ARCBALL: A User Interface for Specifying Three-Dimensional Orientation Using a Mouse

Ken Shoemake
Computer Graphics Laboratory
University of Pennsylvania
Philadelphia, PA 19104

Abstract

Arcball is an input technique for 3-D computer graphics, using a mouse to adjust the spatial orientation of an object. In Arcball, human factors and mathematical fundamentals come together exceptionally well. Arcball provides consistency between free and constrained rotations using any direction as an axis; consistent visual input and feedback; kinesthetic agreement between mouse motion and object rotation; and consistent interpretation of mouse position. Attention to mathematical detail facilitates the tasks of users and implementors. Users say that as a general-purpose rotation controller Arcball is easier to use than its nearest rival, the Virtual Sphere. It is also more powerful, and simpler to implement.

Résumé

Arcball est une interface utilisateur pour un système de visualisation 3D utilisant la souris pour orienter les objets dans l'espace. Dans Arcball, facteurs humains et concepts mathématiques se marient exceptionnellement bien. Arcball assure la consistence entre rotations libres et contraintes quelle que soit la direction de leur axe. Consistence entre l'entrée et la réponse; accord kinésique entre mouvements de la souris et rotations de l'objet; interprétation consistente des positions de la souris. Les utilisateurs affirment que le contrôleur universel de rotation Arcball est plus simple à utiliser que son concurent direct: «the Virtual Sphere». Il est aussi plus puissant et plus simple à implémenter.

Key Words: object movement, view movement, mouse, user interface, interactive graphics, 3D graphics, rotation, orientation, human factors, quaternion

Introduction

In computer simulated scenes, most objects are manipulated as if they were rigid, even when they are rubber balls or robot arms. (Arms are divided into linked rigid pieces.) When one point of a rigid body is fixed by a translation, the remaining variability in position is called the body's orientation. Arcball is an input technique which allows users to adjust orientation using a mouse. The design is unusual in that it considers mathematical fundamentals as well as human factors to address a difficult problem.

Although we find it easy to pick up a small object in one hand, and turn it this way and that to examine it, it has proved much harder to devise a mouse interface which feels nearly as natural. One problem is that changes in orientation can be made in three independent directions—for example, a rotation about a left-right x axis, about an up-down y axis, or about an in-out z axis. In contrast, a mouse can move in only two independent directions.

A deeper problem is the curved geometry of orientation space, which is quite different from the flat space of mouse movements. For example, a 360° rotation leaves the orientation unchanged, while a 180° z rotation gives the same orientation as a 180° x rotation followed by a 180° y rotation. The first case would be like pushing the mouse straight forward and finding it back where it started, while the second would be like raising the mouse in the air by pushing it forward and to the side.

Moreover, closed loops of mouse motion may not produce the closed loops of rotation one expects. Rotate an object by +90° x, +90° y, −90° x, and then −90° y, and it is not back as it started, but off by 120°. This "hysteresis" effect is not inevitable, and would not be tolerated in a translation controller. The point is not that such behavior precludes undoing a complete drag sequence, but that it is unforgiving during dragging. Yet only Arcball avoids hysteresis, by a more careful mapping of mouse input to rotation.

Alternative mouse input techniques were recently evaluated in [Chen 88]. These include simulating sliders or treadmills; selecting a coordinate axis (by menu or by mouse button) then dragging; approximating a trackball; and so on. All separate rotations into independent x, y, and z angles (at least internally), yet psychological studies show that mental models of rotation do not [Carlton 90]. The virtual trackball of [Hultquist 90] computes an instantaneous rotation axis directly, but still exhibits hysteresis.

To avoid hysteresis, we must begin with the mathematical fundamentals. From those, we are immediately led to the basic Arcball design. Then we will see how to augment free rotation with a constraint mode. After pausing to admire an elegant quaternion implementation, we end by evaluating Arcball's success as a general-purpose rotation controller.

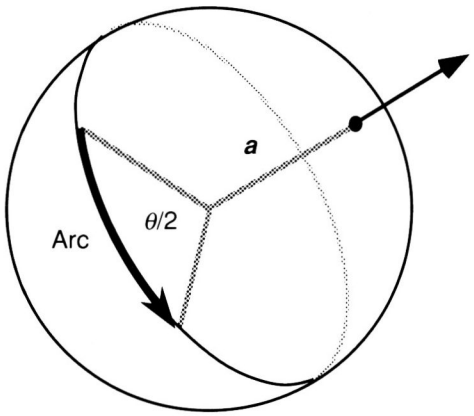

Figure 1. Arc interpretation

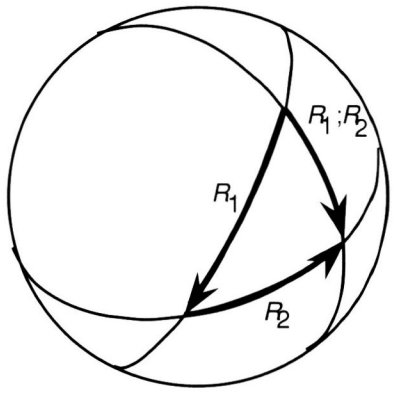

Figure 2. Arc combination

Mathematical Fundamentals

Any orientation of a rigid body can be given by a single rotation, a turn about some axis, starting from an agreed-upon reference orientation. Furthermore, the combination of any number of rotations can be given by a single rotation [Euler 1752] [Goldstein 80].

The combination law takes different forms, depending on the rotation parameters used. Behind the various formulae, however, is simple spherical geometry. A rotation R about axis $\boldsymbol{a}$ by amount θ can be represented on a sphere as any directed arc of length $^1/_2\theta$ in the plane perpendicular to $\boldsymbol{a}$, with positive angles giving a counter-clockwise direction around $\boldsymbol{a}$. (See Fig. 1.) When the end of the first arc is made to coincide with the beginning of the second, two sides of a spherical triangle are formed. The arc completing the triangle, from the beginning of the first arc to the end of the second, represents a single rotation which has the same effect as performing R_1 followed by R_2. (See Fig. 2.) Since rotations do not commute, when the order of combination is reversed, a different arc results. Otherwise, this is like vector addition.

Consider the example given earlier of 180° rotations about x and y giving a 180° rotation around z; this is represented by a 90° arc from the north pole to the equator, followed by a 90° arc along the equator. The third leg of the triangle is indeed a 90° arc, and represents a 180° z rotation. That this is the correct result is easy to verify using a physical object. Furthermore, half-length arcs were essential, as full-length arcs would have predicted a result of no rotation.

The hysteresis example is more complicated, for we must slide arcs around on their great circles in order to add them. First we go from 45° north down to the equator, then 45° east. We then slide the diagonal result arc back by its length, so it ends where we began. We are half done. Now we add a 45° arc to the north pole, giving a result arc also going due north. We slide this down so it ends on the equator, then add a final 45° arc going west. The outcome of all this is a 60° diagonal arc, representing a 120° rotation around the axis (1,-1,-1). This is not at all a simple closed path, but it is how the rotations truly combine.

From these two examples we can see how previous rotation controllers go astray mathematically. Most do not use any kind of spherical model, and those that do rotate "physical" spheres through full-length arcs. Unlike translations, rotations do not admit a choice of "C:D ratio" in this aspect of their model; only half-length arcs are permitted.

The mathematical and physical spheres are similar in that, while arc lengths differ, their planes or axes correspond. While the length problem is surprising, and perhaps confusing, it is also unavoidable. (How many kids have thought adding fractions would be much simpler if we could just add numerators and denominators separately?) For an excellent discussion of the relevant mathematical and physical theory, see Chapter 4 of [Biedenharn 81].

Arcball

Arcball takes its design—and its name—directly from the mathematics. Consequently, rotations can be displayed, as well as input, in a graphically meaningful way.

Suppose some object is selected on the screen. To change its orientation, the user simply draws an arc on the screen projection of a sphere. The arc is a great arc specified by its initial and final points, which are given by the mouse down and up positions. (See Fig. 3.) As the mouse is being dragged, its current position is used to compute the arc. Thus the object—or a faster-drawn stand-in—turns with the mouse. The direction and amount of turn are those of the half-length arc model described above.

A great arc—the shortest spherical path between two points—lies in the plane containing the two points and the center of the sphere. If the end points lie exactly opposite each other (which for Arcball can only happen if they lie on the sphere silhouette), the plane of the arc is not defined. In this application, however, that doesn't matter; opposite points imply a 360° rotation, which leaves the orientation unchanged. In effect, opposite points on the circle are "the same point."

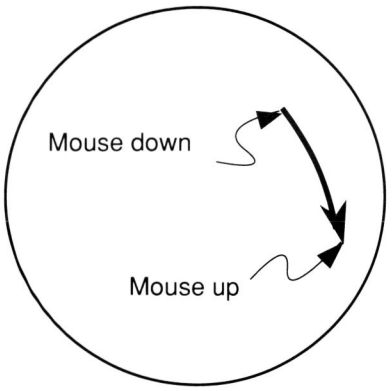

Figure 3. Mouse input

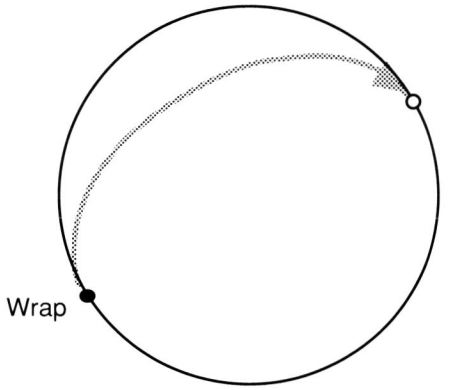

Figure 4. Wrapping.

This permits a useful feature: If the mouse is dragged out of the circle at some point the arc can be made to reenter at—wrap around to—the opposite point. (See Fig. 4.) Although the arc end jumps across the circle, the orientation being controlled changes smoothly and naturally. There is thus no limit on the amount of rotation.

We give the user graphical feedback as the mouse is dragged, by drawing a "rubber-band" arc from the initial point to the current point, and updating a picture of the scene to show the rotation of the body or camera. The sphere and rubber-band arcs can be displayed separately from the scene image, but preferably are drawn transparently on top of it.

Constrained Rotation

Spatial manipulations can be complex, and often are easier with some constraints [Bier 86]. With Arcball, fixed axis constraints are a natural extension. Axes can be chosen from sets of any size, based on the object, the environment, the view, or other sources. Natural axes to allow include the view coordinate axes, the selected object's model space coordinate axes, world space coordinate axes, normals and edges of surfaces, and joint axes of articulated models (such as robot arms).

Remember that every arc lies in a plane perpendicular to its rotation axis. If the two mouse points on the sphere are orthogonally projected onto a fixed plane through the sphere center then radially projected back onto the sphere, they will necessarily become points giving a rotation about the fixed axis perpendicular to that plane. (Fig. 5 illustrates a body *z*-axis constraint.)

A method must be provided to ask for constraint, and to select the axis to use. Many choice mechanisms, such as menus, are possible, but one is particularly attractive. Superimpose on the sphere a great circle (the front half) for each axis from a limited set. Thus for body coordinate axes, three mutually perpendicular arcs would be drawn, tilted with the object. When the mouse is clicked down to initiate a rotation, the constraint axis selected—and the only one shown—will be that of the nearest arc. So that the user doesn't have to guess, we dynamically highlight the nearest arc as the mouse is moved around prior to clicking. Mouse, menu, or keyboard combinations can be used to select among axis sets (e.g., SHIFT-click for camera axes, CTRL-click for body axes, unmodified click for no constraints).

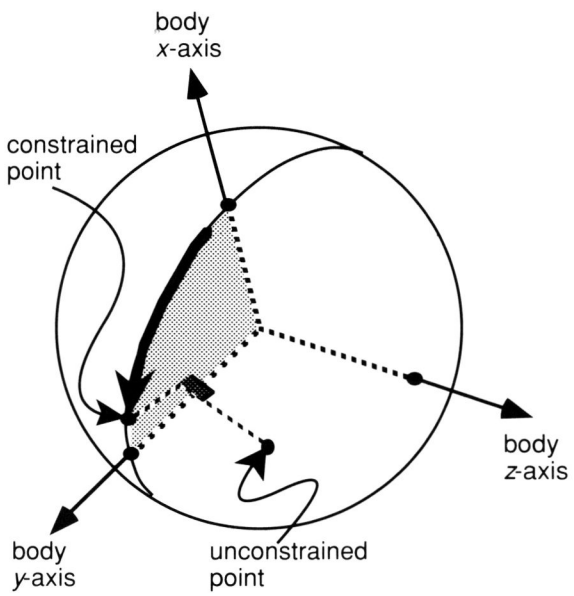

Figure 5. Constrained z rotation

Implementation

All the code for Arcball is here. The sphere can be indicated by drawing its silhouette circle. To transform cursor coordinates on the screen into a point on the sphere, the center of the circle is subtracted from the cursor coordinates giving a radial vector, which is divided by the radius of the circle to give two of the coordinates on the unit sphere. If the cursor lies outside the circle, that is easily corrected now. The third sphere coordinate is obtained as the quantity which makes the sum of the squares 1. Specifically, let the cursor screen coordinates be `screen.x` and `screen.y`, let the center of the circle be at `center.x` and `center.y`, and let the radius on the screen be `radius`. Then the coordinates on the sphere are given by

```
pt.x ← (screen.x - center.x)/radius;
pt.y ← (screen.y - center.y)/radius;
r ← pt.x*pt.x + pt.y*pt.y;
IF r > 1.0
    THEN s ← 1.0/Sqrt[r];
         pt.x ← s*pt.x;
         pt.y ← s*pt.y;
         pt.z ← 0.0;
    ELSE pt.z ← Sqrt[1.0 - r];
```

When a constraint axis is being used, the `sphere` point is projected onto the perpendicular plane, flipped to the front hemisphere if necessary, and renormalized before being used. If the point lies on the axis, an arbitrary point on the plane must be chosen. Flipping exploits the wrap effect.

```
dot ← V3_Dot[pt, axis];
proj ← V3_Sub[pt, V3_Scale[axis, dot]];
norm ← V3_Mag[proj];
IF norm > 0
    THEN s ← 1.0/norm;
         IF proj.z < 0 THEN s ← -s;
         pt ← V3_Scale[proj, s];
ELSE IF axis.z = 1.0
    THEN pt ← [1.0, 0.0, 0.0];
ELSE pt ← V3_Unit[[-axis.y, axis.x, 0]];
```

Incidentally, to find the closest arc simply constrain with each axis in turn, and pick the one that gives the nearest point. The nearest constrained point will have the largest dot product with the free point.

Having obtained the initial and final end points by this means, the rotation in unit quaternion form [Shoemake 85] is the product of the final point times the conjugate of the initial point: $q = p_1 p_0^*$. Essential quaternion facts are: (1) a unit quaternion $q = [\boldsymbol{v}, w] = [x, y, z, w]$ consists of a scalar w which is cos $\theta/2$ (where θ is the rotation angle), and a vector $\boldsymbol{v}$ which is sin $\theta/2$ times a unit vector along the rotation axis; and (2) the product of two quaternions gives the combination of the rotations they represent, and is non-commutative. The formula given amounts to setting the quaternion vector to the cross product of the initial and final points, and the quaternion scalar to their dot product. No trigonometric functions are required, only simple arithmetic.

```
[q.x, q.y, q.z] ← V3_Cross[p0, p1];
q.w ← V3_Dot[p0, p1];
```

The new orientation of the rigid body is given by the product of the quaternion for the orientation when dragging started with the quaternion we've just derived from the user's mouse input:

```
qnow ← QuatMul[q, qstart];
```

This product uses only 16 multiplies and 12 adds, and the result can be converted to a matrix at about the same cost [Shoemake 89]. If a graphics pipeline with 4×4 matrix multiply is available, accumulation and conversion can be done at essentially no cost [Shoemake 92].

From these quaternion formulae we can verify that spherical triangles behave as described earlier. Arc lengths will be half the rotation angle, since we rotate by twice the inverse cosine of the dot product, which is cosine of the arc length. Also, the axis of rotation is taken from the cross product, which is perpendicular to the vectors from the center to the end points. To go from p_0 to p_1, we use $p_1 p_0^*$, and we continue from p_1 to p_2 using $p_2 p_1^*$. The combined effect is given by the product $p_2 p_1^* p_1 p_0^*$, which for these unit quaternions is equivalent to $p_2 p_1^{-1} p_1 p_0^{-1} = p_2 p_0^*$, corresponding to the third leg of the triangle. We can also see that opposite points are equivalent, since $-q$ gives the same rotation as q.

For many purposes, the unit quaternion is most convenient, however a quaternion can easily be converted to other forms [Shoemake 85]. Although modern systems use quaternions already, an Arcball implementation can certainly be done without them. The essential step will still be to compute the dot product and cross product of the arc endpoints.

A few observations may help simplify the arc drawing code. We can approximate an arc with N line segments if we know an endpoint and a point $1/N$ of the arc length away, by reflection. Reflect the first point across the second, then the second across the third, and so on. If δ is the dot product of p_t and $p_{t+\Delta}$, then $p_{t+2\Delta} = 2\delta p_{t+\Delta} - p_t$. We find δ just once, then ignore the z coordinates. Given the endpoints, p_0 and p_1, and the arc length, $\Omega = \cos^{-1} p_0 \cdot p_1$, we can compute the extra point as $(p_0 \sin (N-1)\ \Omega/N + p_1 \sin \Omega/N)/\sin \Omega$. An arc for constraint axis $\boldsymbol{a} = [x, y, z]$ is split in two. If $s = \sqrt{1-z^2}$ is non-zero, the endpoints will be $[-xz/s, -yz/s, s]$ and $\pm[y/s, -x/s, 0]$; otherwise the "arc" is the sphere silhouette.

Orientation graphing

To go the other way, from a unit quaternion to a pair of points on the sphere, first pick an initial point—say a point on the sphere edge—perpendicular to the quaternion vector; then obtain a final point as the product of the quaternion times the initial point. The following suffices.

```
s ← SqRt[q.x*q.x + q.y*q.y];
IF s = 0.0
    THEN p0 ← [0.0, 1.0, 0.0]
    ELSE p0 ← [-q.y/s, q.x/s, 0.0];
p1.x ← q.w*p0.x - q.z*p0.y;
p1.y ← q.w*p0.y + q.z*p0.x;
p1.z ← q.x*p0.y - q.y*p0.x;
```

Also, if desired, the initial rim point can be negated when that would give a shorter arc:

```
IF q.w < 0
    THEN p0 ← [-p0.x, -p0.y, 0.0];
```

Evaluation

At this point in the paper, it's a good bet that the graphics programmers and mathematicians are happy, but the readers who go to SIGCHI conferences are wondering what became of the user in this user interface. Their concern is legitimate, for as [Gentner 90] observed, "good engineering practice can lead to poor user interfaces."

A good user interface is quickly learned and easily remembered; gets the task done quickly and with few errors; and is attractive to users [Foley 90]. These goals are more likely to be met if the design uses simple and natural interactions in "the user's language," requires little memorization, provides feedback and shortcuts, and is consistent—both with itself and other interfaces [Nielsen 90].

How well does Arcball meet these criteria? We can make a preliminary assessment before looking at user tests. It is manifestly simple to use. There are no multiple sliders, knobs, or mouse buttons. Selecting constraints is as simple as holding down a key and clicking on an arc.

Dragging on a sphere is fairly natural. The object turns the same direction the mouse moves, for a natural kinesthetic correspondence. We can repeat an orientation by repeating a position, as holding physical objects leads us to expect. On the other hand, half-length arcs are probably not in "the user's language," so are a potential problem.

Users must remember to hold down a key to get constraints, but they never have to remember which direction is object, world, or view x, y or z. The interface itself quickly reminds users of how it behaves. (As for remembering constraints, it may be helpful to add a persistent constraint mode, toggled with a menu item.)

Arcball is rich in feedback—more so than most controllers. The object rotates for a sense of direct manipulation, and a rubber-band arc shows the net effect of a drag. Constraint mode is signaled both by muscle tension [Buxton 86], and a visual display of arc choices. The arc to be selected is highlighted, for further constraint feedback. The orientation of the object can be graphed as yet another feedback arc. It is, of course, possible to augment Arcball with numeric output (and input) when that is meaningful.

Arcball hardly needs shortcuts, but there are three. First, the availability of constaints can be considered a shortcut for times when they are desirable. (Constraints can also be used as "training wheels.") Second, wrap-around makes large rotations easier; but notice that half-length arcs already let the user rotate by 360° around any axis with a single drag. Third, dragging outside the circle is an easy way imitate the use of a screen normal constraint axis. Angle detents can be added to make, say, 15° angle multiples easier.

Consistency is one of Arcball's strongest features. Object motion is consistent with mouse motion, as noted, and lack of hysteresis is a very powerful kind of consistency. Also, at any time within any drag, mouse movement between the same points will always turn the object exactly the same way. Arcs will be drawn for dragging, for constraints, and for object orientation; all are interpreted within a consistent context. Some interfaces have "trouble spots," such as gimbal lock, at certain angles; Arcball has none. Interfaces like the Virtual Sphere that depend on incremental mouse movements can behave badly if sampling is slow or coordinates are noisy; Arcball will not. There is little difference between use of the mouse in constraint mode and free mode, so Arcball can be a general-purpose controller.

The Arcball consistency of constrained rotation with free rotation means users need learn only one interface for many purposes. Consider the manipulation of a manikin arm with rotary joints [Badler 86]. The shoulder has a ball joint and so can rotate freely, while the elbow can bend around only one axis. With other interfaces, a different style of input might be required for each joint. A thumbwheel is a common choice for the elbow, but using three wheels for the shoulder is awkward. Even for the elbow, the relationship of wheel motion to spatial motion may change when the shoulder is rotated, so a horizontal stroke of the wheel corresponds to a diagonal bend of the elbow. Arcball eliminates all these difficulties, and simplifies new possibilites. For example, the foot can be forced to pivot on a sloping floor without raising the heel or toe, simply by constraining its rotation to be around the normal to the floor.

User interfaces cannot be evaluated adequately on paper, yet empirical measures of advantage are often hard to obtain. Even the study in [Chen 88] failed to find significant performance differences between two very different controllers, and only concluded the superiority of the Virtual Sphere over the method of [Evans 91] based on user comments. Yet informal evaluation should not be quickly dismissed, as studies have shown it can be informative [Nielsen 90].

Informal tests of Arcball suggest that its visual feedback provides important cues for understanding its behavior, that it is valuable to have both free and constrained modes, and that eliminating hysteresis is helpful. Arcball rotates objects twice as far as might be expected, yet few users realized that; when they did, it was not a problem. A number of users were slow learning how to rotate around the screen normal. This confusion was possible because they were given no initial hint about how Arcball worked, in order to learn as much as possible about their expectations and responses. Nevertheless, without exception, they quickly learned to use, and like, Arcball. Many described it as "a sort of trackball." Trackballs usually have only two degrees of freedom, which may explain the difficulty mentioned.

Further comment on visual feedback is in order. Unlike other controllers, Arcball can draw a meaningful rubber band arc while dragging, but it was not clear whether it was worth cluttering the picture. In fact, users appreciated the arc, and noted that it was a strong cue to the spherical nature of the controller. The circular silhouette of the controller sphere might also be eliminated; however without it, Arcball is much more difficult to use. Finally, a constraint axis can be

chosen as the nearest arc from some set; not surprisingly, it is better to draw the arc choices, and to highlight the closest one before the mouse is clicked to begin dragging.

When specifically compared to Chen's Macintosh demo of the Virtual Sphere, Arcball was the clear favorite.† Hysteresis may be hard to describe in writing, but users easily noticed the different feel when they tried the controllers together, and preferred the feel of Arcball. The Virtual Sphere had a bothersome modal distinction between drags that started inside the circle and those that started outside; Arcball did not. Use of the shift key (consistent with some other Macintosh interfaces) invoked a limited constraint mode for the Virtual Sphere, but there was no visual feedback, and users found its behavior hard to understand. Some users were slow to notice the Arcball arc highlighting (a more distinct color should probably be used), but since only the selected arc remained visible when dragging began, they could see their mistakes. In this area, too, Arcball was preferred. As mentioned earlier, half-length arcs were unexpected, but not annoying. Their benefits of wrap-around, greater range of motion, arc feedback, and hysteresis elimination seemed more important than their lack of "physicality."

Conclusions

Arcball is an elegant application of mathematical theory to interface design. Its behavior and its implementation are clean and simple. We can perform both free rotation and constrained rotation. In either case, the direction of mouse motion corresponds to the direction of object rotation. Lack of hysteresis makes Arcball more forgiving than other rotation controllers, since incremental motions are easily undone. Use of half-length arcs brings this and other benefits, without seeming unnatural. More user studies are needed, but mathematically, at least, Arcball is likely to be the best general-purpose rotation controller using a mouse.

That said, there is a bigger picture. Arcball only controls rotation, and typical 3-D interactions certainly require translation [Houde 92] and possibly scaling as well—and along arbitrary axes [Shoemake 92]. Studies indicate important differences between manipulating objects and manipulating views [Ware 90]. For the latter, egocentric controllers [Mackinlay 90] may be more satisfactory. Users may feel that consistency of controllers should encompass all three transformations, or may decide that specific tasks warrant custom rotation controllers. Ultimately, users will choose the interfaces that serve them best.

Since a single mouse position has only two degrees of freedom, a pair of positions—the ends of an arc—are used. This part of Arcball has wider applicability, including a translation controller to be described in a future paper.

† Readers who would like to make the comparison for themselves can get a copy of the Arcball demo for the Macintosh by anonymous ftp to ftp.cis.upenn.edu, directory /pub/graphics.

Acknowledgments

Xerox PARC proved a fertile ground for the blooming of these ideas. Jules Bloomenthal helped me integrate Arcball into existing software. Jock Mackinlay, George Robertson, and Eric Bier gave me enthusiastic support, often at some sacrifice, for which I am very grateful. Thanks to all the hardy folk who took the time to try Arcball and give me their comments. Pierre Crégut translated the abstract.

References

[Badler 86] Badler, Norman I., Manoochehri, Kamran H., and Baraff, David. "Multi-Dimensional Input Techniques and Articulated Figure Positioning by Multiple Constraints," in Pizer, Stephen M., ed., Proceedings 1986 Workshop on Interactive 3D Graphics, ACM, 1986, 151–169.

[Biedenharn 81] Biedenharn, L.C., and Louck, J.D. *Angular Momentum in Quantum Physics: Theory and Application*. As *Encyclopedia of Mathematics and Its Applications*, Gian-Carlo Rota (ed.), Vol. 8. Addison-Wesley, 1981.

[Bier 86] Bier, Eric A. "Skitters and Jacks: Interactive 3-D Positioning Tools," *Proceedings 1986 Workshop on Interactive 3-D Graphics* (Chapel Hill, North Carolina, October 1986), 183–196.

[Buxton 86] Buxton, William. "There's More to Interaction Than Meets the Eye: Issues in Manual Input," in Norman, D., and Draper, S., eds. *User-Centered System Design*, Lawrence Erlbaum, 1986, 319–337.

[Chen 88] Chen, Michael, Mountford, S. Joy, and Sellen, Abigail. "A Study in Interactive 3-D Rotation Using 2-D Control Devices," *Computer Graphics*, **22** (4), August 1988 (SIGGRAPH '88 Proceedings), 121–129.

[Euler 1752] Euler, Leonhard. "Decouverte d'un nouveau principe de méchanique," (1752), *Opera omnia, Ser. secunda, v. 5*, Orel Füsli Turici, Lausannae, 1957, 81–108.

[Evans 81] Evans, Kenneth B., Tanner, Peter P., and Wein, Marceli. "Tablet Based Valuators that Provide One, Two, or Three Degrees of Freedom," *Computer Graphics*, **15** (3), August 1981. (SIGGRAPH '81 Proceedings), 91–97.

[Goldstein 80] Goldstein, Herbert. *Classical Mechanics, second edition*, Addison-Wesley, 1980.

[Houde 92] Houde, Staphanie. "Iterative Design of an Interface for Easy 3-D Direct Manipulation," *CHI '92 Conference Proceedings* (Montery, California, May 3–8, 1992), ACM 1992.

[Hultquist 90] Hultquist, Jeff. "A Virtual Trackball," *Graphics Gems*, Academic Press, 1990, 462–463.

[Mackinlay 90] Mackinlay, Jock D., Card, Stuart K., and Robertson, George G. "Rapid Controlled Movement Through a Virtual 3D Workspace," *Computer Graphics*, **24** (4), August 1990. (SIGGRAPH '90 Proceedings), 171–176.

[Nielsen 90] Nielsen, Jakob, and Molich, Rolf. "Heuristic Evaluation of User Interfaces," *CHI '90 Conference Proceedings* (Seattle, Washington, April 1–5, 1990), ACM 1992.

[Shoemake 85] Shoemake, Ken. "Animating Rotation with Quaternion Curves," *Computer Graphics*, **19** (3), July 1985. (SIGGRAPH '85 Proceedings), 245–254.

[Shoemake 89] Shoemake, Ken. "Quaternion Calculus For Animation," Notes for Course #23, *Math for SIGGRAPH*, SIGGRAPH '89.

[Shoemake 91] Shoemake, Ken. "Quaternions and 4¥4 Matrices," Graphics Gems II, Academic Press, 1991, 351–354.

[Shoemake 92] Shoemake, Ken. "Matrix Animation and Polar Decomposition," *Graphics Interface '92 Proceedings* (Vancouver, British Columbia, May 11–15, 1992).

[Ware 90] Ware, Colin and Osborne, Steve. "Exploration and Virtual Camera Control in Virtual Three Dimensional Environments," *Computer Graphics* **24** (2), March 1990, 175–183.