Six Degree of Freedom Control with a Two-Dimensional Input Device: Intuitive Controls and Simple Implementations

Mark A. Livingston[1,2], Arthur Gregory[2], and Bruce Culbertson[1]

[1]Hewlett-Packard Laboratories
[2]Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

A variety of techniques have been proposed to manipulate objects in three dimensions using only a two-dimensional input device, such as a standard mouse. Some of these methods lacked full six degree of freedom (DOF) control, instead manipulating only the two DOF orientation of the object. Others provide six DOF control at the cost of complex user operations and complex implementations. We extend one popular manipulation paradigm, the ARCBALL controller, to a full six DOF controller. We provide two implementations of six DOF control, with the new implementations of operations made simple by taking advantage of graphics library operations.

Keywords: view control, mouse, user interface, interactive graphics, 3D graphics, rotation, orientation, translation, navigation

1 Introduction

Manipulation of objects in $\Re^3$ has six degrees of freedom (DOFs), three for translation and three for rotation. Equivalent to moving the entire world is manipulation of the camera through the environment. Such control would allow one to generate any desired view of a 3D scene. Standard graphics libraries such as OpenGL [6] include a variety of routines for specifying the pose (position and orientation) of a camera viewing the environment, but leave the control of that pose to the application program.

A major factor in the difficulty of controlling the camera pose with six DOFs is that the standard user interface tool—the mouse—has only two DOFs. This clearly presents a challenge to the interface designer, as a 3D input device has definite advantages over a 2D device in designing a navigation interface [2]. However, until 3D input devices become as ubiquitous as 2D devices, we must continue to design navigation interfaces that take advantage of only those features present in a typical 2D input device. One can thus understand the considerable difficulty the application programmer has in providing a suitable user interface. This is compounded by the fact that the programmers tend to be experienced with a variety of input devices and do not always consider the aesthetic properties of an interface. These issues, however, are beyond the scope of this paper.

In this paper, we present two user interfaces that provide six DOF navigation. One adds translation to a previous interface, another implements the interface anew. Both of these interfaces have the advantage that only the current screen position of the mouse relative to the previous position affects the current view; the *path* through which the user guides the mouse during navigation does not affect the view. We feel this is an important aspect of the user interface; this is why we began by building from an existing interface with this property. A secondary goal of our implementation for both the extension to the existing technique and our new technique was simplicity of implementation. This helps further the goals of efficiency and natural feel to the interface.

We begin by summarizing the architecture of previous interfaces in Section 2. These include orientation-only controllers and a separate implementation of six DOF control. In Section 3, we introduce similar translation operations as the previous six DOF implementation to the architecture of a popular three DOF controller. This yields a system with the aesthetic advantages of the orientation controller. Section 4 presents an alternative, simple implementation that only slightly alters the resulting interface. The interface methods are summarized in Section 5.

2 Previous Work

2.1 Orientation Controllers

One of the most popular systems of controlling the relative orientation of the environment with respect to the camera is ARCBALL [4]. ARCBALL allows the user to directly specify an arc on a sphere which is rigidly attached to the environment. The arc is defined by two vectors which originate at the center of the sphere and terminate at the point on the surface of the sphere where the rotation operation was begun and is currently. In the practical sense, this means the mouse position when the rotation was begun (initiated by a button press) and the current mouse position, both transformed into world coordinates from the camera image plane and then projected onto the surface of the sphere. This arc naturally defines a rotation of the ARCBALL by rotating around the normal to the plane which contains the two vectors, by an angle that is proportional to the length of the arc.

Another orientation controller is the virtual trackball [3]. This interface also transforms mouse motion on the screen to rotations of a sphere affixed to the environment, albeit with different mappings to axes of rotation. The virtual trackball, however, was regarded as difficult to use because the rotation that occurs depends on the path of the mouse across the screen, not solely on the final mouse position. Further, it was difficult to return the camera to its initial setting if both input DOF were active. (A constrained version which utilized only one axis at a time would have been easier, but necessarily less powerful.)

The ARCBALL avoided this problem by using the arcs to map mouse position on the screen to rotations. If the mouse were returned to its initial position, the view would be returned to its initial state. This is a useful property for a user interface for camera manipulation. However, we found the ARCBALL interface made it difficult to control the direction that was vertical on the screen—i.e. which way was "up." We felt the lack of control of that degree of freedom made it difficult to navigate through our environments in which the world inherently defined a vertical direction.

2.2 Six DOF Control

Translation operations complement orientation operations to provide six DOF control. This can lead to a need for multiple user interface controls (e.g. buttons or keystrokes) just for navigation. Interpreting the mouse motion in gestural form can reduce the required number of controls. One of the design goals of the UniCam interface [7] was to require the use of only one button for navigation control. This leads to an interface in which mouse motion not only determines the amount of motion within the DOFs, but also disambiguates between which type of motion the user desires. The operations for orientation are similar to those for the orientation-only controllers described above. Operations for translation consist of translation along the view ray, translation parallel to the camera image plane, and apparent translation by zooming the image[1].

Other applications, for example many VRML browsers, use widgets on screen to map user actions to navigation operations. These tend to be camera-centric operations. We find these interfaces lacking in control of the sense of "up" in the world as the user navigates. Other systems map mouse motion, velocity, and acceleration to different navigation operations. These are similar to the gestural controls in that they require interpretation of how the mouse moves. We feel these interfaces place too many requirements on the user for accurate navigation.

3 Extending ARCBALL to Six DOF

We opt to use the ARCBALL interface as our basis for a navigation system. The most obvious limitation of the ARCBALL interface, as noted in the original description, is that it lacks a translation control. One cannot move closer to an object to examine it in detail. We introduce three operations to the architecture: translation along a specified view ray (an operation also introduced in the gestural interface), zooming around a specified 3D point (also introduced in the gestural interface), and recentering the sphere.

We do not share a need for reserving only one button with the gestural method, and thus prefer to limit the need for precise mouse gesture control on the part of the user. For our new operations, we do not adopt the screen subdivision present in both UniCam and in ARCBALL (and thus our orientation controls). Thus translation once the camera is already near a large, complex object need not require "backing out" to recenter on a nearby point.

3.1 Zoom or Apparent Zoom

Adding zooming to the ARCBALL interface is quite simple. The original implementation [5] simply applied a rotation to the current orientation, which maintains the distance at which the initial view was from the center of rotation. However, we can change the distance or apparent distance as follows. When the mouse button is pressed, it denotes a ray through the camera image plane to the object. We intersect this ray with the object to get a direction in world space. We can provide an apparent zoom effect under either orthographic or perspective projection. For orthographic projection, we simply scale the borders of the projection volume (Figure 1). For perspective projection, we scale the objects in the world (but not the camera position) around the intersection point[2] (Figure 2). These operations are easily performed with the view frustum commands provided in the graphics library.

```
constant zoomFactor = 0.1
zoomValue = 1
scaleValue = 1

ZoomIn( )
    zoomValue *= 1 + zoomFactor

ZoomOut( )
    zoomValue /= 1 + zoomFactor

...in display loop...
xBorder = ( width / height ) / zoomValue
yBorder = 1.0 / zoomValue
glOrtho( -xBorder, xBorder, -yBorder, yBorder,
          zNear, zFar * scaleVal )
```

Figure 1: Implementation of an apparent zoom of the user's view of the environment.

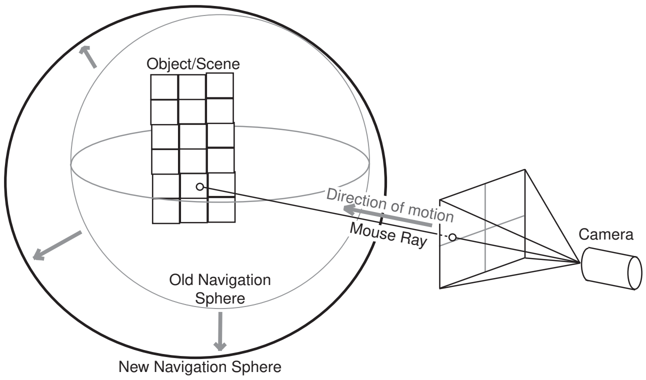

Figure 2: Extension of ARCBALL to include translation by allowing the user to zoom in along a line of sight determined by the mouse. In this scene using perspective projection, we can provide apparent zoom by scaling the world around the intersection point. Alternatively, we can translate the camera along the line of sight. In our technique, we accompany this with a corresponding translation of the center of the ARCBALL sphere, which will influence future navigation operations in a way that the scaling implementation does not. This concept can be applied to orthographic projection as well, as the code in Figures 1 and 4 show.

[1] The orientation-only controllers can also be thought of as translating the camera while maintaining a fixed point of observation, sometimes known as *orbiting* the object. These two operations are indistinguishable from the screen if the entire environment is affected by the operation, and thus we refer to orbiting operations as changes in the orientation, regardless of the implementation.

[2] This point is known as the *hit point* in [7].

3.2 Translation With or Without Recentering

Alternatively, we can translate the camera along the ray defined by the mouse position. This is identical to the operation provided by [7], although we do not adopt their speed control in our implementation. Under orthographic projection, the view generated by this translation will be the same as that generated while only the zoom operation is performed, but we perform an extra operation that makes this operation different. We also change the position of the center of the ARCBALL sphere. The difference will be seen when the user rotates the model. If the user has zoomed in, part of the model may now be behind the camera. A subsequent rotation will then reveal previously hidden portions of the model. This can be a powerful tool for examining the "interior" regions of concave objects (Figure 3).

Under perspective projection, we perform the same two operations: translate the camera position along the specified ray and translate the center of the ARCBALL sphere. This changes not only the apparent size similarly to the way scaling the objects in the world does, but also influences future rotation operations by changing the center of rotation.

These operations allow the user to navigate around the model by "pulling" the camera towards a point on the model (defined by a ray, which is in turn defined by a mouse click), and/or "pushing" the center of rotation towards that same point. (These terms are not meant to be opposites, since they do not affect the same set of parameters. We mean only to convey the sense of motion relative to the camera's point of view.)

The effect of zooming into the interesting point on the model under the mouse cursor is further achieved by scaling it before drawing. With the finite z resolution available in the standard graphics frame buffer, we must also attenuate the farplane distance in the orthographic projection matrix (Figure 4). This is virtually identical in spirit to the "dollying" and "orbiting about a specific point" provided by [7]. Note that the scaling variable used to control the far plane, as shown in Figure 1, is updated in this operation.

```
MoveInit( )
    camRay = ImagePointToWorldRay( x, y,
        world-to-camera-matrix )
    newCenter = Intersect( camRay, model )

MoveIn( )
    MoveInit( )
    centerDispl = newCenter − center
    center = center + centerDispl * zoomFactor
    scaleValue *= 1 + zoomFactor

MoveOut( )
    MoveInit( )
    scaleValue /= 1 + zoomFactor
    if( scaleValue < 1 )
        scaleValue = 1
```

Figure 4: Implementation of a zoom and recentering of the user's view of the environment.

We have found this to be a useful navigation interface which carries the primary advantages of the ARCBALL interface: intuitive control for the user and independence of the operations from the path the mouse takes on the screen. We prefer the multiple buttons to engage the various operations rather than rely on the user to perform gestures on the screen, which, however small they may be, seems to retreat from the ARCBALL's goal of path independence.

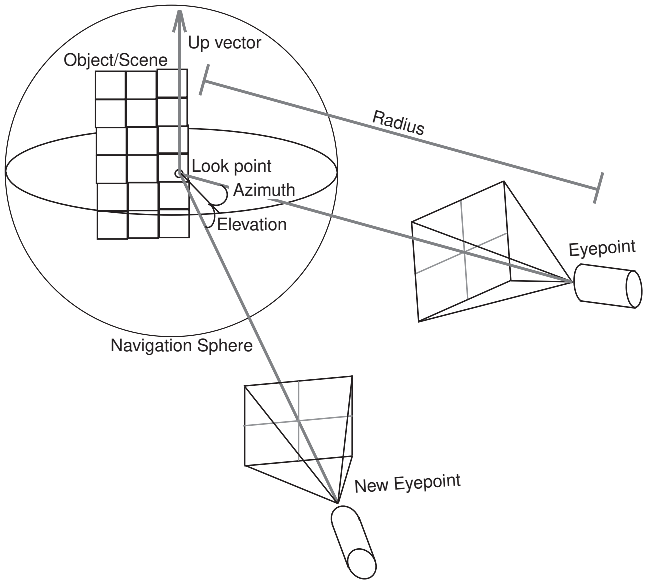

Figure 5: Our new six DOF navigation method maps user interface controls to the azimuth, elevation, and radius of the eye point from the look point, the look point position, and the direction of the up vector as seen by the camera.

Our implementation takes advantage of hardware implementation of the pick function in the OpenGL graphics library utilities [6]. The application can query the graphics engine for which polygons currently project to a given screen point. Hence we only need to compute the intersection point with a single primitive. Although this works well for models that are decomposed into polygons before drawing, some data sets might not be displayed in this fashion. Hence it would be nice to eliminate this necessity.

4 A Simpler Implementation

While the original ARCBALL with the extension to it described in Section 3 or the gestural interface discussed in Section 2 both provide intuitive methods for controlling the view, they have added significant complexity to the implementation. It is time to examine the geometry of the operations from a new perspective, in order to simplify the implementation of these basic operations.

The geometry of the ARCBALL interface can best be understood by examining the "look-at" transformation [1]. ARCBALL offered the user control of three DOF. These were understood as the orientation of the environment (or, more precisely, the orientation of a sphere rigidly affixed to the environment), but could equivalently be described as the direction of the camera (with respect to a fixed point of attention, or *look point*, and a fixed distance from that point, for two DOF) and the world direction that is vertical on the screen. The extension has given us control of the look point (three DOF that partly overlap with camera position, for two new DOF) and the distance to that point (Figure 5).

We can thus remap the controls of the ARCBALL to the elements that we use to specify the standard look-at transformation: the camera position, the look point, and a reference vector to tell us which way is up. While one can

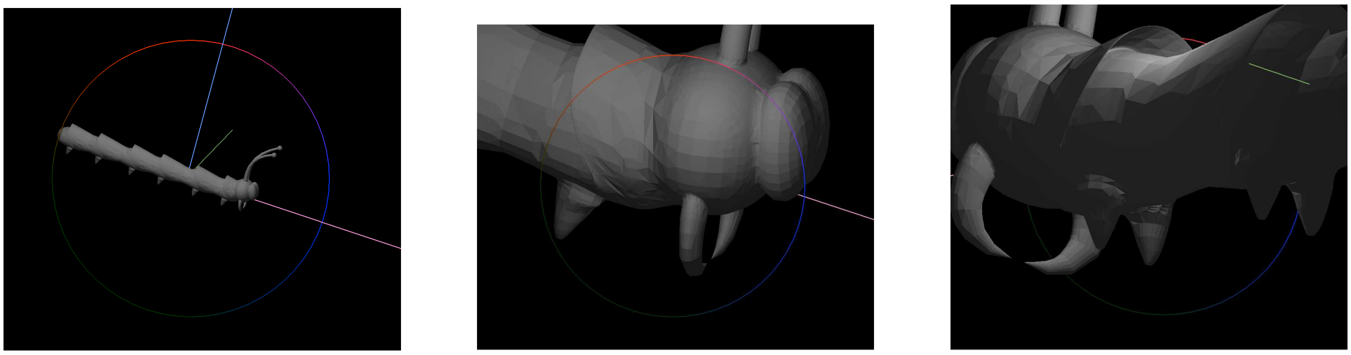

Figure 3: Demonstration of the translation portion of the user interface. The leftmost image shows the initial pose. The user then clicks on the head of the model with the middle button and drags the mouse up, zooming in, until the image looks like the middle image. From this image, the user clicks the left button on the lefthand side of the image and drags to the right, rotating through the body of the model to get a better look at the mouth from behind (rightmost image).

attribute the DOF in this configuration in a variety of ways, it is quite natural to remap our controls to the elements of this transformation.

The ARCBALL orientation change inside the silhouette (rotation by the arc defined on the sphere) maps to a change in the direction from the look point to the camera. The ARCBALL orientation outside the silhouette (rotation around the principal view ray) maps to changing the direction of the up vector on the screen. To implement the translation operations, we introduce the following restrictions from our previous interface. The view ray translation operation is restricted to translation along the principal ray—i.e. it allows a change in the look-to-camera direction and distance. The pick-and-drag operation is restricted to operate parallel to the camera image plane. It translates the look and camera points. Table 1 summarizes this mapping and the user actions we use to initiate the operations.

One cost of this implementation is that we must explicitly prevent gimbal lock by limiting the change in the elevation angle. This maintains the sense of "up" for the world to the user's choice as we navigate.

This interface is simple to implement with vector geometry operations. Code can be found at

`http://www.cs.unc.edu/~livingst/navigate.html`

along with more detailed implementation notes.

We can easily add constraints to the rotate and move operations so that only one DOF is affected by the mouse motion, as was done for ARCBALL. We select the dominant direction of the mouse motion, defined by the one in which the motion first reaches a threshold, and then eliminate any motion in the other dimension for the duration of the current operation. This implies adding some simple bookkeeping and conditional statements to the implementation. Currently, we map the controls to four button/modifier sequences, but we could use the screen-space subdivision used in ARCBALL to reduce this to one button for all orientation changes and one button for all translation changes.

5 Conclusions

Our new navigation interfaces have extended the ARCBALL interface to provide an alternative full six DOF controller. One implementation preserves the operations of the ARCBALL and adds translation by allowing the user to "pull" towards a point in the environment and "push" the center of the ARCBALL to a new position. This interface maintains the simple screen motion interface of the ARCBALL and avoids requiring specific paths for the user to follow. The operations in this interface will appear to the user to be quite similar to those in UniCam [7], so the comments regarding usability of the operations should apply. The use of screen real estate is similar to the ARCBALL, however. We have not performed a formal user study, such as [2], of the usability of the technique, but by basing our interface on previous successful interfaces, we hope to inherit usability from the previous interfaces.

A second, slightly restricted implementation allows the user to perform the same basic operations. Simplification derives from tying the screen motions directly to parameters of the standard look-at transformation of the graphics library. Of the latter interface, a user (computer-savvy but not familiar with any 3D navigation techniques) commented after only a minute of navigating how easy and natural it was to move through the environment. This hardly substitutes for a formal study, but does provide encouragement that the implementation is sound and successfully mimics the usability of ARCBALL and UniCam. We have had success navigating with both methods and find them suitable to our needs.

References

[1] Blinn, J. Where am I? What am I looking at? *IEEE Computer Graphics and Applications 8*, 4 (July 1988), 76–81.

[2] Hinckley, K., Tullio, J., Pausch, R., Proffitt, D., and Kassell, N. Usability analysis of 3d rotation techniques. In 10^{th} *ACM Symposium on User Interface Software & Technology (UIST'97)* (Oct. 1997), pp. 1–10.

[3] Hultquist, J. *A Virtual Trackball.* Graphics Gems I. Academic Press, 1990, pp. 462–463.

[4] Shoemake, K. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of Graphics Interface '92* (May 1992), pp. 151–156.

[5] Shoemake, K. *Arcball Rotation Control.* Graphics Gems IV. Academic Press, 1994, pp. 175–192.

Operation	Description	Action
Yaw and Pitch	Adjust azimuth and elevation of camera	Button-1,drag
Roll	Rotate up vector	Shift-Button-2,drag
Zoom in/out	Translate along principal view ray	Button-2,drag
Pan	Translate parallel to view plane	Button-3,drag

Table 1: Description and mapping of the user actions in the second new interface. The third column gives our mapping to user interface actions. These are merely our choice, and could be changed in another implementation to suit user or programmer tastes. Screen-space subdivision could be used to reduce the number of different buttons required, as was done for the ARCBALL interface.

[6] Woo, M., Neider, J., and Davis, T. *OpenGL Programming Guide*, 2^{nd} ed. Addison Wesley Developers Press, 1997.

[7] Zeleznik, R. C., and Forsberg, A. Unicam–2D gestural camera controls for 3D environments. In *1999 ACM Symposium on Interactive 3D Graphics* (Apr. 1999), pp. 169–174.