# The Rolling Ball

Andrew J. Hanson

Indiana University

Bloomington, IN

Interactive graphics systems often need techniques that allow the user to rotate graphical objects freely in three-dimensional space using commonly-available two-dimensional input devices such as a mouse. Achieving this goal is hampered by the fact that there is no single natural mapping from the two parameters of the input device to the three-parameter space of orientations.

Here we introduce the *rolling-ball* method for mouse-driven three-dimensional orientation control, along with some of its interesting extensions to other scientific visualization problems. This technique exploits a continuous two-dimensional motion (modeled after that of a ball rolling without slipping on a flat table) to reach any arbitrary three-dimensional orientation. Unlike a variety of other methods, the rolling-ball approach has only a single state and is completely *context-free*: one can turn off the mouse cursor and ignore the history or evolving state of the motion, and yet still know *exactly* what the effect of the next incremental mouse motion will be. For applications that benefit from the impression of direct manipulation, this property is very attractive.

It is clear that a mouse can control rotations about two axes (the $x$ and $y$ directions in Figure 1). Surprisingly, the rolling ball also naturally includes the capability of inducing clockwise and counterclockwise rotations with respect to the screen perpendicular (the $z$ axis in Figure 1). According to a fundamental but counterintuitive property of the group theory of spatial rotations, moving a rolling-ball controller in small clockwise circles *must* produce small counterclockwise rotations of the ball, and vice versa. This explains why an apparently impossible third degree of rotational freedom can indeed be generated using a context-free two-degree-of-freedom input device.

The mathematical form of the rolling-ball algorithm given below is in fact included as a part of one of the algorithms studied in the extensive investigation of orientation-control methods by [Chen et al. 1988]; our approach exploits and extends the properties of the algorithm in ways that were not treated in [Chen et al. 1988]. The rolling ball should be understood as a novel, context-free *method* for taking advantage of a known rotation algorithm that is typically used in a context-dependent fashion.

The following treatment consists of two main parts. The first tells how to use the rolling-ball method for three-dimensional orientation control and how to implement it in an interactive graphics system. The second part describes how the rolling ball approach can be extended to other groups of transformations that are extremely important in scientific visualization; the rolling-ball method is then seen to be a fascinating tool in its own right for visualizing the properties of transformation groups.

# The Rolling-Ball Algorithm

## Using the Method

To understand the basic principle of the method, consider a ball lying on a table beneath the horizontal palm of your hand.

Rotations of the ball about any single axis parallel to the table top are executed by moving the hand horizontally in the direction perpendicular to the axis, thus rolling the ball about that axis. Observe that no single motion of this class produces rotations about the vertical axis, the axis perpendicular to the palm of the hand.

However, if you place your hand flat on top of a ball and move your hand in *small horizontal circles*, the ball will actually rotate about the vertical axis in the *opposite* direction.

The *rolling-ball algorithm* for controlling spatial orientation is implemented simply by treating the orientation of the graphical object to be rotated as the orientation of the ball itself, while using the mouse (or similar two-dimensional input device) to emulate the actions of the palm of the hand.

By executing the indicated motions of the mouse (or hand), one can use the rolling-ball algorithm to achieve the following effects on a displayed graphical object:

- **Rotation about a horizontal screen line**, or the $x$-axis, is carried out by moving the mouse forward or backward relative the viewer.

- **Rotation about a vertical screen line**, or the $y$-axis, is carried out by moving the mouse to the left or right.

- **Rotation about a diagonal line** lying in the screen plane, whose direction we denote by the vector $\vec{n}$, is carried out by moving the mouse perpendicular to $\vec{n}$, as though the palm were rotating a cylinder or ball about the axis $\vec{n}$.

- **Small clockwise rotations about the perpendicular to the screen**, or the $z$-axis, are carried out by moving the mouse in small, counterclockwise circles. More pronounced rotations are achieved by using larger circular motions.

- **Small counterclockwise rotations about the perpendicular to the screen** are carried out by moving the mouse in small, clockwise circles.

- **Large rotations about the perpendicular to the screen** are carried out by rotating the object 90° in any direction, rotating the desired amount about the original screen-perpendicular axis (which now lies in the screen plane), and then rotating 90° back to restore the orientation of the original screen-perpendicular axis. This action is essentially a large oblong motion, contrasting with the small circular motions used for small rotations about the screen-perpendicular.

The two most basic actions, rotation about an axis $\vec{n}$ in the screen plane and rotation about the screen-perpendicular axis, are summarized in Figure 1.

The position of the input-device cursor is irrelevant for the rolling-ball algorithm, and it is normally made invisible to the user during rotation operations. Only the difference between the previous and current device position is needed by the computation, so it is often desirable to warp the mouse to the center of the screen after each motion to prevent it from leaving the interactive window. Thus the method is truly context-free, and is well-suited to user interfaces that emphasize direct manipulation.

## Implementation

The rolling-ball algorithm is implemented by taking a given incremental input-device motion to define a vector with components $(dx, dy)$ in a right-handed screen coordinate system. The right-handed axis of rotation $\vec{n}$ is then defined as the following unit vector lying in the screen plane and oriented perpendicular to the input-device motion:

$$n_x = \frac{-dy}{dr} \ , \quad n_y = \frac{+dx}{dr} \ , \quad n_z = 0 \ , \tag{1}$$

where we define the input-device displacement $dr = (dx^2 + dy^2)^{1/2}$.

Next, we introduce the single free parameter of the algorithm, the effective rolling ball radius $R$, which determines the sensitivity of the rotation angle to the displacement $dr$; if $dr$ is a few pixels, a value of $R$ around 100 is appropriate. We choose the rotation angle to be $\theta = \arctan(dr/R) \approx (dr/R)$, so that

$$\begin{aligned} \cos\theta &= \frac{R}{(R^2 + dr^2)^{1/2}} \\ \sin\theta &= \frac{dr}{(R^2 + dr^2)^{1/2}} \ . \end{aligned} \tag{2}$$

The general form of the matrix for a rotation by an angle $\theta$ about the axis $\vec{n}$, where $\vec{n} \cdot \vec{n} = 1$, is (see, e.g., *Matrix Techniques* by M. Pique in Graphics

Gems I, p. 446 [Glassner 1990]):

$$\begin{vmatrix} \cos\theta + (n_x)^2(1-\cos\theta) & n_x n_y(1-\cos\theta) - n_z\sin\theta & n_x n_z(1-\cos\theta) + n_y\sin\theta \\ n_y n_x(1-\cos\theta) + n_z\sin\theta & \cos\theta + (n_y)^2(1-\cos\theta) & n_y n_z(1-\cos\theta) - n_x\sin\theta \\ n_z n_x(1-\cos\theta) - n_y\sin\theta & n_z n_y(1-\cos\theta) + n_x\sin\theta & \cos\theta + (n_z)^2(1-\cos\theta) \end{vmatrix} . \tag{3}$$

When we substitute into Eq. (3) the values of $\vec{n}$ from Eq. (1), we get the rolling-ball rotation matrix

$$\begin{vmatrix} \cos\theta + (dy/dr)^2(1-\cos\theta) & -(dx/dr)(dy/dr)(1-\cos\theta) & +(dx/dr)\sin\theta \\ -(dx/dr)(dy/dr)(1-\cos\theta) & \cos\theta + (dx/dr)^2(1-\cos\theta) & +(dy/dr)\sin\theta \\ -(dx/dr)\sin\theta & -(dy/dr)\sin\theta & \cos\theta \end{vmatrix} , \tag{4}$$

where the values of the trigonometric functions are given by Eq. (2).

**Notes.** We observe the following:

- All vectors must be translated to the desired center of rotation before applying Eq. (4).
- The rotation must be performed in a *single step* as shown in Eq. (4). Carrying out the rotation as a sequence, e.g., first about the $x$-axis and then about the $y$-axis, will give a completely different result (although, for subtle reasons, the difference may be nearly unobservable).

- Changing the overall sign of $\vec{n}$ produces a rotation of the viewpoint around the object instead of a rotation of the object within the view. Small clockwise hand motions will produce small *clockwise* rotations of the viewpoint, but an *object* at the center of the view will continue to rotate *counterclockwise*. This phenomenon derives from a sign difference in the group-theoretical description of body-fixed rotations versus space-fixed rotations [Whittaker 1944].

## Extensions of the Rolling-Ball Method

When we analyze the group-theoretical context of the rolling-ball method, a variety of related applications immediately suggest themselves. Here we summarize the basic group theory involved for ordinary rotations as well as several extensions that are straightforward to implement. These techniques are useful for a number of scientific visualization applications, including building intuition about groups in general. The reader who has no interest in group theory but just wants to know how to implement and use the algorithm need not read further.

## Group Theory of Infinitesimal Rotations

The underlying group theory [Edmonds 1957] involved in the behavior of the rolling ball can be summarized as follows: If we define $L_i$, $i = \{x, y, z\}$, to be the infinitesimal generators of the rotation group $O(3)$ with a right-handed convention for positive rotations, then we have the commutation relations

$$[L_y, L_z] = -L_x \tag{5}$$

$$[L_z, L_x] = -L_y \tag{6}$$

$$[L_x, L_y] = -L_z, \tag{7}$$

where we used the definition $[A, B] = AB - BA$. These infinitesimal generators can be represented as matrices or as differential operators of the form $L_x = y\frac{\partial}{\partial z} - z\frac{\partial}{\partial y}$ and its cyclic permutations. The minus sign in Eq. (7) is not arbitrary, but is determined by our convention that $L_i$ rotate a vector about the $i$th axis using the right-hand rule. This minus sign is directly responsible for the observed counterrotation, and is an inevitable consequence of the properties of the rotation group.

## Quaternion Rotations, $2\times2$ Matrices, and $SU(2)$ Spinors

The rolling-ball transformation works to define quaternion rotations (see, e.g., [Shoemake 1985] or *Using Quaternions* by P.-G. Maillot in Graphics Gems I,

p. 498 [Glassner 1990]) even more naturally than ordinary spatial rotations. This follows from the fact that the quaternion formulation is equivalent to the more standard $2 \times 2$ matrix notation for the group $SU(2)$, which is the double covering of the usual rotation group $O(3)$ [Edmonds 1957]. (Even though these two groups correspond to entirely different topological spaces, their infinitesimal properties exploited by the rolling ball are identical.)

To carry out $SU(2)$ rotations using the rolling ball, we replace Eq. (3) by

$$U = I_2 \cos\frac{\theta}{2} - i\vec{n}\cdot\vec{\sigma}\sin\frac{\theta}{2}, \tag{8}$$

where $I_2$ is the $2 \times 2$ unit matrix, and $\vec{\sigma}$ denotes the $2 \times 2$ matrix basis for $SU(2)$ obeying the cyclic relations ${\sigma_x}^2 = 1, \quad \sigma_x\sigma_y = i\sigma_z$. This is equivalent to a quaternion-based transformation with $(c,\ u) = (\cos(\theta/2),\ \vec{n}\sin(\theta/2))$. Note how much more simply Eq. (8) incorporates the fundamental parameters of the rolling ball than the full matrix, Eq. (3).

Changing the overall sign of $\vec{n}$ produces a rotation of the viewpoint around the object instead of a rotation of the object within the view.

The elements of the matrix Eq. (8) may be used to compute an ordinary vector rotation matrix from Eq. (3) as desired, and may also be used directly as $2 \times 2$ matrices to rotate *spinors* [Edmonds 1957], which are the most fundamental objects upon which the rotation group can act.

## Four Euclidean Dimensions

In four Euclidean dimensions, there are six degrees of rotational freedom from the group $O(4)$ instead of the three that are present in three-dimensional space due to $O(3)$.

The six $O(4)$ rotation operators $L_{\mu\nu},\ \mu,\nu=\{1,2,3,4\}\,,\ L_{\mu\nu}=-L_{\nu\mu}\,,$ can be decomposed into $O(3)\times O(3)$ by defining the following combinations:

$$L_i^{\pm}=\frac{1}{2}\left(\frac{1}{2}\epsilon_{ijk}L_{jk}\pm L_{4i}\right)\ . \tag{9}$$

Here $\epsilon_{ijk}$ is the totally antisymmetric tensor in three dimensions, and we use the convention that repeated roman indices are summed from 1 to 3. Each of these combinations obeys *independent* $O(3)$ commutation relations,

$$[L_i^{\pm},L_j^{\pm}]=-\epsilon_{ijk}L_k^{\pm}\,,\quad [L_i^{\pm},L_j^{\mp}]=0, \tag{10}$$

and therefore can be controlled separately using the $O(3)$ rolling-ball algorithm. The rotation generated in this way can be written as

$$R^{\pm}=I_4\cos\theta+\vec{n}\cdot\vec{L}^{\pm}\sin\theta, \tag{11}$$

where

$$\vec{n}\cdot\vec{L}^{\pm}\ =\ \left|\begin{array}{cccc} 0 & -n_z & n_y & \mp n_x \\ n_z & 0 & -n_x & \mp n_y \\ -n_y & n_x & 0 & \mp n_z \\ \pm n_x & \pm n_y & \pm n_z & 0 \end{array}\right|,$$

and the unit vector $\vec{n}$ would normally be defined by Eq. (1). Thus we can manipulate all the degrees of freedom of *four dimensional* orientation by using *two copies* of the rolling ball, one for $L_i^+$ and one for $L_i^-$.

An alternative technique, which applies also to rotations in $N$-dimensional Euclidean space, is to break up the group $O(4)$ (or $O(N)$ in $N$ dimensions) into $O(3)$ subgroups and treat each as an independent rolling-ball transformation.

## Lorentz Transformations.

Physical systems at very high velocities must be studied using Lorentz transformations of spacetime, rather than Euclidean rotations. Lorentz transformations mix space and time together and preserve a Minkowski-space quadratic form that has one negative component. Pure velocity changes, or "boosts," are similar in form to rotations with hyperbolic functions replacing trigonometric ones. A boost to a frame with velocity $\vec{v} = \hat{v} \tanh \xi$ transforms the vector $(\vec{x}, t)$ by the matrix

$$\left| \begin{array}{cc} \delta_{ij} + \hat{v}_i \hat{v}_j (\cosh \xi - 1) & \hat{v}_j \sinh \xi \\ \hat{v}_i \sinh \xi & \cosh \xi \end{array} \right| . \tag{12}$$

To implement $O(2,1)$ Lorentz transformations, which preserve the form $\mathrm{diag}(1,1,-1)$, we interpret mouse motions as small velocity changes of a "Lorentz rolling ball" in the direction the mouse is moving. (We could also study the transformation group $O(3,1)$ of physical spacetime; unfortunately, the analogy of the argument leading to Eq. (10) requires the introduction of complex vectors.)

The infinitesimal generators of $O(2,1)$ transformations are the boost operators $B_x = t\frac{\partial}{\partial x} + x\frac{\partial}{\partial t}$, $B_y = t\frac{\partial}{\partial y} + y\frac{\partial}{\partial t}$, and the operator $L = x\frac{\partial}{\partial y} - y\frac{\partial}{\partial x}$ producing rotations in the $x-y$ plane. The boost operators transform under rotations as ordinary vectors, $[L, B_x] = -B_y$, $[L, B_y] = +B_x$, while their mutual commutation produces a rotation with the *opposite sign* compared to the analogous $O(3)$ operators, $[B_x, B_y] = +L$.

We relate the mouse input to the $O(2,1)$ transformation Eq. (12) by replacing Eq. (1) by

$$\hat{v}_x = \frac{+dx}{dr} \ , \ \ \hat{v}_y = \frac{+dy}{dr} \ , \tag{13}$$

and choosing the boost parameter to be $\xi = \tanh^{-1}(dr/s) \approx (dr/s)$, where $s$ is a suitable scaling factor that ensures $(dr/s) < 1$.

We then find that moving the input device in small clockwise circles produces a rotation of the spatial part of the coordinate frame in the *clockwise* direction, which is the opposite of the result for standard $O(3)$ rotations! This effect, known as the *Thomas Precession*, makes the rolling-ball technique a very natural one for Lorentz transformations.

**Summary.** In summary, the rolling-ball technique provides an approach to controlling three degrees of rotational freedom in interactive graphics systems with two-dimensional input devices that does not depend on the state, position, or history of the input device. Because of the algorithm's rich group-theoretical origins, a number of related scientific visualization applications naturally present themselves. Becoming fluent with the technique requires some effort on the part of the user. But, once mastered, this method provides context-free, exploratory orientation adjustment that strongly supports the feeling of direct manipulation.

A portion of this work was supported by the National Science Foundation under Grant No. IST-8511751.

# References

[Chen et al. 1988] Chen, Michael, Mountford, S. Joy, and Sellen, Abigail (1988), "A Study in Interactive 3-D Rotation Using 2-D Control Devices," Proceedings of 1988 SIGGRAPH, Computer Graphics **22**, pp. 121–130.

[Edmonds 1957] Edmonds, A.R. (1957), *Angular Momentum in Quantum Mechanics,* Princeton University Press, Princeton, New Jersey.

[Glassner 1990] Glassner, Andrew S., ed. (1990). *Graphics Gems*, Academic Press, Boston, Massachusetts.

[Shoemake 1985] Shoemake, K. (1985), "Animating Rotation with Quaternion Curves," Proceedings of 1985 SIGGRAPH, Computer Graphics **19**, pp. 245–254.

[Whittaker 1944] E.T. Whittaker (1944), *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies,* Dover, New York, New York.

Figure 1: The two basic techniques used in the rolling-ball algorithm to carry out arbitrary spatial rotations of a graphics object. (a) Moving the hand with the mouse in the direction indicated by the black arrow causes the object to rotate about the axis $\vec{n}$ lying in the screen plane, i.e., to rotate the equator in the direction of the hollow arrow. (b) Moving the hand in small circles causes the object to rotate about the normal to the screen plane in the direction *opposite* to the hand motion, again rotating the equator in the direction of the hollow arrow.