The Visual Display Transformation for Virtual Reality

TR94-031
September, 1994

Warren Robinett
Richard Holloway

Head-Mounted Display Project
Department of Computer Science
CB #3175, Sitterson Hall
UNC-Chapel Hill
Chapel Hill, NC 27599-3175

This research was supported by the following grants: ARPA DABT 63-93-C-0048, NSF Cooperative Agreement ASC-8920219, and ARPA "Science and Technology Center for Computer Graphics and Scientific Visualization", ONR N00014-86-K-0680, and NIH 5-R24-RR-02170.

The Visual Display Transformation for Virtual Reality

Warren Robinett[*]
Richard Holloway[†]

Abstract

The visual display transformation for virtual reality (VR) systems is typically much more complex than the standard viewing transformation discussed in the literature for conventional computer graphics. The process can be represented as a series of transformations, some of which contain parameters that must match the physical configuration of the system hardware and the user's body. Because of the number and complexity of the transformations, a systematic approach and a thorough understanding of the mathematical models involved is essential.

This paper presents a complete model for the visual display transformation for a VR system; that is, the series of transformations used to map points from object coordinates to screen coordinates. Virtual objects are typically defined in an object-centered coordinate system (CS), but must be displayed using the screen-centered CSs of the two screens of a head-mounted display (HMD). This particular algorithm for the VR display computation allows multiple users to independently change position, orientation, and scale within the virtual world, allows users to pick up and move virtual objects, uses the measurements from a head tracker to immerse the user in the virtual world, provides an adjustable eye separation for generating two stereoscopic images, uses the off-center perspective projection required by many HMDs, and compensates for the optical distortion introduced by the lenses in an HMD. The implementation of this framework as the core of the UNC VR software is described, and the values of the UNC display parameters are given. We also introduce the vector-quaternion-scalar (VQS) representation for transformations between 3D coordinate systems, which is specifically tailored to the needs of a VR system.

The transformations and CSs presented comprise a complete framework for generating the computer-graphic imagery required in a typical VR system. The model presented here is deliberately abstract in order to be general-purpose; thus, issues of system design and visual perception are not addressed. While the mathematical techniques involved are already well known, there are enough parameters and pitfalls that a detailed description of the entire process should be a useful tool for someone interested in implementing a VR system.

1. Introduction

A typical virtual reality (VR) system uses computer-graphic imagery displayed to a user through a *head-mounted display (HMD)* to create a perception in the user of a surrounding three-dimensional virtual world. It does this by tracking the position and orientation of the user's head and rapidly

---

[*] Virtual Reality Games, Inc., 719 E. Rosemary St., Chapel Hill NC 27514. E-mail: robinettw@aol.com

[†] Department of Computer Science, CB 3175, University of North Carolina, Chapel Hill, NC, 27599-3175. Email: holloway@cs.unc.edu

generating stereoscopic images in coordination with the user's voluntary head movements as the user looks around and moves around in the virtual world.

The hardware for a typical VR system consists of an HMD for visual input, a *tracker* for determining position and orientation of the user's head and hand, a graphics computer for generating the correct images based on the tracker data, and a hand-held *input device* for initiating actions in the virtual world. The visual environment surrounding the user is called the *virtual world*. The world contains *objects*, which are collections of graphics primitives such as polygons. Each object has its own position and orientation within the world, and may also have other attributes. The human being wearing the HMD is called the *user*, and also has a location and orientation within the virtual world.

A good graphics programmer who is given an HMD, a tracker, an input device, and a computer with a graphics library can usually, after some trial and error, produce code to generate a stereoscopic image of a virtual object that, as the user moves to observe it from different viewpoints, appears to hang stably in space. It often takes several months to get to this point. Quite likely, the display code will contain some “magic numbers” which were tweaked by trial and error until the graphics seen through the display looked approximately right. Further work by the programmer will enable the user to use a tracked manual input device to pick up virtual objects and to fly through the virtual world. It takes more work to write code to let the user scale the virtual world up and down, and have virtual objects that stay fixed in room or head or hand space. Making sure that the constants and algorithms in the display code both match the physical geometry of the HMD and produce correctly sized and oriented graphics is very difficult and slow work.

In short, writing the display code for a VR system and managing all of the transformations (or transforms, for short) and coordinate systems can be a daunting task. There are many more coordinate systems and transforms to keep track of than in conventional computer graphics. For this reason, a systematic approach is essential. Our intent here is to explain all of the coordinate systems and transformations necessary for the visual display computation of a typical VR system. We will illustrate the concepts with a complete description of the UNC VR display software, including the values for the various display parameters. In doing so, we will introduce the vector-quaternion-scalar (VQS) representation for 3D transformations and will argue that this data structure is well suited for VR software.

2. Related Work

Sutherland built the first computer-graphics-driven HMD in 1968 (Sutherland, 1968). One version of it was stereoscopic, with both a mechanical and a software adjustment for interpupillary distance. It incorporated a head tracker, and could create the illusion of a surrounding 3D computer graphic environment. The graphics used were very simple monochrome 3D wire-frame images.

The VCASS program at Wright-Patterson Air Force Base built many HMD prototypes as experimental pilot helmets (Buchroeder, Seeley, & Vukobradatovitch, 1981).

The Virtual Environment Workstation project at NASA Ames Research Center put together an HMD system in the mid-80's (Fisher, McGreevy, Humphries, & Robinett, 1986). Some of the early work on the display transform presented in this paper was done there.

Several see-through HMDs were built at the University of North Carolina, along with supporting graphics hardware, starting in 1986 (Holloway, 1987). The development of the display algorithm reported in this paper was begun at UNC in 1989.

CAE Electronics of Quebec developed a fiber-optic head-mounted display intended for flight simulators (CAE, 1986).

VPL Research of Redwood City, California, began selling a commercial 2-user HMD system, called "Reality Built for 2," in 1989 (Blanchard, Burgess, Harvill, Lanier, Lasko, Oberman, & Teitel, 1990).

A prototype see-through HMD targeted for manufacturing applications was built at Boeing in 1992 (Caudell & Mizell, 1992). Its display algorithm and the measurement of the parameters of this algorithm is discussed in (Janin, Mizell & Caudell, 1993).

Many other labs have set up HMD systems in the last few years. Nearly all of these systems have a stereoscopic HMD whose position and orientation is measured by a tracker, with the stereoscopic images generated by a computer of some sort, usually specialized for real-time graphics. Display software was written to make these HMD systems function, but except for the Boeing HMD, we are not aware of any detailed, general description of the display transformation for HMD systems. While geometric transformations have also been treated at length in both the computer graphics and robotics fields (Foley, van Dam, Feiner, & Hughes, 90), (Craig, 86), (Paul, 81), these treatments are not geared toward the subtleties of stereoscopic viewing in a head-mounted display. Therefore, we hope this paper will be useful for those who want to implement the display code for a VR system.

3. Definitions

We will use the symbol T_{A_B} to denote a transformation from coordinate system B to coordinate system A. This notation is similar to the notation $T_{A \leftarrow B}$ used in (Foley, van Dam, Feiner, & Hughes, 90). We use the term "A_B transform" interchangeably with the symbol T_{A_B}. Points will be represented as column vectors. Thus,

$$\mathbf{p}_A = T_{A\_B} \cdot \mathbf{p}_B \tag{3.1}$$

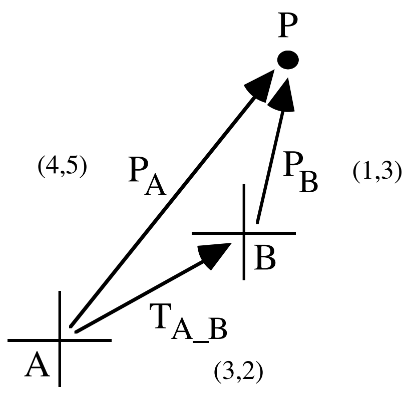
denotes the transformation of the point P_B in coordinate system B by T_{A_B} to coordinate system A. The composition of two transforms is given by:

$$T_{A\_C} = T_{A\_B} \cdot T_{B\_C} \tag{3.2}$$

and transforms a point in coordinate system C into coordinate system A. Note that the subscripts cancel, as in (Pique, 1980), which makes complicated transforms easier to derive. The inverse of a transform is denoted by reversing its subscripts:

$$(T_{A\_B})^{-1} = T_{B\_A} \tag{3.3}$$

Figure 3.1 shows a diagram of a point P and its coordinates in coordinate systems A and B, with some example values given.



$$P_A = T_{A\_B} \cdot P_B$$
$$(4,5) = (3,2) + (1,3)$$

T_{A_B} converts points in B to points in A.
T_{A_B} measures the position of B's origin in A.
The vector runs from A to B.

Figure 3.1. The meaning of transform T_{A_B}.

For simplicity, the transform in Figure 3.1 is limited to translation in 2D. The transform T_{A_B} gives the position of the origin of coordinate system B with respect to coordinate system A, and this matches up with the vector going from A to B in Figure 3.1. However, note that transform T_{A_B} converts the point P from B coordinates ($\mathbf{p}_B$) to A coordinates ($\mathbf{p}_A$) – not from A to B as you might expect from the subscript order.

In general, the transform T_{A_B} converts points from coordinate system B to A, and measures the position, orientation, and scale of coordinate system B with respect to coordinate system A.

4. The VQS Representation

Although the 4x4 homogeneous matrix is the most common representation for transformations used in computer graphics, there are other ways to implement common transformation operations. We introduce here an alternative representation for transforms between 3D coordinate systems which was first implemented for and tailored specifically to the needs of virtual-reality systems.

The *VQS* data structure represents the transform between two 3D coordinate systems as a triple [**v**, **q**, s], consisting of a 3D vector **v**, a unit quaternion **q**, and a scalar *s*. The vector specifies a 3D translation, the quaternion specifies a 3D rotation, and the scalar specifies an amount of uniform scaling (in which each of the three dimensions are scaled by the same factor).

4.1 Advantages of the VQS Representation

The VQS representation handles only rotations, translations, and uniform scaling, which is a subset of the transformations handled by the 4 x 4 homogeneous matrix. It cannot represent shear, non-uniform scaling, or perspective transformations. This is both a limitation and an advantage.

We have found that for the core work in our VR system, translations, rotations and uniform scaling are the only transformations we need. Special cases, such as the perspective transformation, can be handled using 4x4 matrices. For operations such as flying, grabbing, scaling and changing coordinate systems, we have found the VQS representation to be superior for the following reasons:

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.