)`

Find u, the projection of dU onto the perspective frustum.
`u = dU*(0.5*pWidth/aspectRatio)`
Find length h of the hypotenuse of right triangle formed by camera position, translated point (dU, dV) and to-be-determined N-axis.
`h = sqrt( u*u + pDistance*pDistance )`

Angle in radians between vectors H and N-axis: Θ = `-asin( u / h )`

Let `V = U x N.` Q is the quaternion to rotate N clockwise by Θ about axis V.
`rotMatrix = Q.convertTo4x4Matrix()`
`N = rotMatrix.transformVector(N)`

Normalize the new local camera coordinate system axes vectors U, V, and N. Vector N becomes the new camera aim direction vector, U its "right hand", and V its "hat".

7 Implementation

Composition goals are specified via a menu interface. A constraint solver computes an initial solution for projection height, view angle, and avoid occlusion of one subject. In all examples, the preferred view angle is occlusion-free so it only performs ray-to-box queries. If the desired view is occluded the solver would change the view

angle. The Open Scene Graph API performs the rendering. Table 1 gives C++ benchmarks on a 2.66 GHz Intel Xeon running LINUX. Columns represent variants of the scene with differing numbers of similarly configured boxes. The column labeled "5 boxes" gives the benchmarks for the images shown in the figures.

Table 1. Average time to compute camera solution for selected figure screen shots

Figure	1 box	5 boxes	10 boxes
Fig. 1(b)	2.2 ms	4.0 ms	6.6 ms
Fig. 1(c)	3.4 ms	8.2 ms	14.8 ms
Fig. 1(d)	3.4 ms	7.0 ms	12.4 ms
Fig. 6	3.4 ms	6.0 ms	9.8 ms

8 Conclusions and Future Work

This method is fast and easy to install into existing automated camera systems. The method can be used to implement projection size and location constraints for a basic automated camera system. The example implementation does just that. Future work includes fully automating the selection and benefit weighting of composition goals by computing visual attention saliency of visible elements. Unify the separate translate and scale passes by formulating the composition objectives in terms of how to reposition and/or re-size the rectangular bounds of the "picture frame" in one step. Cluster nearby non-subject elements to improve efficiency.

9 Acknowledgements

Thanks to Arnav Jhala, James Lester, Charles McClendon, and the anonymous reviewers for their suggested improvements.

References

1. Banerjee, S., Evans, B.L.: Unsupervised Automation of Photographic Composition Rules in Digital Still Cameras. In Proc. IS&T/SPIE Conf. on Sensors, Color, Cameras, and Systems for Digital Photography, Jan.18-22 2004, vol. 5301, San Jose, CA (2004) 364-373

2. Bares, W., McDermott, S., Boudreaux, C., Thainimit, S.: Virtual 3D Camera Composition from Frame Constraints. In *ACM Multimedia 2000*, Los Angeles, California, October 30th to November 4th (2000) 177-186

3. Blinn, J.: Where am I? what am I looking at? IEEE Computer Graphics and Applications. (1988) 76-81

4. Christie, M., Normand, J-M.: A Semantic Space Partitioning Approach to Virtual Camera Control in Proceedings of the Annual Eurographics Conference, Computer Graphics Forum, Volume 24-3, (2005) 247-256

5. Christie, M., Languénou, E., Granvilliers, L.: Modelling camera control with constrained hypertubes. In Hentenryck, P. V., editor, Principles and Practice of Constraint Programming CP2002 (Lecture Notes in Computer Science), (2002) 618-632

6. Courty, N., Marchand, E.: Computer animation: A new application for image-based visual servoing. In Proceedings of IEEE Int. Conf. on Robotics and Automation, ICRA'2001, volume 1, (2001) 223-228

7. Drucker, S. M., Zeltzer, D.: Intelligent camera control for virtual environments. In Graphics Interface '94. Morgan Kaufmann Publishers (1994) 190-200

8. Gleicher, M., Witkin, A.: Through-the-lens camera control. In Proceedings of ACM SIGGRAPH'92, (1992) 331-340

9. Gooch, B., Reinhard, E., Moulding, C., Shirley, P. Artistic Composition for Image Creation. In Eurographics Workshop on Rendering, (2001) 83-88

10. Halper, N., Helbing, R., Strothotte, T.: A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. In Proceedings of the Eurographics'2001 Conference, volume 20, (2001) 174-183

11. Halper, N., Olivier, P.: CAMPLAN: A Camera Planning Agent. In *Smart Graphics. Papers from the 2000 AAAI Spring Symposium (Stanford, March 20-22, 2000)*, Menlo Park, AAAI Press, (2000) 92-100

12. Jardillier, F., Languénou, E.: Screen-space constraints for camera movements: the virtual cameraman. In Eurographics '98, volume 17, Computer Graphics, Special Issue (1998) 174-186

13. Li-wei He, Cohen, M.F., Salesin, D.H.: The virtual cinematographer: A paradigm for automatic realtime camera control and directing. In *Computer Graphics (Proceedings of SIGGRAPH '96)*, (1996) 217-224

14. O'Brien, M., Sibley, N. The Photographic Eye: Learning to See with a Camera. Davis Publications, Inc., Worchester, Massachusetts (1995)

15. Olivier, P., Pickering, J., Halper, N., Luna, P.: Visual composition as optimisation, *AISB Symposium on AI and Creativity in Entertainment and Visual Art*, Edinburgh (1999) 22-30

16. Phillips, C.B., Badler, N., Granieri, J.: Automatic viewing control for 3D direct manipulation. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, March (1992) 71-74

17. Ward, P.: Picture Composition for Film and Television. Focal Press, Woburn, Massachusetts (1996)