1 Transforming a parallelepiped (10pt)

There are 3 sets of parallel edges. Angles and distances are preserved in a rigid body transformation. So we can transform one point A and its 3 neighbors B, C and D (see figure). That takes $4 \times 16$ multiplications and $4 \times 12$ additions. Then we compute vectors $AB$ and $AC$, which takes $2 \times 3$ additions (subtractions, but it’s the same). The remaining vectors are computed as follows: $E = C + AB$; $G = D + AC$; $H = D + AB$; $F = G + AB$; those are $4 \times 3$ additions. In total, we need $4 \times 16 = 64$ multiplications and $4 \times 12 + 2 \times 3 + 4 \times 3 = 66$ additions.

This solution, or a better one which is correct gets full credit. If they get a similar solution with similar number of operations (below 80 or so of each type), they get up to 8 points. If they use an approach that doesn’t make use of the properties of the shape, or of the rigid transformation, they should get no more than 2-3 points.

2 Sequence of transformations

The original and final configurations are shown in the picture. 3 points for just drawing the figure with the right coordinates. No partial credit.
they say that the same transformation can be achieved by a glTranslatef(1, 1, 0), they get 2 points. But realize that this, although looks the same in this drawing, displays a 90 degrees rotated version of the square. A more correct answer (for full 4 points) would be:

- glTranslatef(1, 1, 0)
- glRotatef(90, 0, 0, 1)

The same transformations in a different order would get 1 point less.

3 More transformations

The images resulting from case 1 and 2 (3 points each, no partial credit) are the following:
To make case 2 be the same as case 1, it should be (4 points, no partial credit):

- `glTranslatef(6,4,0);`
- `glScalef(3,2,0);`

The analysis required is that the translation parameters have to be scaled (multiplied) by the scaling parameters, because in the first case, the scaling matrix is applied before the translation, and the translation gets scaled. This change in the parameters corrects for that. (2 points for the right answer, or 1 for a partial but correct observation).

4 Inverse of rotation matrices (10 pts.)

Row vectors are unit long and perpendicular among themselves (orthonormal), and so are column vectors. In a rotation matrix, this shows up in the fact that the dot product of a row with itself is equal to 1, but the dot product of a row with another row is 0. The product of a rotation matrix \( R \) and its inverse (transpose) \( R^T \) is the identity matrix. That’s natural if we realize that the position \((i,j)\) of the matrix product \(M \cdot N\) is given by the dot product of the row \(i\) of matrix \(M\) with column \(j\) of matrix \(N\).

This question is hard to split in parts, so I’ll give it to you to give partial credit, depending on how close to the solution they are.
5 Change of coordinate systems

The matrix we’re looking for has U, V and W in its three rows:

\[
\begin{pmatrix}
W_x & W_y & W_z \\
U_x & U_y & U_z \\
V_x & V_y & V_z
\end{pmatrix}
\]

If they say it and explain why this is true (i.e. multiply vectors U, V, W by the matrix and see the result), give full credit. If they give the transpose matrix (very frequent answer) gets 80%. Unexplained answers get half of what they would get if they were explained.

6 Commutativity of rotation matrices

They have to prove this by laying out the rotation matrices for both rotations, and seeing how the expressions reduce to the same. For example, for the first one, \( R_x(\theta_1) \cdot R_z(\theta_2) = \begin{pmatrix} \cos \theta_1 & -\sin \theta_1 & 0 \\ \sin \theta_1 & \cos \theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ \sin \theta_2 & \cos \theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c_1c_2 - s_1s_2 & -c_1s_2 - s_1c_2 & 0 \\ c_1s_2 + s_1c_2 & c_1c_2 - s_1s_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \). And if the order of the multiplied matrices is the opposite, the result should be the same. This implies that rotations about the same coordinate axis are commutative. [This part receives 4 points.]

The second part is simpler, and just requires some high school trigonometry, like \( \sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b) \), and \( \cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b) \). [Just 2 points in this part].

Finally, in the third part (worth 4 points), they have to decompose a rotation matrix into simpler ones, like this: \( R(\theta_1) = R_x(-\alpha_1) \cdot R_y(-\lambda_1) \cdot R_x(\theta_1) \cdot R_y(\lambda_1) \cdot R_x(\alpha_1) \).

So when we put two of these expressions together, the result is easily simplified:

\[
\begin{align*}
R(\theta_1) R(\theta_2) &= R_x(-\alpha_1) R_y(-\lambda_1) R_x(\theta_1) R_y(\lambda_1) R_x(\alpha_1) R_y(-\lambda_1) R_x(\theta_2) R_y(\lambda_1) R_x(\alpha_1) \\
&= R_x(-\alpha_1) R_y(-\lambda_1) R_x(\theta_1) R_y(\lambda_1) R_y(-\lambda_1) R_x(\theta_2) R_y(\lambda_1) R_x(\alpha_1) \\
&= R_x(-\alpha_1) R_y(-\lambda_1) R_x(\theta_1) R_y(\lambda_1) R_x(\theta_2) R_y(\lambda_1) R_x(\alpha_1) \\
&= R_x(-\alpha_1) R_y(-\lambda_1) R_x(\theta_1 + \theta_2) R_y(\lambda_1) R_x(\alpha_1) \\
&= R(\theta_1 + \theta_2).
\end{align*}
\]

7 Scaling along a particular vector

This should be done by composing a translation by \((-5, -5, -5)\), a rotation, a scaling by a factor of 3, then the inverse of the rotation and the inverse of the translation. The rotation matrix \( R \) should be such that aligns the vector \((1, 2, 1)\) with a coordinate axis (for example, \(Z\)). You can tell because vector \((1, 2, 1)\) should appear in one of the rows. The scaling matrix \( S \) should be an identity matrix with a 3 in the position of the diagonal which is aligned with
the row of vector (1, 2, 1) in the rotation matrix. So if the student chose to align vector (1, 2, 1) with axis Z, then the diagonal of the scaling matrix should have the third component as a 3, and all others as 1.

The result should look like this: $T(5, 5, 5)R^T S R T (−5, −5, −5)$.

If they don’t give the intermediate steps but just the final matrix, check that it’s right as follows: Post-multiply vector (5, 5, 5) by their matrix, and the result should be (5, 5, 5). Then post-multiply (6, 7, 6), and the result should be (8, 11, 8); finally, post-multiply (1, 5, 1), and the result should remain the same.

It is OK if they don’t give the final matrix, but just the explanation and the decomposition into 5 matrices. [The explanation and the correct result is worth 10 points; If the result is wrong, but the explanation is right, give them 8 points; If the result is fine but the explanation is missing or wrong, give them no more than 3 points].

8 gluLookAt and glFrustum

The first task is to find the “center” position, which is the projection of the eye point (0, 0, 0) onto the image plane. The unnormalized plane normal is (1, 1, 1). It’s easy to infer that the projection of (0, 0, 0) onto the plane has the form $(0, 0, 0) + k (1, 1, 1)$, and it’s exactly $(2, 2, 2)$. As expected, plugging these coordinates into the equation makes it hold true $2 + 2 + 2 = 6$. So now we know that the near clipping plane is at distance $\sqrt{2^2 + 2^2 + 2^2} = \sqrt{12} = 2\sqrt{3}$. This will be useful for the second part of the question. Now we have the eye point (0, 0, 0), the center point (2, 2, 2), and the up vector (0, 2, 0), which normalized is $(0, 1, 0)$.

The calculation process for gluLookAt is well explained here:
(http://pyopengl.sourceforge.net/documentation/manual/gluLookAt.3G.html), although there is a missing normalization in the calculation of $s$.

We let $f$ be the normalized vector (center-eye), or $f = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$, $s = \frac{f \times u}{|f \times u|}$, and $u = s \times f$. We construct matrix $M = \begin{pmatrix} s0 & s1 & s2 & 0 \\ u0 & u1 & u2 & 0 \\ -f0 & -f1 & -f2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, which has the form: $M = \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ -0.40825 & -0.8165 & -0.40825 & 0 \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. [Getting the complete development gets 6 full points, even if the matrix is not the same.]

For the glFrustum, we know n, f, l, r, t and b, which is all we need to compute the matrix as explained in:
(http://www.talisman.org/opengl-1.1/Reference/glFrustum.html).

So we have:

$A = (right + left)/(right − left)$
$B = (top + bottom)/(top − bottom)$
\[ C = -(\text{far} + \text{near})/(\text{far} - \text{near}) \]
\[ D = -(2 \times \text{far} \times \text{near})/(\text{far} - \text{near}) \]
\[ E = (2 \times \text{near})/(\text{right} - \text{left}) \]
\[ F = (2 \times \text{near})/(\text{top} - \text{bottom}) \]

And the matrix is

\[
\begin{bmatrix}
E & 0 & A & 0 \\
0 & F & B & 0 \\
0 & 0 & C & D \\
0 & 0 & -1 & 1 \\
\end{bmatrix} = \begin{bmatrix}
1.73205 & 0 & 0 & 0 \\
0 & -1.73205 & -3 & 0 \\
0 & 0 & -2.06002 & -10.6023 \\
0 & 0 & -1 & 0 \\
\end{bmatrix}.
\]

[Again, 6 points for the development, independently of the result.]

Finally, in order to compute the projected coordinates of \( p = (10, 4, 6)^T \), we simply multiply \( p_{pr} = P \cdot M \cdot p \), and the result should be \((-4.8990, 40.2979, 13.1869, 12.5470)^T = (-0.39045, 3.21175, 1.051, 1)^T \). [The correct result, or approximately correct result gets 2 points; The correct procedure, but wrong result gets 1 point. Partially correct procedure (missing normalizations, etc) get points off according to the importance of the mistake.]

9 Points at infinity

Vector \((x,y,z,0)\) represents the point that is “at infinity” and along the direction of axis \((x,y,z)\), because the Cartesian coordinates are obtained from their homogeneous counterparts by dividing by their \(W\) coordinate.
1 Triangle normals

The normal of triangle ABC (given counterclockwise) is \( \frac{(B-A)(C-A)}{|(B-A)(C-A)|} \).

No partial credit.

2 Vertex normals

The problem with the given method is that it considers the influence of each face on a vertex equally. The right approach would weigh the triangle normals \( N_i \) by the incident angle \( \alpha_i \): \( \frac{N_i \alpha_i + N_B \alpha_B + N_C \alpha_C + N_D \alpha_D}{N_A \alpha_A + N_B \alpha_B + N_C \alpha_C + N_D \alpha_D} \).

3 points for identifying the problem. 1 extra point if they fix it for the cube, and full credit if they give the generic solution.

3 Model transformation

The transformation is a rotation \( R = \begin{pmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \) followed by a translation \( T = \begin{pmatrix} 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \).

3 points for each matrix.

4 Transformation order

The answer is B: TRv.

No partial credit.
5 Identity matrix

It means that the camera is at the origin, looking towards the negative Z axis, with the up vector along the positive Y axis.

6 \( w=0 \) for normals

The interpretation is that normals represent directions, and thus the length of the vector is not relevant.

The proof that translations have no effect on normals is given by:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
0
\end{pmatrix} =
\begin{pmatrix}
x \\
y \\
z \\
0
\end{pmatrix}.
\]

4 points for interpreting the meaning of \( w = 0 \), and another 4 for showing the proof.

7 For normal \( n \), \( TRn \neq RTn \) or \( TRn = RTn \)?

Because a rotation of a normal \( (w = 0) \) gives another normal \( (w = 0) \), and the translation leaves the normal unaffected, \( TRn = RTn = Rn \).

3 points for saying yes; another 3 for proving it.

8 Illumination in the OpenGL pipeline

The illumination is computed between model-view and projection transformations. The reason is that projection applies some transformations (shear, scaling) that modify distances and normals, hence changing the illumination conditions. In fact, this is the only reason why modelview and projection have separate matrices in an OpenGL implementation.

2 points for naming the stage, and the other 6 according to how accurate the explanation is.

9 Gouraud shading highlights

The reason is that whenever the maximum reflectance is on a vertex, it is computed correctly, but when it is inside a triangle, it is missed by OpenGL's lighting. One solution would be using Phong shading. Another would be subdividing the triangles of the model to increase the number of vertices, where illumination is computed.

This question was misleading, so I gave everyone full credit. Most people answered it more or less fine, but a bunch of students were confused by the wording.