Virtual 3D Camera Composition from Frame Constraints

William Bares Scott McDermott Christina Boudreaux Somying Thainimit
Center for Advanced Computer Studies
University of Lousiana at Lafayette
337-482-5697
whbares@cacs.louisiana.edu

ABSTRACT

We have designed a graphical interface that enables 3D visual artists or developers of interactive 3D virtual environments to efficiently define sophisticated camera compositions by creating storyboard frames, indicating how a desired shot should appear. These storyboard frames are then automatically encoded into an extensive set of virtual camera constraints that capture the key visual composition elements of the storyboard frame. Visual composition elements include the size and position of a subject appearing in a camera shot. A recursive heuristic constraint solver then analyzes the space of a given 3D virtual environment to determine camera parameters which produce a shot closely matching the shot in the given storyboard frame. For example, developers of interactive 3D virtual environments can create storyboard frames that visually express how the virtual camera should film a given set of objects. Our heuristic constraint solver takes a given storyboard frame, analyzes the current state of a potentially dynamic, unpredictable 3D virtual environment, and computes a camera shot that closely matches the desired shot depicted by the user in the storyboard frame. This enables developers of interactive 3D environments to specify desired virtual camera shots In contrast, existing methods of automatically positioning cameras in 3D virtual environments typically rely on pre-defined camera placements that cannot account for unanticipated configurations and movement of objects.

Keywords

Animation, Animation w/Constraints, User Interfaces.

1. INTRODUCTION

Automatically planning camera shots in virtual 3D environments requires solving problems similar to those faced by human cinematographers. In traditional cinematography parlance, a *shot* refers to a continuous stream of *frames* (individual images) recorded by a given camera. In the most essential terms, a shot typically communicates some specified visual message or goal. Consequently, the camera must be carefully staged to clearly view the relevant subject(s), properly emphasize the important elements in the shot, and compose an engaging image that holds the viewer's attention [20, 23]. Expert cinematographers and film directors often design the initial concept of a camera shot using a *storyboard frame*, or rough sketch of how the shot should appear [17]. It in effect defines the most essential visual composition elements of a shot. Visual composition elements include the size and position of a subject appearing in a camera shot. Given a storyboard frame, a camera operator then carefully surveys the given setting and chooses a camera position, orientation, and lens angle to create a shot closely matching that in the given storyboard frame. This division of labor allows the cinematographer to express the appearance of the shot at a high level leaving the details of camera placement to the camera operator. In addition the camera operator must be able to think on his or her feet by adjusting the camera to account for unanticipated obstacles or arrangements of objects [14].

Users interact with three-dimensional virtual environments by viewing the scene through the "eyes" of a virtual camera. A virtual 3D camera view is defined by parameters of camera position, aim direction vector, and field of view (lens angle). In many 3D applications, users directly control the camera view. Applications featuring complex interaction tasks or fast-moving action often delegate the task of camera control to automatic computer control allowing the user to focus on the task at hand.

Our approach to automatically computing camera shots in virtual 3D environments employs a constraint-based methodology. The desired appearance of a camera shot in a virtual 3D environment is encoded by constraints declaring how subjects should appear in the frame. For example, set the camera to obtain a close-up shot in which the subject fills the frame with the camera viewing its front right side. A constraint solver module then attempts to automatically find values for each camera parameter so that all constraints are satisfied in the context of a given 3D virtual environment. In contrast, some less sophisticated methods of automatically placing cameras in virtual 3D environments typically define shots by in effect pre-coding the camera parameters to properly view a set of anticipated situations. Constraint-based approaches have the flexibility to analyze the space of possible camera parameter values to successfully find satisfactory shots in spite of viewing obstructions or unanticipated object configurations typical in dynamic interactive 3D worlds.

A constraint-based virtual camera system should provide the following basic features if it is to be effective in a wide-range of applications and virtual 3D environments.

- *Visual Constraint Editor:* Provide a graphical WYSIWYG interface allowing users to directly draw how a desired camera shot is to appear. Constraints are then automatically extracted from the user's storyboard frame drawing and exported to the constraint solver system.

- *Environmental Analysis:* The constraint solver should be able to analyze a virtual 3D environment to account for viewing obstructions or objects which may move into an infinity of possible configurations in dynamic or interactive 3D virtual environments.

- *Evaluation of Shot Quality:* The constraint solver should evaluate computed shots to determine how effectively they satisfy the desired visual constraints.

- *Interactive Performance:* The constraint solver should be able to compute solutions in real-time or near real-time to facilitate interactive 3D applications.

1.1 Current State of the Art

Automatic camera control assistants vary camera position to avoid occlusions of a goal object or satisfy screen-space constraints on how subjects appear on-screen [11, 24]. Automated camera navigation assistants adjust camera speed based on distance to the target or guide the camera along specified optimal vantages as a user navigates over a terrain [13, 19, 26, 27]. Both automated viewing and navigation assistants focus on controlling the camera at a low-level and frequently require considerable user inputs.

Other systems define camera shots by displacements relative from the subject(s) being viewed [1, 8, 16, 25]. For example, the "chase plane" view found in typical flight or driving simulators and popular third-person 3D adventure games automatically positions the camera a set distance behind and slightly above the player's vehicle or character. IBIS features multi-view illustrations and cutaways of occluding obstructions and CATHI has a facility for transparency [4, 9]. Idiom-based planners employ hierarchical encodings to sequence shots of anticipated actions such as conversations between small groups of players [4, 5, 16, 18]. Existing idiom-based systems use variants of the pre-specified relative camera method in lieu of more complex camera placement solvers. The Virtual Cinematographer employs Blinn's method to stage the camera to project an object to a given point in the frame [3]. It can also reposition objects that are not in the anticipated locations to improve camera shots [18]. One recent effort employs a cognitive forumulation of the Virtual Cinematographer's Finite State Machines to film autonomous virtual creatures [10]. However, each of these systems can fail to find acceptable shots when multiple subjects occupy or move into unanticipated relative spatial configurations, structures in the world occlude the subject(s) of interest, or users wish to view unanticipated types of shots.

The CAMDROID constraint system supports a powerful set of camera constraints, but employs a numerical constraint solver that is subject to local minima failures [6, 7]. CAMPLAN utilizes genetic algorithms to generate-and-test candidate camera shots by "mating" those sets of camera parameters which best satisfy its thirteen types of shot constraints [12, 22]. A third constraint-based solver features a systematic solution for handling constraint failures by creating multiple shot solutions or relaxing less important constraints, but supports only four types of constraints [2]. Virtual Camera Man solves small sets of interval constraints to generate animations offline [15]. Existing constraint-based systems have yet to demonstrate real-time performance combined with the expressive power to create artful photographic compositions and a graphical interface to facilitate development of constraint specifications.

1.2 Photographic Composition

Our camera constraints were based on established knowledge in photography and cinematography. Like photographs and shots in motion pictures, camera shots in virtual 3D environments should be composed to clearly depict the objects and actions of interest to the user. *Composition* is the artful arrangement of visual elements in an image [20, 21, 23]. These visual elements include:

- *Subject size:* The size of a subject in the frame is expressed in terms extreme close-up, close-up, medium, long, and extreme long shots in order of decreasing subject size. Nearer camera positions or narrower fields of view yield larger subject sizes.
- *Location:* Subjects may be carefully positioned in the frame to determine visual weight, balance, or emphasis. For example, subjects may be framed so that they form a triangle with the dominant subject at the apex of the triangle.
- *View angle:* The relative orientation from which the camera views the subject. For example, the camera may be positioned at a 45-degree angle from the front of the subject to create a *three-quarter shot.*
- *Occlusion:* Allowing subjects to partially overlap one another can provide an increased degree of interest or additional depth cues.
- *Exclusion:* Good compositions typically exclude unimportant or distracting objects from the frame.
- *Depth:* The camera can be positioned so that subjects lie at varying distances from the camera to better reveal the spatial distribution of objects in a scene.

2. STORYBOARD FRAME INTERFACE

A developer of an interactive 3D virtual environment would begin by creating constraint-based definitions of the types of shots needed. Our constraint solver module, integrated into a 3D application, would analyze the environment to compute camera shots. We have developed a graphical interface to facilitate the creation of constraint-based shot definitions. It is far more intuitive to manipulate an object's appearance so its size in the frame defines a medium shot rather than using a constraint program script to specify that the object should cover 10% of the frame's area. The storyboard frame view (left half of Figure 1) is augmented by overhead or side views (right half of Figure 1) to allow the artist to indicate depth relationships between objects with respect to the camera.

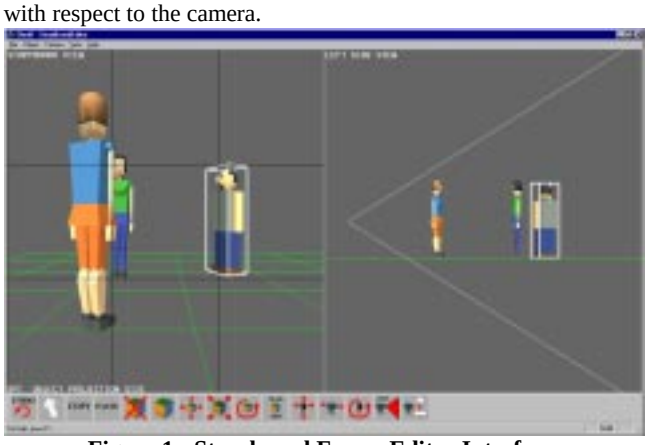
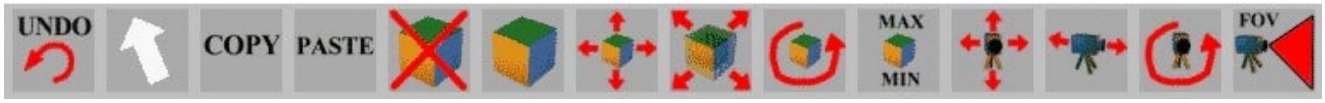


Figure 1: Storyboard Frame Editor Interface.



(a) (b) (c) (d) (e) (f) (g)

Figure 2. Storyboard Frame Editor Toolbar

The Storyboard Frame Editor provides the following functionalities via drop-down menus or a toolbar (Figure 2).

(a) Object Edits. Basic operations to select, copy, paste, delete, and create new storyboard objects.

(b) Location. Move the currently selected object in the storyboard frame view to vary its location in the frame. Move an object in the overhead or side views to vary its relative distance from the camera causing a corresponding increase or decrease in size in the storyboard frame view.

(c) Size. The size of the currently selected object in the storyboard frame view may be varied by dragging the mouse also producing a corresponding decrease or increase in the object's relative distance from the camera in the overhead or side views.

(d) View Angle. Rotate the currently selected object to set the relative viewing angle between it and the camera.

(e) Minimum-Maximum Allowable Ranges. By default the previous object edit operations (location, size, and view angle) specify the optimal framing attributes for a given object in the shot. In complex, unpredictable, or dynamic interactive 3D environments it may not always be possible to obtain the optimal shot framing so our constraint-based method permits satisfactory solutions that are within a specified threshold away from the optimal shot depicted in the Storyboard Editor frame view. By enabling a toggle switch, the user can edit the allowable ranges of values for the selected attribute of the current object. An object's range of allowable minimum and maximum sizes in the frame are represented by two rectangular outlines. The outermost rectangle represents the maximum size that an object may assume in the frame, while the innermost rectangle represents its minimum size. The range of allowable locations in the storyboard is represented by one bounding rectangle (Figure 4). A transparent spherical patch surrounding the current object represents the range of allowable view angles of the camera relative to an object.

(f) Camera Pan, Dolly, and Rotate. The storyboard camera can be moved about to simultaneously change the framing properties of all objects in the shot. A pan moves the camera laterally, while a dolly moves the camera towards or away from the objects, and rotate swings the camera around the center of the object(s).

(g) Field of View. Change the field of view angle (zoom factor) of the camera lens to simultaneously vary the sizes of all objects.

(h) Field of View Inclusion and Occlusion. If an object is partially outside the frame, then this fact may be noted for export to the constraint system. If an object is partially occluded by another object, the user can specify that this overlap is required for a constraint solution (this option only available in the menu).

(i) Design Guides. Users may toggle the display of the "Rule of Thirds Lines" or a horizon plane. The Rule of Thirds suggests placing objects on or at the intersection of horizontal and vertical lines that split the frame into thirds in either dimension [21]. The horizon or ground plane can be used to assist in aligning objects.

3. CAMERA CONSTRAINTS

The camera constraint system supports fifteen types of constraints on either the camera's attributes or how objects appear in the camera shot. Constraint types are designed to support the photographic composition elements from Section 1.2. Constraints specify optimal and allowable ranges of values for compositional attributes including subject size in the frame and relative view angles. Constraints may be applied to one (primary object) or two specified objects (primary and secondary objects) or to the camera itself. Some object constraints, may opt to apply the constraint to a designated point or region of space (bounded by a sphere) displaced from an object's midpoint. This optional construct is referred to as a *locus modifier*. For example, the object projection size constraint can apply a locus modifier to a figure's head requiring its projection to fill a given fraction of the frame.

3.1 Constraint Types

The following lists the constraint types featured in this paper.

1. OBJ_PROJECTION_SIZE: requires that the projection of the primary object's *projection source* cover a specified fraction of the frame. An object's projection source may be either its bounding box or its optional locus modifier sphere.

2. OBJ_PROJECTION_ABSOLUTE: Requires that the specified projection source of the primary object project to lie entirely inside a given rectangular region of the frame. The projection source may be a point, locus sphere, or bounding box.

3. OBJ_VIEW_ANGLE: requires the camera to lie at a specified orientation relative to the untransformed primary object. This relative orientation is expressed in spherical coordinates (horizontal angle theta -180° to 180° and phi -90° to 90°).

4. OBJ_IN_FIELD_OF_VIEW: requires the specified primary object to lie entirely or partially in the camera's field of view.

5. OBJ_OCCLUSION_MINIMIZE: requires that no more than the specified maximum allowable fraction of the given primary object is occluded by other object(s).

6. OBJ_OCCLUSION_PARTIAL: requires the fraction of the primary object occluded by other opaque object(s) to range between specified minimum and maximum allowable values. It is also possible to specify which object should contribute occlusion.

7. OBJ_EXCLUDE_OR_HIDE: requires that the primary object is either completely occluded by other object(s) or lies entirely outside the camera's field of view.

8. CAM_POS_IN_REGION: requires that the camera position lie within the specified region of space, which can be defined by one or more box, sphere, or space partitioning plane regions combined in a hierarchy using union and intersection operations.

Other implemented constraint types include relative projection locations of two objects in the frame (e.g. above, below, left of, or right of, etc.), relative depth of two objects from the camera, field of view angle, and aim the camera at a specified point.

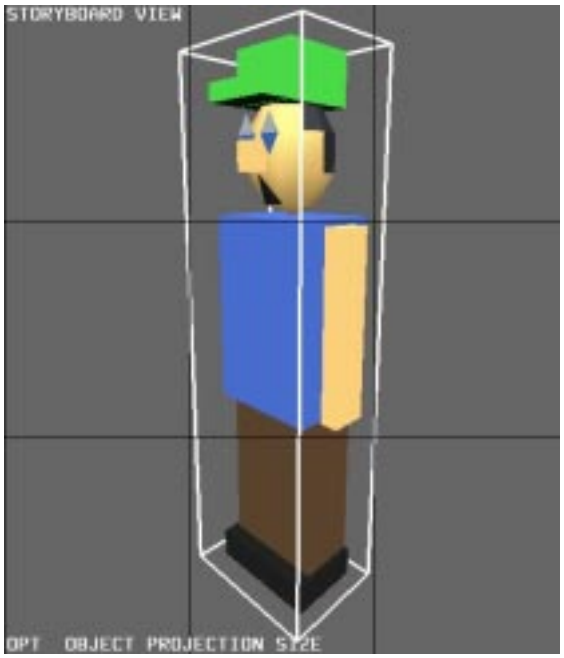
3.2 Storyboard Frame to Constraints

The Storyboard Frame Editor exports shot definition files, which are loaded into a separate 3D application program equipped with our constraint solver. Two forms of shot definition are exported when the user saves a shot in the Storyboard Frame Editor.

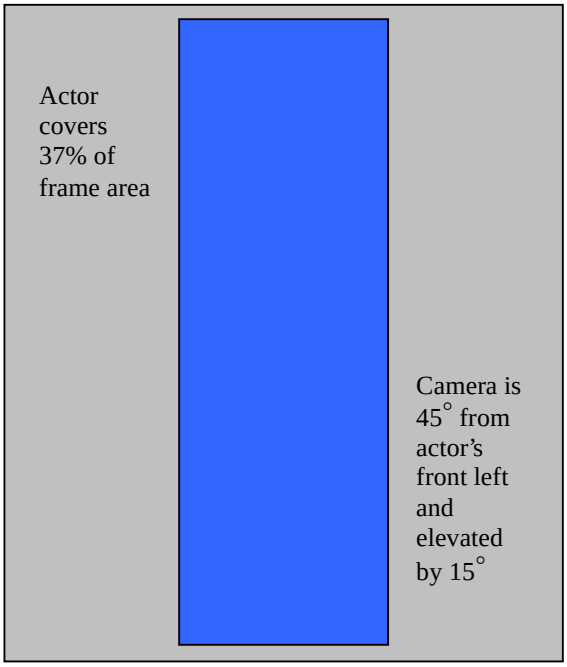
Relative Displacement Definition: The camera position and aim direction are expressed as vectors in a local coordinate system anchored at the first object in the storyboard frame. Our constraint solver includes an implementation of relative displacements to permit an instantaneous solution if the configuration of objects in a scene happens to match those of the Storyboard Frame Editor's internal 3D scene model.

Constraint Definition: The Storyboard Frame Editor automatically exports a set of constraints that define the key visual elements of the shot. These constraints are requirements on how various objects are to appear in the desired shot. Exported constraint definitions are scene independent allowing the same set of storyboard constraints to be used in a variety of different 3D virtual environments. A constraint-based shot definition may also be used to film differing configurations of objects in the same 3D virtual environment.

For example, the storyboard frame in Figure 3(a) depicts a medium three-quarter shot of a virtual actor. The user has manipulated the actor's appearance in the frame view to visually define the desired shot. Select the view angle tool and rotate the object so the camera views the front-left side of the figure. Next, use the size tool to move the object towards or away from the camera until the object appears at the desired size in the frame. With the size tool still selected, enable the minimum-maximum toggle switch to specify first the minimum allowed object projection size and then the maximum allowed object projection size. Do so by stretching the bounding rectangles marking the minimum and maximum allowed projection sizes (Figure 4). The Storyboard Frame Editor automatically extracts the essential visual composition constraints that define this shot. In this case, the image of the virtual actor covers 37% of the frame area and the camera is oriented 45 degrees from the actor's front left and is elevated by 15 degrees. Figure 3(b) illustrates an abstract representation of the constraint parameters for projection size and view angle. The parameter values for projection sizes (optimal, minimum, and maximum) are taken by dividing the area of the object or bounding rectangles by the total frame area. Values are encoded for the desired optimal values and allowable ranges for the virtual actor's OBJ_PROJECTION_SIZE, OBJ_PROJECTION_ABSOLUTE and OBJ_VIEW_ANGLE constraints. Projection location parameters are encoded with (0,0) as the lower left corner of the frame and (1,1) as the top right corner. The constraints automatically extracted from the Storyboard Editor frame view in Figure 3 are given in the constraint script file listing on the following page (Listing 1). Note, a constraint to minimize occlusion of a subject is automatically exported unless otherwise specified by the user.



(a) Storyboard frame view of medium three-quarter shot.



(b) Abstraction of constraints extracted from frame view.

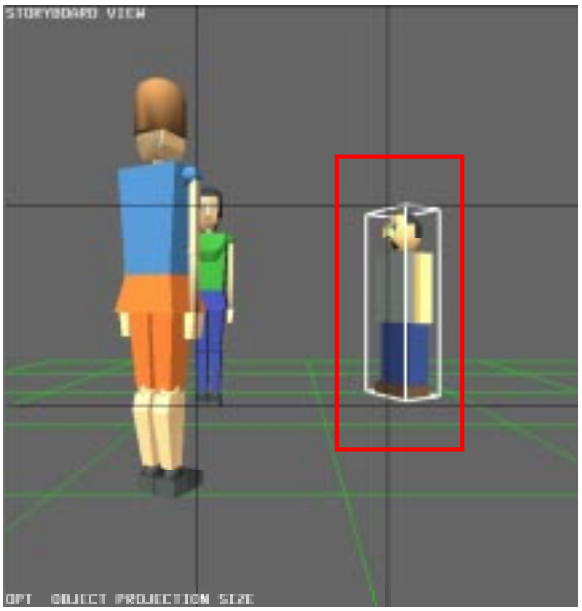
Figure 3. Extraction of projection size and view angle constraints from Storyboard Editor frame view.

```
NumConstraints 5
 // Include BlueActor in camera field of view.
Constraint OBJ_IN_FIELD_OF_VIEW
{  PrimaryObj BlueActor                    Priority 1.0 }

// Projection size of BlueActor
Constraint OBJ_PROJECTION_SIZE
{   PrimaryObj BlueActor
    Parameters
    {
      Source BoundingBox
      MinSize 0.18   OptSize 0.37          MaxSize 0.63
    }
   Priority 1.0  }

// BlueActor's 3D bounding box must project into the
// specified rectangular region of the frame.
Constraint OBJ_PROJECTION_ABSOLUTE
{  PrimaryObj BlueActor
   Parameters
  {
    Source BoundingBox
    BottomLeft 0.079801 0.006981
   TopRight 0.882977 1.044370
  }
  Priority 1.0  }

// Relative view orientation to camera.
Constraint OBJ_VIEW_ANGLE
{   PrimaryObj BlueActor
    Parameters
    {
      optHoriz 45.0   optElev 15.0
      AllowedHorizRange 0.0  90.0
     AllowedElevRange   0.0  35.0
    }
   Priority 1.0   }

// Minimize occlusion by obstacles
Constraint OBJ_OCCLUSION_MINIMIZE
{   PrimaryObj BlueActor
    Parameters
    {        MaxAllowable 0.10             }
   Priority 1.0   }
```

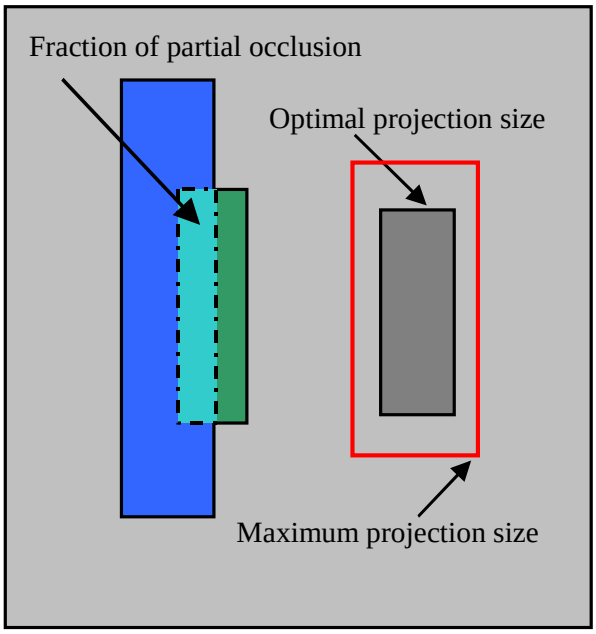
Listing 1. Exported constraint definition from figure 3.

As a second example, consider the shot of three virtual actors in conversation expressed in the Storyboard Editor frame view in Figure 4(a). Projection size, viewing angle, projection location, and occlusion constraints are exported for each character. Figure 4(b) illustrates the abstraction of the extracted optimal and maximum allowable projection size for the gray actor standing on the right side of the frame.

The user manipulates a bounding rectangle in the frame view to set the maximum allowable projection size of the subject in the frame. The Storyboard Frame Editor detects that the nearer blue actress partially obscures the green actress. The user can opt to export a constraint that specifies the optimal and allowed degree to which the green actress is occluded. This constraint is useful to add interest and depth cues to a shot.



(a) Storyboard frame view of desired shot. User adjusts rectangle around rightmost actor to specify its maximum allowable projection size. Note the partial overlap of the other two figures.



(b) Abstraction of extracted constraints.

Figure 4. Specification and extraction of maximum projection size and partial occlusion constraints.

3.3 Measuring Constraint Satisfaction

A computed *camera placement solution* is an assignment of values to the virtual camera's position, aim direction, and field of view angle. It's essential to determine how well a given camera placement satisfies the constraints. The satisfaction rating for a constraint is a value in the range 0.0 to 1.0, which rates how near a camera parameter value is to its specified optimal value for the constraint. A cumulative constraint satisfaction rating is

computed as shown in the below equation, where P_i is the relative priority of the i^{th} constraint, which may be specified in a Storyboard Editor dialog box, and S_i is the satisfaction rating of the i^{th} constraint.

$$satisfaction = \sum_{1}^{N} (Pi \times Si)$$

This cumulative shot constraint satisfaction rating is used to determine the "best" shot found during the solution search process. Constraint priority values, which can be edited by the user in the Storyboard Frame Editor, normally range between 0.0 and 1.0. Constraints having priority values greater than 1.0 are mandatory and must have a non-zero individual constraint satisfaction rating; otherwise, the cumulative satisfaction rating for the shot is degraded to zero. The constraint priority weights distinguish between those that are mandatory for an acceptable shot and those that rank acceptable shots based on how near they are to the optimal shot given in the Storyboard Editor frame view. The constraint priority weights help select the returned solution shot from the set of candidate shots satisfying the given constraints. The selection of these priority weights is largely subjective based on the relative importance of each constraint assigned by an application or user.

Specialized evaluator functions are provided for each type of constraint. Object projection size, location, and inclusion or exclusion from the field of view are evaluated using the bounding box of the relevant object. For object projection size and location, the 8 vertices of an object bounding box are projected onto the camera's viewplane. Then a rectangle is computed to enclose the projected points in the frame of the shot. During solution searches, the evaluation of a shot terminates when it is certain that the shot's cumulative satisfaction rating is less than that of the best shot(s) found so far. Occlusion constraints are evaluated last since they tend to be more costly. The occlusion constraint evaluator can assume one of two variations:

Ray Casting: Cast rays from the camera position to each of the 8 vertices and midpoint of a potential obstruction's bounding box. The number rays resulting in a hit is used to estimate the fraction of the object in occlusion.

Frame Rendering: An object of interest is rendered into the OpenGL backbuffer with accompanying writes to a stencil buffer mask. Potential occluders are rendered using unique color codes. Non-zero pixels having values not equal to the id of the object of interest are added to the fraction of the object in occlusion.

3.4 Camera Constraint Solutions

Our heuristic search method uses the given constraint allowable minimum and maximum values to reduce the size of the 7 Degree of Freedom search space of possible camera positions (x,y,z), aim direction vectors (dx, dy, dz), and field of view angles. The Constraint Solver constructs *valid regions* of space, each of which represents the allowable range of virtual camera parameter values that will satisfy a particular constraint. The solver then examines candidate camera shots by stepping by discrete increments of the camera placement parameters (position, aim direction, and field of view angle) inside the respective valid regions. Each candidate camera placement is evaluated as described in Section 3.3 to determine how well it satisfies the set of constraints. If no camera parameter values lie inside all of the respective valid regions, then the solver reports its failure to find a solution.

3.4.1 Valid Regions

First compute valid regions for those camera constraints that restrict the position of the camera. A view angle constraint requires the camera position to lie inside a spherical wedge, defined by (theta, phi) minima and maxima extending from the midpoint or locus modifier of an object (Figure 5). For example, the optimal 45 degree horizontal and 15 degree vertical elevation orientation between the camera and virtual actor shown previously in Figure 2 would require the camera to lie along a vector *V*. The allowable range of view angles would be defined by sweeping out by user-specified threshold angles along the horizontal and vertical dimensions.

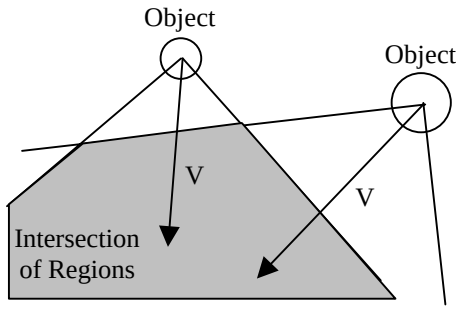


Figure 5: Viewing angle constraint valid regions.

Distance and object projection size constraints require the camera to lie inside two concentric spheres, representing the minimum and maximum distances from camera to object midpoint or locus modifier. For projection size constraints, frame area fractions are converted into corresponding camera to subject distances using the subject's bounding sphere and allowable range of camera field of view angles. If a projection size constraint requires a subject to fill most of the frame, then the camera must be positioned relatively near the subject and vice versa for small projection size.

The camera in region constraint directly expresses its valid region using a set of spatial region primitives including boxes, spheres, and space-partitioning planes combined using Boolean intersection and union operators. This is used to keep the camera inside an irregular-shaped room interior of a 3D environment. The camera can be located only at those points, the so-called *valid camera positions*, lying inside the intersection of all of these valid regions which restrict the allowable camera positions.

Next, each valid camera position can be examined to determine the maximum allowable range of field of view angles. Given a valid camera position, an object's minimum projection size constraint determines its maximum allowable field of view angle, while the maximum projection size determines its minimum allowable field of view angle. Its optimal field of view angle (FOV) may be likewise estimated from the given optimal projection size $f = \pi r^2 / A_F$ in Figure 6, where πr^2 is the area of the projection of the object's bounding sphere and A_F is the frame area. The equation giving the field of view angle is:

$$fov = \arctan\left(\frac{\sqrt{\pi r^2 / f}}{vpdist}\right)$$

where *f* = minimum, optimal, or maximum projection size fraction and *vpdist* is the distance to the projection plane. The per-object field of view ranges are intersected to estimate the overall valid field of view angle range.

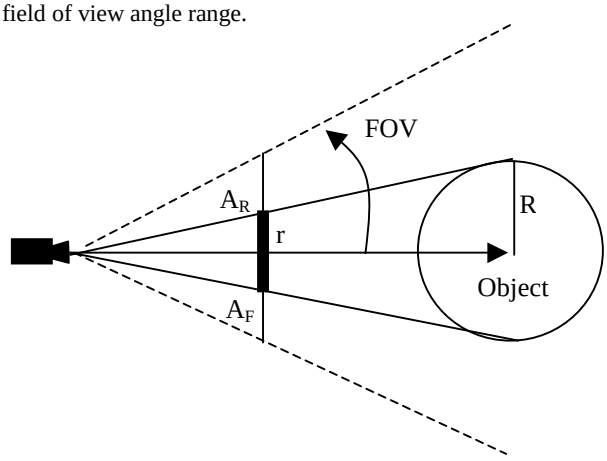


Figure 6: Projection size to field of view.

For a given candidate camera position and object geometry, the maximal allowable range of camera aim directions can be estimated using the OBJ_IN_FIELD_OF_VIEW, OBJ_NOT_IN_FIELD_OF_VIEW, LOOK_AT_POINT, and OBJ_PROJECTION_ABSOLUTE constraints. For example if point *P* must project into frame region *T*, then the valid range of aim directions *S* may be determined in spherical coordinates (Figure 7). The valid aim directions for multiple constraints are intersected using (theta, phi) spherical coordinate intervals.

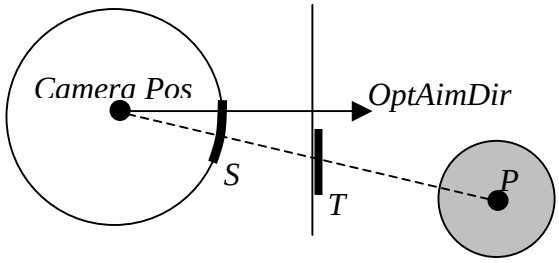


Figure 7: Valid regions for aim direction.

3.4.2 Heuristic Solver Algorithm

The constraint-solver algorithm utilizes a recursive heuristic search over the possible camera parameter values. The constraint valid regions limit the search to consider only those sets of parameter values that may be part of an allowable solution as defined by the given constraints. The search begins at a relatively coarse resolution generating-and-testing candidate shots. The best candidate shots are logged for further refinement of the search via recursion about these most promising candidates.

1. Initialize a log of the *N* best candidate shots found so far. In the examples, the log records the top 5 candidate shots.

2. Optimistically, compute and evaluate the relative-displacement solution. If its constraint satisfaction rating exceeds the specified minimum success threshold, then immediately return this shot as the solution to obtain instantaneous performance in those cases when configured as anticipated.

3. Compute valid regions for constraints that limit the allowable camera positions. These constraints include projection size, object-to-camera distance, object view angle, and camera inside region. Form an axis-aligned bounding box *PosBBox* to enclose the intersection of these camera position valid regions.

4. Loop over candidate camera positions spaced along a regular grid of points spanning the space enclosed by *PosBBox*. The number of points generated is determined by the size of *PosBBox* and the desired degree of solution precision.

> 4(a). For each valid camera position, use the projection size, projection location, aim direction, and camera field of view constraints to compute the allowable and estimated optimal aim direction and field of view angle.
> 4(b). Evaluate the shot using the estimated optimal aim direction and field of view angle and immediately return it as the solution if its satisfaction rating exceeds the specified minimal constraint success threshold.
> 4(c). Update the log of the best *N* shot found so far.

5. If a candidate shot exceeding the given minimal success threshold has not yet been found, then loop over each of the *N* best shots found so far.

> 5(a) For each top candidate shot *TS*, initialize *PosBBox* to enclose space nearby the camera position of shot *TS*.
> 5(c) Initialize a new log of best candidate shots for the next level of search to empty.
> 5(c) Recursively execute the solver algorithm beginning from step number (4) if the maximum recursion depth has not been reached.
> 5(d) If the best candidate shot returned by the recursive call in step 5(c) exceeds the minimal success threshold, then immediately return that shot as the solution.

4. EXAMPLE CAMERA SOLUTIONS

For example, we might want a shot of player1's face as he speaks, in which player1 (gray) is viewed from a camera position behind and over-the-shoulder of player2. In order to compose the desired over-the-shoulder shot of player1 and player2, we could specify the following set of visual composition elements, or camera constraints, by drawing the storyboard frame depicted in Figure 8.

- Player1 and player2 should appear in the field of view, while the other three players are excluded or hidden.
- View the face of player1 who is speaking.
- Player1 should appear slightly to the right of player2 and centered in the frame (Absolute projection location constraint for each player).
- Player1 should be framed in a medium shot and player2 should be in a near shot (Projection size).
- Player1 should be partially occluded by player2 and player2 should not be occluded (Object partial and minimize occlusion constraints).

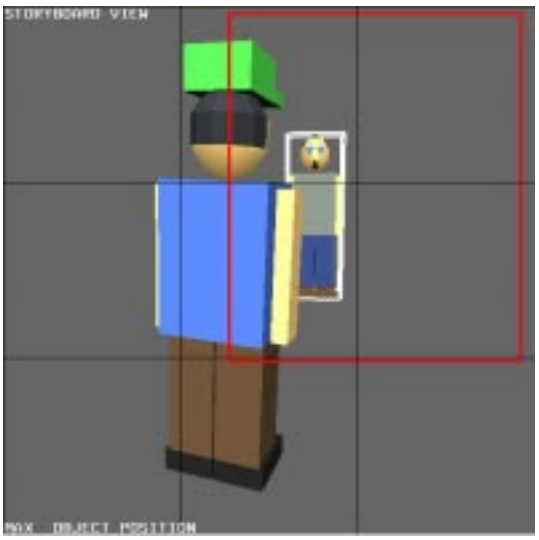


Figure 8: Given storyboard frame of a two shot.

Figure 9: Room interior scene featuring five players.

Figure 10: Computed two shot in given 3D scene.

The image depicted in this storyboard frame represents the optimal location, projection sizes, view angles, and occlusion of the two players. The allowable location of the gray (rightmost) player in the frame has been set by a red rectangle. The optimal viewing angle for the gray player is the front, and the optimal viewing angle for the blue player is rear right.

We could then apply this storyboard frame to a 3D scene of five players, four standing around a table in a room ringed by stone columns with a fifth player standing on a balcony (Figure 9). The virtual camera will need to be carefully staged in order to capture the desired over-the-shoulder shot of player1 and player2, while excluding the three other players. The computed shot appears in Figure 10. A less sophisticated relative displacement method would place the camera behind and slightly to the right of the blue player. However, in this environment, such a shot have the blue player partially occluded by a column. If the camera were moved in front of the column and directly behind the blue player to avoid the occlusion, it would only partly include the blue player in the shot and would also include the green player on the balcony. The heuristic search solver analyzed the environment to find a next-best shot within the allowable bounds specified in the given storyboard frame (Figure 10). This shot takes a camera orientation farther to the side of the players than specified by the optimal storyboard frame definition in Figure 8. This example demonstrates the flexibility of a constraint-based approach, which enables developers of 3D virtual environments to compose desired camera shots (Figure 7) independent of the unanticipated configuration of objects in a given scene (Figure 9).

Figure 11 presents a triangle shot of three players where one of the players is required to partially overlap the balcony player. The storyboard frame used to create this shot appears in Figure 4.

Figure 11: Three shot with required partial occlusion of balcony player by the blue player.

Next, suppose the blue player (standing on the left side) backs away from the table. We can still apply the same set of storyboard frame constraints to compute the acceptable solution depicted in Figure 12.

Figure 12. Solution after player movement.

Figure 13: Cluttered medium three-quarter shot.

Figure 14: Improved composition excludes distractions.

The next example, specifies a medium three-quarter shot of a cathedral. Figure 13 depicts the resulting heuristic search solution of the desired medium three-quarter shot. However, the user feels that the cars surrounding the building clutter the shot distracting attention from the subject. The user then adds constraints to exclude each of the cars from the shot. The heuristic solver positions the camera to obtain the desired three-quarter angle and utilizes the flower arrangement to occlude the distracting cars, which also results in a far more dramatic composition (Figure 14).

5. PERFORMANCE EVALUATIONS

Table 1 provides benchmarks for several individual constraint evaluators. As described in Section 3.3, specialized evaluators are provided to determine how well a given shot satisfies a particular constraint. A typical constraint-based shot definition may include between 6 and 20 different constraints, each of which would be evaluated by its respective type of evaluator method.

Table 1. Constraint evaluator benchmark results.

Evaluator Function	*Time in milliseconds*
Projection size	0.009344
Projection location	0.009088
View angle	0.010140
Object in camera field of view	0.013505
Occlusion by ray casting	0.184320
Occlusion by frame rendering	10.78

Table 2 gives benchmarks of the heuristic search constraint solver using ray casting for evaluating occlusion. Times are given in milliseconds. The test system is a Pentium II 400 MHz Intergraph GL2 computer with 256 MB memory, and a VX113 AGP OpenGL accelerator. In all examples, the first level of search over candidate camera positions scanned a 9x9x9 grid, with the second recursive search over the top 5 shots scanned over a 4x4x4 grid. The third column records the total number of shots evaluated including shots tested to refine the camera field of view and aim direction per each candidate position tested. The fourth column lists the total number of candidate camera positions tested, while the fifth column records the number of candidate camera positions skipped because they were not inside the constraint valid regions.

Table 2. Constraint solution benchmark results.

Figure	*Time*	*Shots Tested*	*Positions*	*Culled Positions*
10	4500 ms	5358	402	711
11	2872	4509	414	699
12	3461	5073	410	703
14	417	609	271	586

The solver can be configured to examine the search space at varying resolutions and can be directed to return the best shot found after testing no more than a specified maximum number of shots. This allows an application to fine-tune the quality of the computed solutions and the solution computation time. For comparison, our implementation of a simple relative displacement camera algorithm computes camera placements in 0.016 milliseconds. An exhaustive search was implemented for comparison purposes and its solution times range from about 1 to 10 minutes in evaluating 2 million shots.

Table 3 gives the cumulative constraint success ratings of our heuristic search and our relative displacement solvers.

Table 3. Comparison of constraint satisfaction.

Figure	*Heuristic Search*	*Relative Displacement*
10	0.877109	0.0
11	0.850244	0.338701
12	0.774565	0.338701
14	0.810209	0.473219

6. CONCLUSIONS AND FUTURE WORK

Together the Storyboard Frame Editor interface and constraint solver provide tools for interactive 3D application developers to define how the virtual camera should film objects of interest leaving the low-level task of placing the camera to the automated constraint solver module. The constraint solver module is currently being integrated into several prototypical types of interactive 3D virtual environments. The search heuristics and constraint evaluators will be further optimized to improve real-time performance for interactive applications. Additional constraint types can be implemented by coding new evaluators and script parsing functions. Techniques such as returning multiple shots or relaxing less important constraints will be added to handle failures to compute constraint solutions [2]. Studies of user interface effectiveness and aesthetics of the computed shots will also be performed.

7. ACKNOWLEDGEMENTS

We would like to thank James Lester and Lou Harrison for their assistance in reviewing the paper. Thanks to visual artists Robert Russett, Yeon Choi, and Patrick FitzGerald for their creative suggestions. Thanks also to Junwei Li for editing the 3D environments and Byungwoo Kim for assisting in testing the examples. This work is partly supported by the Louisiana Board of Regents through the Board of Regents Support Fund contract (1999-2002)-RD-A-48.

8. REFERENCES

[1] André, Elisabeth , W. Finkler, W. Graf, T. Rist, A. Schauder, and W. Walhster. WIP: The automatic synthesis of multimodal presentations. In M.T. Maybury, editor, *Intelligent Multimedia Interfaces*, chapter 3, AAAI Press, 1993.

[2] Bares, W. and J. Lester. Intelligent Multi-Shot Visualization Interfaces for Dynamic 3D Worlds. In *IUI-99: Proceedings of the 1999 International Conference on Intelligent User Interfaces*, Los Angeles, California, 1999, pages 119-126.

[3] Blinn, James. Where Am I? What Am I Looking At? In *IEEE Computer Graphics and Applications*, July, 1988, pages 76-81.

[4] Butz, Andreas. Anymation with CATHI. In *IAAI-97: Proceedings of Innovative Applications of Artificial Intelligence*. Providence, Rhode Island, July 1997, pages 957-962.

[5] Christianson, David, Sean Anderson, Li-wei He, David Salesin, Daniel Weld, and Michel Cohen. Declarative camera control for automatic cinematography. In P*roceedings of the AAAI-96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, August 1996, pages 148-155.

[6] Drucker, Steven. *Intelligent Camera Control in Graphical Environments*. Ph.D. thesis, 1994, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1994.

[7] Drucker, Steven and David Zeltzer. CamDroid: A System for Implementing Intelligent Camera Control. In *1995 Symposium on Interactive 3D Graphics*, pages 139-144, 1995.

[8] Feiner, Steven. APEX: An Experiment in the Automated Creation of Pictorial Explanations. In *IEEE Computer Graphics & Applications*, September, 1985, pages 29-37.

[9] Feiner, Steven and Dorée D. Seligmann. Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations. In *The Visual Computer*, August 1992, pages 292-302.

[10] Funge, John, Xiaoyuan Tu, Demetri Terzopouos. Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In *Proceedings of ACM SIGGRAPH '99*, August 1999, pages 29-38.

[11] Gleicher, Michael and Andrew Witkin. Through-the-lens camera control. In Edwin E. Catmull, editor, *Computer Graphics (Proceedings of SIGGRAPH '92)*, 1992, pages 331-340.

[12] Halper, Nicolas and Patrick Olivier. CAMPLAN: A Camera Planning Agent. In *Smart Graphics 2000 AAAI Spring Symposium*, Stanford, California, March 2000, pages 92-100.

[13] Hanson, Andrew and Eric Wernert. Constrained 3D navigation with 2D controllers. In *Proceedings of IEEE Visualization '97*, 1997, pages 175-182.

[14] Hines, William. Operating Cinematography for Film and Video: A Professional and Practical Guide. Ed-Venture Publishers, Los Angeles, California, 1997.

[15] Jardillier, Frank and Eric Languénou. Screen-Space Constraints for Camera Movements: the Virtual Cameraman. In *Eurographics 1998 Computer Graphics Forum*, 17(3), 1998.

[16] Karp, Peter and Steven Feiner. Automated Presentation Planning of Animation Using Task Decomposition with Heuristic Reasoning. In *Graphics Interface '93*, 1993, pages 118-126.

[17] Katz, Steven. *Film Directing Shot by Shot*. Studio City, CA, Michael Wiese Press, 1991.

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

[19] Mackinlay, Jock, S. Card and G. Robertson. Rapid controlled movement through a virtual 3D workspace. In *Proceedings of ACM SIGGRAPH '90*, pages 171-176, 1990.

[20] Mascelli, Joseph. *The Five C's of Cinematography*. Silman-James Press, Los Angeles, California, 1965.

[21] Millerson, Gerald. *Video Camera Techniques*. Focal Press, Oxford, England, 1994.

[22] Olivier, Patrick, Nicolas Halper, Jon Pickering, and Pamela Luna. Visual Composition as Optimisation. In *Artificial and Simulation of Intelligent Behavior (AISB) Workshop '99 Symposium on AI and Creativity in Entertainment and Visual Arts*. Edinburgh, UK, April 1998, pages 22-30.

[23] Peterson, Bryan. *Learning to See Creatively*. Watson-Guptill, New York, New York, 1988.

[24] Phillips, Cary B., Norman Badler, and John Granieri. Automatic viewing control for 3D direct manipulation. In David Zeltzer, editor, *Computer Graphics (1992 Symposium on Interactive 3D Graphics)*, volume 25, March 1992, pages 71-74.

[25] Seligmann, Dorée and Steven Feiner. Automated Generation of Intent-Based 3D Illustrations. In *Computer Graphics*, July 1991, pages 123-132.

[26] Ware, Colin and S. Osborn. Exploration and virtual camera control in virtual three-dimensional environments. In *1990 Symposium on Interactive 3D Graphics*, pages 175-184, 1990.

[27] Ware, Colin and Daniel Fleet. Context Sensitive Flying Interface. In *Proceedings of the Symposium on Interactive 3D Graphics*. Providence, Rhode Island, 1997, pages 127-130.