0. Follow the instructions above. Failure to do so will be penalized. Feedback is always welcome.

CS417 Part

1. Let us look at two derivations of the distance between a point \( w = (w_1, w_2) \) and the line defined by the origin and a point \( v = (v_1, v_2) \neq (0,0) \). Show all your work.

   (a) Compute the projection \( u \) of \( w \) onto \( v \) and then compute the distance between \( u \) and \( w \). Stick with vector notation (no vector components).

   \[
   \text{Answer: } \sqrt{w \cdot w - \left(\frac{w \cdot v}{v \cdot v}\right)^2 \cdot v \cdot v}.
   \]

   HW#0 tells us projection of \( w \) onto \( v \) is gives

   \[
   u = \frac{w \cdot v}{v \cdot v} v.
   \]

   The distance squared between \( u \) and \( w \) is

   \[
   |u - w|^2 = (u - w) \cdot (u - w) = u \cdot u - 2u \cdot w + w \cdot w
   \]

   \[
   = \left(\frac{w \cdot v}{v \cdot v}\right) \cdot v \cdot v - 2 \left(\frac{w \cdot v}{v \cdot v}\right) \cdot w \cdot w
   \]

   \[
   = \frac{w \cdot v}{v \cdot v} v \cdot v - 2 \frac{w \cdot v}{v \cdot v} w \cdot v + w \cdot w
   \]

   \[
   = w \cdot w - \left(\frac{w \cdot v}{v \cdot v}\right)^2 v \cdot v.
   \]

   (b) Compute a (non-zero) vector \( u \) that is perpendicular to \( v \) (solve for \( u \cdot v = 0 \) or rotate \( v \) by \( \pm 90 \) degrees). Compute the length of the projection of \( w \) onto \( u \). For this part, expand your answer in terms of vector components.

   \[
   \text{Answer: } \frac{v_2 w_1 - v_1 w_2}{\sqrt{v_1^2 + v_2^2}}.
   \]

   Vector \( u = (v_2, -v_1) \) satisfies \( u \cdot v = 0 \), as does unit vector

   \[
   \frac{u}{|u|} = \frac{u}{\sqrt{v_1^2 + v_2^2}}.
   \]

   The length of the projection of \( w \) onto \( \frac{u}{|u|} \) is simply the absolute value of their dot product \( w \cdot \frac{u}{|u|} \).

2. Suppose we rotate, scale (uniformly in all directions), and translate the plane so that the the origin ends up at \( a = (x_a, y_a) \), and the point \((1,0)\) ends up at \( b = (x_b, y_b)\). NOTE: No flipping/reflection (e.g. \((x,y) \mapsto (x,-y)\)) is done.

   Express this sequence of transformations as a single 3-by-3 matrix (that is applied to homogeneous coordinates). Show your work. Your final answer should not involve any angles or trig functions.

   \textbf{Answer: } We scale, then rotate, then translate. Since the scaling is uniform, it commutes with the rotation. The general forms of the rotation matrix \( R \), scale matrix \( S \), and translation matrix \( T \) are
\[ R = \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \]

Let \( w = b - a = (w_1, w_2) \). The scale factor \( s_x = s_y = |w| \) is the distance from \( b \) to \( a \). We can compute \( \sin \theta \) and \( \cos \theta \) from \( w = |w|(\cos \theta, \sin \theta) \), i.e. \( \cos \theta = w_1/|w| \), \( \sin \theta = w_2/|w| \). The translation is simply \((t_x, t_y) = a\). This gives us the final transformation \( TRS \):

\[
\begin{pmatrix} 1 & 0 & x_a \\ 0 & 1 & y_a \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} w_1/|w| & w_2/|w| & 0 \\ -w_2/|w| & w_1/|w| & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_a \\ 0 & 1 & y_a \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} x_b - x_a & y_b - y_a & x_a \\ y_a - y_b & x_b - x_a & y_a \\ 0 & 0 & 1 \end{pmatrix}
\]

3. (Moved into CS418 part)

4. Write a Matlab function so that \([x,y,z]=torus(m,n,r,R)\); \( \text{surf}(x,y,z) \) plots a torus obtained as follows. Place a circle in the \(xz\)-plane with center \( x = R, z = 0 \) and radius \( r \); sample the circle at \( m \) points. Revolve the circle around the \(z\)-axis; sample at \( n \) points, i.e. create \( n \) such circles. Don’t worry about accuracy (the original \textit{(my)}sphere code worries that \( \cos(\pi/2) \) does not yield 0).

Avoid loops. Demonstrate with \( m = 11, n = 40, r = 1, R = 2 \); use axis equal to fix the aspect ratio. Turn in your code and a printout of the plot.

\textbf{Hint:} modify \textit{my}sphere, which does the following. Place half a unit circle in the \(xz\)-plane centered at the origin, sampled at \( m \) points. Revolve the half circle around the \(z\)-axis, sampled at \( n \) points.

\textbf{Answer:} (See Watt page 53 Figure 2.27 for the parametric representation of a torus.)

\begin{verbatim}
function [x,y,z] = torus(m,n,r,R)
%TORUS Generate torus.
% [X,Y,Z] = TORUS(M,N,R,R) generates three (m+1)-by-(n+1)
% matrices so that SURF(X,Y,Z) produces a
% with m latitude and n longitude lines.
% Modified from sphere.m code by Clay M. Thompson 4-24-91, CBM 8-21-92.

theta = (0:n)/n*2*pi; % 0 <= theta <= 2pi is a row vector -- "longitude"
phi = (0:m)'/m*2*pi; % 0 <= phi <= 2pi is a column vector -- "latitude"

% compute cross section
x = R*r*cos(phi); 
z = r*sin(phi);

% rotate cross section
y = x*sin(theta); 
x = x*cos(theta); 
z = z*ones(1,n+1);
\end{verbatim}
5. (Moved out of HW#1)

3. Write a Matlab function so that \( \text{dst} = \text{rotatesuper}(\text{src}, \text{angle}, n) \) returns the source image \( \text{src} \) rotated by \( \text{angle} \) radians. For better accuracy than our code from lecture/newsgroup, divide each source pixel into \( n \times n \) subpixels, map them to destination pixels, and average appropriately. Avoid loops. Turn in your code and a printout of the image from durer rotated by \( \pi/6 \) radians.

**Answer:**

```matlab
function [dst] = rotatesuper(src, angle, n)
    % function [DST] = ROTATESUPER(SRC,ANGLE,N)
    % compute destination image DST = source image SRC rotated by ANGLE radians.
    % For better accuracy, divide each source pixel into nxn subpixels, map them
    % to destination pixels, and average appropriately.

    sz = prod(size(src));  % total # of pixels of source image

    % use (1:n*S)’/n to construct a row vector of n*S evenly spaced points
    % between 1/n to S (inclusive)
    % In effect, expand each source pixel to nxn subpixels in the rotating
    % algorithm given in class

    % create parallel arrays i, j, src that list
    % all the coordinates and values of the sub-pixels (same as enclosing pixel)
    i = (1:n*size(src,1))/n;
    j = (1:n*size(src,2))/n;
    src2 = reshape(src(ceil(i), ceil(j)), 1, sz*n*n);

    % we approximate the color of subpixel (x,y) as the color of the pixel
    % (ceil(x), ceil(y))
    i = reshape(i' * ones(1,n*size(src,2)), 1, sz*n*n);
    j = reshape(ones(n*size(src,1),1)' * j, 1, sz*n*n);

    % rotate all the subpixels; store coordinates in parallel arrays ii, jj
    ii = ceil(cos(angle) * i - sin(angle) * j);
    jj = ceil(cos(angle) * j + sin(angle) * i);

    % Each rotated subpixel (xx,yy) contributes to color of (ceil(xx),ceil(yy))
    % Consistent with the rule for source subpixels for better approximation.

    % build dst image by adding the contributions of all the sub-pixels:
    % each sub-pixel contributes 1/(n*n) of the original pixel’s value
    dst = full(sparse(ii+1-min(ii), jj+1-min(jj), src2))/(n*n);

    It should be noted that in the last step, we assume that each sub-pixel contributes \( \frac{1}{n^2} \) of the
    destination pixel’s value, and average the value of each destination pixel by dividing it by \( n^2 \).
    This is what was expected.
    However, this is only one way of approximation. It turns out that averaging by the actual
    number of subpixels mapped to the same destination pixel looks better:

    dst = full(sparse(ii+1-min(ii), jj+1-min(jj), src2));
    dst = dst ./max(1, full(sparse(ii+1-min(ii), jj+1-min(jj), 1)));
The denominator simply counts (by adding 1 for each hit) the number of source subpixels landing in each destination pixel, and the \( \max(1, \ldots) \) is to avoid dividing by zero.

This looks better because if the original code has gaps, then supersampling puts only a few sub-pixels into the gaps, so dividing by \( n^2 \) makes the “gaps” too dark.