Transformations and Bases in OpenGL and the Viewing Pipeline

Hassan Farhat
Computer Science
University of Nebraska at Omaha

Abstract- *Geometric transformations are fundamental aspects of all computer graphics rendering and are found throughout the computer graphics pipeline. In earlier work we presented mathematical treatment of this topic. The different concepts were combined into a general mathematical procedure based on the concept of vector spaces in linear algebra.*

While we established the mathematical aspects of transformations, we found the application of this in the OpenGL environment was not simple, especially when the modelview matrix was constructed using the viewing matrix obtained by the "gluLookAt" function. When the gluLookAt function is used, incremental changes in viewing were possible by first re-computing the viewing matrix in world coordinates and not by composing with incremental changes in the viewing coordinates.

In this paper we look at alternative methods of application of the mathematical work in OpenGL. The two alternative procedures of rendering (transformations in world and transformation in viewing) are explored in the OpenGL environment.

1 Introduction

An essential aspect of computer graphics is the concept of transformations [6, 7]. Transformations are found through out the graphics pipeline (Figure 1). The figure shows a simplified view of the graphics pipeline in OpenGL [1, 2, 4].

Initially, objects are defined in their own *modeling coordinates* (example a circle with center at the origin and a unit radius). Instances of objects are then created in world coordinates by proper translation, rotation and scaling. Each instance with corresponding transformation corresponds to a copy of the object, properly placed in world coordinates.

In OpenGL, one invokes the MODELVIEW matrix to make copies of objects and place them in world or object coordinates [4, 5].

As the name implies, the model-view matrix can also be used to build viewing coordinates. The viewing coordinates are described by a set of orthonormal vectors and viewing origin. The coordinates of the vectors and the origin are given in world coordinates. In OpenGL, one can build the viewing coordinates using a call to the OpenGL graphics utility library (gluLookAt()).

In [3], we established the mathematical concepts that allowed the computation of coordinates of a point measured in several different coordinates system. When the concepts were applied to OpenGL using gluLookAt for example, we found incremental changes in viewing were not simple. In this paper, we look at how the transformation equations can be used effectively in OpenGL with or without the need to use the gluLookAt function.

The paper is organized as follows. In section 2, we review the computations of transformations needed to describe the coordinates of object in several coordinates system.

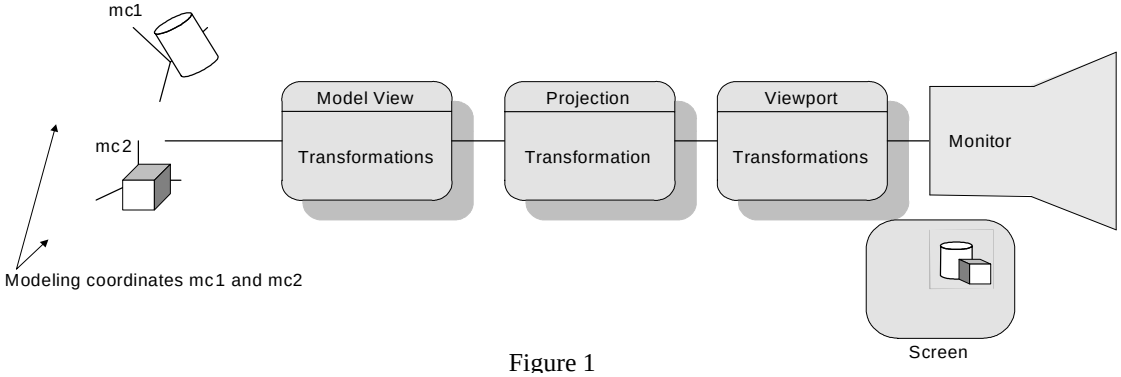

Figure 1

In section 3, we look at the application of these concepts in OpenGL when the gluLookAt function is used. In section 4 we look at alternative procedures where incremental changes in viewing can be applied. The conclusions are given in section 5.

2 Graphics Pipeline and Coordinates Transformations

As suggested by Figure 1, an object can be represented in several coordinate systems including: modeling, world, viewing and device coordinates as examples. Affine transformations are important in determining the coordinates of an object in several coordinate systems. They are also important in transforming an object within the same coordinate system.

Affine transformations for a three-dimensional space for points are described in homogeneous coordinates and are given by the equation

$$\begin{pmatrix} x_{new} \\ y_{new} \\ z_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = AX$$

In above, the a_{ij} values determine the type of transformation applied on the homogeneous point $(x, y, z, 1)^{T}$ [1]. The point $(x_{new}, x_{new}, x_{new}, 1)$ is the transformed point under A. The equation can also serve as a transformation of coordinate systems as follows. The point (x, y, z, 1) can be the coordinate of a point in space with respect to one coordinate system while the same point in space will have the coordinates $(x_{new}, y_{new}, z_{new}, 1)$ in another coordinate system.

When the equation is used as a transformation of coordinate system, we use the concept of a vector space. Even though points and vectors in n-dimensional coordinate system are described using n-tupple. The two differ in the type of operations applied to them. In [5] the distinction is made by associating points with the origin of a given coordinate frame.

A vector, X, written as a linear combination of other elements of a set of vectors, S = {X1, X2, …, Xn}, satisfies the equation

$$X = c_1 X1 + c_2 X2 + ... + c_n Xn \qquad (1)$$

for proper choice of scalars $c_1, c_2, .., c_n$. The set S is linearly independent if no vector in S can be written as a linear combination of other vectors in the set. A set, S, forms a basis for a vector space, V, if: a) every vector, X, in the vector space V can be written as a linear combination of vectors in S, and b) the set S is linearly independent.

For a set, S, forming a basis of a vector space, the elements of S constitute an alternative coordinate system to the traditional natural basis set. A vector, X, given in the natural basis have coordinates $(c_1, c_2, …, c_n)$ in the coordinate system formed from the elements of S.

The coordinate transformation matrix: The coordinate transformation matrix for a general n-dimensional vector space, $\mathcal{V}$, and a subset basis of $\mathcal{V}$, S = {X1, X2, …., Xn} can be generated from the equation

$$X = (x1, x2, …, xn) = c1\ X1 + c2\ X2 + … + cn\ Xn$$

$$\begin{pmatrix} x1 \\ x2 \\ \vdots \\ xn \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} \end{pmatrix} \cdot \begin{pmatrix} c1 \\ c2 \\ \vdots \\ cn \end{pmatrix} \qquad (2)$$

Equation (2) serves as a method of moving from one coordinate system to the other. When the Xi vectors are of unit length and are pair-wise perpendicular; i.e., when the bases are orthonormal, the ci terms can be computed as

$$ci = X \,.\, Xi$$

where the "." is the dot product. The above is then used to compute the needed transformation matrix used to write the coordinates of a vector in two different frames, label as j and j+1, as follows [3].

Given an orthonormal basis of a vector space {X1, X2, …, Xn} and an arbitrary vector X = {x1, x2, …, xn} with S and X given in coordinate frame j. The coordinates of X in frame (j + 1) with basis S (natural bases S) is

$$\begin{pmatrix} c1 \\ c2 \\ \vdots \\ cn \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & \cdots & x_{n2} \\ \vdots & \vdots & \cdots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} \end{pmatrix}^{T} \cdot \begin{pmatrix} x1 \\ x2 \\ \vdots \\ xn \end{pmatrix} = M \,.\, X \qquad (3)$$

Note that due to the orthonormal property, the inverse of the matrix is its transpose.

Using this property, the above equation can be used to transform from one system to another in either direction (world to view or vice versa) without the need to compute the inverse of M. If the position vector C is given in viewing instead of X we obtain X as

[1] The T is the transpose of the column matrix representation of the point. We omit the T for the remaining points.

$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix}^{-1} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} =$$

$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix}^{-1} \cdot \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad (4)$$

Or

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix}^{T} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} & \dots & x_{n1} \\ x_{12} & x_{22} & \dots & x_{n2} \\ x_{13} & x_{23} & \dots & x_{n3} \\ \vdots & \vdots & \dots & \vdots \\ x_{1n} & x_{2n} & \dots & x_{nn} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} \quad (5)$$

For the case where the coordinate systems origins do not match we obtain the equation [3]

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 1 \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} & 0 \\ x_{21} & x_{22} & \cdots & x_{2n} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 & -P_{01} \\ 0 & 1 & \cdots & 0 & -P_{02} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -P_{0n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} & -\sum_{i=1}^{n} x_{1i}.P_{0i} \\ x_{21} & x_{22} & \cdots & x_{2n} & \sum_{i=1}^{n} x_{21i}.P_{0i} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} & \sum_{i=1}^{n} x_{ni}.P_{0i} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ 1 \end{pmatrix} \quad (7)$$

where P0 = (P_{01}, P_{02}, …, P_{0n}) is the coordinate of the origin of the new frame.

Similar to previous, one can move from one coordinate to the other by simple manipulation of the above matrix equation. The coordinates of the point in world given its coordinates in viewing is

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & P_{01} \\ 0 & 1 & \cdots & 0 & P_{02} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & P_{0n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} & 0 \\ x_{12} & x_{22} & \cdots & x_{n2} & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 1 \end{pmatrix} \quad (8)$$

$$= \begin{pmatrix} x_{11} & x_{21} & \cdots & x_{n1} & P_{01} \\ x_{12} & x_{22} & \cdots & x_{n2} & P_{02} \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ x_{1n} & x_{2n} & \cdots & x_{nn} & P_{0n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \\ 1 \end{pmatrix}$$

3 The OpenGL "gluLookAt" Function

In this section and the next section we look at detailed application of the above equations in animation in terms of *incremental* changes of the coordinate systems as a viewer moves around a given object in circular or spherical motion. Hence we work with many frames instead of two. Our objective would be, as the view is changed, to create all the needed frames, compute the coordinates of the object in the new frame, and display the object based on the new coordinates.

We simulate animation by fixing the object in space and viewing it through different coordinate systems. For finer control over the view, the viewing frames are built relative to existing frames. First we generate the needed mathematical framework.

Assume we start with a coordinate system aligned with world coordinates, CS_0. Assume as well we generate a new coordinate system, call CS_k, through generation of several coordinate systems in between call CS_1, CS_2, ….., $CS_{(K-1)}$, in that order, as shown in Figure 2.

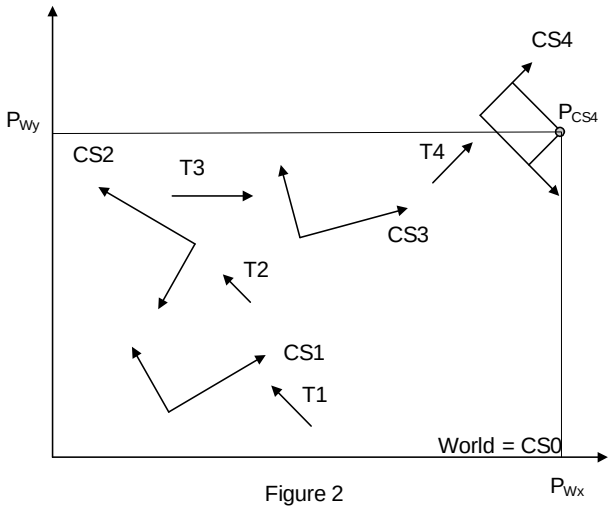

Figure 2

In the figure we assume coordinate system 4 (CS4) is formed by the application of a sequence of intermediate coordinate frames (CS1, CS2, and CS3) where each coordinate system, CS_i, is defined *relative* to the previous coordinate system, $CS_{(i-1)}$. Hence, the origin and basis for CS_1 are defined in terms of the world coordinates. Similarly, the origin and basis for CS3 are defined in the CS2 coordinates frame. Our objective is to compute PW = (P_{wx}, P_{wy}, 1) from knowledge of transformation matrices, Ti's, that move coordinate frame i – 1 to coordinate frame i and from knowledge of the coordinates of a point, P_{CS4}, in frame 4. We would like to compute T in $P_W = T.\ P_{CS4}$. Using the transformation $P_{CS(i-1)} = T_i.\ P_{CSi}$ we have $P_{CS3} = T_4\ .\ P_{CS4}$, $P_{CS2} = T_3\ .\ P_{CS3}$, $P_{CS1} = T_2\ .\ P_{CS2}$, $P_W = P_{CS0} = T_1\ .\ P_{CS1}$. Hence

$$P_W = (T_1\ .\ T_2\ .\ T_3\ .\ T_4)\ .\ P_{CS4} \text{ or } T = T_1\ .\ T_2\ .\ T_3\ .\ T_4 \qquad (9)$$

To form the final coordinate transformation matrix then we post-multiply the matrices corresponding to frame generation in the same order the coordinate frames are generated. When we consider rendering, we are normally given the point in world coordinate; i.e, we are given point Pw. Based on Pw and a viewing frame, the coordinates of Pw, in the viewing frame, Pv, are computed. This point is then rendered based on other transformations in the graphics pipeline. The point, Pv, can be obtained from equation 9 as

$$P_{CS4} = (T_1\ .\ T_2\ .\ T_3\ .\ T_4)^{-1}.\ P_w \text{ or } T = (T_1\ .\ T_2\ .\ T_3\ .\ T_4)^{-1} \qquad (10)$$

That is to form Pv (P_{CS4} above), we pre-multiply the transformation for the last frame $(T_4)^{-1}$ defined relative to frame 3 by the cumulative previous transformations for earlier frames.

In OpenGL, a frame can be generated by invoking the graphics utility library function "gluLookAt(P_{0x}, P_{0y}, P_{0z}, P_{refx}, P_{refy}, P_{refz}, V_{upx}, V_{upy}, P_{upz}). Here the point (P_{0x}, P_{0y}, P_{0z}) corresponds to the origin of the viewing frame in world. The orthonormal viewing vectors are computed from the gluLookAt function and are based on the equations

$$V_3 = \frac{V}{|V|},\quad V_1 = \frac{V_3 \times (V_{upx}, V_{upy}, V_{upz})}{|V_3 \times i|},\quad V_2 = V_3 \times V_1 \qquad (11)$$

where $|\ |$ represents the length of a vector and V = (P_{0x}, P_{0y}, P_{0z}) – (P_{refx}, P_{refy}, P_{refz}). In the equation, V_1, V_2, and V_3 correspond to the coordinates of the viewing vectors in world and the set {V1, V2, V3} forms an alternative basis for the set R^3.

Now suppose we have started animation by generating frame, i, using the gluLookAt function. To generate additional frames incrementally we need to define motion relative to frame, i, and accordingly generate frame i + 1. From equation (10), to obtain the coordinates of a point in frame i +1 we *pre-multiply* the new transformation matrix by the cumulative transformation generated from previous frames. Unfortunately, this is not easily done in OpenGL since the transformation generated from cumulative LookAt functions are post-multiplied by the current transformation matrix (CT). We next look at alternative solutions to the above.

4 Incremental Viewing Frame Generation

We consider three solutions: 1) build a library of transformation matrices (here we accumulate (compose) matrices by pre-multiplying the transformation of a new frame by the cumulative transformations), 2) by making use of the function graphics library glLoadMatrix(), and 3) by incorporating the gluLookAt and the glLoadMatrix().

1) Build a Library of transformation matrices: In this method, we construct the needed matrix operations (multiply, translate, etc.) and accumulate results by pre-multiplying a new matrix generation by the cumulative matrix. Once the cumulative matrix is obtained, we have to modify the object description by: a) computing the transformed vertices as (seen in viewing), and b) rendering the points.

2) Making use of the glLoadMatrix: In this solution we keep a copy of the cumulative matrix. On generating a new frame, we: a) compute the viewing transformation for the new frame relative to the previous frame, b) form a new composite matrix by pre-multiplying the generated matrix with the composite matrix, and c) set the current Model-View matrix of OpenGL to the newly generated matrix using the glLoadMatrix. Here, unlike the previous method, we do not need to compute the new viewing coordinates of an object before rendering them. Instead, when we render the object, OpenGL does the transformation by first pre-multiplying each vertex by the current transformation matrix.

3) Incorporating the gluLookAt and the glLoadMatrix(): This option is similar to option 2 but we start by making use of the gluLookAt function as follows: a) in a separate routine we accumulate the transformation matrices similar to method 2 above but with the exception of the initial frame, b) we set the current transformation matrix to the computed accumulated matrix, and c) we use the gluLookAt for the initial frame. OpenGL then post-multiply CT by the matrix generated from gluLookAt. And, as result, produce the correct order of transformations (last to first).

Example: To simplify our analysis we consider an example where an object (a cube) is defined as centered in world coordinates with the initial viewing frame placed at world coordinates (P_{0x}, P_{0y}, P_{0z}) = (0, 0, 1), (P_{refx}, P_{refy}, P_{refz}) = (0, 0, 0) and (V_{upx}, V_{upy}, P_{upz}) = (0, 1, 0); i.e., the viewing coordinates are the same as the world coordinates but with

origin translated along the z axis (z = 1). We will simulate moving about the object by generating frames about a unit circle generated by turning about the x-axis in small angles dtheta[2]. Consider Figure 3.

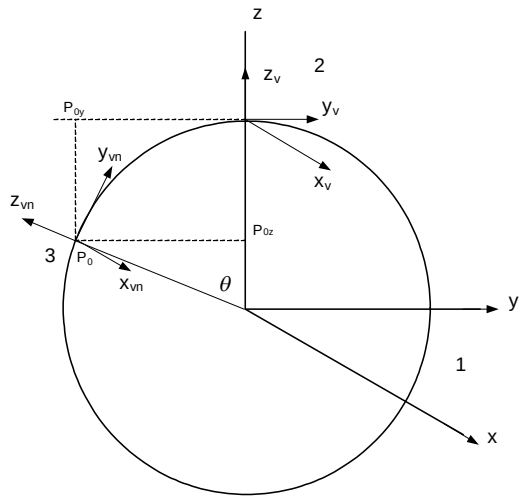

Figure 3

The figure shows three coordinates systems: of the three 2 are viewing coordinates (labeled 2 and 3); the third is the original world coordinate (labeled 1). In the figure viewing coordinates 2 was obtained from viewing coordinates 1 by counterclockwise rotation by angle theta about the x-axis. In the diagram we chose large theta for the purpose of computation the origin and vectors geometrically. Given a point in world coordinates; our task is to generate viewing coordinates by successive small rotations. Hence once a rotation of viewing coordinates is performed, we need to determine the coordinates of the new orthonormal bases and the coordinates of the new origin relative to the previous viewing coordinates. From the figure the coordinates of the new origin are computed as

$$P_{0x} = 0$$
$$P_{oy} = -\sin\theta$$
$$P_{0z} = -1 + \cos\theta$$

The look at point remains the origin which has coordinates (0, 0, –1) in viewing frame 2. The coordinates of the basis for z_{vn} are read from the figure as (0, –sinθ, cosθ). The y_{vn} basis is obtained as a right-turn from z_{vn}, y_{vn} = (0, cosθ, sinθ). With the above we could form the transformation matrix from viewing 2 to viewing 3 as

$$T = \begin{pmatrix} x_{vn} & -x_{vn}.p_{on} \\ y_{vn} & -y_{vn}.p_{on} \\ z_{vn} & -z_{vn}.p_{on} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta & -1+\cos\theta \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

The above matrix is used to compute the coordinates of a point in frame 3 given its coordinates in frame 2.

OpenGL realization: In the OpenGL model-view matrix mode, applying a transformation, T, followed by rendering an object, O, causes a transformation of the vertices of the object based on the equation

$$O_{new} = (CT) \,.\, (T) \,.\, O$$

where CT is the current transformation matrix. As a result, the order of applying transformation in model-view is to first set the CT matrix to the viewing matrix, V, second set the modeling matrix, M, and third call the rendering of a given object. This results in the desired product V . M . O. The order, however, makes it difficult to realize the alternative analogy of rendering with the viewing coordinates changing instead of the set object description. This makes it difficult to realize incremental changes to viewing frames where the coordinates of a new frame are defined in terms of the coordinates of current frame. What is desired is a pre-multiply of new viewing, V1 matrix, by current viewing, CT = V2 matrix, V1 . CT. The gluLookAt function fixes the CT matrix to viewing V2 based on equation 11. Incremental changes using additional gluLookAt functions results in post-multiply of new viewing V2.

In the rendering of example 2, we considered the previous proposed three solutions.

Procedure 1: In this procedure we do not use the gluLookAt function; instead we developed a matrix of geometric operations, functions, (rotate, translate, scale and accumulate for example). The accumulate function causes the user-defined current matrix to be pre-multiplied by a desired transformation such rotation, for example. To render an object based on the accumulated matrix, we first changed the vertices of the object by pre-multiplying each by the accumulated matrix.

Procedure 2: We followed a procedure similar to procedure 2. Here, however, we made use of the OpenGL glLoadMatrixf(). The advantage of this method is in removing the need to write code to change the vertices of the object to be rendered; OpenGL accomplishes this. Using the glutKeyboardFunc(), a new viewing frame V(i+1) defined relative to Vi was computed based on equation 12. Keyboard presses generated new viewing frames. We compute the initial viewing frame V2 (origin of viewing at

[2] It is well known in computer graphics, the effects of generating the frames and rendering are the same as rotating the abject about the x-axis in world. The discussion is intended to provide alternatives that require mathematical analysis and better understanding of OpenGL transformations and graphics pipeline.

(0, 0, 1)) and make the current transformation matrix through the glLoadMatrixf(). To render an object, the object is first transformed based on the given CT matrix. Now to display an object based on viewing V3 defined relative to V2, we: 1) computed the new viewing transformation matrix, V3, using equation 7 (this results in equation 12), 2) formed the matrix product V = V3 . V2, and 3) made V = CT using the glLoadMatrix() function. Subsequent incremental changes in viewing frames can be accomplished similarly.

Procedure3: In this procedure we updated the viewing matrix as done in procedure 2. We first computed the *new* viewing frame matrix and made it CT. We then invoked the gluLookAt() function with parameters that generated the viewing matrix V2. This produces the correct order of matrix multiplication CT = CT . V2 = V3.V2.

A realization of the three procedures on a simple cube with the keyboard used to create new viewing frames based on equation 12, produced the correct result.

5 Conclusion

In this paper we looked at transformations in computer graphics as related to vector spaces from linear algebra in the context of OpenGL. In earlier related work we developed the theoretical framework between transformations and vector spaces. We looked at applications in OpenGL and the concept of incremental change in viewing coordinates. The OpenGL viewing pipeline is built on the concept of current transformation and a post-multiplication of a new transformation by the current transformation matrix (CT). This, however, is not suitable when we look at the alternative interpretation of rendering in computer graphics, where the viewing coordinates change instead of the object world coordinates. We considered several solutions that meet the alternative interpretation.

References:

[1] E. Angel. "Interactive Computer Graphics, A Top-Down Approach Using OpenGL, 5th Edition". Addison Wesley, 2009.

[2] E. Angel. "OpenGL, A Primer, 2nd Edition". Addison Wesley, 2005.

[3] H. Farhat. "Transformations and Basis in the Computer Graphics Viewing Pipeline," International Conference on Frontiers in Education, FECS 2009, pp. 267-271.

[4] D. Hearn, M. Baker, W. Carithers. "Computer Graphics with OpenGL, 4th Edition". Prentice Hall, 2011.

[5] F. Hill, S. Kelly. "Computer Graphics using OpenGL, 3rd Edition". Prentice Hall, 2007.

[6] D. Rogers, J. Adams. "Mathematical Elements for Computer Graphics, 2nd edition". McGraw Hill, 1990.

[7] Foley, van Dam, Feiner, Hughes. "Computer Graphics Principles and Practice, 2nd edition". Addison Wesley, 1990.