Improvements in Arcball Interfaces (Oct 2021)

M. Elks

Abstract—A new arcball method is described that circumvents the limitations of current methods around the periphery of the view area. The method does not exhibit the discontinuities of previous methods. Notes are included on an improvement to user functionality traversing global space. The proposed method offers a far more intuitive user experience.

Index Terms—Computer Graphics, User Interface, Computer Modeling

1 Introduction

Arcball [1, 2] has severe limitations that impair the user experience, a new method is proposed to eliminate the non-intuitive aspects of the previous methods. The nature of arcball rotation near the screen periphery is improved. In addition, a method of movement around global space is discussed. A novel improved Method for user interaction with camera rotation is proposed along with notes on translation and zoom functionality.

2 Background

2.1 Previous Work

The original arcball method proposed by Shoemake involved projecting the screen coordinates onto a virtual hemisphere that inscribed the screen. The mouse click and release points on this hemisphere are used to calculate the rotation of the camera. The initial work published had obvious problems around the edge of the hemisphere that inscribed the screen area. The discontinuity of the motion when clicked near or outside the border the hemisphere left much to be desired [1]. A much improved solution published by Shoemake [2] combines the hemisphere with a hyperbolic function. This solution gives a smoother result past the hemispheres edge; however, it is not a full solution. An area of concern is the rapidly devolving hyperbolic approximation as the mouse approaches the edge of the screen. This leads to non-intuitive control.

2.2 Structure

The goal of the arcball calculation pipeline is to create a View-Projection Matrix. The view matrix is selected to be an orthogonal basis created from the cameras position and up vectors. A useful property of this matrix is that the inverse of the matrix is simply the transpose of the matrix. Thus the inverse of the View-Projection Matrix is quick to calculate.

Entirety of the methods described henceforth are based upon performing computation in a lower dimensional local space. The local space vectors are then transferred into global space to finish the computation. This local space is less computationally intensive than first converting to and then working in global space.

The general rotation method is best conceptualized as a circle perpendicular to the “local coordinates” of the screen. This allows for excellent visualization. It is not interacting with the actual local coordinate system of the screen; however, it is based upon the local coordinate system of the camera. A visual representation of this is given in figure one.

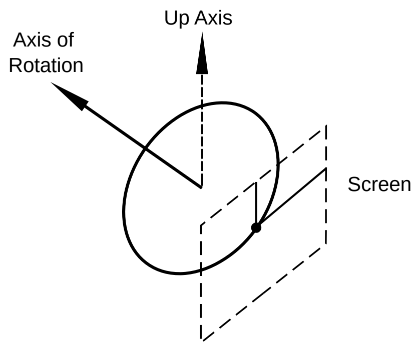

Fig 1. Geometric Visualization of Rotation Method

3 Method

3.1 Rotation

A circle is created in a two-dimensional plane that is perpendicular to the “screen.” The radius of the circle is given by the distance from the camera to the arcball center. The rotation of the circle relative to the up vector is given by the direction vector of the dragged mouse, or stylus on touchscreen devices. This direction vector is relative to the screen coordinates; however, it is also relative to the cameras local coordinate system. The up vector is always in the same direction as the screen’s y-coordinate.

The Creation of the circle allows for rotation of a vector in local coordinates, which in turn allows for rotation in global coordinates. This is done through addition of a vector. This vector can be derived from figure two and is given in Equation one.

- *The author is not with any organizations, E-mail: mattelks43216@gmail.com*

This vector is added to the camera's position vector to get the new camera position. The addition is done in global space. The vector can be projected into global space by the inverse of the view matrix.

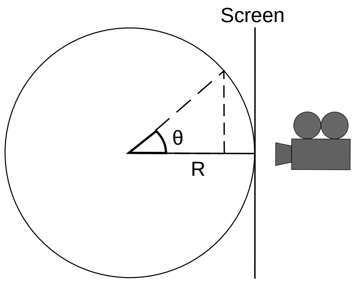

Fig 2. Side View of the Vertical Imaginary Circle

Figure two gives the geometric intuition of the formulae listed below. Since the derivation is simple, it has been omitted. The y coordinate of the vector must be reversed due to the formality of the upper left of the screen as the origin with y being positive down.

$$\theta = sensitivity * |V_{camera}| \tag{1}$$

$$X_{camera} = \frac{-\Delta x * R * \sin(\theta)}{|V_{camera}|} \tag{2}$$

$$Y_{camera} = \frac{\Delta y * R * \sin(\theta)}{|V_{camera}|} \tag{3}$$

$$Z_{camera} = R * \cos(\theta) - R \tag{4}$$

$$V_{Global\ Camera} = V_{Global\ Camera} + B^T V_{Local\ Camera} \tag{5}$$

The up vector must also be rotated. The up vector is not always perpendicular to the axis of rotation and is also perpendicular to the camera vector. To account for the former, the vector is scaled by the dot product of the up vector with the normalized mouse vector. To adjust the problem of the latter, it is as simple as deriving the equations as before, but for a vector rotated ninety degrees counter-clockwise with a radius of one. The result is formalized in the following formulae.

$$X_{local\ up} = \frac{-\Delta x * \Delta y * (1 - \cos(\theta))}{|V_{camera}|^2} \tag{6}$$

$$Z_{local\ up} = \frac{-\Delta y * \sin(\theta)}{|V_{camera}|} \tag{7}$$

$$Y_{local\ up} = \frac{-\Delta y^2 * (1 - \cos(\theta))}{|V_{camera}|^2} \tag{8}$$

$$V_{Global\ up} = V_{Global\ up} + B^T V_{local\ up} \tag{9}$$

3.2 Translation

With an orthogonal view matrix implementation, there is no need to calculate the full inverse of the projection matrix for translation calculations. A more efficient method is utilize the parallel relation between the movement vectors plane and the camera's view plane. The plane of the movement vector is coincident with the arcball's center position. The vector must be scaled in this plane according to the field of view of the camera. The movement vector is given below and can be found easily using the visualization in figure three.

$$X_{move} = \frac{-\Delta x * R * \tan(0.5 * FOV)}{0.5 * window\ width} \tag{10}$$

$$Y_{move} = \frac{-\Delta y * R * \tan(0.5 * FOV)}{0.5 * window\ height} \tag{11}$$

$$Z_{move} = 0 \tag{12}$$

$$V_{Camera} = V_{Camera} + B^T V_{move} \tag{13}$$

$$V_{Center} = V_{Center} + B^T V_{move} \tag{14}$$

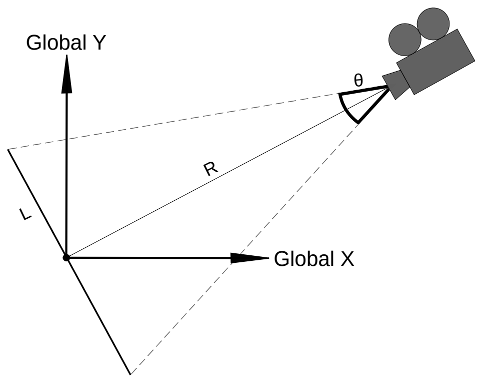

Fig 3. Geometric Visualization of Translation

3.3 Zoom

Many arcball implementations have difficulty with movement around global space. It is very difficult for the user to adjust the global position of the camera accurately with translation and pure zooming. The solution to this problem exists in very few software offerings. An alternative to translation and pure zoom is to slightly translate the arcball as the camera zooms. This is present in many 2D applications and transfers well to 3D. The translation of the camera is based upon the mouse's deviation from the screen center. One must multiply the mouse deviation by a sensitivity factor, found empirically by trial and error. It gives the advantage of easily positioning objects in the arcball's center. The ease of centering allows faster and more accurate traversal of global space.

7 Conclusion

The arcball implementation presented averts the limitations of the previous techniques. No discontinuities exist to hamper the users intuition. The user experience of the new implementation is also much improved with use of the notes on translation and zoom functionality. The user experience has increased positivity due to the intuitive and predictable nature of the method.

Acknowledgment

The author wishes to thank IEEE Computer Society for publishing their template, which was used as guide for this paper.

References

1 Shoemake, Ken. "ARCBALL: a user interface for specifying three-dimensional orientation using a mouse." (1992).

2 Shoemake, Ken. "Arcball Rotation Control." Graphics Gems (1994).

Matthew Elks B.S. Mechanical Engineering 2020. Current Research interests include: Discrete Learning Algorithms, Machine Learning, and Continuous Learning Control Systems