Analysis of Vector Rotation Alorithms

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Shubham Jindal & Ankit Kr. Singh
(150101071 & 150101086)

under the guidance of

Dr. Pinaki Mitra

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Analysis of Vector Rotation Alorithms**" is a bonafide work of **Shubham Jindal & Ankit Kr. Singh** (**Roll No. 150101071 & 150101086**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Pinaki Mitra**

Associate Professor,

April, 2019 Department of Computer Science &

Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

Acknowledgements

I would like to take this opportunity to thank my supervisor Dr. Pinaki Mitra who has been supporting us throughout this study. Also I would like to thank the following individuals who have helped me in carrying my study:

- Prof. Benny George K for attending the presentation.
- Prof. Diganta Goswami for attending the presentation.
- Our family - for supporting us in our endeavors.

Abstract

Currently there are several techniques of rotating a vector in 3D space. In our previous work, comparison of two main techniques of rotation namely, rotation using Euler angle and rotation using quaternions is done [gsc], brief introduction of which is also included in this report. Current work done is about the error analysis of the two rotation techniques for small angle rotations. Theoretical analysis of the error introduced by the rotation of a point about an axis by small angle is included. Error analysis for both rotation techniques is also shown (from practical purpose point of view) by measuring change in volume of the closed 3D body, formed by Delaunay tetrahedralization of the point cloud, when the body is rotated by small angle. Also, finding out the threshold above which the error for the chain of rotations by small angles becomes too large.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We start by discussing concepts and formulas involved in Euler rotation in three dimension for an arbitrary angle rotation which is later compared with the rotation using quaternions. Also we review few concepts involving quaternions.

1.1 Rotation in three dimensions using Euler Angle

A standard coordinate frame in three dimension is shown in the figure below. A vector/position (tip of vector with tail at origin) in three dimensions is given in Cartesian coordinate as (x,y,z). This vector can be transformed along any one particular axis[Sla99]

In the case of rotation about an arbitrary angle, we use the following procedure:

- Rotate axis of rotation in appropriate fashion to coincide it with the z-coordinate axis.
- Now, rotate by the angle β about z-axis .
- Rotate the axis of rotation by the inverse of the transformation done in step 1.

The matrix for arbitrary rotations around these axes is obtained by multiplying the matrices for each axis: rotation matrix for rotation along z-axis by the angle γ, rotation

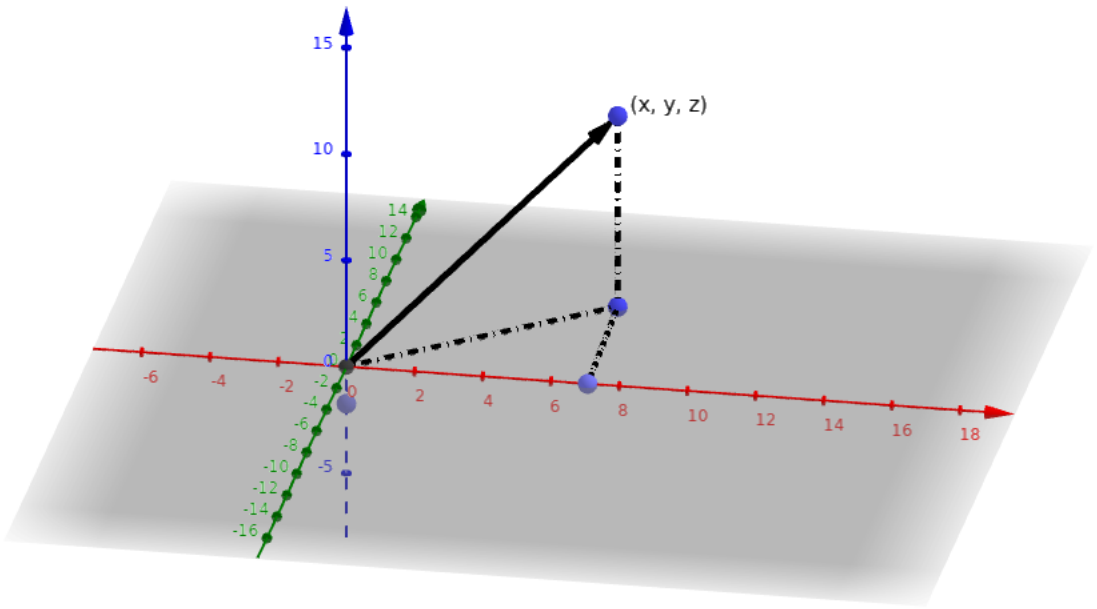

Fig. 1.1 Cartesian coordinate system

matrix for rotation along y-axis by the angle β and rotation matrix for rotation along x-axis by the angle α. The resulting matrix is computed as follows. First multiply the rotation matrix for rotation about the x-axis then rotation matrix for rotation about y-axis.

$$R_{y(\beta)}R_{x(\alpha)} = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha \\ 0 & sin\alpha & cos\alpha \end{bmatrix} = \begin{bmatrix} cos\beta & sin\beta sin\alpha & sin\beta cos\alpha \\ 0 & cos\alpha & -sin\alpha \\ -sin\beta & cos\beta sin\alpha & cos\beta cos\alpha \end{bmatrix}$$

Next rotate the resultant vector about the z-axis

$$R_{z(\alpha)y(\beta)x(\alpha)} = R_{y(\beta)}R_{x(\alpha)} = \begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & sin\beta sin\alpha & sin\beta cos\alpha \\ 0 & cos\alpha & -sin\alpha \\ -sin\beta & cos\beta sin\alpha & cos\beta cos\alpha \end{bmatrix}$$

$$R_{z(\alpha)y(\beta)x(\alpha)} = R_{y(\beta)}R_{x(\alpha)} = \begin{bmatrix} cos\gamma cos\beta & cos\gamma sin\beta sin\alpha - sin\gamma cos\alpha & cos\gamma sin\beta cos\alpha + sin\gamma sin\alpha \\ sin\gamma cos\beta & sin\gamma sin\beta sin\alpha + cos\gamma cos\alpha & sin\gamma sin\beta cos\alpha + cos\gamma sin\alpha \\ -sin\beta & cos\beta sin\alpha & cos\beta cos\alpha \end{bmatrix}$$

1.1.1 Rotation Matrix Chaining in Euclidean vector rotation

If R_1, R_2 are the two rotation matrices for arbitrary axis rotation in three-dimensional space. These two rotation operation can be composed as the first rotation and then after that the second rotation. In terms of matrix rotation operation this can be written as:

$$L_{R2}(L_{R1}(v)) = R_2 * (R_1 * v) = (R_2 * R_1) * v = L_{R2*R1}(v)$$

Thus we can represent a series operation L_{R1} followed by L_{R2} by the Matrix product $R_2 * R_1$. The composition of matrix rotation can be generalized for any arbitrary number of rotations.

1.2 Quaternions

We can generalize the concept of complex number in 4D [BML98] using **Quaternions**. Irish mathematician William Rowan Hamilton first introduced it in 1843[Ham63]. Pure and applied mathematics use quaternions extensively. They find a wide range of application [Muk02] in computer graphics, robotics, mechanics, virtual reality[Vin95], crystallographic texture analysis and computer vision. In particular they are used for three dimensional rotation as an alternative to Euler rotation matrix. Quotient of two vectors is defined as

quaternion by Hamilton. General form of quaternions is: $a_0 + a_1\hat{\imath} + a_2\hat{j} + a_3\hat{k}$ where i, j, and k are the fundamental quaternion units and $a_0, a_1, a_2, and a_3$ are real numbers which satisfies following set of rules.

1. $i^2 = j^2 = k^2 = ijk$ = -1
2. $ij = k$, ji = -k
3. $jk = i$, kj = -i
4. $ki = j$, ik = -j

Also, quaternions do not commute w.r.t to multiplication and quaternions form a non-abelian group under addition.

1.3 Quaternions Algebra

1.3.1 Quaternions Arithmetic

Arithmetic in quaternions is performed formally and the results are simplified using the identities mentioned above. Addition and Multiplication of two quaternions are shown below :

$$\mathbf{q} + \mathbf{p} = (q_0 + q_1 i + q_2 j + q_3 k) + (p_0 + p_1 i + p_2 j + p_3 k)$$
$$= (q_0 + p_0) + (q_1 + p_1)i + (q_2 + p_2)j + (q_3 + p_3)k$$
$$\mathbf{q} * \mathbf{p} = (q_0 + q_1 i + q_2 j + q_3 k) * (p_0 + p_1 i + p_2 j + p_3 k)$$
$$= (q_0 p_0 - q_1 p_1 - q_2 p_2 - q_3 p_3) + (q_0 p_1 + q_1 p_0 + q_2 p_3 - q_3 p_2)i +$$
$$(\ q_0 p_2 + q_2 p_0 - q_1 p_3 + q_3 p_1)j + (q_0 p_3 + q_3 p_0 + q_1 p_2 - q_2 p_1)k.$$

1.3.2 Quaternion Conjugate, Norm and Inverse

We can define the norm and conjugate of quaternions in the similar way as we did for complex numbers. Given $q = q_0 + q_1 i + q_2 j + q_3 k$

- Conjugate is defined as:

$$q^* = q_0 - q_1 i - q_2 j - q_3 k$$

- Norm is defined as:

$$|q| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

- Inverse is defined as:

$$q * q^* = (q_0 + q_1 i + q_2 j + q_3 k) * (q_0 - q_1 i - q_2 j - q_3 k)$$

$$= (q_0 q_0 + q_1 q_1 + q_2 q_2 + q_3 q_3) + (-q_0 q_1 + q_1 q_0 - q_2 q_3 + q_3 q_2) i + (-q_0 q_2 + q_2 q_0 + q_1 q_3 -$$

$$q_3 q_1) j + (-q_0 q_3 + q_3 q_0 - q_1 q_2 + q_2 q_1) k.$$

$$= q_0^2 + q_1^2 + q_2^2 + q_3^2.$$

$$q^{-1} = \frac{q_*}{|q|^2}$$

The logarithm and Euler form representation of quaternions are described in[BML98].

Chapter 2

Problem definition and Proposed solution

2.1 Problem definition

Comparing number of operations required for rotations using quaternions and rotation using Euler angles and explain why quaternions perform better when we have chain of rotation operations and why in the field of computer graphics quaternions have become popular. The angles by which three-dimensional coordinate frame are rotated are called Euler angles. We can use matrix of trigonometric functions of the angles to represent rotation of Euler angles. Quaternions are an algebraic structure that extends the familiar concept of complex numbers . While quaternions are much less intuitive than angles, rotations defined by quaternions can be computed more efficiently and with more stability, and therefore are widely used. Comparison study will be done for single rotation and for chain of rotations. Comparison will be based on space and time complexities. Analysis based on time complexity will further be broken down in two stages, for small angle rotations and general angle rotations.

2.2 Proposed Solution

2.2.1 Quaternion Rotation

Derivation

If $\vec{v}$ is vector in 3D Cartesian space then quaternion corresponding to it $(\vec{0},\vec{v})$. Now to rotate this vector about the axis $\vec{n} = \mathrm{n}_x\ \hat{\mathrm{i}} + \mathrm{n}_y\ \hat{\mathrm{j}} + \mathrm{n}_z\ \hat{\mathrm{k}}$ by an angle α, following formula is used

$$\vec{v'} = q\vec{v}q^*$$

where,

$$\mathrm{q} = e^{(\alpha/2)\vec{n}} = \cos(\alpha/2) + \mathrm{n}_x \sin(\alpha/2)\ \hat{\mathrm{i}} + \mathrm{n}_y \sin(\alpha/2)\ \hat{\mathrm{j}} + \mathrm{n}_z \sin(\alpha/2)\ \hat{\mathrm{k}}$$
$$\mathrm{q}^* = e^{-(\alpha/2)\vec{n}} = \cos(\alpha/2) - \mathrm{n}_x \sin(\alpha/2)\ \hat{\mathrm{i}} - \mathrm{n}_y \sin(\alpha/2)\ \hat{\mathrm{j}} - \mathrm{n}_z \sin(\alpha/2)\ \hat{\mathrm{k}}$$

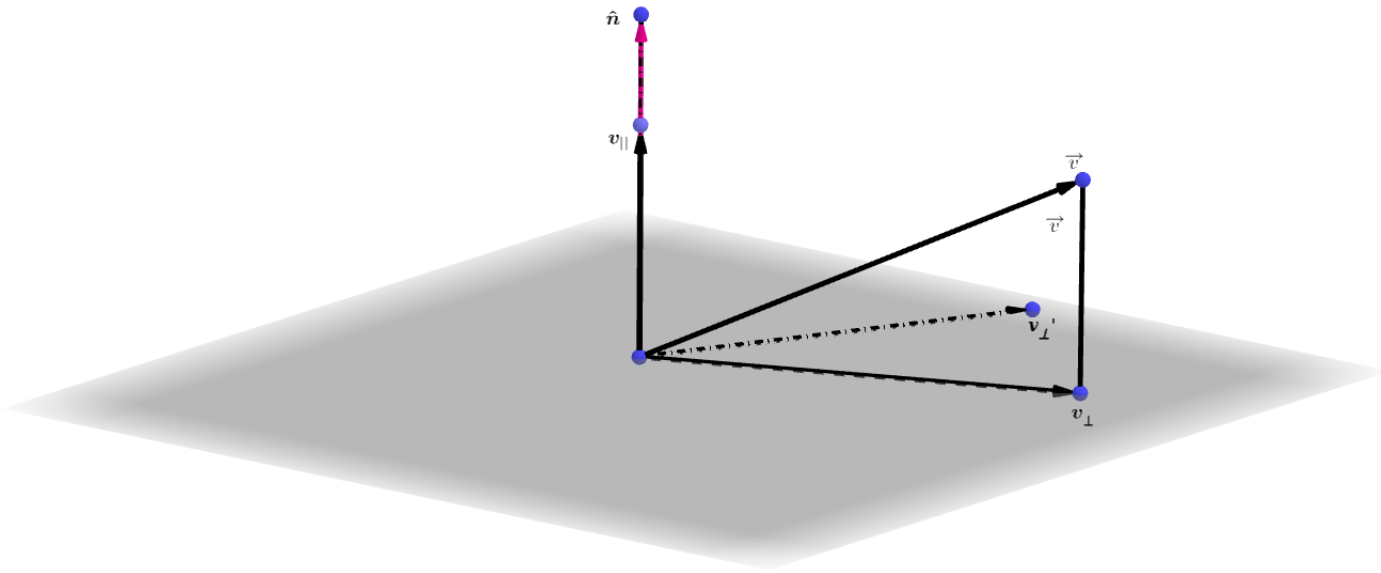

Fig. 2.1 Quaternion Rotation

Proof:

$$\vec{v} = \vec{v}_{||} + \vec{v}_{\perp}$$

$$\vec{v'} = \vec{v'}_{||} + \vec{v'}_{\perp}$$

$$\vec{v'}_{||} = \vec{v}_{||}$$

$$\vec{v'} = \vec{v}_{||} + \vec{v'}_{\perp}$$

$$\vec{v'}_{\perp} = \cos\alpha\ \vec{v}_{\perp} + \sin\alpha\ (\vec{n}\ \text{x}\ \vec{v}_{\perp})$$

$$\vec{v}_{\perp} = \vec{v}\ \text{-}\ \vec{v}_{||}$$

$$\vec{n}\ \text{x}\ \vec{v} = \vec{n}\ \text{x}\ (\vec{v}_{||} + \vec{v}_{\perp})$$

$$\vec{n}\ \text{x}\ \vec{v}_{||} = \vec{0}$$

$$\vec{n}\ \text{x}\ \vec{v} = \vec{n}\ \text{x}\ \vec{v}_{\perp}$$

$$\vec{v'} = (1\text{-}\cos\alpha)(\vec{v} \cdot \hat{\text{n}})\hat{\text{n}} + \vec{v}\cos\alpha + (\vec{n}\ \text{x}\ \vec{v})\sin\alpha$$

$$\textbf{where, } \vec{v}_{\ ||} = (\vec{v} \cdot \hat{\text{n}})\hat{\text{n}}$$

Now we know that in 3D Cartesian space to rotate a vector by angle α the formula is given by,

$$\vec{v'} = e^{\alpha\vec{n}}\ \vec{v}$$

$$\text{and } \vec{v'} = \vec{v}_{\ ||} + \vec{v'}_{\ \perp}$$

$$\vec{v'} = \vec{v}_{\ ||} + e^{\alpha\vec{n}}\ \vec{v}_{\ \perp}$$

$$\vec{v'} = \vec{v}_{\ ||} + (\cos(\alpha/2) \text{ - } \text{n}_x \sin(\alpha/2)\ \hat{\text{i}} \text{ - } \text{n}_y \sin(\alpha/2)\ \hat{\text{j}} \text{ - } \text{n}_z \sin(\alpha/2)\ \hat{\text{k}})\ \vec{v}_{\ \perp}$$

Note that,

$$e^{\alpha\vec{n}}\ \vec{v}_{\perp} = \vec{v}_{\perp}\ e^{\alpha\vec{n}}$$

$$(\cos\alpha,\ \vec{n}\sin\alpha)\ (\vec{0},\ \vec{v}_{\perp}) = (\vec{0},\ \vec{v}_{\perp})\ (\cos\alpha,\ \text{-}\vec{n}\sin\alpha)$$

$$(\vec{0},\ \cos\alpha\ \vec{v}_{\perp} + \sin\alpha(\vec{n}\ \text{x}\ \vec{v}_{\perp})) = (\vec{0},\ \cos\alpha\ \vec{v}_{\perp} \text{ - } \sin\alpha(\vec{v}_{\ \perp}\ \text{x}\ \vec{n}))$$

$$(\vec{0},\ \cos\alpha\ \vec{v}_{\perp} \text{ - } \sin\alpha(\vec{v}_{\ \perp}\ \text{x}\ \vec{n}) = (\vec{0},\ \cos\alpha\ \vec{v}_{\perp} \text{ - } \sin\alpha(\vec{v}_{\ \perp}\ \text{x}\ \vec{n}))$$

Also,

$$e^{\alpha\vec{n}}\ \vec{v}_{\ ||} = \vec{v}_{\ ||}\ e^{\alpha\vec{n}}$$

$$\text{where, } \sin\alpha\ (\hat{\text{n}}\ \text{x}\ \vec{v}_{\ ||}) = \vec{0}$$

Finally,

$$\vec{v'} = \vec{v}_{||} + e^{\alpha\vec{n}}\ \vec{v}_{\perp}$$
$$\vec{v'} = e^{(\alpha/2)\vec{n}}\ e^{-(\alpha/2)\vec{n}}\ \vec{v}_{||} + e^{(\alpha/2)\vec{n}}\ e^{(\alpha/2)\vec{n}}\ \vec{v}_{\perp}$$
$$\vec{v'} = e^{(\alpha/2)\vec{n}}\ \vec{v}_{||}\ e^{-(\alpha/2)\vec{n}} + e^{(\alpha/2)\vec{n}}\ \vec{v}_{\perp}\ e^{-(\alpha/2)\vec{n}}$$
$$\vec{v'} = e^{(\alpha/2)\vec{n}}\ (\vec{v}_{||} + \vec{v}_{\perp})\ e^{-(\alpha/2)\vec{n}}$$
$$\vec{v'} = e^{(\alpha/2)\vec{n}}\ \vec{v}\ e^{-(\alpha/2)\vec{n}}$$

where,

$$e^{(\alpha/2)\vec{n}} = \cos(\alpha/2) + \mathrm{n}_x \sin(\alpha/2)\ \hat{\mathrm{i}} + \mathrm{n}_y \sin(\alpha/2)\ \hat{\mathrm{j}} + \mathrm{n}_z \sin(\alpha/2)\ \hat{\mathrm{k}}$$
$$e^{-(\alpha/2)\vec{n}} = \cos(\alpha/2) - \mathrm{n}_x \sin(\alpha/2)\ \hat{\mathrm{i}} - \mathrm{n}_y \sin(\alpha/2)\ \hat{\mathrm{j}} - \mathrm{n}_z \sin(\alpha/2)\ \hat{\mathrm{k}}$$
$$\mathbf{q} = e^{(\alpha/2)\vec{n}}$$
$$\mathbf{q}^* = e^{-(\alpha/2)\vec{n}}$$

Hence we proved that,

$$\vec{v'} = \mathbf{q}\vec{v}\mathbf{q}^*$$

2.2.2 Chain of Quaternion vector rotation Operations

Let us suppose we have two quaternions q_1, q_2 representing arbitrary rotations in three-dimensional space.These two rotation operation can be composed as the first rotation followed by the second. In terms to quaternion rotation operation this can be written as:

$$L_{q_2}(L_{q_1}(v)) = q_2(q_1\vec{v}q_1^*)q_2^* = (q_2q_1)\vec{v}(q_1^*q_2^*) = L_{q_2q_1}(v)$$

Hence, a series of operators L_{q_1} followed by L_{q_2} is equal to quaternion rotation operator $L_{q_2q_1}$. The composition of quaternion rotation can be generalized for any arbitrary number of rotations.

Chapter 3

Experimental Results

Based on the above comparison of the two formalism for rotation we have following results:

3.1 Storage Required

Rotation Method	Storage Required per Unit Structure
Euler Rotation Matrix	9 or 16
Unit Quaternion	4

Table 3.1 Comparison of Space Required

Explanation

- Rotational matrix needs 9 (3*3) matrix or 16(4*4) matrix when homogeneous transformation is considered.
- Unit Quaternions only require 4 elements (q_0, q_1, q_2, q_3) or 3 elements since we can get the fourth element using other 3 since its a unit quaternion.
- Hence for n rotation operations Rotation matrix will require 5*n more space.

3.2 Operations required

Comparison of flops required for single vector rotation for general angle rotation and small angle rotation is summarized below:

3.2.1 General Angle Rotation

Rotation Method	#multiplies/divisions	#add/sub	#sin/cos	total operations
Rotation Matrix	15	13	2	30
Rotated Vector ($\mathbf{M}$*$\mathbf{v}$)	9	6	0	15
Unit Quaternion	32	20	2	54

Table 3.2 Comparison of FLOPs Required for General Angle Rotation

Explanation

- Total operations required for Euclidean vector rotation is 45.
- Total operations required for Vector rotation using Quaternions is 54.

3.2.2 Small Angle Rotation

Rotation Method	#multiplies/divisions	#add/sub	#sin/cos	total operations
Rotation Matrix	6	0	0	6
Rotated Vector ($\mathbf{M}$*$\mathbf{v}$)	9	6	0	15
Unit Quaternion	32	20	0	52

Table 3.3 Comparison of FLOPs Required for Small Angle rotation

When we perform a sequence of such rotation operations the resulting error (due to the approximation of sinθ to θ and cosθ to 1 in each of the individual rotations) may cause the matrix $\mathbf{M}$ (direction cosine matrix $\mathbf{M}$) to become non-orthogonal, and hence pure rotation could not be represented by the transformation. To normalize the matrix $\mathbf{M}$, replace M by the matrix $\mathbf{M}(Det(\mathbf{M}^T\mathbf{M})^{-\frac{1}{2}})$. The corresponding operation in the quaternion domain

is computationally less expensive than in Euler angle domain. In quaternion domain, we only need to divide by $||\mathbf{q}||$.

For Normalization of

Rotation matrix : compute $\mathbf{M}(Det(\mathbf{M}^T\mathbf{M}))^{-\frac{1}{2}}$

Quaternions : division by $||\mathbf{q}||$, where $||\mathbf{q}||^{\,2} = \mathbf{q}\mathbf{q}^* = {q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2$

Explanation

- Total operations required for Euclidean vector rotation is 21.
- Total operations required for Vector rotation using Quaternions is 52.

3.3 Rotation Chaining Operations

Rotation Method	#multiplies/division	#add/subtract	total operations
Euler Rotation Matrix	27	18	45
Unit Quaternion	16	12	28

Table 3.4 Comparison of FLOPs Required for Rotation Chaining

Explanation

- Multiplying two 3*3 Matrices requires 27 Multiplications and 18 additions.
- Unit Quaternions only require 4 elements (q_0, q_1, q_2, q_3) or 3 elements since we can get the fourth element using other 3 since its a unit quaternion.Multiplying two quaternions requires 16 Multiplications and 18 additions.
- If we chain n rotation operations then Euler angles will require 11*n extra multiplication. Hence quaternion rotations more optimized in terms of flops required.

3.4 Problem of Gimbal Lock

When Euler Angles are used the graphic system faces the problem of gimbal lock. On the other hand, quaternions helps to avoid this problem.

Chapter 4

Error Analysis

Volume enclosed by an arbitrary n-points cloud in an invariant property that remains unchanged due to 3D rotation. No error(change in enclosed volume) gets introduced for general angle rotation of n-points cloud about any arbitrary axis for both Euler angle rotation and quaternion rotation techniques. Also this can be ensured if the matrix of rotation is ortho-normal which is the case with Euler angle rotation.

We define error in terms of change in volume or deviation from orthonormality. We give two kinds of analysis: theoretical and for production use. Once we have a cumulative error we set the tolerance limit and recompute the final point.

For small angle rotation, we take an arbitrary number of points and we analyze the change in enclosed volume for chain of rotations about a fixed axis(say X-axis). Considering fixed axis won't affect the analysis since the point cloud is selected randomly.

4.1 Theoretical Analysis

This section contains the details of the theoretical error, which can be used for the analysis part but can't be used in a graphics software, since it recomputes the final point on the

basis of series of rotation a point has under gone through.

4.1.1 Euler Small Angle Rotation

For small angle rotation, the rotation matrix for chain of rotations about X-axis will be,

$$M = R^1_{x(\theta_1)} R^2_{x(\theta_2)} ... R^n_{x(\theta_n)} = \begin{bmatrix} 1 & -\theta_1 & 0 \\ \theta_1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\theta_2 & 0 \\ \theta_2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} ... \begin{bmatrix} 1 & -\theta_n & 0 \\ \theta_n & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We define error due to the loss of normality of the rotation matrix. Using approximation for small angle rotation, the orthogonality of matrix is preserved but the normality is lost. For rotation about X-axis the self dot product for the 2 columns is $1 + \theta^2$. We define this quantity as normality measure. For chain of rotations this is given by

$$\epsilon = \Pi_{i=1}^{n}(1 + \theta_i^2) - 1$$

and to recompute the actual transformed point we use the history of transformations(exact Euler rotation matrix) the original point has gone through. We can also normalize the rotation matrix, by replacing $\mathbf{M}$ by $\mathbf{M}(Det(\mathbf{M}^T\mathbf{M}))^{-\frac{1}{2}}$, where

$$Det(\mathbf{M}^T\mathbf{M}) = \Pi_{i=1}^{n}(1 + \theta_i^2)$$

4.1.2 Quaternion Small Angle Rotation

For chain of rotations (m rotations) about an arbitrary axis $\vec{n}$, we have,

$$L_{q_m}...(L_{q_2}(L_{q_1}(v)) = (q_m...(q_2(q_1)))\ \vec{v}\ (q_m^*...(q_2^*(q_1^*)))$$
$$L_{q_m}...(L_{q_2}(L_{q_1}(v)) = (q_m...q_2q_1)\ \vec{v}\ (q_m^*...q_2^*q_1^*)$$
$$L_{q_m}...(L_{q_2}(L_{q_1}(v)) = L_{q_m...q_2q_1}(v)$$

So the normalization factor $||\mathbf{q}||$ is given by,

$$\sqrt{1 + \frac{m^2\theta^2}{4}(n_x^2 + n_y^2 + n_z^2)}$$

4.2 Analysis using Delaunay Tetrahedralization

We take the arbitrary n-points cloud and rotate them about X-axis, and analyze the change in volume enclosed. In order to find the enclosed volume we first tetrahedralize the point cloud, which involves dividing the enclosed volume with non-overlapping tetrahedral units. The volume enclosed is the sum of the individual units. Code is available at [eas]

In order to tetrahedralize or 3D triangulate the points we use Delaunay Triangulation algorithm. Delaunay triangulation for a n-point cloud C in a plane is a method of triangulation such that no point in the point cloud is in the circumcircle of any unit(triangular in case of plane) in the triangulation. Mathematically the problem of delaunay triangulation can be written as :

$$\text{set of triangles s.t. maximize } \textit{min(angles of triangles in triangulation)}$$

It is named after Boris Delaunay for his contribution since 1934. On the similar line we can define Delaunay Tetrahedralization.

Delaunay Tetrahedralization is applied on the set of points and the volume (V_{init}) is calculated for the constructed 3D object. Then this set of points is rotated about an arbitrary axis using either quaternion or euler angle rotation. Then the volume (V_{final}) is calculated for the constructed 3D object from the new set of points obtained after rotation by small angle θ. So we need to minimize the change in volume while rotating an object formed using point cloud.

$$\text{Loss} = \min(|V_{final} - V_{init}|)$$

Pseudo Code for typical algorithm implementing Delaunay Tetrahedralization.

Algorithm 1 Bowyer Watson Implementation of N-Dimensional Delaunay triangulation

```
Require: pointList
  //pointList is a set of coordinates defining the points to be triangulate
  Triangulation ← empty triangle mesh data structure
  //Super-triangle must be large enough to completely contain all the points in pointList
  Add Super-triangle to Triangulation
  //add all the points one at a time to the triangulation
  for point ∈ pointlist do
    BadTriangle ← emptySet
    //first find all the triangles that are no longer valid due to the insertion
    for Triangle ∈ Triangulation do
      if point ∈ circumcircle of Triangle then
        Add Triangle to BadTriangles
      end if
    end for
    Polygon ← empty set
    for Triangle ∈ BadTriangle do
      //find the boundary of the polygonal hole
      for edge ∈ Triangle do
        if edge is not shared by any other triangles in BadTriangles then
          add edge to Polygon
        end if
      end for
    end for
    //remove them from the data structure
    for Triangle ∈ BadTriangle do
      remove Triangle from Triangulation
    end for
    //re-triangulate the polygonal hole
    for edge ∈ Polygon do
      newTri ← form a triangle from edge to point
      add newTri to Triangulation
    end for
  end for
  //done inserting points, now clean up
  for Triangle ∈ Triangulation do
    if Triangle contains a vertex from original Super-triangle then
      remove Triangle from Triangulation
    end if
  end for
  return Triangulation
```

We have plotted graphs for three different cases by taking 1000 points in the points cloud and computed the percent error(volume change) using Delaunay triangulation for volume approximation. This is done for Euler and Quaternion general and small angle rotation. Other observations is available at [gra]

Case 1: The point cloud was rotated by 1^o for 1000 steps. Volume and percentage error was recorded as depicted in graph below. Due to the precision error, error is introduced and this behaviour is shown in the graphs by the negligible error(of order 10e-10) in graphs for Euler and quaternion general angle rotations.

Also the error percent in case of small angle rotation is directly proportional to the rotation angle in both Euler and Quaternion rotation.

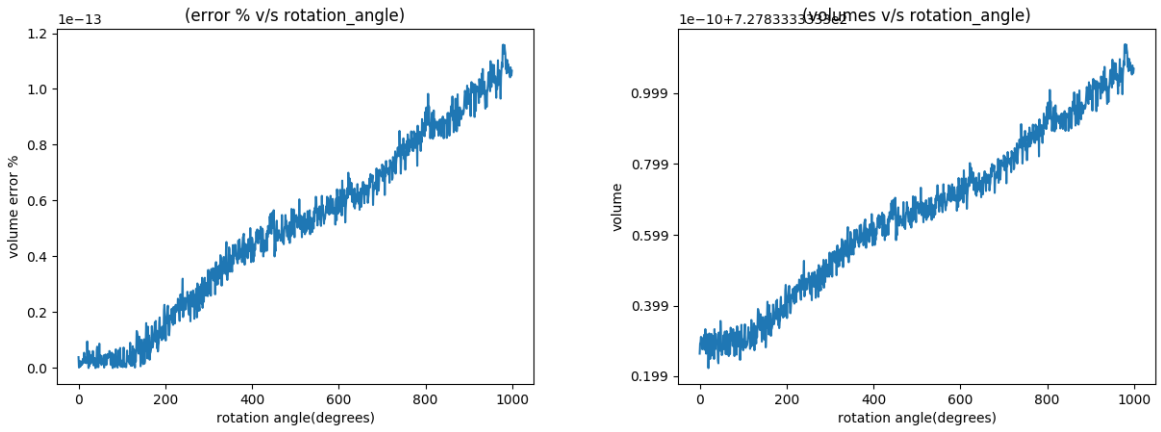

Fig. 4.1 CASE 1 : Error Analysis for Euler General Angle Rotation

Case 2: Similar experiment was performed but this time the approx matrix was normalized during each transformation. Hence the error in each both quaternion and Euler case decreases significantly. But as mentioned previously the cost incurred for Matrix normalization will be higher than quaternion normalization. Also decrease in volume can be observed in both cases after normalization.

Case 3: Again similar set experiment was performed but this time a error threshold was set. Hence if the volume error reaches a threshold value(5% in our case) the actual

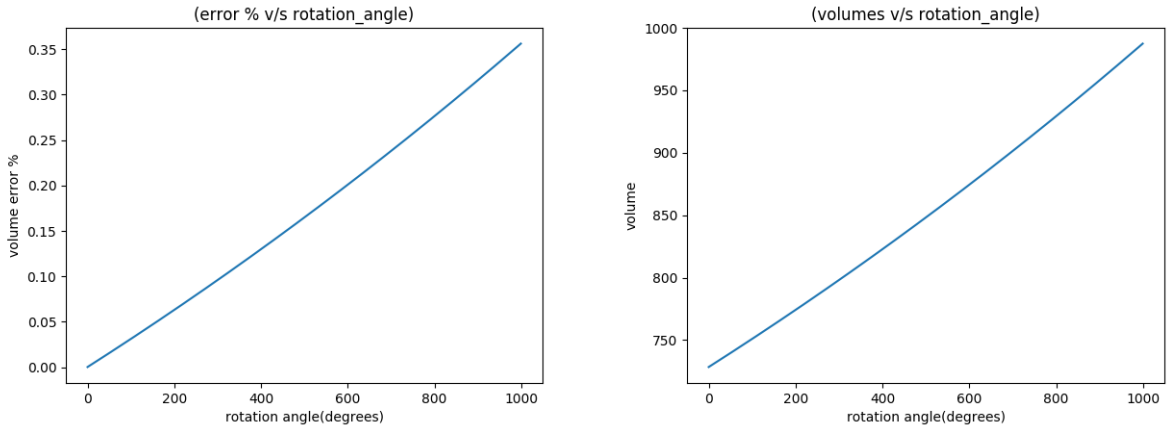

Fig. 4.2 CASE 1 : Error Analysis for Euler Small Angle Rotation

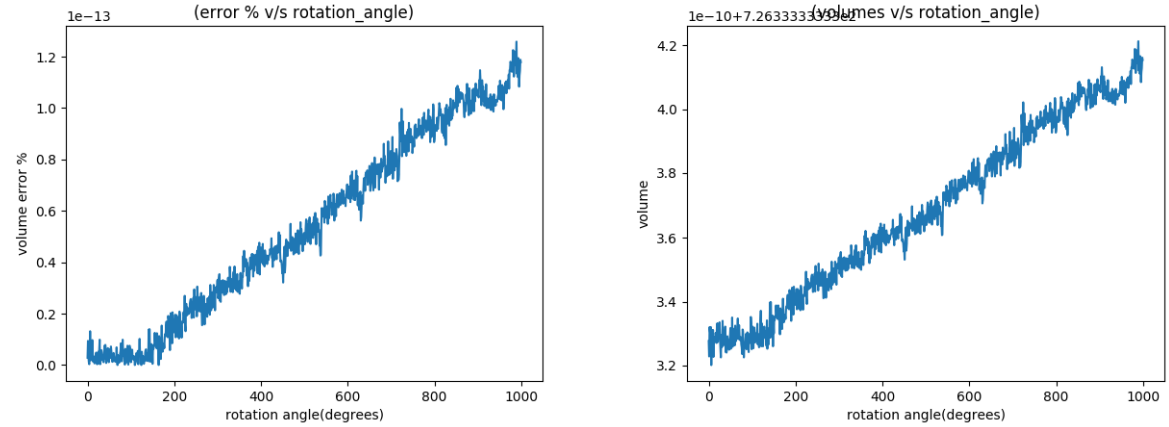

Fig. 4.3 CASE 1 : Error Analysis for Quaternion General Angle Rotation

transformed vector is recomputed. Hence the error drops down to zero. It can be observed that the frequency of re-computation is lower in Quaternion case. Hence it is preferable for small angle rotation.

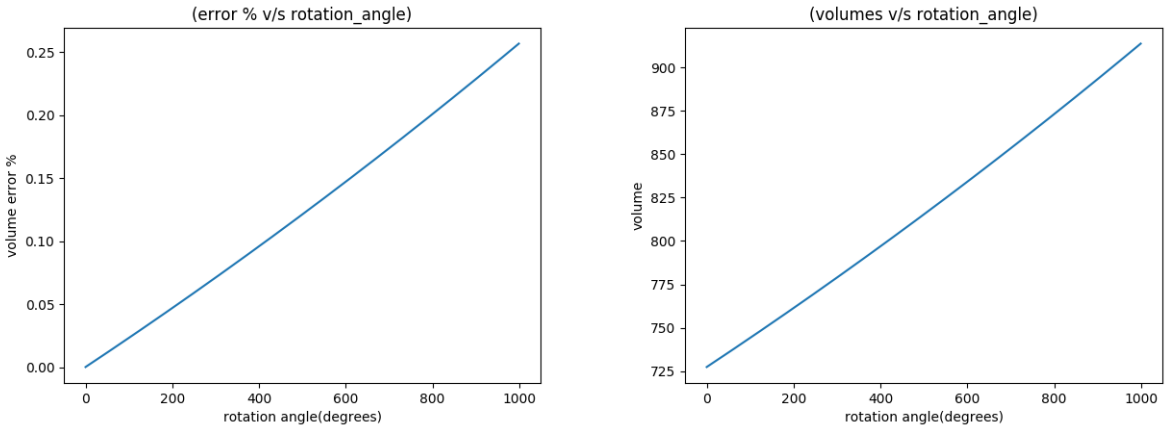

Fig. 4.4 CASE 1 : Error Analysis for Quaternion Small Angle Rotation

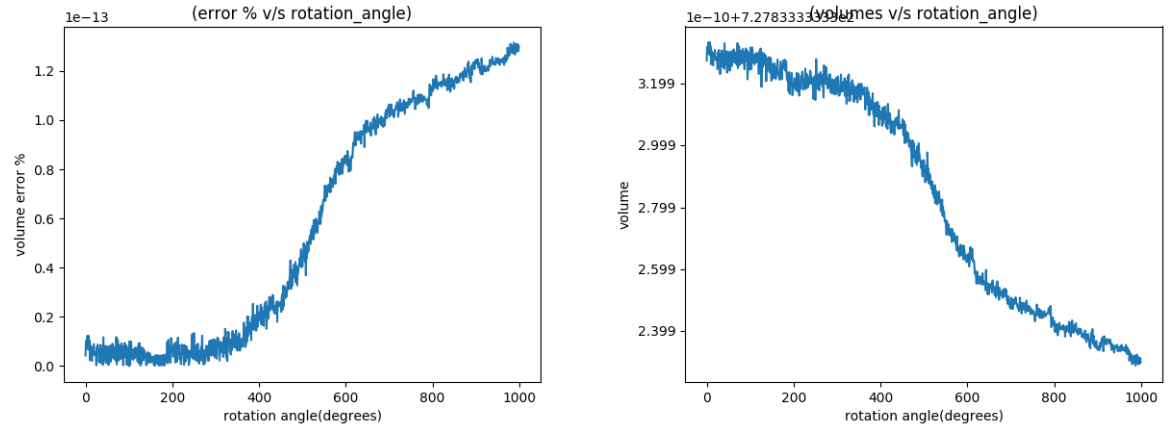

Fig. 4.5 CASE 2 : Error Analysis for Euler Small Angle Rotation

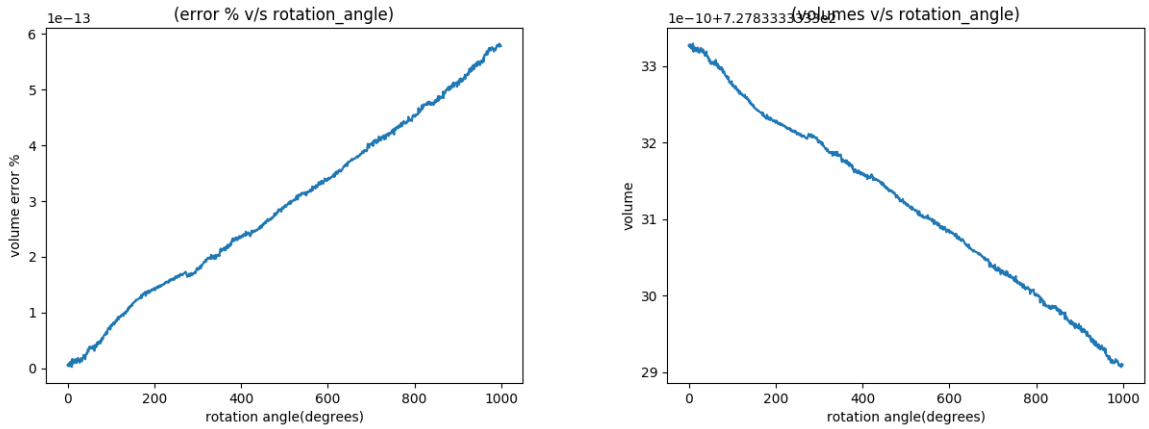

Fig. 4.6 CASE 2 : Error Analysis for Quaternion Small Angle Rotation

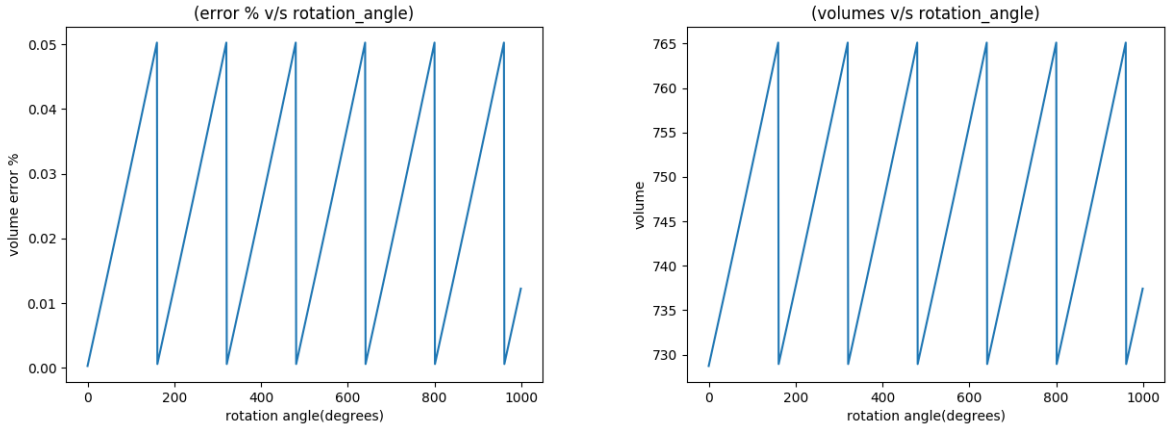

Fig. 4.7 CASE 3 : Error Analysis for Euler Small Angle Rotation

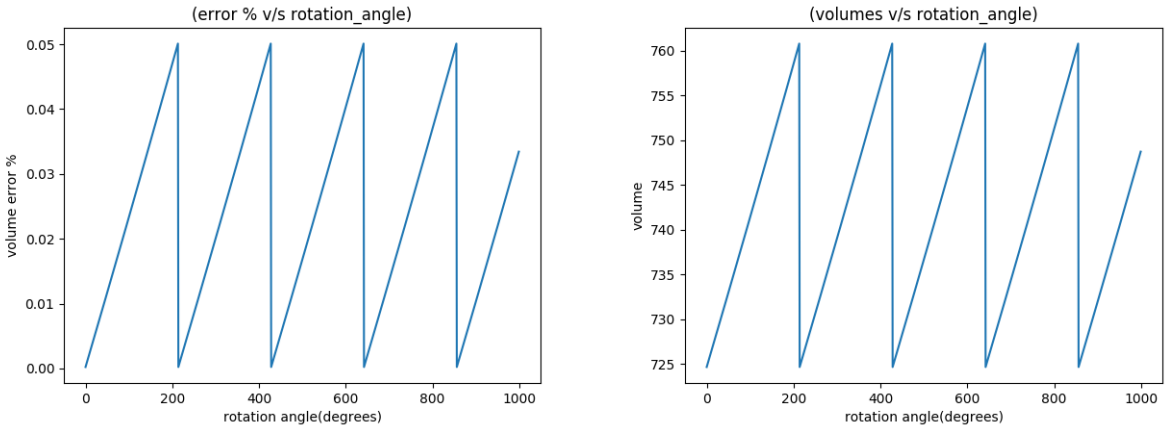

Fig. 4.8 CASE 3 : Error Analysis for Quaternion Small Angle Rotation

Chapter 5

Conclusions

- Rotations of vectors by quaternions has more benefits than the rotations using Euler rotation matrix
  - **Storage :** When we rotate a vector along a given arbitrary axis using quaternions then it involves only the axis (vector) of rotation and the angle, but when we rotate a vector along an arbitrary axis using Euler angles then it requires composition of three independent rotations about the coordinate axes.
  - **Chaining Operations :** When n rotation operations are chained then, quaternions provide a significant better performance.
  - **Small Angle Rotation :** For small angle rotation, we need to maintain the orthogonality of the matrix which require the normalization. Normalization in case of quaternion is more efficient than in case of matrix rotation.
  - **Gimbal Lock :** Loss of one degree of freedom when the property of mutual independence of rotation using Euler angle breaks down, which happens when the second Euler angle becomes $\frac{\pi}{2}$ degrees due to "Gimbal Lock"[WW92]. This singularity is not removed even if we try to come up with various other formulations of the Euler angles in the rotation matrix. Hence, frustrating results in graphics animation could result due to the condition of the gimbal lock.

- Error analysis using Delaunay tetrahedralization is done.
  - Percent error in case of small angle rotation is directly proportional to the rotation angle in both Euler and Quaternion rotation.
  - Negligible error is introduced in Euler and Quaternion general angle rotation due to precision in sin and cos calculation.
  - Due to normalization, error in both quaternion and Euler case decreases significantly.
  - Frequency of re-computation is higher in Euler case as compared to quaternion rotation. Hence it is preferable for small angle rotation.

References

[BML98] Garrett Birkhoff and Saunders Mac Lane. *A survey of modern algebra*. AK Peters/CRC Press, 1998.

[eas] *Error Analysis Source Code*.

[gra] *Graphs and Other Observations*.

[gsc] *Github Source Code*.

[Ham63] Sir William R. Hamilton. *Elements of Quaternions*. Chelsea Publishing Co., New York, 3rd edition, 1963.

[Muk02] R Mukundan. Quaternions: From classical mechanics to computer graphics, and beyond. In *Proceedings of the 7th Asian Technology conference in Mathematics*, pages 97–105, 2002.

[Sla99] Gregory G Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on August*, 6(2000):39–63, 1999.

[Vin95] John Vince. *Virtual reality systems*. Pearson Education India, 1995.

[WW92] Alan Watt and Mark Watt. Advanced animation and rendering techniques. 1992.