CubeCam: A screen-space camera manipulation tool

Nisha Sudarsanam, Cindy Grimm, Karan Singh

Abstract— **We present CubeCam, an image-space camera manipulation widget that supports visualization of the relationship of the camera with respect to the scene. The shape of the widget presents the user with natural affordances for camera manipulation. The widget supports ghosting of the scene which helps novice users remember the functions associated with different parts of the widget. Pie-menus provide a natural interface for presenting the different non-camera actions to the user. Finally, we provide a novel method for visualizing camera bookmarks.**

Index Terms— **Camera control, Projection, Perspective**

I. Introduction

A. Motivation

Virtual 3D environments are becoming larger and increasingly realistic. Manipulating a virtual camera in such 3D environments is a crucial concern. Unfortunately, effective camera manipulation techniques have been difficult to develop. In order to completely specify a camera, 11 distinct degrees of freedom need to be specified : six for positioning and orienting the camera and five for controlling the projection of the image. Standard mouse-based techniques can control only a subset of this large 11-dimensional space. In order to map the whole camera space to the 2D space of the mouse, current techniques use an array of menu options, shortcut keys and key modifiers. This mapping is not always transparent to the user. Given the myriad of tools available to manipulate the camera, it is hard for the user to know which tool to use in order to obtain their desired view.

Another problem with current 3D applications is the method used to represent "camera bookmarks", or previously saved camera views. Current techniques represent camera bookmarks as a textual list that users can organize. However, a textual representation of a bookmark conveys no visual information about the view point corresponding to that bookmark. There is also no way to find bookmarks close to the current viewpoint.

email:nsudarsa@cs.wustl.edu
email:cmg@cse.wustl.edu
email:karan@dgp.toronto.edu

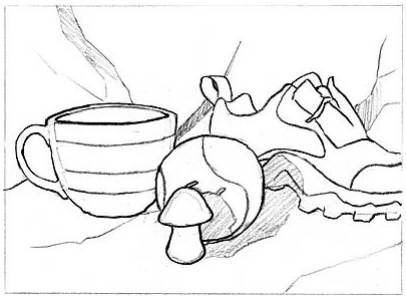

Fig. 1. Geometric primitives are used by artists for simplifying objects

This paper presents a camera manipulation interface that moves away from the menu-driven approach of current techniques. We present the user with an interface that encapsulates camera operations within intuitive controls: the user specifies a camera operation in terms of the change they want to see in the projected image. In contrase to previous approaches, our system provides visual selection aids to select different camera operations. Finally, we visualize camera bookmarks and the relationship between a camera bookmark and the current view-point.

B. Background

In linear perspective, the perspective effects depends on the relative position of the camera with respect to the 3D scene. Artists explicitly visualize this relationship using geometric proxies which are used to reduce objects to sets of points, lines and curves (Figure 1). Thus, geometric proxies visually reflect the artist's perception of the 3D scene. We borrow this idea by presenting the user with **camera primitives** which not only serve as geometric proxies for visualizing the current camera but also can modify the projection of the scene. These camera primitives allow camera operations to be defined with respect to any point

in the scene and also provide visual feedback with regard to how specific camera operations affect the projected image. Thus, our system explicitly visualizes the camera-scene relationship in the image plane.

C. Novel features of the interface

The camera is visualized through the projection of a cube in the scene. The cube can be viewed in three different perspective views, namely 1-pt perspective, 2-pt perspective and 3-pt perspective (Figure 2). We collectively label the three perspective views of the cube as CubeCam.

Each camera primitive encapsulates a subset of all possible camera operations. This subset is selected based on the operations that map naturally to the shape of the cube in that particular view. This allows camera operations to be defined in terms of the change the user want to see in the camera primitive. For example, when the cube is in 1-pt perspective making the the cube bigger results in the camera being zoomed in or out This is analogous to saying "perform the camera operation that makes the cube appear bigger". In this case, the camera operation is a camera zoom.

Object-Centric CubeCam: CubeCam can either be centered in a scene or centered on a particular object in the scene(**scene v/s obj picture**). If it is centered on an object, all camera operations are defined with respect to that object. For example, we can rotate the camera with respect to any point in or on an object or around any of the principal axes of the object.

Ghosting: To help users find out which operations are associated each primitive, we provide "ghosting". When ghosting is active, users interact with CubeCam as they would normally. However, instead of changing the scene camera, the user changes a "ghost camera". The ghost camera is used to render a ghosted projection of the scene on top of the current scene (Figure 14). Ghosting allows new users to get acquainted with the interface. It also allows users to test out possible camera changes before actually making them.

Focus-depth: CubeCam allows users to visualize and change the "focus-distance" (depth of focus). Specifying the focus-distance helps users indirectly specify which objects need to stay the same size (Figure 6) or stay at the same spot on the screen (Figure 9) when changing the perspective distortion of the scene. The focus-distance is visualized through a semi-transparent plane (focus plane) drawn at a distance equal to the focus-distance along the view-direction of the camera. Users can slide this plane through the scene thereby changing the focus-distance.

Camera bookmarks: Camera bookmarks are displayed in our system. Each bookmark is represented as an icon whose image is the scene viewed from the saved viewpoint. Mousing over the icon renders a ghosted version of the scene over the current scene. In addition to visualizing the bookmarks themselves we also visualize the relationship between a bookmarked viewpoint and the current viewpoint. Bookmarks are arranged in a circle in a clockwise manner at the center of the image. The closer an icon is to the 12:00 position, the "closer" a bookmarked viewpoint is to the current viewpoint. As the current view changes, so does the position of a given bookmark. The metric used to measure the distance between two view-points depends on the currently active view of CubeCam (Figure 15, 16). In addition to automatic placement, we also allow the user to manually place the bookmark icons anywhere on the screen. In this case, the position of the bookmarks remain fixed even if the current view changes.

Selection: Dividing camera operations among the three views of the cube simplifies the interface of a given camera primitive, but requires the user to switch between them. CubeCam also allows users to turn ghosting on and off, bookmark a view and view all of the bookmarks. Mapping these options to different shortcut keys increases the amount of information users have to remember about the interface. We incorporate these options into a pie-menu [1]. We provide a button (which we will refer to as the pie-menu button) for invoking the pie-menu. The pie-menu button is located in the top-right corner of the screen (Figure 2). Crossing the pie-menu button pops up a circular menu of options (Figure 4). Releasing the mouse over one of these options, selects that particular option. User studies [1] have shown that once users grow accustomed to the location of the menu-options on a pie-menu, the time taken to select a pie-menu option is less than the time taken to select an option on a traditional menu.

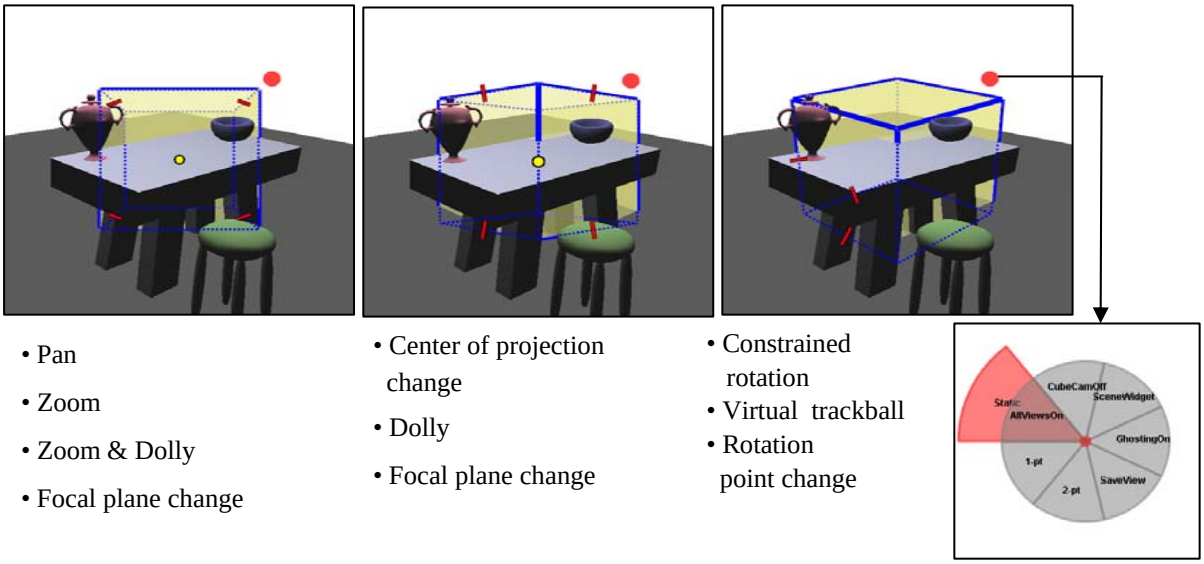

Fig. 2. The three perspective views of a cube form CubeCam. Each primitive is associated with a set of camera operations, listed underneath it. The red icon in the top right corner is the pie-menu button which when crossed pops up a pie-menu of options shown in the box.

Camera rotation: In the 3-pt perspective view of the cube, the user can perform two types of camera rotation: constrained rotation and a virtual trackball rotation. Crossing a single edge allows the user to perform constrained rotation about the axis parallel to the selected edge (Figure 11). Crossing multiple edges, allows for virtual trackball (Figure 12). Thus, both these options can be specified naturally in the CubeCam interface without introducing additional menu-options.

Rotation planes: Analogous to visualizing the focal plane of the camera, CubeCam explicitly visualizes the point about which the camera is rotated as the intersection of three rotation planes (Figure 13). The rotation planes are aligned along the principal axes of the cube. Users can change the pivot point by changing any of the rotation planes.

The rationale for choosing a cube: We chose a cube as the basic shape of our camera primitives since the different perspective views of the cube are easily recognized and familiar to all users. Artists frequently use the cube as a geometric proxy to specify their viewpoint for this reason. Finally, each of the perspective views lend themselves naturally to being associated with camera operations.

D. Contributions

CubeCam is a simple, intuitive camera manipulation interface. CubeCam can visualize both the relationship of the camera with the scene and the state of the camera itself, while staying in the 2D image plane. Specifically, CubeCam explicitly visualizes important aspects of any camera manipulation interface such as the camera's focal plane and pivot point about which camera rotations take place. CubeCam provides an intuitive interface because it allows users to define camera operations in terms of the change they want to see in the projected image.

Visual aids such as ghosting of the scene help novice users learn the different functions of CubeCam and also allows advanced users to experiment with possible camera changes before actually making them. Finally, CubeCam allows users to find and visualize nearby camera bookmarks.

E. Overview

The paper has the following structure: Section II places this paper in context with previous work in this area. Section III describes the features of the interface while Section IV discusses ghosting. Section V describes camera bookmarks while Section VI discusses some implementation details of CubeCam. Section IX provides the conclusion.

II. Related work

For mouse-based systems, camera control paradigms fall roughly into two categories, camera-centric and object-centric. In the camera-centric paradigm, operations are applied to the camera as if it were a real object in the scene. This mirrors camera placement in the real world, and many of the camera operations (dolly, pan, and roll) reflect that. The external parameters, position and orientation, can be specified either "through the lens", or by manipulating a pictorial representation of the camera in a second window. The internal camera parameters, with the exception of focal length, are changed through textual input.

In the object-centric paradigm, the camera is centered on an object and the viewpoint is rotated relative to the object (as if there were a virtual trackball around the object [2], [3]). The camera can also be zoomed in and out. This paradigm is useful when there is a single object in the scene (or one object of importance) and the user is simply choosing a direction from which to view it.

Three or six degrees of freedom devices permit other interesting navigation techniques [4], such as the palm-top world [5], the "grab and pull" approach [6] and virtual fly-throughs [7]. The latter can also be used in mouse or keyboard-based systems if the camera's movement is restricted to a well-defined floor plane (most first-person shooters use this approach).

An alternative approach to directly specifying the camera is to use image-space constraints [8], [9]. In this approach, points in the scene are constrained to appear at particular locations, or to move in a specified direction, and the system solves for the camera parameters that meet those constraints.

Pie-menus [1] are an alternative to linear menus. Pie-menus display a set of menu options in a circle on the screen. User studies [1] have shown that pie-menus are a faster way to selecting menu items.Using pie-menus we allow users to switch between different views of CubeCam without resorting to linear menus or key modifiers for performing the same task.

A. *CubeCam versus IBar*

The recently-introduced IBar [10] is, in some sense, a specialization of the constraint approach, where the points are the points of the edge of a cube.

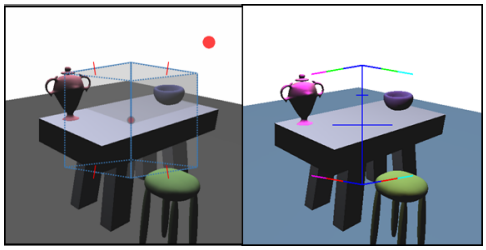

Fig. 3. (a) 2-Point perspective view of CubeCam.(b) IBar placed in the scene.

Like CubeCam, the IBar is a screen-space widget where changing the widget changes one or two camera parameters. The IBar and CubeCam have similar goals; both systems move beyond current menu-based camera manipulation techniques to a unified screen-space camera primitive. The underlying mathematical framework of the two systems are similar. Thus, both systems support the same set of camera operations. However, CubeCam improves and extends the interface presented to the user.

- **(improved) Rendering:** The IBar is a camera primitive representing an edge on two point perspective rendering of the cube. It is made up of the projected edges of a cube thus giving perspective but no depth information (Figure 3).
- **(improved) Camera primitive Interface:** In the IBar interface, all of the camera operations that can be performed are associated with a single camera primitive. Each part of the widget performs different functions depending on where the user has clicked. This tends to be confusing for a novice user.
- **(improved) Rotation:** CubeCam improves the camera rotation operation by allowing a camera rotation around an arbitrary 3D point and about an arbitrary axis in the scene.
- **New Features:** Ghosting, visualizing the focal plane and the rotation point(through the rotation planes) are features unique to CubeCam. Finally, visualizing camera bookmarks is an entirely novel feature of CubeCam. Classification of bookmarks in CubeCam is made easier due to the different perspective views.

III. The CubeCam Interface

A. *Pie-Menu*

All camera operations are associated with the camera primitives. The non-camera actions namely

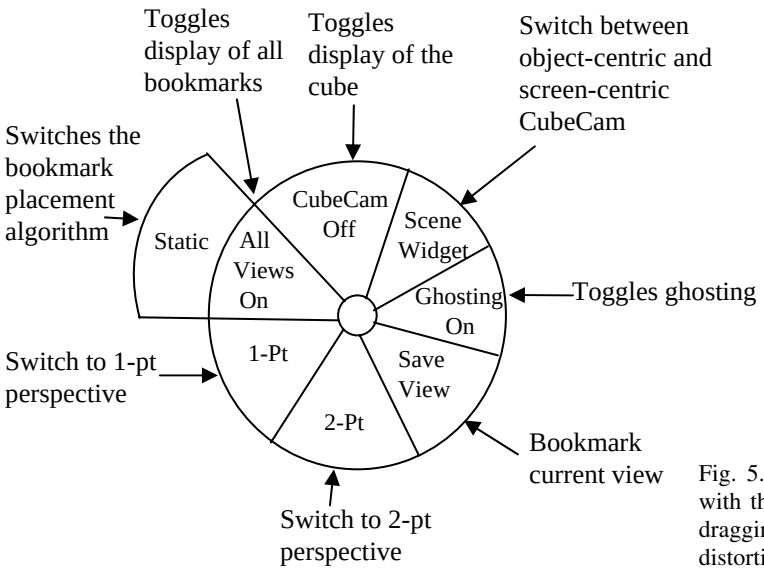

Fig. 4. The pie-menu encapsulates the various non-camera options.

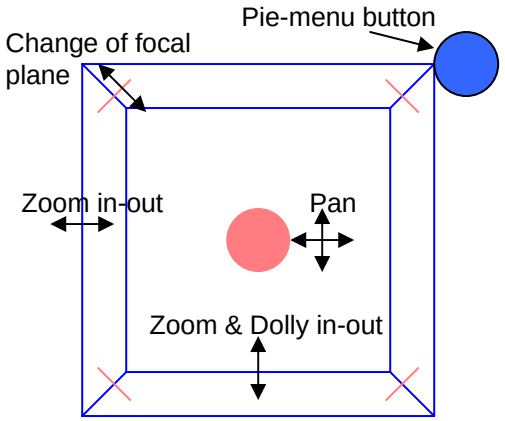

Fig. 5. The above figure shows the camera operations associated with the 1-pt perspective view. The camera is zoomed in or out by dragging the outer edges of the cube outward or inward. Perspective distortion is achieved by dragging the inner edges. The camera is panned by moving the whole cube. The focal plane is slid through the scene using the sliders on the receding edges of the cube. The pie-menu button is used to bring up the pie-menu.

switching between the bookmark placement algorithms, turning bookmarks on or off, switching between object-centric and scene-centric CubeCam, turning ghosting on or off, bookmarking the current view and switching between the different camera primitives can be invoked through a pie-menu (Figure 4). We designate a button located near the top-right corner of the screen as the pie-menu button (Figure 2). When this button is crossed, the current camera primitive disappears and a pie-menu appears. Releasing the mouse over an option selects it. Once an option is selected, the pie-menu disappears and the camera primitive reappears. Thus, pie-menus allow us to naturally integrate different menu-options into our interface.

B. 1-Point Perspective

1-pt perspective allows for camera zooming, camera panning and perspective manipulation (Figure 5).

Perspective Distortion

Perspective distortion is a function of the distance of the camera eye point to the object in question. Unfortunately, changing the camera distance also changes the size of the object in the scene. To counter-act this, the camera is zoomed out simultaneously to keep the object the same size [11] [10]. Thus, objects at a specific distance d along the `look` vector remain the same size in the image plane. This distance is visualized by a plane drawn at depth d, which can be changed by moving a slider along the receding edges of the cube in the 1-pt perspective view (Figure 6). Thus, CubeCam allows users to control the perspective distortion in the scene.

C. 2-Point Perspective

2-pt perspective (Figure 7) allows for camera dollying (Figure 8) and center of projection change.

Center of projection: We extend the focal plane idea to the 2-point perspective view. When the center of projection is changed the whole scene slides in the opposite direction. To prevent this, the camera is panned in the opposite direction ensuring that objects at a depth d remain stationary. A focal plane is rendered at this depth and its position is controlled by sliders located on the slanting edges of the cube. The position of the focal plane controls which objects stay fixed on the screen (Figure 9).

D. 3-Point Perspective

3-pt perspective is used only for either constrained rotation or virtual trackball of the camera (see Figure 10).

Rotation

In the 3-pt perspective view, the user can performed either a constrained rotation or virtual trackball

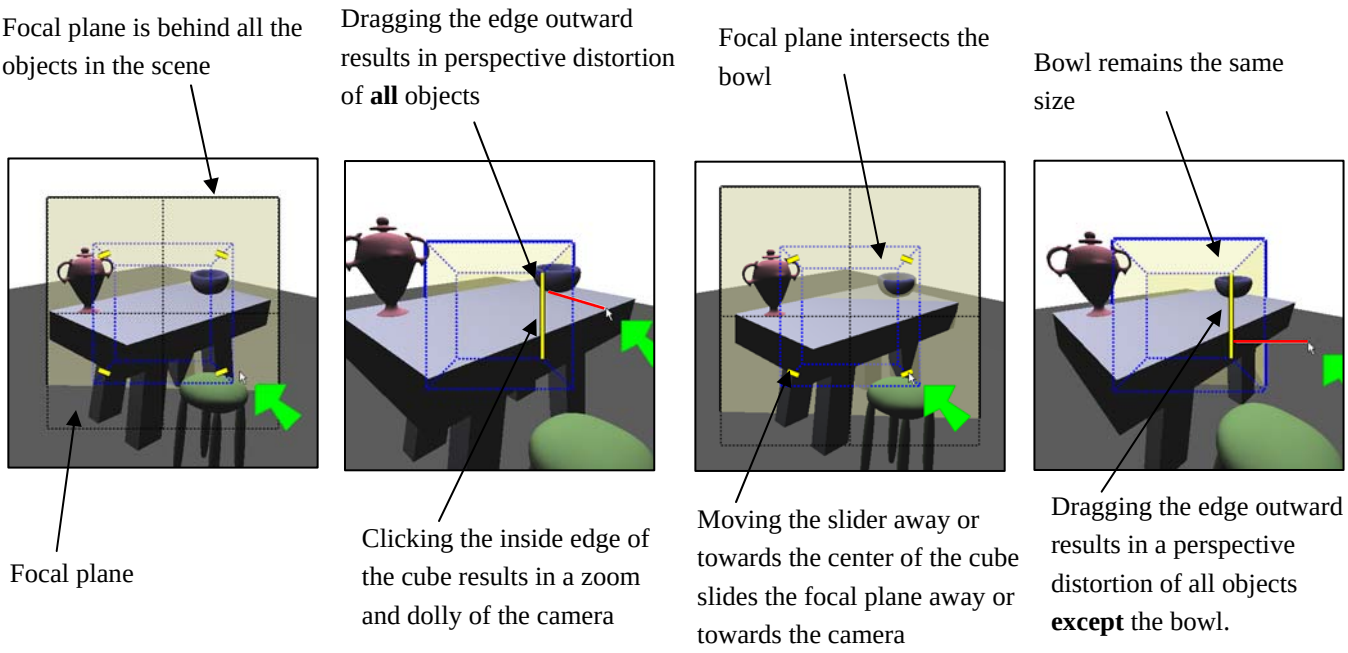

Fig. 6. Perspective Distortion controlled by the focal plane.

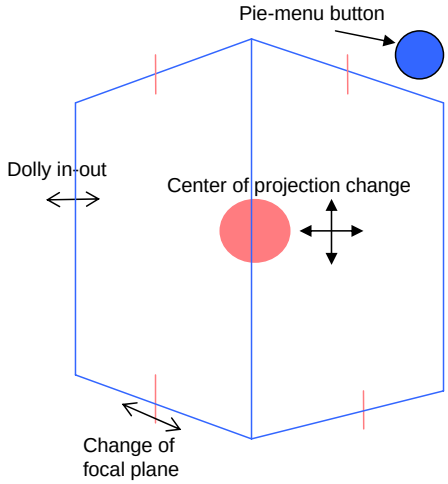

Fig. 7. The above figure shows the camera operations associated with the 2-pt perspective view. The camera is dollied in or out by dragging the outer edges of the cube outward or inward. The camera's center of projection is changed by moving the whole cube. The sliders on the cube are used to change the position of the focal plane in the scene.

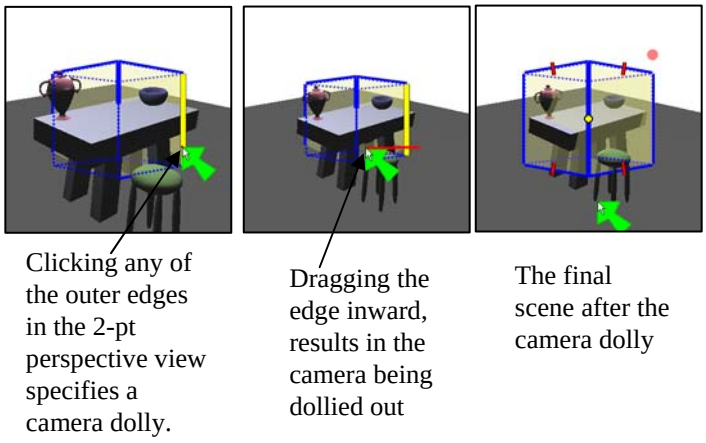

Fig. 8. Camera dolly in the 2-pt perspective view

of the camera. A constrained rotation is specified by crossing a single cube edge. In constrained rotation, the camera is rotated about a single axis. The rotation axis depends on whether the primitive is centered in the scene or on an object. If centered in the scene, the camera is rotated about one of the X,Y,Z world axes, depending on the selected cube edge. For example, if the cube edge selected is vertical, the camera will be rotated about the Y axis. If centered on an object, the camera is rotated about one of the X,Y,Z axes of the object. A virtual trackball rotation is specified by crossing multiple multiple edges of the cube. Once the rotation type has been selected, the user can click and drag any point on the cube to perform the rotation (Figure 11, 12).

picture of object centered rotation

Rotation Planes

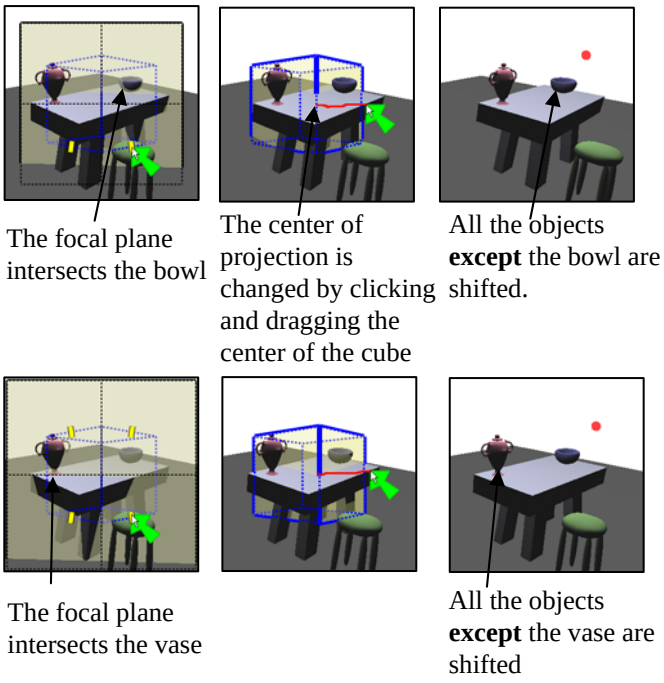

Fig. 9. Center of Projection and Camera Pan

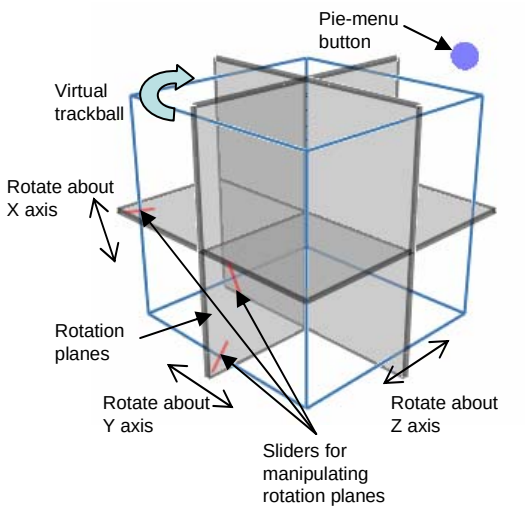

Fig. 10. The above figure shows the camera operations associated with the 3-pt perspective view. The semi-transparent planes are rotation planes. The intersection of these planes form the rotation point about which the camera can be rotated. The position of each rotation plane is controlled using the corresponding slider located on an edge of the cube. Constrained rotation about a particular axis is selected by crossing the appropriate edge. Clicking and dragging any point on the cube then performs the actual rotation. A virtual trackball rotation of the camera is selected by crossing multiple edges of the cube. The pie-menu can be used to display the set of pie-menu options.

Our system visualizes the rotation point about which a camera rotation takes place. This is done using three semi-transparent planes. The planes are aligned along the principal axes of the cube in the 3-pt perspective view (Figure 10). Sliders located on three edges of the cube, are used to change the position of the rotation planes. The final rotation point is the intersection of the rotation planes (Figure 13).

E. Scene-centric CubeCam versus Object-centric CubeCam

CubeCam can be either centered in the scene or at the center of an object in the scene. Users can switch between the two configurations using the pie-menu. Clicking on an object centers CubeCam about that object. All the previously described camera operations can be performed in both configurations of CubeCam with the only difference being the axis used for rotating the camera in the 3-pt perspective. If CubeCam is scene centered then the camera primitives snap back to the original position after the user has modified them. In the case of object-centric CubeCam, the position of the camera primitives change with the position of the object. **picture of scene centered and object centered**

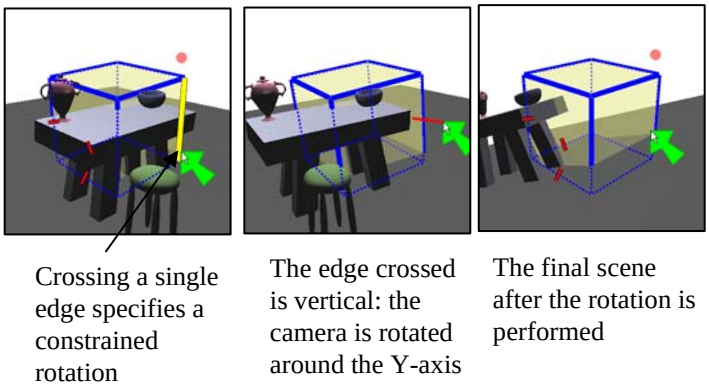

Fig. 11. Constrained rotation

IV. Ghosting

Our goal is to present the user with a simple camera manipulation interface. By creating multiple camera primitives we simplify the overall interface but introduce the problem of expecting the user to remember the functions associated with each primitive. In order to help the user learn these functions we provide ghosting. When ghosting is active, users manipulate the camera primitive as

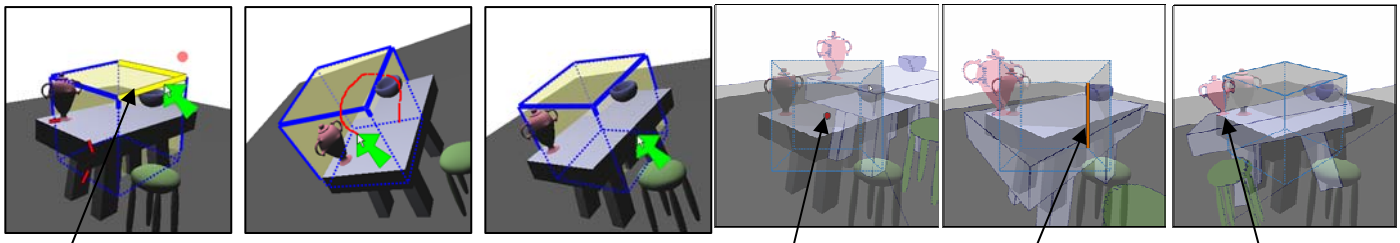

Fig. 12. Virtual trackball

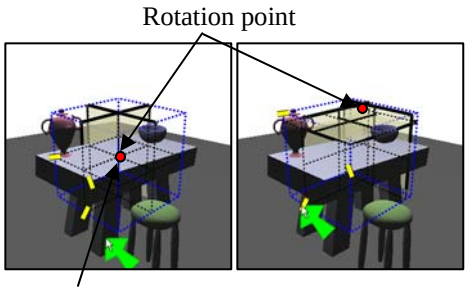

Fig. 13. Rotation Planes

they would normally. However, as they change the camera primitive, a "ghost camera" is changed. This ghost camera is used to render a ghosted scene rendered over the current scene. The original scene camera remains unaffected (Figure 14). For advanced users, this mode provides an opportunity to test out possible camera changes before actually making them.

V. Camera Bookmarks

It is very useful to be able to save cameras and snap back to them at will. For example, when modeling a surface, a user might bookmark a handful of orthogonal views and close-ups of complex geometry. An animator might also use bookmarks to start laying out an animation sequence. In both of these cases, we need to provide the user with a method for quickly searching through existing cameras. Although the user could simply create a text list, appropriately naming each camera, we believe that a visual search mechanism is more useful. We have implemented two bookmark placement algorithms, one of which is static placement and

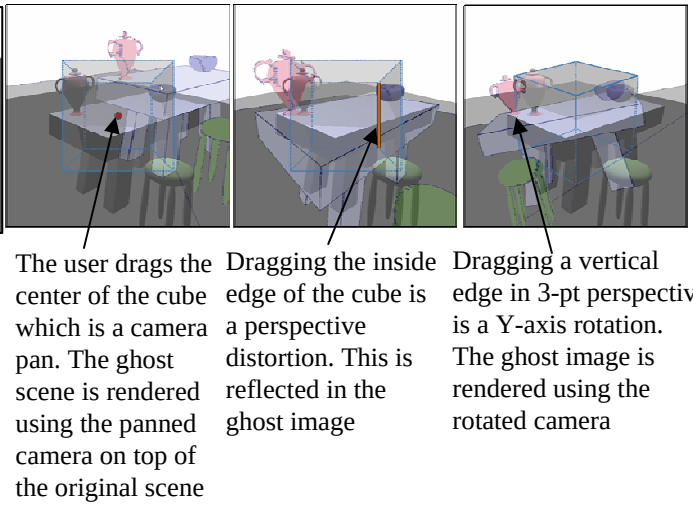

Fig. 14. Ghosting helps users learn the various functions of each camera primitive

the other is automatic placement. Each bookmark is represented as an icon with an image of the scene. In static placement, the user simply places the icon on the image plane. As the user changes the current camera, the bookmark icons remain at the same place on the screen. Mousing over an icon renders a ghost image of the bookmarked view on top of the current view. Double clicking an icon switches to the bookmark.

In the automatic method, bookmark icons are arranged in a circular fashion around the center of the screen. The cameras are arranged in an increasing order of distance from the current camera in a clockwise manner. If the 1-point or 2-point perspective view is active, the ordering is based on the translation parameters otherwise it is based on the rotation parameters. To order cameras based on the translation parameters, each bookmarked camera's eye point is projected onto the film plane of the current camera. The magnitude of the vector from the origin to the projected eye point is used to order the bookmarks (Figure 15). The rotation distance is calculated by measuring the length of the geodesic path between the quaternion of the current camera and the quaternion of the bookmarked camera (Figure 16).

$$GeodesicPath(q_1, q_2) = Log({q_1}^{-1}, q_2)$$

q_1: Quaternion corresponding to the current scene camera.

q_2: Quaternion corresponding to the bookmarked camera.

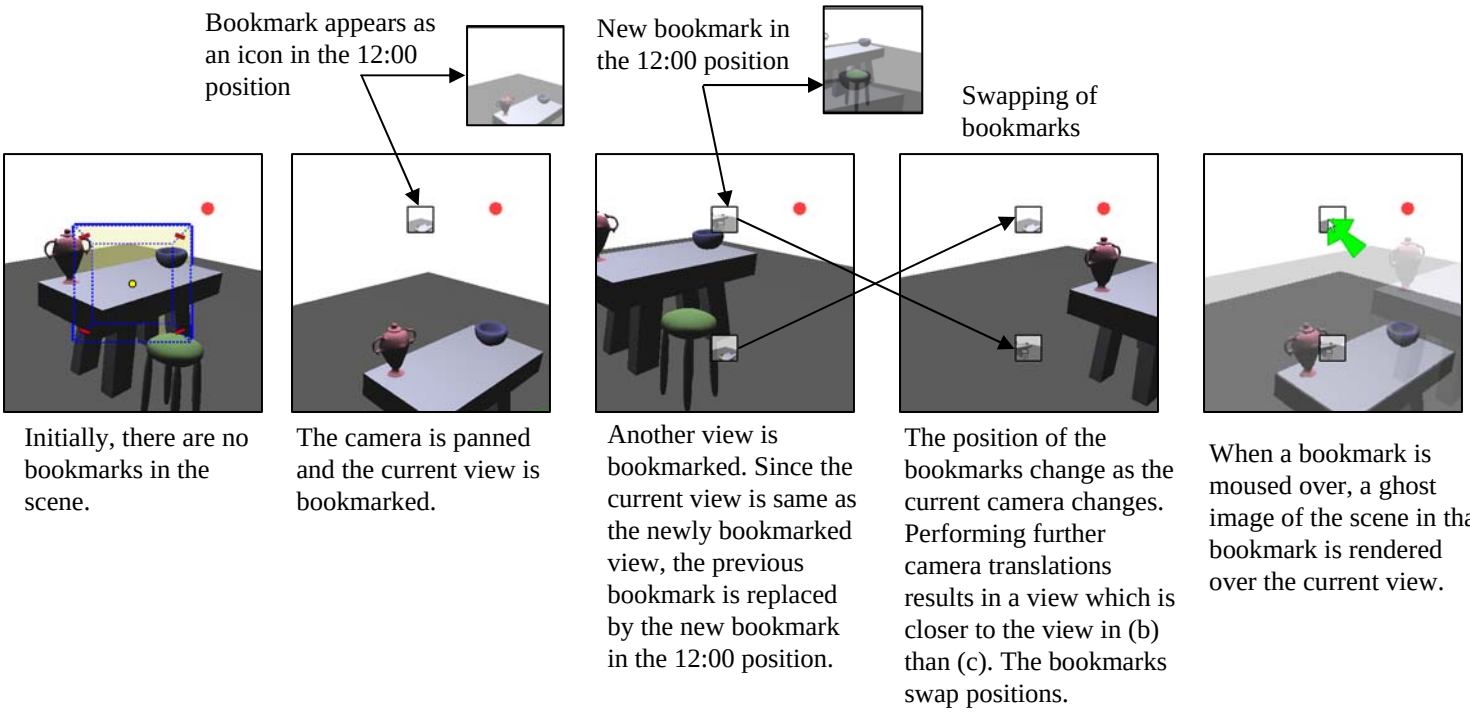

Fig. 15. Bookmarks ordered based on translation parameters

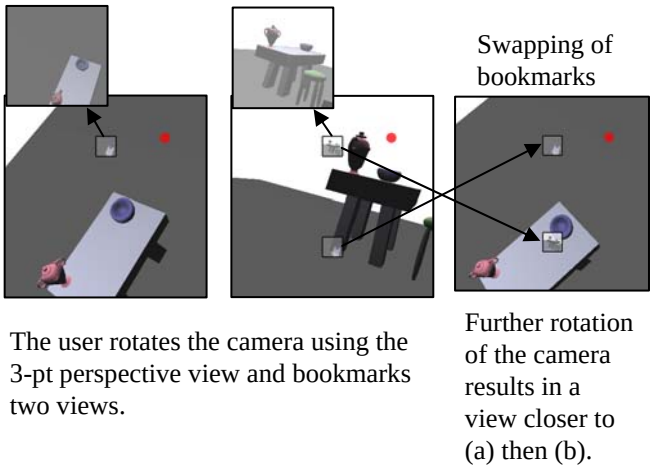

Fig. 16. Bookmarks ordered based on rotation parameters

As the user changes the current scene camera, bookmarked cameras move further or closer to the current camera. Accordingly, the ordering of the bookmark icons change.

VI. Implementation

A. Rendering a ghost scene

The current scene is rendered to the back-buffer using the ghost camera. If ghosting is turned on, the ghost camera is the camera created by modifying the current camera. If bookmarks are turned on, the ghost camera is the bookmarked camera. The scene is rendered under the original lighting in a non-photorealistic style with the silhouette edges highlighted. The contents of the back-buffer are copied into a texture which is alpha-blended on top of the original scene which is rendered using the current camera.

B. Bookmark Icons

As the number of bookmarks in the scene increases, the circle of bookmarks created by the automatic placement algorithm becomes more crowded. To prevent the occlusion of bookmarks in the circle, we vary the size of the icon as a function of the number of bookmarks in the scene.

$$SizeOfIcon = \sqrt{2} \times r \times \frac{2\pi}{3n}$$

r : Radius of the bookmarks circle
n : Number of bookmarks

VII. An Example Sequence

I plan to summarize key points of the video in this section.

VIII. User Study

We performed a small user study to compare people's performance and preferences when using CubeCam versus using the IBar. Our user study consisted of 10 participants (**proficiency**).

A. Experimental Task

Each user was asked to develop n storylines of varying difficulty with both widgets. Each storyline was placed in one of three categories (easy, medium, hard) based on their difficulty.(**give an example of an easy storyline and a hard one**). We used different storylines with each widget since building the same storyline again would be easier the second time around irrespective of the widget used. However, the storylines developed with both the widgets were equivalent in difficulty. Every storyline was specified as a sequence of conceptual camera shots(**example of a storyline**). As the user manipulated each widget, they saved each camera shot as a camera bookmark. In the case of the IBar, camera bookmarks were represented as a textual list. At the end of the session, the user answered a questionnaire. The questionnaire contained questions regarding the user's opinion of specific features of CubeCam such as ghosting and bookmarks. The questionnaire also gathered information about their preferred interface.

B. Procedure

1) The experimenter explained the functions of the IBar and CubeCam, demonstrated a few examples with each widget and explained the task to the user.
2) The user then had a practise session in which they interacted with each widget until they were comfortable with both.
3) Then, the user was randomly assigned to start with either the IBar or CubeCam. They were then given $n/2$ storylines in increasing difficulty to build with the first widget. They stopped building a given storyline when they were generally satisfied with the result. They repeated the same process with the second widget, constructing different storylines.
4) Once the session was complete, participants filled out a questionnaire.

C. Evaluation

We measured the performance of a user by the time taken to develop a storyline and the degree of satisfaction (expressed as a number between 0-5, 5 being completely satisfied with the developed storyline). Thus, the independent variables are the type of widget used and the level of difficulty of a storyline. The dependent variables are the time taken and the degree of satisfaction.

D. Results

IX. Conclusion

We have presented CubeCam, a simple intuitive screen-space camera manipulation widget. CubeCam allows users to define camera operations in terms of the change they want to see in the projected image. We visualize in image-space the relationship of the camera with respect to the scene. This is achieved by explicitly visualizing the camera's focal-plane and the camera's rotation point. Users can easily manipulate the focal plane to achieve different perspective distortions. Visual aids such as ghosting help users remember the different operations associated with each camera primitive. Finally, camera bookmarks and the relationship between them are explicitly visualized in our system.

References

[1] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman, "An empirical comparison of pie vs. linear menus," in *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM Press, 1988, pp. 95–100.

[2] J. Hultquist, "A virtual trackball," in *Graphics Gems*, 1990, pp. 462–463.

[3] K. Henriksen, J. Sporring, and K. Hornbaek, "Virtual trackballs revisited," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, Mar 2004, pp. 206–216.

[4] D. A. Bowman, D. Koller, and L. F. Hodges, "Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques," *IEEE Proceedings of VRAIS'97*, no. 7, pp. 45–52, 1997. [Online]. Available: citeseer.nj.nec.com/bowman97travel.html

[5] R. Stoakley, M. J. Conway, and R. Pausch, "Virtual reality on a WIM: Interactive worlds in miniature," in *Proceedings CHI'95*, 1995. [Online]. Available: citeseer.nj.nec.com/stoakley95virtual.html

[6] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa, "The go-go interaction technique: Non-linear mapping for direct manipulation in VR," in *ACM Symposium on User Interface Software and Technology*, 1996, pp. 79–80. [Online]. Available: citeseer.nj.nec.com/76070.html

[7] M. M. Wloka and E. Greenfield, "The virtual tricorder: A uniform interface for virtual reality," in *ACM Symposium on User Interface Software and Technology*, 1995, pp. 39–40. [Online]. Available: citeseer.nj.nec.com/wloka95virtual.html

[8] J. Blinn, "Where am i? what am i looking at?" in *IEEE Computer Graphics and Applications*, vol. 22, 1988, pp. 179–188.

[9] M. Gleicher and A. Witkin, "Through-the-lens camera control," in *Siggraph*, E. E. Catmull, Ed., vol. 26, no. 2, July 1992, pp. 331–340, iSBN 0-201-51585-7. Held in Chicago, Illinois.

[10] K. Singh, C. Grimm, and N. Sudarsanam, "The ibar: A perspective-based camera widget," in *UIST*, October 2004.

[11] C. Grimm, K. Singh, and N. Sudarsanam, "The ibar: A perspective-based camera widget," Washington university in St. Louis, Tech. Rep. 7, 2004. [Online]. Available: www.cs.wustl.edu

[12] M. Chen, S. J. Mountford, and A. Sellen, "A study in interactive 3d rotation using 2d input devices," in *Siggraph*, vol. 22, no. 4, August 1988, pp. 121–130, proc. of Siggraph '88.

[13] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics : Principles and Practice*. Addison Wesley, 1990.

[14] R. V. Cole, *Perspective for Artists*. Dover Publications, 1976.

[15] A. Gooch, B. Gooch, P. Shirley, and E. Cohen, "A non-photorealistic lighting model for automatic technical illustration," *Computer Graphics*, vol. 32, no. Annual Conference Series, pp. 447–452, 1998. [Online]. Available: citeseer.nj.nec.com/gooch98nonphotorealistic.html

[16] C. o'Connor Jr., T. Kier, and D. Burghy, *Perspective Drawing and Application*. Prentice Hall, 1998.

[17] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes, "Sketch: An interface for sketching 3d scenes," in *Siggraph*, Aug. 1996, pp. 163–170.

[18] I. Carlbom and J. Paciorek, "Planar geometric projections and viewing transformations," in *ACM Computing Surveys (CSUR)*, vol. 10, no. 4, December 1978.

[19] J. C. Michener and I. B. Carlbom, "Natural and efficient viewing parameters," in *Computer Graphics (Proceedings of SIGGRAPH 80)*, vol. 14, no. 3, July 1980, pp. 238–245.