
Computer graphics relies heavily on geometric transformations. Most common are transformations such as rotations, translations, and scales. In the following, \( T(dx,dy) \) refers to a translation by \((dx, dy)\), \( R(a) \) refers to a rotation by \(a\) degrees, and \( S(sx,sy) \) refers to a scaling by \((sx, sy)\). For simplicity, assume all transformations are 2D.

The order of transformations may matter. Also, sometimes the order may be rearranged, but the arguments will change. State whether the following statements are True or False.

\[ T(1,0) S(2,2) = T(2,0) S(1,1) \]

False. Applied to point \((1,1)\) the first yields \((3,2)\) and the second \((3,1)\).

\[ T(-1,0) T(0,2) = T(0,1) T(-1,1) \]

True. \( T(a,b)T(x,y) = T(a+x, b+y) \)

\[ R(180) = R(-180) \]

True. The angles are equal mod 360 degrees.

\[ R(45) S(2,1) = S(2,1) R(45) \]

False. Apply to point \((\sqrt{2},0)\) to get \((2, 2)\) and \((2, 1)\) respectively.

\[ T(1,0) R(90) = T(0,1) \]

False. Apply to point \((0,k)\) to get \((1-k,0)\) vs. \((0,1+k)\)

\[ R(90) T(1,0) R(90) = T(0,-1) \]

True. The critical observation is that \( T(1,0)R(90) = R(90)T(0,-1) \).

Transformations have inverses. Applying a transformation followed by its inverse has no effect. State whether the following formulas involving inverses are True or False.

\[ R(180)^{-1} = R(180) \]

True. \( R(180)R(180) = R(360) = R(0) = I \)

\[ T(1,1)^{-1} = T(-1,-1) \]

True. \( T(1,1)T(-1,-1) = T(1-1,1-1) = T(0,0) = I \)

\[ [R(45) T(1,0)]^{-1} = T(1,0) R(45) \]

False. \([A B]^T = B^T A^T\) so \([R(45)T(1,0)]^{-1} = T(-1,0) R(-45)\)

\[ [R(45) S(2,2)]^{-1} = R(-45) S(0.5,0.5) \]

True. Remember that rotations commute with uniform scales.
2. [20 points] Windows

Suppose we have a display that has size (ScreenWidth, ScreenHeight). The screen is shown below with the thickest border. On the screen is a window. The window has size (w,h) and its upper left corner is at (x,y). On the left and right the window is in different positions and has a different size.

We want to draw a clock face and have it appear to stick to the screen. That is, as the window is moved and resized, the clock sticks to the display and does not move with the window.

Assume the window coordinates would be set to (xmin,xmax,ymin,ymax) if the window occupied the full screen (ScreenWidth, ScreenHeight).

For this problem, GLUT has been modified to call back the function reposition when the window is changed. Describe the sequence of OpenGL calls with the proper arguments so that when the clock is drawn, it appears in the correct position. Make sure to include glViewport.

```c
void reposition( int x, int y, int w, int h )
{
    glViewport(0,0, w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    /* Left Right Bottom Top */
    gluOrtho2D(x, x+w, y-h, y);
    glutPostRedisplay();
}
```

We were looking for something along the lines of the code below, but accepted solutions that may have positioned the clock differently but kept it in a consistent position on resizing. 1 point was deducted for missing either glMatrixMode or glLoadIdentity, 2 for both. Additional points were deducted if the specified calls only translated the clock without considering scale, or scaled the clock without translating it appropriately.
3. [20 points] Geometry

Points and vectors are not the same. In particular, points and vectors do not transform in the same way.

3A [5 points]. We are given a point \( P=(px,py,pz) \) and a vector \( V=(vx,vy,vz) \). We now translate by \((tx,ty,tz)\). What is the new position of the point? What are the new coordinates of the vector?

\[
P' = (px+tx, py+ty, pz+tz), \quad V = (vx, vy, vz).
\]

It's important to note that points and vectors behave differently here. A correct solution was worth 5/5. Getting one out of two was worth 2/5.

3B [5 points]. Suppose we rotate the coordinate system by 90 degrees about z. What is the new position of the point? And the vector? (In this question, ignore the translation in 3A).

\[
P' = (py,-px, pz), \quad V' = (vy,-vx, vz).
\]

Alternatively, \( P'=(-py, px, pz), \quad V'=(-vy, vx, vz) \) were also accepted.

If you missed the sign change, you were deducted one point for each mistake. If you used the rotation matrix (with trigonometric functions) but applied it incorrectly, one point was deducted. Up to three points were deducted for conceptual mistakes, such as assuming that \( P \) or \( V \) do not change. Getting one out of two correctly received at least 2/5.
3C [5 points]. We wrote a program that computes a new point from two points using the expression \( p = a \cdot p_1 + b \cdot p_2 \), where \( p \), \( p_1 \) and \( p_2 \) are points, and \( a \) and \( b \) are floats. Suppose the points \( p_1 \) and \( p_2 \) are translated by \((tx,ty,tz)\). That is, \( p_1' = T(tx,ty,tz) \cdot p_1 \) and \( p_2' = T(tx,ty,tz) \cdot p_2 \). We would expect the point \( p' = T(tx,ty,tz) \cdot p \). Prove whether this is true or false.

The statement \( p' = T(tx, ty, tz) \cdot p \) is false.
A simple counter example is \( a=b=0 \). Then, \( p' = a \cdot p_1' + b \cdot p_2' = 0 \) regardless of the values of \( p_1' \) and \( p_2' \). However, \( T(tx, ty, tz) \cdot p \) is not identically zero. For instance, one could let \( tx=p1_x=p2_x=1 \).

"Proofs" that the statement is true did not receive credit. Often these arguments assumed that translation commutes with scaling, or that it is distributive.

A baseline of 1/5 was given to students who asserted the statement was false. A correct proof or a counterexample received 5/5. Proofs that had incorrect steps received partial credit, depending on the severity of the mistake.

Credit was not given for cases in which the student claimed that the problem is ill-formed and bypassed the problem. Interpolation/extrapolation are valid operations on pairs of points.

3D [5 points]. Suppose we compute \( p = (1-a) \cdot p_1 + a \cdot p_2 \). Now we translate \( p_1 \) and \( p_2 \) as we did in 3C. Again, we would expect \( p' = T(tx,ty,tz) \cdot p \). Is this true or false?

The statement \( p' = T(tx, ty, tz) \cdot p \) is true. The proof is as follows:
By definition, \( p' = (1-a) \cdot p_1' + a \cdot p_2' \). Therefore,
\[
p'\cdot = (1-a) \cdot p_1'\cdot + a \cdot p_2'\cdot = (1-a)(p1x+tx) + a (p2x + tx) = (1-a)p1x + (a)p2x + tx = px + tx.
\]
Here \( x \) as suffix means the x-component of the vector. The same holds for the y- and z-components.

A baseline of 1/5 was given to students who asserted the statement was true. If you argued that it was true simply as a special case of 3C (which is in fact false), you did not receive further credit. If you otherwise argued that the statement was true based on incorrect reasoning (such as proof by example), you were deducted up to 2 points. Geometric arguments with a convincing figure received full credit.

Overall, parsing proofs in 3C/3D required discretion of the grader.
4. [20 points] Bezier Curve

The most widely used curve in computer graphics is the Bezier curve. The quadratic Bezier curve is given by 3 points, P0, P1, and P2. The cubic Bezier curve is given by 4 points, P0, P1, P2 and P3.

To draw another curve, we need to convert it to graphics primitives. Normally we draw a curve by drawing a set of line segments. However, it is possible to convert a curve to a Bezier curve. The advantage of this approach is that it takes many fewer Bezier curves than line segments to closely approximate our curve.

In this problem, we want to draw the parabola \( y = x^2 \).

We want to draw a section of the parabola from (0,0) to (1,1).

4A [10 points] Determine the positions of the 3 controls points P0, P1, and P2 of the quadratic Bezier curve. Position these points so that the Bezier curve goes through (0,0) and (1,1) and is tangent to the parabola at these two points.

\[
\begin{align*}
\text{P0} & = (0, 0) \\
\text{P1} & = (0.5, 0) \\
\text{P2} & = (1, 1)
\end{align*}
\]

The end points must be (0, 0) and (1, 1). To get the correct tangent at (0, 0), \( P1 = (x, 0) \). The slope of the tangent at (1, 1) is 2. Therefore, we solve for \( x \) in the following: \( 2 = (1 – 0) / (1 – x) \). This yields, \( x = 0.5 \).

-1 Small errors like flipping x/y-axis, control points out of order

+2 / +4 Demonstrated general understanding of quadratic Bezier curves.

+1.5 / +3 Curve has correct endpoints at (0, 0) and (1, 1).

+1.5 / +3 Curve has correct tangents.
4B [10 points] Calculate the positions of the 4 controls points P0, P1, P2 and P3 of the cubic Bezier curve. Position these points so that the Bezier curve goes through (0,0) and (1,1) and is tangent to the parabola at these points. Also require that the point (1/2,1/4) on the parabola be on the Bezier curve. That is, P(1/2) = (1/2,1/4).

P0 = (0, 0)
P1 = (1/3, 0)
P2 = (2/3, 1/3)
P3 = (1, 1)

There were many ways to get this answer. Here is one approach. To get the correct end points, we know P0 = (0, 0) and P3 = (1, 1).

To get the correct tangents at the end points, we have P1 = (x, 0) and P2 = (a, b) where 2 = (1 – b) / (1 – a).

For either x or y, when t = 1/2:
P_{(x,y)}(1/2) = 1/8 P0 + 3/8 P1 + 3/8 P2 + 1/8 P3

In y:
P_y(1/2) = 1/8 * 0 + 3/8 * 0 + 3/8 b + 1/8 * 1 = 1/4

Solving for b, yields b = 1/3. Plug this into the slope formula:

2 = (1 – 1/3) / (1 – a)
   = (2/3) / (1 – a)
   1 – a = 1/3
This yields, a = 2/3.

Now, we can solve for x:
P_x(1/2) = 1/8 * 0 + 3/8 * x + 3/8 * 2/3 + 1/8 * 1 = 1/2
3x + 2 + 1 = 4
x = 1/3

-3 Used TrueType instead of Cubic Bezier control points (-4 if TrueType curve incorrect)

+2 / +4 Demonstrates reasonable understanding of cubic Bezier curves
+1 / +2 Curve has correct endpoints at (0, 0) and (1, 1)
+1 / +2 Curve has correct tangents
+1 / +2 Curve at t=1/2 goes through (1/2, 1/4)
5 [20 points] Input

5A [10 points]. When designing a graphics application, it is often necessary to select different objects drawn on the screen. In order to implement selection, you need a hit-testing procedure that determines whether the mouse position x, y is over the object.

In the MicroUI assignment, you created UI widgets at different positions on the screen. In this problem, we want to add the capability to transform the UI widgets to any location on the screen using a sequence of OpenGL transformations. For example,

```cpp
// make a rectangular widget of the given width w and height h
UIWidget widget(w, h);

// transform and draw
glLoadIdentity();
glTranslate(x1,y1,0);
glRotate(a1, 0,0,1);
glPushMatrix();
    glTranslate(x2,y2,0);
    glRotate(a2, 0,0,1)
    widget.Draw();
glPopMatrix();
```

Describe in detail how would you detect whether a mouse location x, y is inside the widget? Provide all the mathematical details.

To test whether the point (x,y) is within the widget, we need to transform (x,y) by the inverse of the transformation applied to the widget. The transformation applied to the widget is:

\[
M = T(x_1,y_1) R(a_1) T(x_2, y_2) R(a_2)
\]

Thus we need to transform (x,y) using the matrix:

\[
M^{-1} = R(-a_2)T(-x_2,-y_2)R(-a_1)T(-x_1,-y_1)
\]

We calculate (x’, y’) as:

\[
(x’, y’) = M^{-1} \cdot (x, y)
\]

We can then test whether (x’, y’) is within the bounds of the un-transformed widget, e.g. by using widget.HitTest().
Grading: 10 points total.

For solutions in the expected vein:
+5 Tried to solve by transforming (x,y) and then testing (x’,y’) against widget
+3 Correctly identified that (x,y) should be transformed by the inverse of the widget transform
+2 Gave correct mathematical details for getting/using the inverse

Some common mistakes were to forget that \([AB]^{-1} = B^{-1}A^{-1}\), to forget the way that OpenGL matrices accumulate, or to compute \((x,y)\cdot M^{-1}\) instead of \(M^{-1}\cdot (x,y)\).

An additional family of approaches was to transform the widget instead of the point:
+5 Tried to solve by transforming the widget and then testing (x,y)
+3 Described approach would give correct results if implemented correctly
+2 Provided all relevant mathematical details (and details were correct)

A common mistake in this case was to assume that an axis-aligned rectangle (i.e. defined entirely by pMin and pMax) would still be axis-aligned after transformation. This is not true if the transformation involves rotation.

For solutions outside of these cases:
+3 Some effort, but approach was not sound (e.g. tried to transform both (x,y) and the widget)
+10 Unexpected solution, but would work in practice and details were correct
5B [10 points]. A very common method for converting rotary motion to a direction is to use quadrature encoding. Trackballs and mechanical mice use quadrature encoding.

Quadrature encoders provide two signals A and B. The values of A and B are either 0 or 1. As rotation occurs, the values of A and B change as shown in the diagram. The waveforms are different depending on whether the motion is clockwise (CW) or counter-clockwise (CCW). In the figure below, we are rotating CW and A is on the top and B is on the bottom. If we were to rotate CCW, the waveform would be reversed and the signals A and B would run backward.

Write in pseudo-code a procedure that keeps track of A and B and returns whether the motion is CW or CCW.

A solution that does not need to maintain much state:
// called whenever A goes from low to high
Direction getDir(A, B) {
    if (B == 0) return CW;
    return CCW;
}

Another solution that maintains oldA value
// Called whenever A changes.
oldA = ...; // some initial value
Direction getDir(A, B) {
    dA = A - oldA;
    oldA = A;
    if ((dA > 0 && B == 0) || (dA < 0 && B == 1)) return CW;
    return CCW;
}

There were several solutions that looked like:
Direction getDir() {
    s = getState();
    newS = getState();
    While (s == newS) { newS = getState(); }
    // logic here to return CW or CCW
}
We treated these solutions as not maintaining state (the state is kept local) and moreover, a misunderstanding of how input systems work.

+3 Demonstrates understanding of how input signals work
+3 Demonstrates understanding of how quadrature encoding works
+2 Indicates how code is called (interrupt or polling) and maintains state
  • if this was not specified but code still worked with both polling and interrupt, we gave full credit.
+2 Pseudo-code works. We were forgiving of small mistakes with logic and also the parity (which direction CW/CCW were)