Matrix and Quaternions FAQ

SE Intern Training

Exported on 2025-03-04 08:28:06

Table of Contents

**No table of contents entries found.**

|  |
| --- |
| **License Warning**  You are using an UNLICENSED copy of **Scroll Word Exporter**.  Do you find Scroll Word Exporter useful?  Consider purchasing it today: <https://marketplace.atlassian.com/apps/24982/scroll-word-exporter-for-confluence?tab=overview&hosting=datacenter> |

From: <http://www.j3d.org/matrix_faq/matrfaq_latest.html>

The Matrix and Quaternions FAQ  
==============================  
  
Version 1.21 30th November 2003  
-------------------------------  
Please mail feedback to [matrix\_faq@j3d.org](mailto:matrix_faq@j3d.org)with a subject starting with  **MATRIX-FAQ** (otherwise my spam filter will simply kill your message).  
Any additional suggestions or related questions are welcome.  
Just send E-mail to the above address.  
The latest copy of this FAQ can be found at the following web page: [http://www.j3d.org/matrix\_faq/matrfaq\_latest.html](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html)Feel free to distribute or copy this FAQ as you please.  
  
Contributions  
-------------  
[Introduction I1](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#I1): [Steve ?????](mailto:steve@%28no-spam%29mred.bgm.link.com) Correction to [Q55](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q55) until [Q59](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q59):[Andreas Junghanns](mailto:andreas.junghanns@%28no-spam%29dcx.com) Correction to [Q50](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q50): [Morten Ofstad](mailto:morten@%28no-spam%29innerloop.no) Note to [Q39](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q39): [Tom Nuydens](mailto:t.nuydens@%28no-spam%29vrcontext.com) Corrections to [Q29](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q29) and [Q37](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q37): [Eric Reiss](mailto:eric@%28no-spam%29ee.pitt.edu) Clarification to [Q56](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q56): [Duncan Murdoch](mailto:murdoch@%28no-spam%29fisher.stats.uwo.ca) Clarification to [Q37](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q37): [Ron Avitzur](mailto:avitzur@%28no-spam%29PacificT.com) Correction to [Q1](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q1): [Mona Wong](mailto:mona@%28no-spam%29ncmir.ucsd.edu) Corrections to [Q36](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q36) and [Q37](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q37): [Eric Reiss](mailto:eric@%28no-spam%29ee.pitt.edu) Improvement to [Q34](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q34) and [Q38](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q38): [Jon Watte](mailto:hplus-mail@%28no-spam%29mindcontrol.org) Warning and alternative to [Q58](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q58) and [Q59](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q59): [Paul Pedriana](mailto:PPedriana@%28no-spam%29maxis.com) Correction (and optimization [Lee]) to [Q53](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q53): [Eleanor Groundwater](mailto:ellieg@%28no-spam%29cableinet.co.uk) and [Lee Morgan](mailto:LeeMorgan@%28no-spam%29lee-morgan.net) Improvement to [Q39](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q39): [jhunpingco](mailto:jhunpingco@%28no-spam%29yahoo.com) Corrections to [Q11](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q11) and optimization to [Q12](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q12): [Gordon](mailto:GDW33@%28no-spam%29student.canterbury.ac.nz) Corrections to [Q54](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q54) to [Q60](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q60): [Eleanor Groundwater](mailto:eleanorg@%28no-spam%29owl.co.uk) Corrections and improvements to [Q23](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q23) and [Q24](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q24): [Ben Houston](mailto:ben@%28no-spam%29exocortex.org) Addition to [Q39](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q39): [Jon Watte](mailto:hplus@%28no-spam%29mindcontrol.org) Correction to [Q61](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q61): [Adam D. Moss](mailto:adam@%28no-spam%29gimp.org) Addition of [Q63](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q63): [Mike Cline](mailto:cline@%28no-spam%29cs.ubc.ca) Addition of [I2](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#I2): [Jacob Marner](mailto:jacob@%28no-spam%29marner.dk) Correction to [Q38](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q38) and inception of [I2](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#I2): [Armin Müller](mailto:armuller@%28no-spam%29ira.uka.de) Addition of [Q60](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q60): [Pablo Figueroa](mailto:pfiguero@%28no-spam%29cs.ualberta.ca) Correntions and additions to [Q14](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q14), [Q16](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q16), [Q21](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q21) and [Q34](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q34): [Tronster Hartley](mailto:tronster321@%28no-spam%29hotmail.com) Correction to [Q12](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q12) and [Q54](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q54): [Frank DJ](mailto:frankdj@%28no-spam%29mailhost.ph.tn.tudelft.nl) Correction to [Q34](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q34): [Robert Funnell](mailto:robert.funnell@i%28no-spam%29mcgill.ca)History  
-------  
I (Andreas) tried to find "[hexapod@(no-spam)netcom.com](mailto:hexapod@%28no-spam%29netcom.com)" who seemed to have maintained  
this for a while, but the site at netcom.com doesn't exist anymore,  
emails bounce. Since I (and colleques) wasted quite some time figuring out  
what was wrong with some of the algorithms given in the earlier versions of  
this document, I decided to correct it and put it back on the web.  
The formerly given sites for the location of these documents do  
not exist anymore:  
 ftp://ftp.netcom.com/pub/he/hexapod/index.html  
 http://www.glue.umd.edu/~rsrodger  
 Versions, dates and links to local copies (so you can compare):[matrfaq\_1.02.html: Version 1.2 2nd September 1997](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.02.html)[matrfaq\_1.04.html: Version 1.4 26th December 1998](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.04.html)[matrfaq\_1.06.html: Version 1.6 30th September 2000](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.06.html)[matrfaq\_1.07.html: Version 1.7 20th December 2000](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.07.html)[matrfaq\_1.08.html: Version 1.8 21th December 2000](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.08.html)[matrfaq\_1.09.html: Version 1.9 16th January 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.09.html)[matrfaq\_1.10.html: Version 1.10 30th January 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.10.html)[matrfaq\_1.11.html: Version 1.11 9th February 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.11.html)[matrfaq\_1.12.html: Version 1.12 26th March 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.12.html)[matrfaq\_1.13.html: Version 1.13 20th July 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.13.html)[matrfaq\_1.14.html: Version 1.14 17th August 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.14.html)[matrfaq\_1.15.html: Version 1.15 20th August 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.15.html)[matrfaq\_1.16.html: Version 1.16 2nd October 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.16.html)[matrfaq\_1.17.html: Version 1.17 30th November 2001](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.17.html)[matrfaq\_1.18.html: Version 1.18 27th January 2002](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.18.html)[matrfaq\_1.19.html: Version 1.19 20th March 2002](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.19.html)[matrfaq\_1.20.html: Version 1.20 31st January 2002](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.20.html)[matrfaq\_1.21.html: Version 1.21 30th November 2003](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_1.21.html)   
  
Please refrain from asking me math questions. I am only maintaining this FAQ  
and have very little knowledge about the subject. But, if you have a  
question that is not answered by this FAQ and later happen to find the  
answer and believe it to be relevant for this FAQ (or its readers), please  
send all relevant information, hopefully in a pre-digested form, to me to  
be included here. Thanks!  
If you prefer to appear as "anonymous" in the contributions list, let me  
know, otherwise I'll just put you down with whatever name I can gather from  
your email header.  
  
Introduction  
------------[I1. Important note relating to OpenGL and this document](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#I1)[I2. Important note with respect to normalized inputs](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#I2)Questions  
---------  
  
BASICS  
====== [Q1. What is a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q1) [Q2. What is the order of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q2) [Q3. How do I represent a matrix using the C/C++ programming languages?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q3) [Q4. What are the advantages of using matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q4) [Q5. How do matrices relate to coordinate systems?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q5)ARITHMETIC  
========== [Q6. What is the identity matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q6) [Q7. What is the major diagonal matrix of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q7) [Q8. What is the transpose of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q8) [Q9. How do I add two matrices together?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q9)[Q10. How do I subtract two matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q10)[Q11. How do I multiply two matrices together?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q11)[Q12. How do I square or raise a matrix to a power?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q12)[Q13. How do I multiply one or more vectors by a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q13)DETERMINANTS AND INVERSES  
=========================[Q14. What is the determinant of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q14)[Q15. How do I calculate the determinant of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q15)[Q16. What are Isotropic and Anisotropic matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q16)[Q17. What is the inverse of a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q17)[Q18. How do I calculate the inverse of an arbitary matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q18)[Q19. How do I calculate the inverse of an identity matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q19)[Q20. How do I calculate the inverse of a rotation matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q20)[Q21. How do I calculate the inverse of a matrix using Kramer's rule?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q21)[Q22. How do I calculate the inverse of a 2x2 matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q22)[Q23. How do I calculate the inverse of a 3x3 matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q23)[Q24. How do I calculate the inverse of a 4x4 matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q24)[Q25. How do I calculate the inverse of a matrix using linear equations?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q25)TRANSFORMS  
==========[Q26. What is a rotation matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q26)[Q27. How do rotation matrices relate to coordinate systems?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q27)[Q28. How do I generate a rotation matrix in the X-axis?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q28)[Q29. How do I generate a rotation matrix in the Y-axis?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q29)[Q30. How do I generate a rotation matrix in the Z-axis?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q30)[Q31. What are Euler angles?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q31)[Q32. What are yaw, roll and pitch?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q32)[Q33. How do I combine rotation matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q33)[Q34. What is Gimbal Lock?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q34)[Q35. What is the correct way to combine rotation matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q35)[Q36. How do I generate a rotation matrix from Euler angles?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q36)[Q37. How do I generate Euler angles from a rotation matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q37)[Q38. How do I generate a rotation matrix for a selected axis and angle?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q38)[Q39. How do I generate a rotation matrix to map one vector onto another?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q39)[Q40. How do I use matrices to convert between two coordinate systems?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q40)[Q41. What is a translation matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q41)[Q42. What is a scaling matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q42)[Q43. What is a shearing matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q43)[Q44. How do I perform linear interpolation between two matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q44)[Q45. How do I perform cubic interpolation between four matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q45)[Q46. How can I render a matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q46)QUATERNIONS  
===========[Q47. What are quaternions?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q47)[Q48. How do quaternions relate to 3D animation?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q48)[Q49. How do I calculate the conjugate of a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q49)[Q50. How do I calculate the inverse of a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q50)[Q51. How do I calculate the magnitude of a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q51)[Q52. How do I normalise a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q52)[Q53. How do I multiply two quaternions together?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q53)[Q54. How do I convert a quaternion to a rotation matrix?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q54)[Q55. How do I convert a rotation matrix to a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q55)[Q56. How do I convert a rotation axis and angle to a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q56)[Q57. How do I convert a quaternion to a rotation axis and angle?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q57)[Q58. How do I convert spherical rotation angles to a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q58)[Q59. How do I convert a quaternion to spherical rotation angles?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q59)[Q60. How do I convert Euler rotation angles to a quaternion?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q60)[Q61. How do I use quaternions to perform linear interpolation between matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q61)[Q62. How do I use quaternions to perform cubic interpolation between matrices?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q62)[Q63. How do I use quaternions to rotate a vector?](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q63)Introduction  
------------   
  
I1. Important note relating to OpenGl and this document  
-------------------------------------------------------  
 In this document (as in most math textbooks), all matrices are drawn  
 in the standard mathematical manner. Unfortunately graphics libraries  
 like IrisGL, OpenGL and SGI's Performer all represent them with the  
 rows and columns swapped.  
 Hence, in this document you will see (for example) a 4x4 Translation  
 matrix represented as follows:  
  
 | 1 0 0 X |  
 | |  
 | 0 1 0 Y |  
 M = | |  
 | 0 0 1 Z |  
 | |  
 | 0 0 0 1 |  
  
 In Performer (for example) this would be populated as follows:  
  
 M[0][1] = M[0][2] = M[0][3] =  
 M[1][0] = M[1][2] = M[1][3] =  
 M[2][0] = M[2][1] = M[2][3] = 0 ;  
 M[0][0] = M[1][1] = M[2][2] = m[3][3] = 1 ;  
 M[3][0] = X ;  
 M[3][1] = Y ;  
 M[3][2] = Z ;  
  
 ie, the matrix is stored like this:  
  
 | M[0][0] M[1][0] M[2][0] M[3][0] |  
 | |  
 | M[0][1] M[1][1] M[2][1] M[3][1] |  
 M = | |  
 | M[0][2] M[1][2] M[2][2] M[3][2] |  
 | |  
 | M[0][3] M[1][3] M[2][3] M[3][3] |  
  
 OpenGL uses a one-dimensional array to store matrices - but fortunately,  
 the packing order results in the same layout of bytes in memory - so  
 taking the address of a pfMatrix and casting it to a float\* will allow  
 you to pass it directly into routines like glLoadMatrixf.  
 In the code snippets scattered throughout this document, a one-dimensional  
 array is used to store a matrix. The ordering of the array elements is  
 transposed with respect to OpenGL.  
   
 This Document OpenGL  
 | 0 1 2 3 | | 0 4 8 12 |  
 | | | |  
 | 4 5 6 7 | | 1 5 9 13 |  
 M = | | M = | |  
 | 8 9 10 11 | | 2 6 10 14 |  
 | | | |  
 | 12 13 14 15 | | 3 7 11 15 |  
  
I2. Important note with respect to normalized inputs  
----------------------------------------------------  
 Note that most algorithms assume normalized inputs, such as vectors of  
 union length, or matrices with normalized main diagonal etc. It is possible,  
 and often enough the case, that algorithms (and the code snippets provided here)  
 work correctly with arbitrary inputs, but it is usually considered bad practise  
 (and you will pay in debugging time if you fail to observe this suggestion) to  
 rely on this property.  
   
  
Answers  
-------  
  
BASICS  
======  
Q1. What is a matrix?  
----------------------  
 A matrix is a two dimensional array of numeric data, where each  
 row or column consists of one or more numeric values.  
 Arithmetic operations which can be performed with matrices include  
 addition, subtraction, multiplication and division.  
 The size of a matrix is defined in terms of the number of rows  
 and columns.  
 A matrix with M rows and N columns is defined as a MxN matrix.  
 Individual elements of the matrix are referenced using two index  
 values. Using mathematical notation these are usually assigned the  
 variables 'i' and 'j'. The order is row first, column second  
 For example, if a matrix M with order 4x4 exists, then the elements  
 of the matrix are indexed by the following row:column pairs:  
  
 | 00 01 02 03 |  
 M = | 10 11 12 13 |  
 | 20 21 22 23 |  
 | 30 31 32 33 |  
  
 The element at the top right of the matrix has i=0 and j=3  
 This is referenced as follows:  
  
 M = M  
 i,j 0,3  
  
 In computer animation, the most commonly used matrices have either  
 2, 3 or 4 rows and columns. These are referred to as 2x2, 3x3 and 4x4  
 matrices respectively.  
 2x2 matrices are used to perform rotations, shears and other types  
 of image processing. General purpose NxN matrices can be used to  
 perform image processing functions such as convolution.  
 3x3 matrices are used to perform low-budget 3D animation. Operations  
 such as rotation and multiplication can be performed using matrix  
 operations, but perspective depth projection is performed using  
 standard optimised into pure divide operations.  
 4x4 matrices are used to perform high-end 3D animation. Operations  
 such as multiplication and perspective depth projection can be  
 performed using matrix mathematics.  
   
Q2. What is the "order" of a matrix?  
-------------------------------------  
 The "order" of a matrix is another name for the size of the matrix.  
 A matrix with M rows and N columns is said to have order MxN.  
   
Q3. How do I represent a matrix using the C/C++ programming languages?  
-----------------------------------------------------------------------  
 The simplest way of defining a matrix using the C/C++ programming  
 languages is to make use of the "typedef" keyword. Both 3x3 and 4x4  
 matrices may be defined in this way ie:  
  
 typedef float MATRIX3[9];  
 typedef float MATRIX4[16];  
  
 Since each type of matrix has dimensions 3x3 and 4x4, this requires  
 9 and 16 data elements respectively.  
 At first glance, the use of a single linear array of data values may  
 seem counter-intuitive. The use of two dimensional arrays may seem  
 more convenient ie.  
  
 typedef float MATRIX3[3][3];  
 typedef float MATRIX4[4][4];  
  
 However, the use of two reference systems for each matrix element  
 very often leads to confusion. With mathemetics, the order is row  
 first (i), column second (j) ie.  
  
 Mij  
  
 Using C/C++, this becomes  
  
 matrix[j][i]  
  
 Using two dimensional arrays also incurs a CPU performance penalty in  
 that C compilers will often make use of multiplication operations to  
 resolve array index operations.  
 So, it is more efficient to stick with linear arrays. However, one issue  
 still remains to be resolved. How is an two dimensional matrix mapped  
 onto a linear array? Since there are only two methods (row first/column  
 second or column first/row column).  
 The performance differences between the two are subtle. If all for-next  
 loops are unravelled, then there is very little difference in the  
 performance for operations such as matrix-matrix multiplication.  
 Using the C/C++ programming languages the linear ordering of each  
 matrix is as follows:  
  
 mat[0] = M mat[3] = M  
 00 03  
 mat[12] = M mat[15] = M  
 30 33  
  
 | 0 1 2 3 |  
 | | | 0 1 2 |  
 | 4 5 6 7 | | |  
 M = | | M = | 3 4 5 |  
 | 8 9 10 11 | | |  
 | | | 6 7 8 |  
 | 12 13 14 15 |  
  
Q4. What are the advantages of using matrices?  
-----------------------------------------------  
 One of the first questions asked about the use of matrices in computer  
 animation is why they should be used at all in the first place.  
 Intuitively, it would appear that the overhead of for-next loops and  
 matrix multiplication would slow down an application.  
 Arguments that resolve these objections can be pointed out. These include  
 the use of CPU registers to handle loop counters on-board data caches  
 to optimise memory accesses.  
 Advantages can also be pointed out. By following a mathematical approach  
 to defining 3D algorithms, it is possible to predict and plan the  
 design of a 3D animation system. Such mathematical approaches allow  
 for the implementation of character animation, spline curves and inverse  
 kinematics.  
 However, one objection that frequently comes up is that it would be  
 quicker to just multiply each pair of coordinates by the rotation  
 coefficients for that axis, rather than perform a full vector-matrix  
 multiplication.  
 ie. Rotation in X transforms Y and Z  
 Rotation in Y transforms X and Z  
 Rotation in Z transforms X and Y  
 The argument to this goes as follows:  
 Given a vertex V = (x,y,z), rotation angles (A,B and C) and translation  
 (D,E,F). A the algorithm  
 is defined as follows:  
 ---------------------------  
 sx = sin(A) // Setup - only done once  
 cx = cos(A)  
 sy = sin(B)  
 cy = cos(B)  
 sz = sin(C)  
 cz = cos(C)  
 x1 = x \* cz + y \* sz // Rotation of each vertex  
 y1 = y \* cz - x \* sz  
 z1 = z  
 x2 = x1 \* cy + z1 \* sy  
 y2 = z1  
 z2 = z1 \* cy - x1 \* sy  
 x3 = x2  
 y3 = y2 \* cx + z1 \* sx  
 z3 = z2 \* cx - x1 \* sx  
 xr = x3 + D // Translation of each vertex  
 yr = y3 + E  
 zr = z3 + F  
 ---------------------------  
 Altogether, this algorithm will use the following amounts of processing  
 time:  
 Set-up Per-vertex  
 ------------------------- ------------------------  
 6 trigonometric functions  
 6 assignment operations. 12 assignment  
 12 multiplication  
 9 addition  
 ------------------------- ------------------------  
 Assume that the same operations is being performed using matrix  
 multiplication.  
 With a 4x4 matrix, the procesing time is used as follows:  
 Set-up Change Per-vertex Change  
 -------------------------- ------ ------------------------ ------  
 6 trigonometric functions 0 0  
 18 assignment operation -12 3 assignment -9  
 12 multiplication +12 9 multiplication -3  
 6 subtraction +6 6 addition -3  
 -------------------------- ------ ------------------------ ------  
 Comparing the two tables, it can be seen that setting up a rotation  
 matrix costs at least 12 multiplication calculations and an extra  
 18 assignment calls.  
 However, while this may seem extravagant, the savings come from  
 processing each vertex. Using matrix multiplication, the savings made  
 from processing just 4 vertices, will outweigh the additional set-up  
 cost.  
   
Q5. How do matrices relate to coordinate systems?  
--------------------------------------------------  
 With either 3x3 or 4x4 rotation, translation or shearing matrices, there  
 is a simple relationship between each matrix and the resulting coordinate  
 system.  
 The first three columns of the matrix define the direction vector of the  
 X, Y and Z axii respectively.  
 If a 4x4 matrix is defined as:  
  
 | A B C D |  
 M = | E F G H |  
 | I J K L |  
 | M N O P |  
  
 Then the direction vector for each axis is as follows:  
  
 X-axis = [ A E I ]  
 Y-axis = [ B F J ]  
 Z-axis = [ C G K ]  
   
ARITHMETIC  
==========  
  
Q6. What is the identity matrix?  
---------------------------------  
 The identity matrix is matrix in which has an identical number of rows  
 and columns. Also, all the elements in which i=j are set one. All others  
 are set to zero. For example a 4x4 identity matrix is as follows:  
  
 | 1 0 0 0 |  
 M = | 0 1 0 0 |  
 | 0 0 1 0 |  
 | 0 0 0 1 |  
  
Q7. What is the major diagonal of a matrix?  
--------------------------------------------  
 The major diagonal of a matrix is the set of elements where the  
 row number is equal to the column number ie.  
  
 M where i=j  
 ij  
  
 In the case of the identity matrix, only the elements on the major  
 diagonal are set to 1, while all others are set to 0.  
   
Q8. What is the transpose of a matrix?  
---------------------------------------  
 The transpose of matrix is the matrix generated when every element in  
 the matrix is swapped with the opposite relative to the major diagonal  
 This can be expressed as the mathematical operation:  
  
 M' = M  
 ij ji  
  
 However, this can only be performed if a matrix has an equal number  
 of rows and columns.  
 If the matrix M is defined as:  
  
 | 0.707 -0.866 |  
 M = | |  
 | 0.866 0.707 |  
  
 Then the transpose is equal to:  
  
 | 0.707 0.866 |  
 T = | |  
 | -0.866 0.707 |  
  
 If the matrix is a rotation matrix, then the transpose is guaranteed  
 to be the inverse of the matrix.  
   
Q9. How do I add two matrices together?  
----------------------------------------  
 The rule of thumb with adding two matrices together is:  
 "add row and column to row and column"  
 This can be expressed mathematically as:  
  
 R = M + L  
 ij ij ij  
  
 However, both matrices must be identical in size.  
 For example, if the 2x2 matrix M is added with the 2x2 matrix L then  
 the result is as follow:  
  
 R = M + L  
 | A B C | | J K L |  
 | | | |  
 = | D E F | + | M N O |  
 | | | |  
 | G H I | | P Q R |  
  
 | A+J B+K C+L |  
 | |  
 = | D+M E+N F+O |  
 | |  
 | G+P H+Q I+R |  
  
Q10. How do I subtract two matrices?  
-------------------------------------  
 The rule of thumb with subtracting two matrices is:  
 "subtract row and column from row and column"  
 This can be expressed mathematically as:  
  
 R = M - L  
 ij ij ij  
  
 However, both matrices must be identical in size.  
 For example, if the 2x2 matrix L is subtracted from the 2x2 matrix M then  
 the result is as follows:  
  
 R = M - L  
 | A B C | | J K L |  
 | | | |  
 = | D E F | - | M N O |  
 | | | |  
 | G H I | | P Q R |  
  
 | A-J B-K C-L |  
 | |  
 = | D-M E-N F-O |  
 | |  
 | G-P H-Q I-R |  
  
Q11. How do I multiply two matrices together?  
---------------------------------------------  
 The rule of thumb with multiplying two matrices together is:  
 "multiply row into column and sum the result".  
 This can be expressed mathematically as:  
 n  
  
 --  
 R = \ M x L  
 ij / ik kj  
 --  
 k=1  
  
 If the two matrices to be multiplied together have orders:  
  
 M = AxB and L = CxD  
  
 then the two values B and C must be identical.  
 Also, the resulting matrix has an order of AxD  
 Thus, it is possible to multiply a Nx4 matrix with a 4x4 matrix  
 but not the other way around.  
 For example, if the 4x4 matrix M is defined as:  
  
 | A B C D |  
 M = | E F G H |  
 | I J K L |  
 | M N O P |  
  
 and a 4x2 matrix L is defined as:  
  
 L = | Q R |  
 | S T |  
 | U V |  
 | W X |  
  
 then the size of the resulting matrix is 4x2. The resulting matrix  
 is defined as:  
  
 R = M x L  
  
 | A B C D | | Q R |  
 = | E F G H | x | S T |  
 | I J K L | | U V |  
 | M N O P | | W X |  
  
 | AQ+BS+CU+DW AR+BT+CV+DX |  
 = | EQ+FS+GU+HW ER+FT+GV+HX |  
 | IQ+JS+KU+LW IR+JT+KV+LX |  
 | MQ+NS+OU+PW MR+NT+OV+PX |  
   
Q12. How do I square or raise a matrix to a power?  
--------------------------------------------------  
 A matrix may be squared or even raised to an integer power. However  
 there are several restrictions. For all powers, the matrix must be  
 square, that is orthogonal and the same width and height  
 For example,  
  
 -1  
 M is the inverse of the matrix  
  
 0  
 M generates the identity matrix  
  
 1  
 M leaves the matrix unchanged.  
  
 2  
 M squares the matrix and  
  
 3  
 M generates the cube of the matrix  
  
 Raising a matrix to a power greater than one involves multiplying a matrix  
 by itself a specific number of times.  
 For example,  
  
 2  
 M = M . M  
  
 3  
 M = M . M . M  
  
 and so on.  
   
 Raising the identity matrix to any power always generates the identity  
 matrix ie.  
  
 n  
 I = I  
  
 One can be bit faster using the following piece of code, note that m and i  
 are both getting changed:  
 m -> matrix to be raised to a power  
 i -> power to raise matrix to  
 a -> matrix that will contain the result  
 I -> identity matrix  
 / -> integer division (ie round down to nearest whole number)  
 % -> integer remainder operation  
 \* -> matrix multiplication  
   
  
 a=I  
 while(i>0){  
 if(i%2) a=a\*m  
 i=i/2  
 m=m\*m  
 }  
  
Q13. How do I multiply one or more vectors by a matrix?  
-------------------------------------------------------  
 The best way to perform this task is to treat the list of vectors as  
 a single matrix, with each vector represented as a column vector.  
 If N vectors are to be multiplied by a 4x4 matrix, then they can be  
 treated as a single 4xN matrix:  
 If the matrix is defined as:  
  
 | A B C D |  
 M = | E F G H |  
 | I J K L |  
 | M N O P |  
  
 and the list of vectors is defined as:  
  
 | x1 x2 x3 x4 x5|  
 V = | y1 y2 y3 y4 y5|  
 | z1 z2 z3 z4 z5|  
 | 1 1 1 1 1|  
  
 Note that an additional row of constant terms is added to the vector  
 list, all of which are set to 1.0. In real life, this row does not  
 exist. It is simply used to make the orders of the matrix M and the  
 vector list V match.  
 Then the multiplication is performed as follows:  
  
 M . V = V'  
  
 | A B C D | | x1 x2 x3 x4 x5 | | A.x1+B.y1+C.z1+D A.x2+B.y2+C.z2+D ... |  
 | E F G H | . | y1 y2 y3 y4 y5 | = | E.x1+F.y1+G.z1+H E.x2+F.y2+G.z2+H ... |  
 | I J K L | | z1 z2 y3 y4 z5 | | I.x1+J.y1+K.z1+L I.x2+J.y2+K.z2+L ... |  
 | M N O P | | 1 1 1 1 1 | | M.x1+N.y1+O.z1+P M.x2+N.y2+O.z2+P ... |  
  
 For each vector in the list there will be a total of 12 multiplication  
 16 addition and 1 division operation (for perspective).  
 If the matrix is known not to be a rotation or translation matrix then the  
 division operation can be skipped.  
   
DETERMINANTS AND INVERSES  
=========================  
  
Q14. What is the determinant of a matrix?  
-----------------------------------------  
 The determinant of a matrix is a floating point value which is used to  
 indicate whether the matrix has an inverse or not. If zero, then no  
 inverse exists. If non-zero, then an inverse exists.  
 As an example, consider a matrix consisting of a single element:  
  
 M = [ 1 ].  
  
 For a matrix of this size, the determinant is simply the value of the   
 single element.  
 Also, the inverse is simply the reciprocal of this single element:   
  
 -1  
 M = [ 1 / M[0][0] ]   
  
 If this single value is non-zero, then an inverse exists. In the case   
 of the identity matrix, this happens to be 1 / 1 or 1.0   
 However, if the value of this single element is zero, then the determinant  
 is also zero.  
   
 Attempting to calculate the reciprocal of zero, generates a value of  
 infinity. This isn't permitted as far a matrices are concerned, so no  
 inverse of the matrix exists.  
 For an identity matrix, the determinant is always equal to one.  
 Any matrix with a determinant of 1.0 is said to be isotropic.  
 Thus all rotation matrices are said to be isotropic, since the  
 determinant is always equal to 1.0.  
 This can be proved as follows:  
  
 | A B | | cos X -sin X |  
 M = | | = | |  
 | C D | | sin X cos X |  
  
 D = AD - BC  
  
 D = (cos X . cos X) - (-sin X . sin X)  
  
 2 2  
 D = (cos X ) + (sin X)  
  
 2 2  
 But, cos X + sin X = 1  
  
 Therefore,  
 D = 1  
  
Q15. How do I calculate the determinant of a matrix?  
----------------------------------------------------  
 The determinant of a matrix is calculated using Kramer's rule, where  
 the value can be calculated by breaking the matrix into smaller  
 matrices.  
 For a 2x2 matrix M, the determinant D is calculated as follows:  
  
 | A B |  
 M = | |  
 | C D |  
  
 D = AD - BC  
  
 For 3x3 and 4x4 matrices, this is more complicated, but can be solved  
 by methods such as Kramer's Rule.  
   
Q16. What are Isotropic and Anisotropic matrices?  
-------------------------------------------------  
 An Isotropic matrix is one in which the sum of the squares of all  
 three rows or columns add up to one.  
 A matrix in which this is not the case, is said to be Anisotropic.  
 When 3x3 or 4x4 matrices are used to rotate and scale an object, it  
 is sometimes necessary to enlarge or shrink one axis more than the  
 others.  
 For example, with seismic surveys, it is convenient to enlarge the  
 Z-axis by a factor or 50 or more, while letting the X and Y axii  
 remain the same.  
 Another example is the implementation of "squash" and "stretch"  
 with character animation. When a character is hit by a heavy object  
 eg. an anvil, the desired effect is to make the character look stretched out  
 sideways and squashed vertically:  
 A suitable matrix would be as follows:  
  
 | 2 0 0 0 |  
 M = | 0 2 0 0 |  
 | 0 0 0.5 0 |  
 | 0 0 0 1 |  
  
 However, there is problem looming ahead. While this matrix will cause  
 no problems with the transformation of vertex data, it will cause  
 problems with gouraud shading using outward normals.  
 Because the transformation stage is implemented using matrix  
 multiplication, both vertex data and outward normal data will be  
 multiplied with this matrix.  
 While this is not a problem with vertex data (it is the desired effect)  
 it causes a major headache with the outward normal data.  
 After raw multiplication, each outward normal will no longer be  
 normalised and consequently will affect other calculations such as  
 shading and back-face culling.  
   
Q17. What is the inverse of a matrix?  
-------------------------------------  
 -1  
 Given a matrix M, then the inverse of that matrix, denoted as M , is  
 the matrix which satisfies the following expression:  
  
 -1  
 M . M = I  
  
 where I is the identity matrix.  
 Thus, multiplying a matrix with its inverse will generate the identity  
 matrix. However, several requirements must be satisfied before the  
 inverse of a matrix can be calculated.  
 These include that the width and height of the matrix are identical and  
 that the determinant of the matrix is non-zero.  
 Calculating the inverse of a matrix is a task often performed in order  
 to implement inverse kinematics using spline curves.  
   
Q18. How do I calculate the inverse of an arbitary matrix?  
----------------------------------------------------------  
 Depending upon the size of the matrix, the calculation of the inverse  
 can be trivial or extremely complicated.  
 For example, the inverse of a 1x1 matrix is simply the reciprocal of  
 the single element:  
  
 ie. M = | x |  
  
 Then the inverse is defined as:  
  
 -1 | 1 |  
 M = | - |  
 | x |  
  
 Solving 2x2 matrices and larger can be achieved by using Kramer's Rule  
 or by solving as a set of simultaneous equations.  
 However, in certain cases, such as identity or rotation matrices, the  
 inverse is already known or can be determined from taking the transpose  
 of the matrix.  
   
Q19. How do I calculate the inverse of an identity matrix?  
----------------------------------------------------------  
 Don't even bother. The inverse of an identity matrix is the identity  
 matrix. ie.  
  
 -1  
 I . I = I  
  
 Any identity matrix will always have a determinant of +1.  
   
Q20. How do I calculate the inverse of a rotation matrix?  
---------------------------------------------------------  
 Since a rotation matrix always generates a determinant of +1,  
 calculating the inverse is equivalent of calculating the transpose.  
 Alternatively, if the rotation angle is known, then the rotation  
 angle can be negated and used to calculate a new rotation matrix.  
   
Q21. How do I calculate the inverse of a matrix using Kramer's rule?  
--------------------------------------------------------------------  
 Given a 3x3 matrix M:  
  
 | A B C |  
 | |  
 M = | D E F |  
 | |  
 | G H I |  
  
 Then the determinant is calculated as follows:  
  
 n  
 ---  
 \ i  
 det M = / M \* submat M \* -1  
 --- 0,i 0,i  
 i=1  
 where  
  
 submat M defines the matrix composed of all rows and columns of M  
 ij  
  
 excluding row i and column j. submat may be called recursively.  
 ij  
 | A B C |  
 M = | D E F | becomes submat = | E F |  
 | G H I | 11 | H I |  
   
 If the determinant is non-zero then the inverse of the matrix exists.  
 In this case, the value of each matrix element is defined by:  
  
 -1 1 i+j  
 M = ----- \* det submat M \* -1  
 j,i det M i,j  
  
Q22. How do I calculate the inverse of a 2x2 matrix?  
----------------------------------------------------  
 For a 2x2 matrix, the calculation is slightly harder. If the matrix is  
 defined as follows:  
  
 | A B |  
 M = | |  
 | C D |  
  
 Then the determinant is defined as:  
  
 det = AD - BC  
  
 And the inverse is defined as:  
  
 -1 1 | D -B |  
 M = --- | |  
 det | -C A |  
  
 This can be proved using Kramer's rule. Given the matrix M:  
  
 | A B |  
 M = | |  
 | C D |  
  
 Then the determinant is:  
  
 0 1  
 det = M \* submat M \* -1 + M \* submat M \* -1  
 0,0 0,0 0,1 0,1  
  
 <=> M \* M \* 1 + M \* M \* -1  
 0,0 1,1 0,1 1,0  
  
 <=> A \* D + B \* C \* -1  
  
 <=> AD + BC . -1  
  
 <=> AD - BC  
  
 And the inverse is derived from:  
  
 -1 0+0 -1  
 M = det submat \* -1 <=> M = M \* 1 <=> D  
 0,0 0,0 0,0 1,1  
  
 -1 1+0 -1  
 M = det submat \* -1 <=> M = M \* -1 <=> C \* -1  
 0,1 1,0 0,1 1,0  
  
 -1 0+1 -1  
 M = det submat \* -1 <=> M = M \* -1 <=> B \* -1  
 1,0 0,1 1,0 0,1  
  
 -1 1+1 -1  
 M = det submat \* -1 <=> M = M \* 1 <=> A  
 1,1 1,1 1,1 0,0  
  
 Then the inverse matrix is equal to:  
  
 -1 1 | D -C |  
 M = --- | |  
 det | -B A |  
  
 Providing that the determinant is not zero.  
   
Q23. How do I calculate the inverse of a 3x3 matrix?  
----------------------------------------------------  
 For 3x3 matrices and larger, the inverse can be calculated by  
 either applying Kramer's rule or by solving as a set of linear  
 equations.  
 If Kramer's rule is applied to a matrix M:  
  
 | A B C |  
 M = | D E F |  
 | G H I |  
  
 then the determinant is calculated as follows:  
 det M = A \* (EI - HF) - B \* (DI - GF) + C \* (DH - GE)  
  
 Providing that the determinant is non-zero, then the inverse is  
 calculated as:  
  
 -1 1 | EI-FH -(BI-HC) BF-EC |  
 M = ----- . | -(DI-FG) AI-GC -(AF-DC) |  
 det M | DH-GE -(AH-GB) AE-BD |  
  
 This can be implemented using a pair of 'C' functions:  
  
 VFLOAT m3\_det( MATRIX3 mat )  
 {  
 VFLOAT det;  
 det = mat[0] \* ( mat[4]\*mat[8] - mat[7]\*mat[5] )  
 - mat[1] \* ( mat[3]\*mat[8] - mat[6]\*mat[5] )  
 + mat[2] \* ( mat[3]\*mat[7] - mat[6]\*mat[4] );  
 return( det );  
 }  
  
 int m3\_inverse( MATRIX3 mr, MATRIX3 ma )  
 {  
 VFLOAT det = m3\_det( ma );  
 if ( fabs( det ) < 0.0005 )  
 {  
  
 m3\_identity( mr );  
 return(0);  
 }  
  
 mr[0] = ma[4]\*ma[8] - ma[5]\*ma[7] / det;  
 mr[1] = -( ma[1]\*ma[8] - ma[7]\*ma[2] ) / det;  
 mr[2] = ma[1]\*ma[5] - ma[4]\*ma[2] / det;  
 mr[3] = -( ma[3]\*ma[8] - ma[5]\*ma[6] ) / det;  
 mr[4] = ma[0]\*ma[8] - ma[6]\*ma[2] / det;  
 mr[5] = -( ma[0]\*ma[5] - ma[3]\*ma[2] ) / det;  
 mr[6] = ma[3]\*ma[7] - ma[6]\*ma[4] / det;  
 mr[7] = -( ma[0]\*ma[7] - ma[6]\*ma[1] ) / det;  
 mr[8] = ma[0]\*ma[4] - ma[1]\*ma[3] / det;  
 return(1);  
 }  
  
Q24. How do I calculate the inverse of a 4x4 matrix?  
----------------------------------------------------  
 As with 3x3 matrices, either Kramer's rule can be applied or the  
 matrix can be solved as a set of linear equations.  
 An efficient way is to make use of the existing 'C' functions defined  
 to calculate the determinant and inverse of a 3x3 matrix.  
 In order to implement Kramer's rule with 4x4 matrices, it is necessary  
 to determine individual sub-matrices. This is achieved by the following  
 routine:  
  
 void m4\_submat( MATRIX4 mr, MATRIX3 mb, int i, int j ) {  
 int di, dj, si, sj;  
 // loop through 3x3 submatrix  
 for( di = 0; di < 3; di ++ ) {  
 for( dj = 0; dj < 3; dj ++ ) {  
 // map 3x3 element (destination) to 4x4 element (source)  
 si = di + ( ( di >= i ) ? 1 : 0 );  
 sj = dj + ( ( dj >= j ) ? 1 : 0 );  
 // copy element  
 mb[di \* 3 + dj] = mr[si \* 4 + sj];  
 }  
 }  
 }  
  
 The determinant of a 4x4 matrix can be calculated as follows:  
  
 VFLOAT m4\_det( MATRIX4 mr )  
 {  
 VFLOAT det, result = 0, i = 1;  
 MATRIX3 msub3;  
 int n;  
 for ( n = 0; n < 4; n++, i \*= -1 )  
 {  
  
 m4\_submat( mr, msub3, 0, n );  
 det = m3\_det( msub3 );  
 result += mr[n] \* det \* i;  
 }  
 return( result );  
 }  
  
 And the inverse can be calculated as follows:  
  
 int m4\_inverse( MATRIX4 mr, MATRIX4 ma )  
 {  
  
 VFLOAT mdet = m4\_det( ma );  
 MATRIX3 mtemp;  
 int i, j, sign;  
 if ( fabs( mdet ) < 0.0005 )  
 m4\_identity( mr );  
 return( 0 );  
 for ( i = 0; i < 4; i++ )  
 for ( j = 0; j < 4; j++ )  
 {  
 sign = 1 - ( (i +j) % 2 ) \* 2;  
 m4\_submat( ma, mtemp, i, j );  
 mr[i+j\*4] = ( m3\_det( mtemp ) \* sign ) / mdet;  
 }  
 return( 1 );  
 }  
  
 Having a function that can calculate the inverse of any 4x4 matrix is  
 an incredibly useful tool. Application include being able to calculate  
 the base matrix for splines, inverse rotations and rearranging matrix  
 equations.  
   
Q25. How do I calculate the inverse of a matrix using linear equations?  
-----------------------------------------------------------------------  
 If a matrix M exists, such that:  
  
 | A B C |  
 M = | D E F |  
 | G H I |  
  
 then the inverse exists:  
  
 | P Q R |  
 M' = | S T U |  
 | V W X |  
  
 and the following expression is valid:  
  
 -1  
 M . M = I  
  
 | A B C | | P Q R | | 1 0 0 |  
 | D E F | . | S T U | = | 0 1 0 |  
 | G H I | | V W X | | 0 0 1 |  
  
 The inverse can then be calculated through the solution as a set of  
 linear equations ie.:  
  
 | AP + BS + CV | | 1 | Column 0 (X)  
 | DP + ES + FV | = | 0 |  
 | GP + HS + IV | | 0 |  
  
 | AQ + BT + CW | | 0 | Column 1 (Y)  
 | DQ + ET + FW | = | 1 |  
 | GQ + HT + IW | | 0 |  
  
 | AR + BU + CX | | 0 | Column 2 (Z)  
 | DR + EU + FX | = | 0 |  
 | GR + HU + IX | | 1 |  
  
TRANSFORMS  
==========  
  
Q26. What is a rotation matrix?  
-------------------------------  
 A rotation matrix is used to rotate a set of points within a  
 coordinate system. While the individual points are assigned new  
 coordinates, their relative distances do not change.  
 All rotations are defined using the trigonometric "sine" and "cosine"  
 functions.  
 For a two-dimensional coordinate system, the rotation matrix is as  
 follows:  
  
 | cos(A) -sin(A) |  
 | |  
 | sin(A) cos(A) |  
  
 With the rotation angle A set to zero, this generates the identity  
 matrix:  
  
 | 1 0 |  
 I = | |  
 | 0 1 |  
  
 If the rotation is set to +90 degrees, then the matrix is as follows:  
  
 | 0 -1 |  
 M = | |  
 | 1 0 |  
  
 If the rotation is set to -90 degrees, then the matrix is as follows:  
  
 | 0 1 |  
 M = | |  
 | -1 0 |  
  
 Negating the rotation angle is equivalent to generating the transpose  
 of the matrix.  
 If a rotation matrix is multiplied with its transpose, the result is  
 the identity matrix.  
  
Q27. How do rotation matrices relate to coordinate systems?  
------------------------------------------------------  
 Rotation matrices relate to coordinate systems in the following way.  
 Mathematical convention requires that a positive rotation angle  
 generates a clockwise rotation when looking from the origin towards  
 the positive end of the rotation axis.  
 Applying this rule, allows for the derivation of three Cartesian  
 rotation matrices. Consider a right-handed coordinate system. For each  
 rotation axis, look from the origin towards the positive end of the  
 selected axis. This generates the following three views:  
 +----------------------------------------+  
 | |  
 | X-axis Y-axis Z-axis |  
 | |  
 | |  
 | ^ Y ^ Z Y ^ |  
 | | | | |  
 | | | | |  
 | | | | |  
 | | | | |  
 | O----> Z O----> X X <----O |  
 | |  
 +----------------------------------------+  
 Since a positive rotation angle generates a clockwise rotation, it is  
 possible to generate a set of coordinate mappings for each rotation.  
 For simplicity, a rotation of +90 will be considered:  
 Starting with the X-axis:  
  
 ( 0, 1, 0 ) -> ( 0, 0, 1 )  
 ( 0, 0, 1 ) -> ( 0,-1, 0 )  
 ( 0,-1, 0 ) -> ( 0, 0,-1 )  
 ( 0, 0,-1 ) -> ( 0, 1, 0 )  
  
 These can be simplified to:  
  
 X' = X  
 Y' = -Z  
 Z' = Y  
  
 These can then be placed into a matrix:  
  
 | 1 0 0 |  
 Rx = | 0 cos A -sin A |  
 | 0 sin A cos A |  
  
 Doing the same for the Y-axis:  
  
 ( 0, 0, 1) -> ( 1,0, 0)  
 ( 1, 0, 0) -> ( 0,0,-1)  
 ( 0, 0,-1) -> (-1,0, 0)  
 (-1, 0, 0) -> ( 0,0, 1)  
  
 These can be simplified to:  
  
 X' = Z  
 Y' = Y  
 Z' = -X  
  
 These can then be placed into a matrix:  
  
 | cos A 0 sin A |  
 Ry = | 0 1 0 |  
 | -sin A 0 cos A |  
  
 And finally for the Z-axis:  
  
 ( 0, 1, 0 ) -> ( -1, 0, 0 )  
 (-1, 0, 0 ) -> ( 0, -1, 0 )  
 ( 0,-1, 0 ) -> ( 1, 0, 0 )  
 ( 1, 0, 0 ) -> ( 0, 1, 0 )  
  
 These can be simplified to:  
  
 X' = -Y  
 Y' = X  
 Z' = Z  
  
 Placing these into a matrix:  
  
 | cos A -sin A 0 |  
 Rz = | sin A cos A 0 |  
 | 0 0 1 |  
  
 These are the three basic rotation matrices used by OpenGL.  
   
Q28. How do I generate a rotation matrix in the X-axis?  
-------------------------------------------------------  
 Use the 4x4 matrix:  
  
 | 1 0 0 0 |  
 M = | 0 cos(A) -sin(A) 0 |  
 | 0 sin(A) cos(A) 0 |  
 | 0 0 0 1 |  
  
Q29. How do I generate a rotation matrix in the Y-axis?  
-------------------------------------------------------  
 Use the 4x4 matrix:  
  
 | cos(A) 0 sin(A) 0 |  
 M = | 0 1 0 0 |  
 | -sin(A) 0 cos(A) 0 |  
 | 0 0 0 1 |  
  
Q30. How do I generate a rotation matrix in the Z-axis?  
-------------------------------------------------------  
 Use the 4x4 matrix:  
  
 | cos(A) -sin(A) 0 0 |  
 M = | sin(A) cos(A) 0 0 |  
 | 0 0 1 0 |  
 | 0 0 0 1 |  
  
Q31. What are Euler angles?  
---------------------------  
 Euler angles are the name given to the set of rotation angles which  
 specify the rotation in each of the X, Y and Z rotation axii.  
 These are specfied in vector format eg. |x y z| and can be stored  
 as a VECTOR data structure.  
 For example, the set  
  
 | 0 0 0 |  
  
 will always generate the identity matrix.  
 Other angles are represented as follows:  
  
 | 90 0 0 | is a rotation of +90 degrees in the X-axis.  
 | 0 90 0 | is a rotation of +90 degrees in the Y-axis and  
 | 0 0 90 | is a rotation of +90 degrees in the Z-axis.  
  
 Euler angles can be represented using a single vector data structure.  
   
Q32. What are Yaw, Roll and Pitch?  
----------------------------------  
 Yaw, Roll and Pitch are aeronautical terms for rotation using the  
 Euclidean coordinate system (Euler angles), relative to the local  
 coordinate system of an aeroplane.  
 Imagine you are viewing an aeroplane from above and from directly behind.  
 The Z-axis is lined up with the tail and nose of the aeroplane.  
 The X-axis runs from the tip of the left wing to the tip of the right  
 wing.  
 The Y axis points straight up from the ground.  
 Pitch then becomes rotation in the X-axis, Yaw becomes rotation in the  
 Y-axis and Roll becomes rotation in the Z-axis.  
   
Q33. How do I combine rotation matrices?  
----------------------------------------  
 Rotation matrices are combined together using matrix multiplication.  
 As a result, the order of multiplication is very important.  
   
Q34. What is Gimbal lock?  
-------------------------  
 Gimbal lock is the name given to a problem that occurs with the use of  
 Euler angles. Because the final rotation matrix depends on the order  
 of multiplication, it is sometimes the case that the rotation in one  
 axis will be mapped onto another rotation axis.  
 Even worse, it may become impossible to rotate an object in a desired  
 axis. This is called Gimbal lock.  
 For example, assume that an object is being rotated in the order Z,Y,X  
 and that the rotation in the Y-axis is 90 degrees.  
 In this case, rotation in the Z-axis is performed first and therefore  
 correctly. The Y-axis is also rotated correctly. However, after  
 rotation in the Y axis, the X-axis is rotated onto the Z-axis.  
 Thus, any rotation in the X-axis actually rotates the object in the  
 Z-axis. Even worse, it becomes impossible to rotate the object in the X-axis.  
 A convenient solution to this problem is to make use of Quaternions.  
   
Q35. What is the correct way to combine rotation matrices?  
----------------------------------------------------------  
 Really, there is no "correct way" of combining rotation matrices.  
 However, in order to be able to predict the result of combining  
 matrices together, some organisation is required. This is also  
 necessary if a full 3D matrix library is to be built.  
 The simplest way to rotate an object is to multiply the matrices  
 using the order:  
  
 M = X.Y.Z  
  
 where M is the final rotation matrix, and X,Y,Z are the individual  
 rotation matrices. This defines a rotation in the X-axis (pitch) first,  
 followed by the Y-axis (yaw) and a final rotation in the Z-axis (roll).  
 However, whenever the view from the camera viewpoint is being  
 evaluated, then the order and signs of the rotation is reversed.  
 For example, if you are standing up, and turn to your left, everything  
 in your field of view appears to move towards the right.  
 However, someone else facing you will say that you turned towards their  
 right.  
 Thus the view from the camera is modelled using the order:  
  
 M = -Z.-Y.-X  
  
 This is the inverse (or transpose) of the rotation matrix generated  
 if the camera were being rendered as another object.  
   
Q36. How do I generate a rotation matrix from Euler angles?  
-----------------------------------------------------------  
 At first glance, the most obvious method to generate a rotation matrix  
 from a set of Euler angles is to generate each matrix individually and  
 multiply all three together ie.  
  
 m3\_rotx( mat\_x, vec -> angle\_x );  
 m3\_roty( mat\_y, vec -> angle\_y );  
 m3\_rotz( mat\_z, vec -> angle\_z );  
 m3\_mult( mat\_tmp, mat\_z, mat\_y );  
 m3\_mult( mat\_final, mat\_tmp, mat\_x );  
  
 This set of calls could be placed in a separate routine eg.  
  
 m3\_fromeuler( MATRIX \*mat\_final, VECTOR3 \*euler )  
  
 However, to perform this sequence of calls is very wasteful in terms  
 of processing time. Given that each 4x4 rotation matrix is guaranteed  
 to have 10 elements with value zero (0), 2 elements with value one (1)  
 and four others of arbitary value, over 75% of every matrix operation  
 is wasted. This does not include the set up and initialisation of each  
 matrix.  
 Altogether, over 75% of all matrix operations are spent processing  
 arithmetic expressions which lead to either zero or one.  
 A more efficient way must be found. Fortunately, there is another way  
 of determining the final resulting matrix.  
 If all three matrices are combined in algebraic format, the following  
 expression is defined:  
  
 M = X.Y.Z  
  
 where M is the final matrix,  
 X is the rotation matrix for the X-axis,  
 Y is the rotation matrix for the Y-axis,  
 Z is the rotation matrix for the Z-axis.  
  
 Expanding into rotation matrices in algebraic format gives:  
  
 | 1 0 0 |  
 X = | 0 A -B |  
 | 0 B A |  
  
 | C 0 D |  
 Y = | 0 1 0 |  
 | -D 0 C |  
  
 | E -F 0 |  
 Z = | F E 0 |  
 | 0 0 1 |  
  
 where A,B are the cosine and sine of the X-axis rotation axis,  
 C,D are the cosine and sine of the Y-axis rotation axis,  
 E,F are the cosine and sine of the Z-axis rotation axis.  
  
 Then the expression:  
  
 M = X.Y.Z  
  
 can be split into two matrix multiplications:  
  
 M' = X.Y  
 M = M'.Z  
  
 Evaluating M' first:  
  
 M' = X.Y  
  
 | 1 0 0 | | C 0 D |  
 M' = | 0 A -B | . | 0 1 0 |  
 | 0 B A | |-D 0 C |  
  
 | 1.C + 0.0 + 0.-D 1.0 + 0.1 + 0.0 1.D + 0.0 + 0.C |  
 M' = | 0.C + A.0 + -B.-D 0.0 + A.1 + -B.0 0.D + A.0 + -B.C |  
 | 0.C + B.0 + A.-D 0.0 + B.1 + A.0 0.D + B.0 + A.C |  
  
 Simplifying M' gives:  
  
 | C 0 D |  
 M' = | B.D A -B.C |  
 | -A.D B A.C |  
  
 Evaluating M gives:  
  
 M = M'.Z  
  
 | C 0 D | | E -F 0 |  
 M = | BD A -BC | . | F E 0 |  
 | -AD B AC | | 0 0 1 |  
  
 | C.E + 0.F + D.0 C.-F + 0.E + D.0 C.0 + 0.0 + D.1 |  
 M = | BD.E + A.F + -BC.0 BD.-F + A.E + -BC.0 BD.0 + A.0 + -BC.1 |  
 | -AD.E + B.F + AC.0 -AD.-F + B.E + AC.0 -AD.0 + 0.0 + AC.1 |  
  
 Simplifying M gives a 3x3 matrix:  
  
 | CE -CF D |  
 M = | BDE+AF -BDF+AE -BC |  
 | -ADE+BF ADF+BE AC |  
  
 This is the final rotation matrix. As a 4x4 matrix this is:  
  
 | CE -CF D 0 |  
 M = | BDE+AF -BDF+AE -BC 0 |  
 | -ADE+BF ADF+BE AC 0 |  
 | 0 0 0 1 |  
  
 The individual values of A,B,C,D,E and F are evaluated first. Also, the  
 values of BD and AD are also evaluated since they occur more than once.  
 Thus, the final algorithm is as follows:  
  
 A = cos(angle\_x);  
 B = sin(angle\_x);  
 C = cos(angle\_y);  
 D = sin(angle\_y);  
 E = cos(angle\_z);  
 F = sin(angle\_z);  
 AD = A \* D;  
 BD = B \* D;  
 mat[0] = C \* E;  
 mat[1] = -C \* F;  
 mat[2] = D;  
  
 mat[4] = BD \* E + A \* F;  
 mat[5] = -BD \* F + A \* E;  
 mat[6] = -B \* C;  
 mat[8] = -AD \* E + B \* F;  
 mat[9] = AD \* F + B \* E;  
 mat[10] = A \* C;  
 mat[3] = mat[7] = mat[11] = mat[12] = mat[13] = mat[14] = 0;  
 mat[15] = 1;  
  
 Using basic matrix calculations, the operation count would reach  
 128 multiplications, 96 additions and 80 assignments operations.  
 Using the optimised algorithm, only 12 multiplications, 6 subtractions  
 and 18 assignment operations are required.  
 So, it is obvious that by using the optimised algorithm, a performance  
 achievement of 1000% is achieved!  
  
Q37. How do I convert a rotation matrix to Euler angles?  
--------------------------------------------------------  
 This operation is the exact opposite to the one answered in the question  
 above. Given that the rotation matrix is:  
  
 | CE -CF D 0 |  
 M = | BDE+AF -BDF+AE -BC 0 |  
 | -ADE+BF ADF+BE AC 0 |  
 | 0 0 0 1 |  
  
 where A,B are the cosine and sine of the X-axis rotation axis,  
 C,D are the cosine and sine of the Y-axis rotation axis,  
 E,F are the cosine and sine of the Z-axis rotation axis.  
  
 Using the C data structure for a 4x4 matrix, the index values are  
 as follows:  
  
 | 0 1 2 3 |  
 M = | 4 5 6 7 |  
 | 8 9 10 11 |  
 | 12 13 14 15 |  
  
 From looking at these two tables, it can be see that array element  
 [2] has the value of D or sin(Y). Then the rotation angle in the  
 Y-axis can be calculated from a call to to the inverse-sine function.  
 Passing this value to the cosine function then gives the value of C.  
  
 If C is not zero, then the rotation angles in each of the X and Z  
 axii, can be derived from the terms on the third column and first row  
 respectively. These are as follows:  
  
 X-axis: M[6] = -BC  
 M[10] = AC  
  
 Z-axis: M[0] = CE  
 M[1] = -CF  
  
 The actual rotation angles can be derived by taking each pair of values  
 dividing by C and passing the results to the inverse tangent function.  
 If C is zero, then these calculations are not possible. In this case  
 the rotation angle in the Y-axis will be either -90 or +90. Thus D  
 will either have the value of 1 or -1.  
 In this case, Gimbal Lock will have occurred. Rotations in both the  
 X and Z axii will appear to be in the same axis. This can be seen  
 through the evaluation of the rotation axis.  
  
 | 0.E -0.F 1 0 |  
 M = | B.1.E+AF -B.1.F+AE -B.0 0 |  
 | -A.1.E+BF A.1.F+BE A.0 0 |  
 | 0 0 0 1 |  
  
 Multiplying out each term gives:  
  
 | 0 0 1 0 |  
 M = | BE+AF -BF+AE 0 0 |  
 | -AE+BF AF+BE 0 0 |  
 | 0 0 0 1 |  
  
 Rearranging these terms gives:  
  
 | 0 0 1 0 |  
 M = | BE+AF AE-BF 0 0 |  
 | -(AE-BF) BE+AF 0 0 |  
 | 0 0 0 1 |  
  
 Then it can be seen that the matrix is really of the form:  
  
 | 0 0 1 0 |  
 M = | V W 0 0 |  
 | -W V 0 0 |  
 | 0 0 0 1 |  
  
 Where V has the value of BE+AF and  
 W has the value of AE-BF  
  
 These two values can be considered to be the sin and cosine of a  
 single rotation axis.  
 The final algorithm is then as follows:  
  
 angle\_y = D = asin( mat[2]); /\* Calculate Y-axis angle \*/  
 C = cos( angle\_y );  
 angle\_y \*= RADIANS;  
 if ( fabs( C ) > 0.005 ) /\* Gimball lock? \*/  
 {  
 trx = mat[10] / C; /\* No, so get X-axis angle \*/  
 try = -mat[6] / C;  
 angle\_x = atan2( try, trx ) \* RADIANS;  
 trx = mat[0] / C; /\* Get Z-axis angle \*/  
 try = -mat[1] / C;  
 angle\_z = atan2( try, trx ) \* RADIANS;  
 }  
 else /\* Gimball lock has occurred \*/  
 {  
 angle\_x = 0; /\* Set X-axis angle to zero \*/  
 trx = mat[5]; /\* And calculate Z-axis angle \*/  
 try = mat[4];  
 angle\_z = atan2( try, trx ) \* RADIANS;  
 }  
  
 /\* return only positive angles in [0,360] \*/  
 if (angle\_x < 0) angle\_x += 360;  
 if (angle\_y < 0) angle\_y += 360;  
 if (angle\_z < 0) angle\_z += 360;  
 -----------------------------------  
  
Q38. How do I generate a rotation matrix for a selected axis and angle?  
-----------------------------------------------------------------------  
 The simplest way to generate this type of rotation matrix is through the  
 use of quaternion mathematics.  
 See question [[Q53](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q53): How do I convert a quaternion to a rotation matrix?]  
 for further details.  
 The following code snipped does most of the work (phi being the rotation  
 angle and (u,v,w) the rotation (axis) vector):  
  
 rcos = cos(phi);  
 rsin = sin(phi);  
 matrix[0][0] = rcos + u\*u\*(1-rcos);  
 matrix[1][0] = w \* rsin + v\*u\*(1-rcos);  
 matrix[2][0] = -v \* rsin + w\*u\*(1-rcos);  
 matrix[0][1] = -w \* rsin + u\*v\*(1-rcos);  
 matrix[1][1] = rcos + v\*v\*(1-rcos);  
 matrix[2][1] = u \* rsin + w\*v\*(1-rcos);  
 matrix[0][2] = v \* rsin + u\*w\*(1-rcos);  
 matrix[1][2] = -u \* rsin + v\*w\*(1-rcos);  
 matrix[2][2] = rcos + w\*w\*(1-rcos);  
  
 Don't forget to set the rest of the matrix to 0 (1 at [3][3]) if  
 you are using 4x4 matrices!  
  
Q39. How do I generate a rotation matrix to map one vector onto another?  
------------------------------------------------------------------------  
 When developing animation software, a common requirement is to find  
 a rotation matrix that will map one direction vector onto another.  
 This problem may be visualised by considering the two direction  
 vectors to be attached at their starting points. Then the entire  
 rotation space forms a unit sphere.  
 In theory, there are an infinite number of rotation axii and angles  
 that will map one vector onto the other. All of these axii lie on the  
 plane where all of the points are the exact same distance from both  
 vectors.  
 However, only one solution is of practical interest. This is the path  
 which covers the shortest angular distance between the two vectors.  
 The rotation axis to this path is calculated by taking the cross  
 product between the two vectors:  
  
 Vaxis = Vs x Vf  
  
 The rotation angle is calculated by taking the dot product between the  
 two vectors:  
  
 -1  
 Vangle = cos ( Vs . Vf )  
  
 One practical application of the solution to this problem is finding  
 the shortest flight path between two cities. In this case, each city  
 is represented as a direction vector generated from spherical  
 coordinates. Since planet Earth is spherical, the desired flight path  
 is the shortest angular rotation between the two cities.  
  
 Note: If Vs and Vf are colinear, the cross product returns (0,0,0).  
 You should test for that case and use any of the 90 degree rotations  
 of either Vs or Vf as a rotation axis, e.g. (y,z,x). If x==y==z, then  
 using one of x,y, or z from the second vector might help.  
 --> Is there a better way?  
  
 Having the rotation angle and vector, generating the corresponding matrix  
 is easy, see [[Q38](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q38): How do I generate a rotation matrix for  
 a selected axis and angle?] for details.  
  
 Don't let the spectre of Gimbal Lock fool you: Euler angles are still  
 a complete representation of any rotation in 3D space; it's just that  
 the actual Euler angles needed to achieve some particular desired  
 rotation may be rather unintuitive.  
   
Q40. How do I use matrices to convert one coordinate system to another?  
-----------------------------------------------------------------------  
 Similar to the previous problem, the requirement is to map one  
 coordinate system onto another. However, instead of just trying to  
 map one coordinate axis onto another, all three axii have to be matched.  
 Both coordinate systems are therefore represented as either 3x3 or  
 4x4 matrices.  
  
 The problem is therefore to find the rotation matrix that will map one  
 matrix onto another. This can be expressed mathematically:  
  
 Mfinal = Mrot . Morig  
  
 where Mfinal is the final coordinate system matrix,  
 Morig is the original coordinate system and  
 Mrot is the desired rotation matrix.  
  
 The goal is then to find the matrix Mrot. This can be achieved by  
 rearranging the equation to give:  
  
 -1  
 Mfinal . Morig = Mrot  
  
 -1  
 Mrot = Mfinal . Morig  
  
 Thus, the desired rotation matrix can be by calculatng the inverse of  
 the original coordinate system and multiplying it with the final  
 rotation matrix.  
  
 As a check, consider the cases when either the original or final  
 rotation matrices are the identity matrix. In each case, the rotation  
 matrix should match the final matrix and the inverse of the final  
 matrix respectively.  
  
 Once calculated, the rotation matrix may be converted into a  
 quaternion.  
  
Q41. What is a translation matrix?  
----------------------------------  
 A translation matrix is used to position an object within 3D space  
 without rotating in any way. Translation operations using matrix  
 multiplication can only be performed using 4x4 matrices.  
   
 If the translation is defined by the vector [X Y Z ], then the 4x4  
 matrix to implement translation is as follows:  
  
 | 1 0 0 X |  
 | |  
 | 0 1 0 Y |  
 M = | |  
 | 0 0 1 Z |  
 | |  
 | 0 0 0 1 |  
  
 If the vector is [0 0 0] then the vertex list will remain as before.  
   
Q42. What is a scaling matrix?  
------------------------------  
 A scaling matrix is used to enlarge or shrink the size of a 3D model.  
 If the scaling vector is [X Y Z] then the matrix to perform this is  
 as follows:  
  
 | X 0 0 0 |  
 | |  
 | 0 Y 0 0 |  
 M = | |  
 | 0 0 Z 0 |  
 | |  
 | 0 0 0 1 |  
  
 If the scaling vector is [1 1 1], then this generates the identity  
 matrix and vertex geometry will remain unchanged.  
   
Q43. What is a shearing matrix?  
-------------------------------  
 A shearing matrix is used to make a 3D model appear to slant sideways.  
 For example, "italic" text requires each character to slant towards the  
 right.  
 In three dimensions six possible shearing directions exist:  
  
 o shear X by Y  
 o shear X by Z  
 o shear Y by X  
 o shear Y by Z  
 o shear Z by X  
 o shear Z by Y  
  
 All six shearing directions may be combined into a single matrix:  
  
 | 1 Syx Szx 0 |  
 | |  
 | Sxy 1 Szy 0 |  
 M = | |  
 | Sxz Syz 1 0 |  
 | |  
 | 0 0 0 1 |  
 | |  
  
 Where Sij implements a shear of I by J  
 Thus, Sxy shears X by Y  
  
 In theory, rotation in three dimensions may be considered a combination  
 of six shearing directions.  
   
Q44. How do I perform linear interpolation between two matrices?  
----------------------------------------------------------------  
 Given two rotation matrices, the problem is to find a way of  
 determining intermediate positions specified by a parametric  
 variable t, where t ranges from 0.0 to 1.0  
 This can be achieved by converting the two matrices into either  
 Euler angles or Spherical rotation angles (via quaternions) and  
 a translation vector.  
  
 In either case, each matrix is converted into a pair of 3D vectors.  
 Interpolation between these two vectors can then be performed  
 through the use of the standard linear interpolation equation:  
  
 Vr = Va + t .(Vb - Va )  
  
 where Vr is the resulting vector  
 Va is the start position vector  
 Vb is the final position vector  
  
 This equation may be applied to both translation and rotation vectors.  
 Once determined, the resulting translation and rotation are then  
 converted back into the desired intermediate matrix.  
   
Q45. How do I perform cubic interpolation between four matrices?  
----------------------------------------------------------------  
 Given four rotation or translation matrices, the problem is to  
 find a way of determining intermediate positions specified by a  
 parametric variable t.  
  
 This can be achieved by making use of cubic interpolation. As with  
 linear interpolation, the four matrices are converted into their  
 corresponding translation and rotation vectors (Again, either Euler  
 angles or spherical rotation angles).  
  
 Each set of four vectors is then converted into a single geometry  
 vector G. Through the use of spline mathematics, this geometry vector  
 is converted into an interpolation matrix M.  
  
 If the geometry vector is defined as:  
  
 | x1 x2 x3 x4 |  
 G = | y1 y2 y3 y4 |  
 | z1 z2 z3 z4 |  
  
 Then multiplication by the base matrix:  
  
 | -4.5 9.0 -5.5 1.0 |  
 Mb = | 13.5 -22.5 9.0 0.0 |  
 | -13.5 18.0 -4.5 0.0 |  
 | 4.5 -4.5 1.0 0.0 |  
  
 will generate the 3x4 interpolation matrix Mi:  
  
 Mi = G .Mb  
  
 This can be implemented through a standard matrix-vector multiplication.  
 Interpolation can then be performed by the use of the parametric  
 variable t:  
  
 R = Mi . t  
  
 |t^3|  
 | xr | | A B C D | |t^2|  
 | yr | = | E F G H | . |t |  
 | zr | | I J K L | |1 |  
  
 The result vector can then be converted back into a rotation or  
 translation matrix.  
  
 It should be noted that the rotation paths that are generated may  
 occasionally become rather loopy. This is normal, as the algorithm  
 is trying to find the path with the least amount of rotation between  
 all four vectors.  
  
 Of the two methods, spherical rotation angles will usually be seen to  
 provide the cleanest interpolation paths for rotation.  
   
Q46. How can I render a matrix?  
-------------------------------  
 When using a graphics window for 3D animation, it is convenient to be  
 able to view a rotation matrix concurrently with the animation.  
 However, displaying a rotation matrix as an array of numeric values  
 does not provide a very meaningful context.  
 An alternative to rendering numeric data is to make use of graphical  
 display methods such as bar-graphs.  
  
 Much like a graphic equalizer on a stereo, a rotation matrix may be  
 displayed in a bar graph format. Each element of the rotation matrix  
 is rendered as an individual bar-graph in the range -1 to +1.  
 A 3x3 matrix would look like the following:  
  
 +--+ +--+ +--+  
 |##| | | | |  
 +--+ +--+ +--+  
 | | | | | |  
 +--+ +--+ +--+  
  
 +--+ +--+ +--+  
 | | |##| | |  
 +--+ +--+ +--+  
 | | | | | |  
 +--+ +--+ +--+  
  
 +--+ +--+ +--+  
 | | | | |##|  
 +--+ +--+ +--+  
 | | | | | |  
 +--+ +--+ +--+  
  
 In this case, the rotation matrix is the identity matrix, since each  
 element in the major diagonal is +1, and all others are zero.  
 For added visual clarity, parameters which are negative may shaded  
 in a different colour than those which are positive.  
   
QUATERNIONS  
===========  
  
Q47. What are quaternions?  
--------------------------  
 Quaternions extend the concept of rotation in three dimensions to  
 rotation in four dimensions. This avoids the problem of "gimbal-lock"  
 and allows for the implementation of smooth and continuous rotation.  
 In effect, they may be considered to add a additional rotation angle  
 to spherical coordinates ie. Longitude, Latitude and Rotation angles  
 A Quaternion is defined using four floating point values |x y z w|.  
 These are calculated from the combination of the three coordinates  
 of the rotation axis and the rotation angle.  
   
Q48. How do quaternions relate to 3D animation?  
-----------------------------------------------  
 As mentioned before, Euler angles have the disadvantage of being  
 susceptible to "Gimbal lock" where attempts to rotate an  
 object fail to appear as expected, due to the order in which the   
 rotations are performed.  
  
 Quaternions are a solution to this problem. Instead of rotating an  
 object through a series of successive rotations, quaternions allow  
 the programmer to rotate an object through an arbitary rotation axis  
 and angle.   
  
 The rotation is still performed using matrix mathematics. However,  
 instead of multiplying matrices together, quaternions representing  
 the axii of rotation are multiplied together. The final resulting   
 quaternion is then converted to the desired rotation matrix.  
 Because the rotation axis is specifed as a unit direction vector,  
 it may also be calculated through vector mathematics or from   
 spherical coordinates ie (longitude/latitude).  
 Quaternions offer another advantage in that they be interpolated.  
 This allows for smooth and predictable rotation effects.  
   
Q49. How do I calculate the conjugate of a quaternion?  
------------------------------------------------------  
 This can be achieved by reversing the polarity (or negating) the  
 vector part of the quaternion, ie:  
  
 Qr = ( Qr.scalar, -Qr.vector )  
 ----------------------------------------------------------------  
 quaternion\_conjugate( QUAT \*qr, QUAT \*qa )  
 {  
 qr -> qw = qa -> qw;  
 qr -> qx = -qa -> qx;  
 qr -> qy = -qa -> qy;  
 qr -> qz = -qa -> qz;  
 }  
  
Q50. How do I calculate the inverse of a quaternion?  
----------------------------------------------------  
 This is equivalent to calculating the conjugate of the quaternion,  
 if the quaternion is normalized (or a unit quaternion).  
 In all other cases, the magnitude of the inverse is 1/|q|.  
 See [Q49](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q49): How do I calculate the conjugate of a quaternion?  
  
Q51. How do I calculate the magnitude of a quaternion?  
------------------------------------------------------  
 The magnitude of a quaternion is calculated by multiplying the  
 quaternion with its conjugate ie:  
 ------------  
 / --  
 |Qr| = \/ Qr.Qr  
 This can be implemented as the following code sequence:  
 -------------------------------------------------------------------  
 QFLOAT quaternion\_magnitude( QUAT \*qa )  
 {  
 return( sqrt(qa->qw\*qa->qw+  
 qa->qx\*qa->qx+ qa->qy\*qa->qy+qa->qz\*qa->qz) )  
 }  
  
Q52. How do I normalise a quaternion?  
-------------------------------------  
 A quaternion can be normalised in a way similar to vectors. The  
 magnitude of the quaternion is calculated first. Then both the  
 scalar and vector part of the quaternion are divided by this value.  
 A unit quaternion will always have a magnitude of 1.0  
  
Q53. How do I multiply two quaternions together?  
------------------------------------------------  
 Given two quaternions Q1 and Q2, the goal is to calculate the  
 combined rotation Qr:  
  
 Qr = Q1.Q2  
  
 This is achieved through the expression:  
  
 Qr = Q1.Q2 = ( w1.w2 - v1.v2, w1.v2 + w2.v1 + v1 x v2 )  
  
 where v1 = (x,y,z) of Q1  
 w1 = (w) of Q1  
 v2 = (x,y,z) of Q2  
 w2 = (w) of Q2  
  
 and both . and x are the standard vector dot and cross products.  
 This can be implemented using the following code segment:  
 ---------------------------------------------------  
 quaternion\_multiply( QUAT \*qr, QUAT \*qa, QUAT \*qb )  
 {  
 qr.scalar = qa->scalar \* qb->scalar - v3\_dot( &qa->vector, &qb->vector );  
 v3\_cross( &va, &qa->vector, &qb->vector );  
 v3\_scalef( &vb, &qa->vector, &qb->scalar );  
 v3\_scalef( &vc, &qb->vector, &qa->scalar );  
 v3\_add( &va, &va, &vb );  
 v3\_add( &qr->vector, &va, &vc );  
 quaternion\_normalise( qr );  
 }  
  
 ---------------------------------------------------  
 An optimization can also be made by rearranging to  
 w = w1w2 - x1x2 - y1y2 - z1z2  
 x = w1x2 + x1w2 + y1z2 - z1y2  
 y = w1y2 + y1w2 + z1x2 - x1z2  
 z = w1z2 + z1w2 + x1y2 - y1x2  
  
Q54. How do I convert a quaternion to a rotation matrix?  
--------------------------------------------------------  
 Assuming that a quaternion has been created in the form:  
  
 Q = |X Y Z W|  
  
 Then the quaternion can then be converted into a 4x4 rotation  
 matrix using the following expression (Warning: you might have to  
 transpose this matrix if you (do not) follow the OpenGL order!):  
  
 ¦ 2 2 ¦  
 ¦ 1 - (2Y + 2Z ) 2XY + 2ZW 2XZ - 2YW ¦  
 ¦ ¦  
 ¦ 2 2 ¦  
 M = ¦ 2XY - 2ZW 1 - (2X + 2Z ) 2YZ + 2XW ¦  
 ¦ ¦  
 ¦ 2 2 ¦  
 ¦ 2XZ + 2YW 2YZ - 2XW 1 - (2X + 2Y ) ¦  
 ¦ ¦  
  
 If a 4x4 matrix is required, then the bottom row and right-most column  
 may be added.  
 The matrix may be generated using the following expression:  
  
 xx = X \* X;  
 xy = X \* Y;  
 xz = X \* Z;  
 xw = X \* W;  
 yy = Y \* Y;  
 yz = Y \* Z;  
 yw = Y \* W;  
 zz = Z \* Z;  
 zw = Z \* W;  
 mat[0] = 1 - 2 \* ( yy + zz );  
 mat[1] = 2 \* ( xy - zw );  
 mat[2] = 2 \* ( xz + yw );  
 mat[4] = 2 \* ( xy + zw );  
 mat[5] = 1 - 2 \* ( xx + zz );  
 mat[6] = 2 \* ( yz - xw );  
 mat[8] = 2 \* ( xz - yw );  
 mat[9] = 2 \* ( yz + xw );  
 mat[10] = 1 - 2 \* ( xx + yy );  
 mat[3] = mat[7] = mat[11] = mat[12] = mat[13] = mat[14] = 0;  
 mat[15] = 1;  
  
 The resulting matrix uses the following positions:  
  
 ¦ mat[0] mat[4] mat[ 8] mat[12] ¦  
 M = ¦ mat[1] mat[5] mat[ 9] mat[13] ¦  
 ¦ mat[2] mat[6] mat[10] mat[14] ¦  
 ¦ mat[3] mat[7] mat[11] mat[15] ¦  
   
Q55. How do I convert a rotation matrix to a quaternion?  
--------------------------------------------------------  
 A rotation may be converted back to a quaternion through the use of  
 the following algorithm:  
 The process is performed in the following stages, which are as follows:  
  
 Calculate the trace of the matrix T from the equation:  
  
 2 2 2  
 T = 4 - 4x - 4y - 4z  
  
 2 2 2  
 = 4( 1 -x - y - z )  
  
 = 1 + mat[0] + mat[5] + mat[10]  
  
  
 If the trace of the matrix is greater than zero, then  
 perform an "instant" calculation.  
 Important note wrt. rouning errors:  
  
 Test if ( T > 0.00000001 ) to avoid large distortions!  
  
 S = sqrt(T) \* 2;  
 X = ( mat[9] - mat[6] ) / S;  
 Y = ( mat[2] - mat[8] ) / S;  
 Z = ( mat[4] - mat[1] ) / S;  
 W = 0.25 \* S;  
  
 If the trace of the matrix is equal to zero then identify  
 which major diagonal element has the greatest value.  
 Depending on this, calculate the following:  
  
 if ( mat[0] > mat[5] && mat[0] > mat[10] ) { // Column 0:   
 S = sqrt( 1.0 + mat[0] - mat[5] - mat[10] ) \* 2;  
 X = 0.25 \* S;  
 Y = (mat[4] + mat[1] ) / S;  
 Z = (mat[2] + mat[8] ) / S;  
 W = (mat[9] - mat[6] ) / S;  
 } else if ( mat[5] > mat[10] ) { // Column 1:   
 S = sqrt( 1.0 + mat[5] - mat[0] - mat[10] ) \* 2;  
 X = (mat[4] + mat[1] ) / S;  
 Y = 0.25 \* S;  
 Z = (mat[9] + mat[6] ) / S;  
 W = (mat[2] - mat[8] ) / S;  
 } else { // Column 2:  
 S = sqrt( 1.0 + mat[10] - mat[0] - mat[5] ) \* 2;  
 X = (mat[2] + mat[8] ) / S;  
 Y = (mat[9] + mat[6] ) / S;  
 Z = 0.25 \* S;  
 W = (mat[4] - mat[1] ) / S;  
 }  
  
 The quaternion is then defined as:  
 Q = | X Y Z W |  
  
Q56. How do I convert a rotation axis and angle to a quaternion?  
----------------------------------------------------------------  
 Given a rotation axis and angle, the following  
 algorithm may be used to generate a quaternion:  
  
 vector\_normalize(axis);  
 sin\_a = sin( angle / 2 );  
 cos\_a = cos( angle / 2 );  
 X = axis -> x \* sin\_a;  
 Y = axis -> y \* sin\_a;  
 Z = axis -> z \* sin\_a;  
 W = cos\_a;  
  
 It is necessary to normalise the quaternion in case any values are   
 very close to zero.  
   
Q57. How do I convert a quaternion to a rotation axis and angle?  
----------------------------------------------------------------  
 A quaternion can be converted back to a rotation axis and angle  
 using the following algorithm:  
   
 quaternion\_normalise( |X,Y,Z,W| );  
 cos\_a = W;  
 angle = acos( cos\_a ) \* 2;  
 sin\_a = sqrt( 1.0 - cos\_a \* cos\_a );  
 if ( fabs( sin\_a ) < 0.0005 ) sin\_a = 1;  
 axis -> x = X / sin\_a;  
 axis -> y = Y / sin\_a;  
 axis -> z = Z / sin\_a;  
  
Q58. How do I convert spherical rotation angles to a quaternion?  
----------------------------------------------------------------  
 A rotation axis itself may be defined using spherical coordinates  
 (latitude and longitude) and a rotation angle  
 In this case, the quaternion can be calculated as follows:  
  
 sin\_a = sin( angle / 2 )  
 cos\_a = cos( angle / 2 )  
 sin\_lat = sin( latitude )  
 cos\_lat = cos( latitude )  
 sin\_long = sin( longitude )  
 cos\_long = cos( longitude )  
 X = sin\_a \* cos\_lat \* sin\_long  
 Y = sin\_a \* sin\_lat  
 Z = sin\_a \* sin\_lat \* cos\_long  
 W = cos\_a  
  
 WARNING: There might be a problem in this code.  
 An alternative is the code snipped given in [[Q60](http://web.archive.org/web/20041029003853/http://web.archive.org/web/20041029003853/http://www.j3d.org/matrix_faq/matrfaq_latest.html#Q60): How  
 do I convert Euler rotation angles to a quaternion?"].  
   
Q59. How do I convert a quaternion to spherical rotation angles?  
----------------------------------------------------------------  
 A quaternion can be converted to spherical coordinates by extending  
 the conversion process:  
  
 cos\_a = W;  
 sin\_a = sqrt( 1.0 - cos\_a \* cos\_a );  
 angle = acos( cos\_a ) \* 2;  
 if ( fabs( sin\_angle ) < 0.0005 ) sin\_a = 1;  
 tx = X / sin\_a;  
 ty = Y / sin\_a;  
 tz = Z / sin\_a;  
 latitude = -asin( ty );  
 if ( tx \* tx + tz \* tz < 0.0005 )  
 longitude = 0;  
 else  
 longitude = atan2( tx, tz );  
 if ( longitude < 0 )  
 longitude += 360.0;  
  
 WARNING: In this code might still be a problem.  
 Please let me know what it is and how to fix this.  
   
Q60. How do I convert Euler rotation angles to a quaternion?  
-------------------------------------------------------------------  
 Converting Euler rotation angles to quaterions can be achieved through  
 the use of quaternion multiplication. Each rotation angle is converted   
 to an axis-angle pair, with the axis corresponding to one of the   
 Euclidean axii. The axis-angle pairs are converted to quaternions and   
 multiplied together. The final quaternion is the desired result.  
   
  
 The following code segment demonstrates this:  
  
 quaternion\_from\_euler( QUATERNION \*q, VFLOAT ax, VFLOAT ay, VFLOAT az )  
 {  
 VECTOR3 vx = { 1, 0, 0 }, vy = { 0, 1, 0 }, vz = { 0, 0, 1 };  
 QUATERNION qx, qy, qz, qt;  
 quaternion\_from\_axisangle( qx, &vx, rx );  
 quaternion\_from\_axisangle( qy, &vy, ry );  
 quaternion\_from\_axisangle( qz, &vz, rz );  
 quaternion\_multiply( &qt, &qx, &qy );  
 quaternion\_multiply( &q, &qt, &qz );  
 }  
  
 The following more or less comes from:  
 http://vered.rose.utoronto.ca/people/david\_dir/GEMS/GEMS.html  
  
 //Pitch->X axis, Yaw->Y axis, Roll->Z axis  
 Quaternion::Quaternion(float fPitch, float fYaw, float fRoll)  
 {  
 const float fSinPitch(sin(fPitch\*0.5F));  
 const float fCosPitch(cos(fPitch\*0.5F));  
 const float fSinYaw(sin(fYaw\*0.5F));  
 const float fCosYaw(cos(fYaw\*0.5F));  
 const float fSinRoll(sin(fRoll\*0.5F));  
 const float fCosRoll(cos(fRoll\*0.5F));  
 const float fCosPitchCosYaw(fCosPitch\*fCosYaw);  
 const float fSinPitchSinYaw(fSinPitch\*fSinYaw);  
 X = fSinRoll \* fCosPitchCosYaw - fCosRoll \* fSinPitchSinYaw;  
 Y = fCosRoll \* fSinPitch \* fCosYaw + fSinRoll \* fCosPitch \* fSinYaw;  
 Z = fCosRoll \* fCosPitch \* fSinYaw - fSinRoll \* fSinPitch \* fCosYaw;  
 W = fCosRoll \* fCosPitchCosYaw + fSinRoll \* fSinPitchSinYaw;  
 }  
  
 The following might also work:   
 QUATERNION qx = { cos(pitch/2f), sin(pitch/2f), 0, 0 }  
 QUATERNION qy = { cos(yaw/2f), 0, sin(yaw/2f), 0 }  
 QUATERNION qz = { cos(roll/2f), 0, 0, sin(roll/2f) }  
 quaternion\_multiply( &qt, &qx, &qy );  
 quaternion\_multiply( &q, &qt, &qz );  
  
 In Java looks like this:  
  
 public static void setQ( Quat4f q, float pitch, float yaw, float roll )  
 {  
 Quat4f qx = new Quat4f((float) Math.cos(pitch/2f),   
 (float) Math.sin(pitch/2f), 0, 0);  
 Quat4f qy = new Quat4f((float) Math.cos(yaw/2f), 0,   
 (float) Math.sin(yaw/2f),0);  
 Quat4f qz = new Quat4f((float) Math.cos(roll/2f), 0, 0,   
 (float) Math.sin(roll/2f) );  
 Quat4f qt = new Quat4f();  
 qt.set( qx );  
 qt.mul( qy );  
 qt.mul( qz );  
 q.set( qt );  
 }  
  
Q61. How do I use quaternions to perform linear interpolation between matrices?  
-------------------------------------------------------------------------------  
 For many animation applications, it is necessary to interpolate  
 between two rotation positions of a given object. These positions may  
 have been specified using keyframe animation or inverse kinematics.  
 Using either method, at least two rotation matrices must be known, and  
 the desired goal is to interpolate between them. The two matrices are  
 referred to as the starting and finish matrices( MS and MF).  
 Using linear interpolation, the interpolated rotation matrix is  
 generated using a blending equation with the parameter T, which  
 ranges from 0.0 to 1.0.  
  
 At T=0, the interpolated matrix is equal to the starting matrix.  
 At T=1, the interpolated matrix is equal to the finishing matrix.  
 Then the interpolated rotation matrix (MI) is specified as:  
  
 MI = F( MS, MF, T )  
  
 where F is a blending function.  
 The first stage in interpolating between the two matrices is to  
 determine the rotation matrix that will convert MS to MF.  
 This is achieved using the following expression:  
  
 -1  
 T = Ms . Mf  
  
 where Ms is the start matrix,  
 Mf is the finish matrix,  
 and T is the intermediate matrix.  
  
 The next stage is to convert this matrix into a rotation axis and  
 angle. This is achieved by converting the matrix into a quaternion  
 and finally into the required rotation axis and angle.  
 In order to generate the interpolated rotation matrix, it is only  
 necessary to scale the rotation angle and convert this angle and  
 the rotation axis back into a rotation matrix.  
  
 Using a 4x4 matrix library, this is as follows:  
  
 m4\_transpose( mt, ms ); /\* Inverse \*/  
 m4\_mult( ms, mt, mb ); /\* Rotation matrix \*/  
 m4\_to\_axisangle( ms, axis, angle ); /\* Rotation axis/angle \*/  
 for ( t = 0; t < 1.0; t += 0.05 )  
 {  
 m4\_from\_axisangle( mi, axis, angle \* t ); /\* Final interpolation \*/  
 ... whatever ...  
 }  
  
 where t is the interpolation factor ranging from 0.0 to 1.0  
  
Q62. How do I use quaternions to perform cubic interpolation between matrices?  
------------------------------------------------------------------------------  
 For some applications, it may not be convenient or possible to use linear  
 interpolation for animation purposes. In this case, cubic interpolation  
 is another alternative.  
 In order to use cubic interpolation, at least four rotation matrices must  
 be known.  
  
 Each of these is then converted into a set of spherical rotations  
 via quaternions and spherical rotation angles (ie. longitude, latitude  
 and rotation angle).  
  
 These are then multiplied with the base matrix for a Cardinal spline  
 curve. This interpolation matrix can then be used to determine the  
 intermediate spherical rotation angles.  
 Once the interpolated coordinates are known (latitude, longitude and  
 rotation angle), the interpolated rotation matrix can then be generated  
 through the conversion to quaternions.  
  
 Using a 4x4 matrix library, the algorithm is as follows:  
  
 for ( n = 0; n < 4; n++ )  
 m4\_to\_spherical( mat[n], &v\_sph[n] ); /\* Spherical coordinates \*/  
 m4\_multspline( m\_cardinal, v\_sph, v\_interp ); /\* Interpolation vector \*/  
 ...  
 v3\_cubic( v\_pos, v\_interp, t ); /\* Interpolation \*/  
 m4\_from\_spherical( m\_rot, v\_pos ); /\* Back to a matrix \*/  
  
Q63. How do I use quaternions to rotate a vector?  
------------------------------------------------------------------------------  
 A rather elegant way to rotate a vector using a quaternion directly  
 is the following (qr being the rotation quaternion):  
  
 -1  
 v' = qr \* v \* qr  
  
 This can easily be realised and is most likely faster then the transformation  
 using a rotation matrix.