revolves around the sun and rotates around its center.

Therefore, the whole robot arm will move together as the planet base of the robot should be fixed on the surface of the planet.

The separate function (as illustrated by Redbook Example 3-4). The Redbook CH. 3 solar system by putting the robot drawing code in a

3. (30 pts.) Put the moving or movable robot on the planet of the

or animated control, your choice.

(30 pts.) Make those fingers move independently (via keyboard, mouse)

rigidly attached; same for the fingers when the elbow moves.

course, when the shoulder moves, the forearm and fingers move as if

fingers must be attached and pivot at the end of the forearm. Of

1. (30 pts.) Add fingers (at least two) to the Redbook CH. 3 robot. The

Project 3: Due Wed. Oct 26. (2 weeks from today)

dot/cross products.

HW3: 2-d rectangular transformations, practice with angles in space, and

and Modelling)

Readings: Hughes CH. 5, start CH. 6 and 7; read Redbook CH. 3 Viewing
current assignments:

University at Albany Computer Science Dept.
4. (10 pts) Implementing a much better visualization by going beyond where frame objects and using a better viewing setup.

New study: HB 5-11 Three-dimensional rotations.
5. Apply the inverse of the transformation of step 1.

4. Apply the inverse of the transformation of step 2. (That inverse is
gotten by transposing the matrix because the inverse of a rotation
matrix is always the transpose, unlike ordinary matrices).

3. Apply the rotation by angle 0 around the z-axis. (That was your
homework.)

2. Rotate this with the rotation that moves THE

\[
(1z - z', f'i, x - x) = (z', f'i, x)
\]

1. Translate (z', f'i, x) with the translation that moves that

\[
(1z') = \begin{pmatrix} z' \\ f'i \\ x \end{pmatrix}
\]

\[
\text{We form R by multiplying 3 matrices corresponding to the following steps:}
\]

the line \( P \)

point \( P \)

rotate \( R \) by angle 0 around

rotate \( \Theta \) by angle

and \( \Theta \) Goal:

Calculate the 4x4 matrix \( R \) so \( R(x,y,z) = P \)

Problem: Given: \( P \)

\[
\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]

University at Albany Computer Science Dept.
Of course real CS people would do it more efficiently (how?)

\[ \frac{z(Iz - z\hat{z}) + z(\partial I - z\hat{z}) + z(Ix - z\hat{x})}{Ix - z\hat{x}} = a \]

etc.

In which we want to rotate, a for example is calculated from the given by

The components \((a', q', e)\) are the direction cosines of the direction around

\[ (Iz - z\hat{z}, I\hat{h} - z\hat{h}, Ix - z\hat{x}) = \overrightarrow{d} - z\overrightarrow{d} = \Lambda \]

Guide to HB pages 268-269:

\[ \Theta (z, z, z, x) = z(\overrightarrow{d}, (Iz, I\hat{h}, Ix)) = \overrightarrow{d} \]

The GIVEN numbers are:
calculated from the first 6 given.

Let's see how to find \( R_x(\alpha) \) in terms of the \( \mathbf{R}_y(\beta) \). \( \mathbf{R}_x(\gamma) \) we had

\[ \mathbf{R}_x(\gamma) \]

vector in the \( z \) direction.

\[ \mathbf{R}_x(\gamma) \]

rotate \( \mathbf{n} \) around the \( y \) axis so \( \mathbf{n} \) is transformed into the unit

\[ \mathbf{R}_x(\gamma) \]

the \( xx \) plane. \( \mathbf{R}_x(\gamma) \) is this vector \( \mathbf{n} \)

two rotation matrices

\[ \mathbf{R}_x(\gamma) \]

will form the matrix for step 2 (rotate \( \mathbf{n} \) into the \( z \) axis) by multiplying

\[ \mathbf{R}_x(\gamma) \]}
To solve the problem, analyze situation in the yz plane only:

Imagining the shadow.

Look at this scene.

Imagine the shadow's tip has coordinates \((0, b, c)\).

The shadow's plane is the yz plane.

The unit coordinate vector of \(\mathbf{u}'\) is in the x=0 plane! 

Well, here is the rotation around the x-axis that rotates \(\mathbf{u} \rightarrow \mathbf{u}'\) (like \(a\)).

Since a rotation around the x-axis doesn't change x-coordinates (like \(a\)), what will rotate \((a', q', c)\) into \((a, 0, 0)\) something? Since a rotation around the x-axis, \(\mathbf{u}'\) will rotate \(\mathbf{u}\) into the z-axis. This will rotate \((a, q', c)\) into \((0, q', c)\) into the x-axis.

The unit coordinate vector is \((0, b, c)\).
\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
= (\alpha)^x \mathbf{R}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
= (\alpha)^x \mathbf{R}
\]

So \( (p/\alpha, p/q, 0) \) is the desired rotation matrix for step 2 in the transpose of this matrix.

Let \((0, p/\alpha, p/q, 0)\) be a unit vector in the \(yz\) plane.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & \frac{p}{q} & \frac{p}{q} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
= (\alpha-)^x \mathbf{R}
\]

\[
\mathbf{z}^2 + \mathbf{zq} = p
\]

University at Albany Computer Science Dept.
Length, $p$, along the $z$ axis: $(0', 0', p)$.

The rotation rotates this $(0', q', c)$ into the vector with the same plane. The rotation rotates this $(0', q', c)$, "shadows" into the $yz$ plane, $(p, 0', q', c)$. $p$ is the length of the projection $(0', q', c) = n$ into $R_x^x (v) (a)$ doesn't change the $x$ component. It rotates $R$. University at Albany Computer Science Dept.
\[ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & p & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & p & 0 & p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & p & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & p & 0 & p \end{bmatrix} = (g')^r \]

\[ u = (a, b, c) \]

\[ u' = (0, b, c) \]

\[ u' \text{ is IN THE } x=0 \text{ PLANE!} \]

\[ y=b \]

\[ d = \text{length of } u'. \]

\[ x=a \text{ plane.} \]

\[ \text{Our first rotation rotated } u \text{ into } u'' \text{ and } u' \text{ into } (0,0,d). \]

\[ u''= (a,0,d) \]

\[ \text{We finish with } R_Y(\beta) \]

\[ \text{which rotates } u'' = (a,0,d) \text{ into the } \]

\[ z \text{ unit vector } (0,0,1). \]

\[ (0,0,d) \]

\[ s \]

\[ (\beta) \]

\[ (\beta') \]

\[ (0,0,d) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,1) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]

\[ \text{— - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -} \]

\[ (0,0,0) \]
The problem is solved by

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & \cos \Theta & \sin \Theta \\
0 & 0 & \sin \Theta & -\cos \Theta
\end{bmatrix}
= \begin{pmatrix} \Theta \end{pmatrix}
\]

So, with
{ /* use stuff below */ }
{
{ /* just use \( z^\theta \) */ }

\( \mathbf{R} = \Theta = 0 \). If \( \theta = 0 \), we needn't work hard. So, first test rotation.

METHOD 2: Remember \( \mathbf{u} \). The unit vector pointing along the axis of
multiplications actually have to be computed when \( \mathbf{R} = \Theta \) is computed.

Only \( a, q, c \), \( p, q \), \( p/q \), \( p/q^2 \), and \( \sin \Theta, \cos \Theta \), and the matrix \( \mathbf{R} \) are necessary, and would be a waste of time.

Although HB explains the rotation matrices with angles \( \theta \) and \( \phi \), numerically computing \( \alpha \), \( \beta \), and their sines and cosines,
right-handed coordinate system.

\[ z_n \times n, x_n \] was chosen to make \[ n \times n \] rather than \[ z_n \times n \] because these two vectors have unit length and are perpendicular to each other (so the sine of the angle between them is \( \pm 1 \)).

This is guaranteed to be unit length and perpendicular to both \( z_n \) and \( n \),

\[ z_n \times n = x_n \]

Next, calculate the magnitude of \( z_n \times n \) by the geometric test above.

\[ |z_n \times n| = n \times n \]

This is guaranteed to be non-zero. The case when \( z_n \times n \) has the same direction as \( n \) was eliminated by the test above.

\[ \frac{|z_n \times n|}{z_n \times n} = n \]

Calculate: \(\tilde{z}_n \) by \( n \), \(\tilde{q}_n \) by \( q \), and \( q \) if both or at least one is non-zero.

So, \( n \) is
And the inverse of this matrix is its transpose.

So the inverse of this matrix will rotate our desired axis direction.

Vector $n$ into the $z$-axis.

Rotes the original axes into orthogonal axes in which $z$ is rotated into

$$\begin{bmatrix} z_n & h_n & x_n \end{bmatrix}$$

Now the matrix:
METHOD 4: Quaternions: Cool for continuous animated rotation.

METHOD 3: Algebraic work on the matrices—no visualization needed.

(Professor must do homework.)

(future handout).
corner or endpoint vertices. When the object is rendered, the fill or edge colors are interpolated from vertices. You can change the current color between calls to

GL_VERTEX* (...);

For example, COLOR: Current color assigns a color to all subsequent previous commands.

The effect of each command is determined by modifications done by

Each command modifies part of the state:

OpenGL API style:
The state also includes 3 matrix stacks.

mapping

OpenGL state includes 3 matrices: modelview, projection and texture
Now the MODELVIEW matrix equals I.

```
GLMatrixMode(GL_MODELVIEW);
GLLoadIdentity();
```

Subsequent matrix ops apply to the MODELVIEW matrix.

GLMatrixMode(GL_MODELVIEW);

To make the current modelview matrix be the identity matrix:
stage at each clock step, stages all operate simultaneously; each command progresses from stage to stage in a series of separate

Modern graphics (and GPU) hardware is *pipelined*; a series of separate

modern graphics card hardware.

OpenGL’s access to its matrices is restricted because OpenGL drivers
accept them.

So, it’s your job to create modeling and viewing transformations to create

*both* modeling or viewing.

The OpenGL *modelview* matrix is called the modelview matrix because

Confused?

related in OpenGL and are in fact combined into a single modelview

Redbook p.104: "Viewing and modelview transformations are intricately
coordinates.

to given rectangle in the screen or graphics window, expressed in pixel
coordinates [0,1] or [0,1,1] coordinates maps the 2-d to 0,1

6.  **Viewport Transformation** (outside the view volume)

primatives outside the view volume
coordinates within the view volume to a

5.  **Normalization Transformation and Clipping** (normalizes)

projection when z coordinate is ignored.

4.  **Projection Transformation** (produces a perspective or other

sometimes called viewing coordinates)

3.  **Viewing Transformation** which moves the world in front of the

2.  **Modeling Transformation** to produce an instance (in the world)

1.  Import of Master or template model

Study HB 6-1 to 6-4, 7-1, 7-2, 7-3, 2 and 3-d Rendering

University at Albany Computer Science Dept.

18
matrix that translates (0, 0, 0) to (20.0, 30.0, 40.0).

\[ mw = mw \cdot \begin{bmatrix} 1 \end{bmatrix} \]

The given matrix on the right

\[ \text{multimatrix} \]

\[ \text{multimatrix} \]

\[ \text{left} \]

\[ \lambda = m \cdot \lambda \]

by

or indirectly via \text{GLUT} models \text{like \GLUTsphere} (\cdot \cdot \cdot)\]
don't worry-F3u makes it easy to code viewing transformations
coordinates describe fixed objects.
Think: Transformations CHANGE the SYSTEM by which numeric
draws an image of the WORLD as if the camera were centered at point
__draw-world()__;

\text{ETranslate}((-100,-100,0)\right)

Think: Transformations MOVE or CHANGE OBJECTS all described

\text{draw-wheel-and-bolts()}

ETranslate((20,0,30,0,40,0)\right)

(1) Using the translation (20,30,40) translation for Modeling: (See Example 3-4 in

the RedBook)

\text{ETranslate}((-100,100,0)\right)

Think: Transformations CHANGE the SYSTEM by which numeric
draws an image of the WORLD as if the camera were centered at point
\text{draw-the-world()};

using a fixed coordinate system.

Think: Transformations MOVE or CHANGE OBJECTS all described
REMEmber the OpenGL **right matrix multiply** rule:

glTranslatef(20.0,30.0,40.0);
glRotatef(45.0, 1.0, 0.0, 0.0);
draw_wheel_and_bolts();

draws wheel and bolts AFTER

1. FIRST, rotating the model around $x$ axis by $45^\circ$,

2. and SECOND, translating (the rotated model) so its origin is at $(20,30,40)$. 
// RESTORE saved MY matrix.

draw_model();

// uses MODELING coordinates.

rotate_model();

// where I want my rotated model to go.

translater();

// SAVE current MY matrix.

pushmatrix();

// FIRST, initialize the modelview matrix with the VIEWING

// SECOND, use the modelview matrix stack to save the current matrix when

// you temporarily want a MODELING transformation to apply to

// your own model-drawing functions.

// SEE matrix, get models, or your own model-drawing functions!
revolves around the sun and rotates around its center. Therefore, the whole robot arm will move together as the planet base of the robot should be fixed on the surface of the planet. The separate function (as illustrated by Redbook Example 3-4). The Redbook Ch. 3 solar system by putting the robot drawing code in a

3. (30 pts.) Put the moving or movable robot on the planet of the

or animated control, your choice.

2. (30 pts.) Make those fingers move independently (via keyboard, mouse

rigidly attached; same for the fingers when the elbow moves.

in course, when the shoulder moves, the forearm and fingers move as if

fingers must be attached and pivot at the end of the forearm. Of

(1 (30 pts.) Add fingers (at least two) to the RedBook Ch. 3 robot. The

Project 3: Due Wed. Oct 26. (2 weeks from today)

dot/cross products.

HW3: 2-d rectangular transformations, practice with angles in space, and

and Modeling)

Readings: Hinch Ch. 5, start Ch. 6 and 7; read Redbook Ch. 3 (Viewing

Current Assignments:

University at Albany Computer Science Dept.

23
wire frame objects and using a better viewing setup.

4. (10 pts) Implementing a much better visualization by going beyond
In anticlockwise order of drawing the upper arm as a 2 x 0.4 x 1 rectangular solid, coordinate system.

cube parallel to the axes and CENTERED at the origin of OpenGL's world.

For example, robot uses glLoadIdentity(1,0) which draws a 1 x 1 x 1 scale within your scene.

model (coordinate) model, it will be drawn in your desired location and orientation.

COORDINATES so when the model is defined using its OWN (local) COORDINATES

1. The current model transformation applies to the vertices.

2. The current model transformation transforms.

The two ways to understand model transformation:

so the pattern for model transformation usage is

25 University at Albany Computer Science Dept.
getPushMatrixMatrix();

getPushMatrixMatrix();

upper arm.  

// future objects relative to the rotated
old system now, because we want to position
the rotated upper arm. Let's call this the
system now aligned with its x-axis alone

// Alternative view: The current coordinate

Here is a case of (Tiny)(Rot)(T)

// of our solid when they are drawn.

// objects will be rotated around the left end

// the 2x0.4x1 rectangle solid and all other future

getTranslated(1.0, 0.0, 0.0);
getTranslated((-1.0, 0.0, 0.0);
getTranslated(0.0, 0.0, 0.0.NotNil)
getPushMatrixMatrix();

getClear (GL_COLOR_BUFFER_BIT);
are now rotated relative to
NEW-NEW system so the x- and y-directions
also, changes to coordinate system to a
end of the upper arm.
(point (1, 0, 0) in OLD system is at the
WILL rotate around THE NEW origin

That is the far end of the upper arm.
in the OLD SYSTEM.
THE ORIGIN in NEW SYSTEM
TRANSFORM THE COORDINATE SYSTEM

No permanent change to MY transformation.
Scaled cube as been drawn around the origin.

StopMatrix();
MatrixCube (1.0);
MatrixCube (2.0, 0.4, 1.0);
}
the robot.

undo the damage to MY caused by drawing

\texttt{StopMatrix}();

\texttt{Matrix} to put the center of the forearm.

\texttt{NEW-NEW origin, which is the proper}

a scaled cube is drawn CENTERED at THE

\texttt{StopMatrix}();

\texttt{Matrix} (1.0);

\texttt{Matrix} (2.0, 0.4, 1.0);

\texttt{PushMatrix}();

is \((x=1, 0, 0)\) in the \texttt{NEW-NEW system}.

or\(\) in the \texttt{NEW-NEW system}

\texttt{Transform} the \texttt{COORDINATE SYSTEM} so the

\texttt{Translate} \((1.0, 0.0, 0.0)\);

the old x- and y-directions.

University at Albany Computer Science Dept.
This approach is what you should use for ... articulated robot arms.

order in the code.

With this approach, the matrix multiplications now appear in the natural

All operations occur relative to this changing coordinate system.

the object you’re drawing.

Multiplying a Local Coordinate System: Another way to view matrix

\texttt{\textit{translate}()}\texttt{, etc.}.


transformation from the right by \texttt{\textit{multiplyMatrix}()} \texttt{\textit{rotate}()}\texttt{, etc.}.

\texttt{\textit{sd}}:: \texttt{\textit{well, they do! New transformations are multiplied into the current}

\texttt{\textit{from how they appear in the code.}}

\texttt{\textit{you have to think of the multiplications as occurring in the opposite order}}

\texttt{\textit{of your model—}}

\texttt{\textit{in which matrix multiplications affect the position, orientation and scaling}}

\texttt{\textit{Grand, fixed coordinate system—}}

\texttt{\textit{Grand, fixed coordinate system}}:: \texttt{\textit{If you like to think in terms of a}}

So, quoting from the RedBook, Ch. 3:

\texttt{University at Albany Computer Science Dept.}
think about it.

now to paraphrase: The results and code are the same either way you
arm (left end).

Purpose: Set the pivot point for shoulder rotation to the base of the upper arm.

Thus, in the robot's LOCAL COORDINATE SYSTEM, the transformation from the old origin to (-1.0, 0.0, 0.0) relative to the old origin changes the origin from the old origin to the transformed position (1.0, 0.0, 0.0).

Tip: When you specify a transformation to be appended, such as...
Reason: The current coordinate system is centered at the (current) origin.

draws its cube centered at the (current) origin.

relative to the current coordinate system, because it is specified base to the CENTER OF THE Upper Arm.

The new x-axes of the current coordinate system is aligned along the upper arm.

The degrees relative to the original axes.

orientation of the axes so the x-axes and y-axes are rotated by shoulder.

changes the
Notice the demo writer installeld the scale transformation very temporarily:
{ 
  /* Regular Robot Color */
  glEnd();
  /* Draw Z-axis */
  glVertex3f(0.0, 0.0, 1.0);
  glVertex3f(0.0, 0.0, 0.0);
  /* Blue for Z */
  glColor3f(0.0, 0.0, 1.0);
  /* Draw Y-axis */
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  /* Green for Y */
  glColor3f(0.0, 1.0, 0.0);
  /* Draw X-axis */
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(0.0, 0.0, 0.0);
  /* Red for X */
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_LINES);
  glEnd();
}

void coordis()
{
  in a series of transformations. Good for debugging!
  Try this! Call this function to see the local coordinate system at any point
  University at Albany Computer Science Dept.
Very useful to know: Each local coordinate system is installed temporarily in NESTED HIERARCHICAL ORDER when a HIERARCHIAL model is drawn.

The MATRIX STACK is perfectly suited to RETURN TO a pervious coordinate system when any BRANCH of the hierarchy is finished.

For example, you can use a different coordinate system to draw each finger.

//The local coordinate system is for the wrist.
glPushMatrix();
... transform so origin is the base of finger 1
... Draw finger 1
glPopMatrix();

//The local coordinate system is for the wrist. (again)
glPushMatrix();
... transform so origin is the base of finger 2
... Draw finger 2
glPopMatrix();
Using $L^{-1}$ is like bringing somebody to a photography studio. Original objects are transformed to objects as the same as the custom camera picture of the transformed objects is the same as the standard picture of the objects and the camera so the standard picture of the OpenGL standard camera. The transformation preserves relationships between the objects and the camera you want to use, then $L^{-1}$ is used for GL viewing.

Suppose you want a picture taken with a differently shaped, directed and positioned camera. If $L$ transforms the standard GL camera into the $z$-axis and takes a fixed, standard picture,

**MINUS ZEE** defines a FIXED STANDARD "camera" looks toward the $z$-axis and takes a fixed, standard picture.

The OpenGL Camera.
Display of hidden lines or surfaces behind the visible surfaces.

Exploded or Cutaway Views: Typical of engineering, scientific,
and for how surfaces reflect or scatter light toward the eye.
Surface Rendering: use computational models for lighting conditions
Identifying Visible Lines and Surfaces: and only rasterize them.

Depth Cueing: more distant objects have less intense color.
Other techniques:

Perspective projections and their approximations are the first technique.
Typically, the normalized view volume implements clipping.

normalized view volume
coordinate by a projection onto a frustum or, for sophisticated
coordinates
"Point the Camera" means transform world coordinates into viewing
coordinates

3D Viewing: Chapter 7.
eye, each the same way as for one eye.

Stereoscopic display devices: Just calculate two images, one for each.

User controlled or animated rotation.
coordinate systems by translation so the new origin is the view point. \( T \) translates the view point to the origin. In other words, transforms

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & z_u & n_u & x_u \\
0 & z_n & n_n & x_n \\
0 & z_n & n_n & x_n
\end{bmatrix}
\] = \( R \) and

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & z & 1 & 0 \\
0 & h & 0 & 1 \\
0 & x & 0 & 1
\end{bmatrix}
\] = \( T \)

Calculation:

1. Direction on the display.

2. The projection will then transform the \( y \) direction into the \( \hat{y} \) viewing direction.

3. The \textit{up vector} is some vector that will be transformed to into the \( \hat{y} \) direction.

4. The \textit{eye vector} is the direction from which the eye is going to camera position.

5. The \textit{view point} aka viewing position aka eye position aka

The viewing transformation is typically determined by
$\mathbf{n} \times \mathbf{u} = \mathbf{\Lambda}$

2) Failure if user gives parallel $\mathbf{N}$ and $\mathbf{\Lambda}$.

\[ \mathbf{n} \times \mathbf{\Lambda} \]

Note: (1) Only division by $\mathbf{\Lambda}$ is necessary, no need to calculate length of $\mathbf{n}$.

\[ t(Zn, \hat{n}, x_n) = \frac{|\mathbf{\Lambda}|}{\mathbf{u} \times \mathbf{\Lambda}} = \mathbf{n} \]

\[ t(Zu, \hat{u}, x_u) = \frac{|\mathbf{\Lambda}|}{\mathbf{\Lambda}} = \mathbf{u} \]

Calculate:

\[ \mathbf{\Lambda} : \text{up vector} \]

\[ \mathbf{\Lambda} \] : View plane normal.

\[ \mathbf{\Lambda} \] : View plane normal.

How to get $\mathbf{\Lambda}$?

GIVENS:

Coordinate systems so the new $x$ axis is along the view plane normal, etc.

The up vector is in the $yz$ plane. IN OTHER WORDS, transforms rotate space so the view plane normal is aligned along the $z$ axis, and the $z$ axis is along the view plane normal, etc.
$0 < \mathbf{\Lambda} \cdot \mathbf{\Lambda}$

That is, make products really make $\mathbf{\Lambda}$ point in the right direction. We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.

We must carefully check that the right-hand-rule definition of the two cross products really makes $\mathbf{\Lambda}$ point in the right direction.
Do it twice; once for \( x \) and once for \( y \).

It is easy to figure it out with similar triangles:

\[
\hat{z} = (x', 0, 0)
\]

from the eye \((0', 0', 0)\) to the point \((x', y', z)\).

\[\text{Where in the } z \text{-} \text{proj? Plane do we draw the intersection of the } \text{RAY}\]

\[\text{Plane is given by the equation } z \text{-} \text{proj}.\]

Perspective projection: Suppose the eye is at \((0', 0', 0)\) and the projection

\[\text{really simple, eh?}\]

\[0 = z \text{ and } \hat{y} = \hat{y} \text{ and } x = x\]

Orthogonal projection onto the \( x \)-plane:

Now let's do two simple projections: