

UniCam - 2D Gestural Camera Controls for 3D Environments

Robert Zeleznik*

Andrew Forsberg[†]

Brown University[‡]

Abstract

We present a novel approach to controlling a virtual 3D camera with a 2D mouse or stylus input device that is based on gestural interaction. Our approach to 3D camera manipulation, UniCam, requires only a *single-button* stylus or mouse to directly invoke specific camera operations within a single 3D view. No 2D widgets or keyboard modifiers are necessary. By gesturally invoking all camera functionality, UniCam reduces the modal nature of typical desktop 3D graphics applications, since remaining mouse or stylus buttons can be used for other application functionality. In addition, the particular choice and refinement of UniCam controls efficiently enables a wide range of camera manipulation tasks, including translation in three-dimensions, orbiting about a point, animated focusing on an object surface, animated navigation around an object, zooming in upon a region, and saving and restoring viewpoints. UniCam's efficiency derives primarily from improved transitions between techniques and to a lesser degree from adaptations of existing techniques. Although we have conducted no formal user evaluations, UniCam is the result of years of iterative development with approximately one hundred users. During this time more conventional modifier key and menu-based implementations were implemented and either refined or rejected. Informally, users appear to require a few hours of training to become proficient with UniCam controls, but then typically choose them almost exclusively even when widget-based controls, such as those provided by an Inventor viewer, are simultaneously available.

CR Categories: H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Graphics user interfaces (GUI); H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Input devices and strategies; H.5.1 [Information Interfaces and Presentation (e.g., HCI)]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques;

Keywords: gesture, interaction, camera control, camera manipulation, direct-manipulation, 3D viewing

*bcz@cs.brown.edu

[†]asf@cs.brown.edu

[‡]Department of Computer Science, Box 1910, Brown University, Providence, RI 02912.

1 INTRODUCTION

Interactively manipulating a virtual camera through a 3D space is a fundamental concern, perhaps even the defining characteristic, of virtually all 3D graphics applications. Since 3D camera specifications include at least six distinct degrees of freedom (DOFs) (3 translational and 3 rotational)¹, mouse-based² camera interfaces simultaneously control only a subset or a subspace of the camera's six DOFs. Therefore, to cover all of the 6D camera space, multiple *independent* interaction techniques, used one at a time, are required. However, merely covering the full 6D camera space may be insufficient, and so applications typically support multiple camera controls that are optimized for different, overlapping subspaces of the camera's 6 DOFs.

In general, user interfaces for selecting between camera interaction techniques are based on using multiple mouse and/or modifier buttons, manipulating different 3D widgets embedded within the 3D application space, or choosing 2D controls external to the 3D view. There are fundamental drawbacks with each of these approaches. Using multiple mouse buttons for camera manipulation means buttons must be re-mapped to perform other non-camera operations where remapping can be confusing, time consuming and disruptive. The approach of using modifier keys to switch between camera interactions extends the functionality of a single mouse button, but requires coordinated interaction between hands (or feet if using foot pedals), often induces uncomfortable working postures, and can be hard to learn and extend.

UniCam is an *integrated* suite of gestural camera controls that has evolved over a number of years based on the input of approximately 100 computer science and visual arts students and staff. The UniCam controls efficiently cover the entire 6D camera space, in part by improving existing techniques but primarily by improving the transitions between techniques. Treated individually, UniCam techniques are modest adaptations of well-known interactions. However, considered collectively, UniCam significantly advances the user interface for camera manipulation by selecting and adapting a set of techniques that complement each other and also by enabling fluid gestural transitions between camera techniques. Since all camera techniques are directly invoked with only one button of a mouse, UniCam reduces the transition time and effort when switching between camera manipulation and application-specific interactions that are mapped to other mouse buttons. Finally, requiring only a single mouse button, UniCam is well-suited toward pen-based applications where keyboards may not be present, screen display space may be limited, and 2D or 3D widgets may be inappropriate.

¹Additional camera degrees of freedom, including clipping plane distances, viewing angles, and film plane direction are typically of importance to a relatively smaller set of 3D graphics applications and are not considered fundamental for the scope of this discussion

²For the scope of this paper, *mouse* should be treated synonymously with a common three-button tablet stylus.

2 PREVIOUS WORK

Although virtually all 3D applications provide support for manipulating a virtual camera, there is essentially no standard suite of camera controls. Instead there are relatively common individual direct manipulation techniques for orbiting and translating, and automatic techniques for flying.

Orbiting techniques map the XY motion of a mouse to motions over the surface of a (often invisible) 3D primitive such as a cylinder or sphere that is embedded in the 3D view. Such techniques only cover a subspace of the camera's full 6 DOFs since the camera's position and orientation is essentially restricted to the surface of the 3D primitive. The most widely known orbiting techniques derive from Chen's virtual sphere[1]. Other approaches to orbiting techniques, popularized by standard Inventor viewers[7], are based on manipulating a 2D control external to the scene that maps to a horizontal or vertical circular orbit around the center of the scene. Each of these orbiting techniques offers a tradeoff between general movement and optimized movement through only a specific subspace of the camera's 6 DOFs.

Techniques for translating the camera either parallel to the film plane or along either one or two orthogonal axes of a 3D scene are commonplace. Phillips [4] describes a typical approach where XY mouse motion is mapped to one of the three principle 3D axes as determined by modifier keys and chorded mouse buttons. Osborne[6] explored three metaphors for direct camera manipulation techniques using a 6DOF input device. Although such input devices are outside the scope of this paper, their scene-in-hand metaphor is similar to mouse-based camera translation techniques, and their discussion of user difficulties with eye-in-hand metaphors is relevant. Our translation and rotation techniques are also similar to Pierce's immersive VR navigation techniques [5].

The flying metaphor is, perhaps, the most prevalent approach for navigating through virtual environments. This metaphor allows the user to steer a momentum-based camera through an environment typically using just a single mouse button. Based on the location of the pointer in the display screen or the state of keys on the keyboard, the camera may turn left or right, up or down, or in some cases fly forward or backward. While this metaphor is effective for navigation tasks in which the environment is large, it is much less appropriate for object viewing tasks[6].

A different approach to camera manipulation is to have the camera animate from one location to another in response to user actions. For example, users may explicitly select a feature to look at causing the camera to smoothly animate to a "good" location [3], or the system may automatically initiate an animated camera change when it perceives that the user *needs* to be in a different location to perform an interaction task [4].

Transitioning between camera techniques has been largely unaddressed in the literature. Zeleznik [8] discusses fluid gestural transitions between camera interaction techniques in a system related to UniCam that, however, requires two 2-DOF input devices. Gleicher also enables elegant transitions between camera manipulations using Through-the-Lens Camera Controls [2] – controls that enable a camera to be manipulated subject to image-plane constraints. Still, Gleicher focused primarily on the constraint-solving issues and not a complete gestural interface for camera control. The most effective techniques for transitioning between camera operations use specialty devices such as joysticks or a keyboard that maps camera operations to different keys. Although these approaches enable fluid transitions, they are only appropriate in some contexts and can require considerable learning before a user can perform rapidly and accurately.

3 OVERVIEW

Our integrated suite of camera manipulation techniques consists of eight distinct camera controls:

- Film plane translation
- Dollying³
- Orbiting about the center of the display
- Orbiting about a specific 3D point
- Click-to-focus on surface points
- Click-to-focus on silhouette edges
- Region zooming
- Saving and restoring a viewpoints

In the following sections, we describe each of these techniques and indicate how each technique is invoked with a single mouse button.

4 CAMERA CONTROLS

Our viewing window is broken up into two regions, a rectangular *center* region, and an outer *border* region whose border width is 1/10th the width of the center rectangle. Neither of these regions is explicitly demarcated, however both regions support a different set of gestural camera controls (see figure 1).

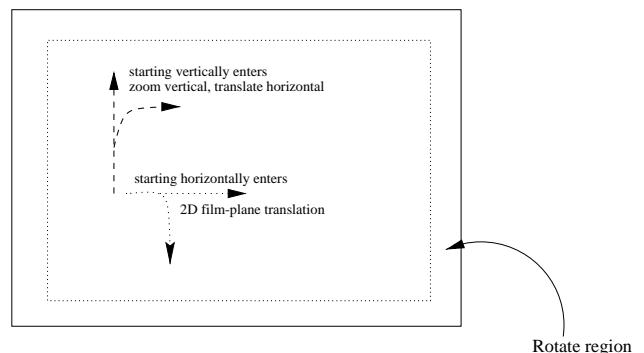


Figure 1: Gestural controls for translation and orbiting.

4.1 Camera Translation

Clicking in the *center* region of the viewing window initiates either a film plane translation mode or a dollying mode. The choice of interaction modes is determined by comparing the first few mouse deltas after the initial button press to a horizontal and vertical line. The number of deltas to wait before choosing a camera mode is arbitrary, but we have informally found that users are disturbed by time delays greater than 1/10th of a second. In addition, we have found that it is necessary for the mouse to move far enough so that an accurate direction can be made (we use 15 pixels for a 1280 x 1024 pixel display).

³For this paper, we define dollying as translating a perspective camera along a view vector, or scaling the film plane of an orthographic camera.

4.1.1 film plane translation

If the first few deltas indicate a horizontal line, then the interface transitions to a 2D film plane translation mode in which the camera translates parallel to the film plane but in the opposite direction of the mouse. Thus for example, if the mouse moves up to the right, the camera moves down to the left which in turn causes the image to move in the direction the mouse is moving. The amount of camera translation is chosen such that the *hit* point⁴ always maintains exact pick correlation with the cursor during the interaction. For perspective projections this translation delta can be computed by the finding the vector between the intersection points of two rays that start at the camera's from point, pass through the cursor's new and old positions respectively, and finally intersecting a plane parallel to the camera's film plane and containing the *hit* point. This interaction has a similar manipulation "feel" to grabbing a piece of paper and moving it.

Note that since this camera translation mode is only entered if the first few deltas indicate a horizontal line, the user must perform a "gesture" to get the camera to move straight up or down. This gesture is to move the mouse slightly horizontally at first to engage the film plane translation mode before moving up or down. The slight horizontal deviation induced by this gesture can be corrected if necessary during the subsequent interaction. The indirection of this gesture, although not ideal, is nearly unnoticeable as users become accustomed to the technique and develop muscle memory.

4.1.2 dollying

If the first few deltas of motion indicate a vertical line, then the interface transitions to a dollying mode in which horizontal motions of the mouse still translate the camera parallel to the film in the same way as the previous technique. However, vertical mouse motions translate⁵ the camera directly along the ray to the *hit* point, thus combining film plane translation with conventional dollying along the film plane normal. It is critical to note that the rate of translation toward the *hit* point is a function of how far the camera is from the *hit* point. 2D mouse motion distances in our dollying technique map to percentages of the 3D distance between the camera and the *hit* point such that a delta from the top of the screen to the bottom would translate the camera to the *hit* point no matter how far away it is in 3D. Since the camera will move slower when it is close to the *hit* point and fast when it is far away, the user can quickly reach very distant points while still being able to make fine-grained translations relative to near objects.

Our choice of how to gesturally map the 3 DOFs of camera translation to 2D mouse movements involves some apparently arbitrary choices. For example, we experimented with swapping the initial mouse delta discriminators so that the film plane translation mode was entered with an initially vertical motion and the dollying mode was entered with an initially horizontal motion. Surprisingly, we found that users strongly disliked this seemingly equivalent mapping. In lieu of an explanation, we note that from our experience with gestural interaction, the most reliable technique for insuring usable interactions is through empirical evaluations.

⁴The *hit* point is the surface point in the 3D view under the 2D cursor when the mouse button is pressed. If there is no such surface point, then the hit point is the projection of the previous hit point onto the view vector through the 2D cursor.

⁵For an orthographic camera, we change the film plane width and height by proportionally the same amount that we would have translated a perspective camera.

4.2 Camera Orbiting

Our set of camera controls includes techniques for orbiting the camera about the center of the display or a specific point in the 3D scene. Although any of the common variants of virtual sphere rotation can be used, our orbiting technique preserves a sense of "up" which is appropriate for the floor-dominated environments that we use [9]. This orbiting technique is implemented by converting horizontal motions of the mouse to rotations around the normal to the floor of the 3D scene (the world-space Y axis in our implementation), and vertical motions to rotations around the camera's right vector (the vector in the film plane that points to the right on the display). The mapping is defined such that a motion from the left to the right of the screen will rotate the camera by 360 degrees around the floor normal, and a motion from the top of the screen to the bottom rotates the camera by 180 degrees around the right vector. To preserve the sense of "up", camera rotation around the right vector is clipped if the rotation would result in a negative dot product between the film plane's up vector and the world-space Y axis. However, the critically important issue for orbiting techniques is choosing the center point for the camera's rotation. We provide two variants of our basic orbiting technique that each choose the center of rotation differently.

4.2.1 orbiting about the view's center

The first orbiting technique is initiated by clicking in the *border* region of the viewing window (see figure 1). The center of rotation for this orbiting technique will be a point at some depth along the camera's lookat vector. However, since the center of the view is arbitrary, we provide a heuristic that selects and continually updates this depth based on previous camera manipulations. Specifically, this heuristic automatically sets the depth during any dollying, film plane translation, or focusing operation to be the distance from the camera's focal point to the projection of the *hit* point onto the lookat vector. This choice is critical because, we have observed, users will generally grab the object of their interest and move it toward the center of the screen in order to subsequently orbit about that object. Thus, by projecting the *hit* point onto the lookat vector, subsequent rotations will be relative to the surface of the last object touched by the user which in turn is generally their object of interest.

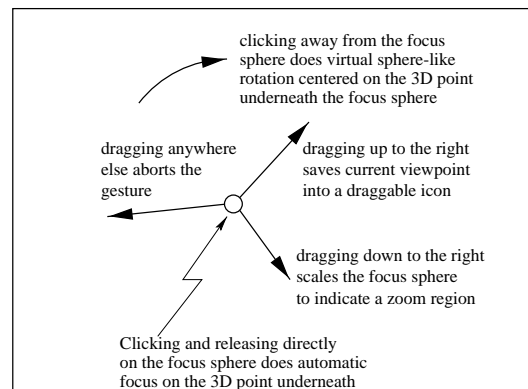


Figure 2: Camera control gestures using the focus dot

4.2.2 orbiting about a specific point

We also provide a variant orbiting technique that allows the user to orbit around any visible point. This second technique is invoked by first clicking and immediately releasing when the cursor is over

any object in the scene. Such a click results in a small *focus dot* (an approximately 10 pixel diameter blue sphere) being placed in the scene at the *hit point*⁶ (see figure 2). Then, if the user clicks a second time at any place other than on top of the focus dot, the camera rather than orbiting around the center of the display, will orbit around the focus dot.

4.3 Automatic Camera Motion

A different approach to translating and rotating the camera is to specify a lookat point. Given such a point, the camera will automatically animate from its current position to a new position that looks at the specified point. This technique has two variations that both depend upon first clicking and releasing to place a focus dot. Placement of the focus dot is the same as in the orbiting technique just described.

4.3.1 click-to-focus

In the first variation, clicking and releasing *on top* of the focus dot causes the camera to animate to a position that provides an intentionally oblique view of the surface⁷. Our oblique views are defined to look at the focus dot, but from a viewpoint that is slightly above and to the right (or the left, whichever is closer to the camera's initial orientation) of the surface normal at the *hit point*. If the normal is approximately vertical (within 10 degrees), then the viewing angle is adjusted to lean (10 degrees) toward the camera's initial position. Normally we attempt to preserve the distance from the camera to the focus point. We do this by placing the camera so that the distance from the focal point to the *hit point* is equal to the distance from the focal point to the projection of the *hit point* onto the lookat vector before focusing. In addition, we allow for a user preference that sets the distance from the camera to the focus dot so that the bounding box of the object that was clicked fills the viewing window.

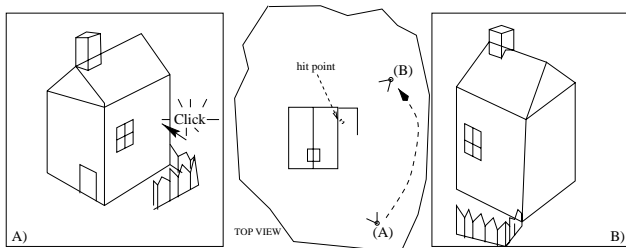


Figure 3: Viewpoint animation when click-to-focus'ing on a silhouette edge. The middle image depicts a top view of the 3D scene and the animation path for the camera from its initial viewpoint A (leftmost image) to its ending viewpoint B (rightmost image).

If the focus dot lies near a silhouette edge of an object (an edge that borders both a front, camera-facing and a rear facing object surface), then the camera's new viewpoint is set as above, except the average of the front and rear facing surface normals is used in place of the surface normal at the *hit point*. In our implementation, we determine closeness to a silhouette edge by casting rays around (10 pixels left, right, up, and down) the cursor point and checking if they all intersect the same object at approximately the same

⁶If the user clicks on empty space, then the hit point is the intersection of the ray through the click point and the plane parallel to the film plane passing through the view's "center" point.

⁷We have found slightly oblique views of surfaces to be useful for a variety of applications. However, it may be appropriate to use a different heuristic depending on the type of application.

depth. The purpose of this adjustment is to make it easy to click on a silhouette edge to navigate *around* an object (see figure 3). For example, by successively clicking four times on the leftmost vertical edge of a floor-aligned cube, a user can quickly navigate 360 degrees around the cube.

4.3.2 region "zooming"

The second variation of automatic camera motion requires selection of a region of the display that that camera will dolly in on. The user selects this region by again clicking on the semi-transparent focus dot. This time, however, by continuing to hold the button down and dragging down and to the right from the focus dot, the user interactively scales the dot into a semi-transparent sphere to indicate a "zoom-region". When the user releases the mouse button, the sphere disappears and the camera animates as it orients towards the center of the sphere and dollies until the sphere fills the smaller dimension of the camera's width or height (thus filling the display surface).

We chose to define the zoom region by placing a focus dot and expanding its radius because its effect is well-defined and easy to implement. Selecting a zoom region by dragging a rectangle from corner-to-corner is a popular technique for orthographic views, but presents difficulties for a perspective view. Although it is possible to distort the camera's perspective to effect a zoom on a rectangular region, this is generally not desirable. Without distorting the perspective, there is no clear, intuitive mapping from a 2D rectangle on the film plane to a new camera position.



Figure 4: Result of dragging up to the right on a focus dot is an inset window icon that saves the current camera viewpoint.

4.4 Camera Saving and Restoring

The last two functions in our suite of camera controls are used to save and restore camera positions. To save a camera position, the user clicks and releases to create a focus dot. Then by clicking and holding the mouse button and dragging up and to the right of the center of the focus dot, the user will automatically enter a mode where an icon depicting the current view appears out of the focus dot and attaches itself to the cursor (see figure 4). This icon will follow the cursor until the user releases the mouse button at which

point the icon just remains on the display. This icon represents the stored camera viewpoint, although the image on the icon is actually a live view of the 3D scene. In addition, the user can interact with the icon in four ways. First, the user can click and drag on the icon's border to reposition it on the viewing surface. Second, the user can drag any corner of the icon to resize it. Third, the user can just click and release over the icon without moving to automatically generate an animated transition from view's current camera viewpoint to the camera viewpoint stored in the icon. Fourth, the user can delete the icon by clicking on the icon and dragging quickly up and to the right.

5 FUTURE WORK

Although our individual camera control techniques are representative of the best and most commonly used 2D camera controls, they are still not as powerful as some techniques designed for 6 DOF input devices. We believe that more research still needs to be directed toward evaluating the potential of special purpose input devices versus further refining interaction with more general purpose devices. In particular, we believe that it may be possible to augment standard 2D input devices with additional controls that could make both interaction in three dimensions and transitions between interactions even more transparent. For example, we plan to explore both the use of the index finger dial on PC mice, and haptic input devices to enhance 3D camera interaction.

6 CONCLUSIONS

In order to improve the fluidity and efficiency of 2D camera controls for 3D environments, we developed a suite of camera controls that covers the typical 6 DOF space of camera movement. Our controls are based on a gestural interface in which a single mouse button is used to gesturally select between from a set of direct manipulation camera control techniques. Although many of our individual techniques are similar to existing camera controls, the combined set of techniques is unique for three reasons. First, our gestural approach supports seamless transitions between individual camera techniques. Secondly, our specific choice of techniques is ideally suited for a broad range of applications that require world-in-hand-style viewing (i.e., not for navigating through VR worlds). Finally, since our techniques require only one button, they fluidly integrate with application functionality that is mapped to the remaining buttons.

7 ACKNOWLEDGMENTS

This work is supported in part by the NSF Graphics and Visualization Center, Advanced Network and Services, Alias/Wavefront, Autodesk, IBM, Intel, Microsoft, National Tele-Immersion Initiative, Sun Microsystems, and TACO.

References

- [1] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-d rotation using 2-d control devices. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):121–129, August 1988.
- [2] Michael Gleicher and Andrew Witkin. Through-the-lens camera control. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):331–340, July 1992.
- [3] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. Rapid controlled movement through a virtual 3d workspace. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 171–176, October 1986.
- [4] Cary B. Phillips, Norman I. Badler, and John Granieri. Automatic viewing control for 3d direct manipulation. In *1990 Symposium on Interactive 3D Graphics*, volume 24, pages 71–74, March 1990.
- [5] Jeffrey S. Pierce, Andrew S. Forsberg, Matthew J. Conway, Seung Hong, Robert C. Zeleznik, and Mark Mine. Image plane interaction techniques in 3d immersive environments. *Computer Graphics (1997 Symposium on Interactive 3D Graphics)*, pages 39–44, April 1997.
- [6] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):175–183, March 1990.
- [7] Josie Wernecke. *The Inventor Mentor*. Addison-Wesley, 1994.
- [8] Robert C. Zeleznik, Andrew S. Forsberg, and Paul S. Strauss. Two pointer input for 3d interaction. *Computer Graphics (1997 Symposium on Interactive 3D Graphics)*, pages 115–120, April 1997.
- [9] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 163–170, August 1996.