B481 / Fall 2022 – Homework 02

Zach Graber (zegraber)

September 16, 2022

1. (Reading assignment)
2. **Vector Spaces:**

Q:

(a) It can be shown that three vertices determine a triangle, if they do not belong to the same flat/straight line. Write pseudocode for a simple function that returns True if three given vertices are "collinear" and False otherwise.

(b) A Computer Graphics API could be based on an Object-Oriented System providing scalars, vectors, and points as basic types. However, graphics APIs do not provide such graphics primitives.

Provide your own explanation as to why do you think that standard Graphics APIs do not provide such graphics primitives?

(in particular, consider that such an Object-Oriented System could provide vectors as coordinate-free elements, as examined at lecture time)

A:

(a) Consider the following:

```
// Assuming that for point x, `x.x` accesses the x-coordinate, `x.y` accesses
   the y-coordinate, etc
function COLLINEAR_CHECK(point a, point b, point c) {
    // If the points a, b, and c are collinear, the area of the triangle
       formed by them should be 0.

    let inputArray = [[a.x, a.y, 1],
                      [b.x, b.y, 1],
                      [c.x, c.y, 1]];

    let area = 0.5 * DET3(inputArray);

    if (area is 0) {
        return True;
    }
    else {
        return False;
    }
}

// helper function for 3x3 determinant
// assume array is 0-indexed, and A[1][2] denotes the item in the row 1, col 2
function DET3(array A) {
```

```
    /* A[0][0] A[0][1] A[0][2]
     * A[1][0] A[1][1] A[1][2]
     * A[2][0] A[2][1] A[2][2] */
    let c1 = (A[1][1] * A[2][2]) - (A[2][1] * A[1][2]);
    let c2 = (A[1][0] * A[2][2]) - (A[2][0] * A[1][2]);
    let c3 = (A[1][0] * A[2][1]) - (A[2][0] * A[1][1]);

    return c1 - c2 + c3;
}
```

(b) While standard graphics APIs *could* provide pre-baked object-oriented structures for graphics primitives like scalars, vectors, and points, there are several reasons why it makes sense not to. First is the argument of universal compatibility. By operating as a pure state machine, APIs like OpenGL are able to offer more clear hardware abstraction. If they were centered around a programming paradigm like object-oriented design, they may be restricted to certain ecosystems, programming languages, etc. Further is that a lack of pre-provisioned structures allows more flexibility in implementation of higher-level implementations; for example, a programmer might want to implement a coordinate/location representation system that clashes with what the graphics API provides. Another major consideration is computational overhead. Object-oriented programming and languages that follow the paradigm tend to come with a *ton* of overhead (e.g. garbage collection, extra function calls for abstraction/protection, etc), thereby hurting overall performance.

3. **Line Equations:**

Q:

(a) Write down the traditional equation of a flat/straight line $y = f(x)$ with explicit parameters for the slope and intercept.

(b) Now modify this equation so that it is guaranteed to pass through the point (x_0, y_0); that is, the parameters $(x0, y0)$ should now appear in your equation $y = f(x)$. If you use any other parameters, state their meaning.

A:

(a) Let a line be defined by $y = f(x) = mx + b$, where m is the slope and b is the y-intercept.

(b) We know it must be true that, given a point (x_0, y_0) on a line, $y - y_0 = m(x - x_0)$ (where m is the slope, since y is linearly related to x). Or, with y as a function of x, we have $y = f(x) = m(x-x_0)+y_0$

4. **2D Affine Transformations:**

Q:

(a) For the following two transformations, write explicitly the two complete 3x3 matrices acting on the homogeneous column vector $\langle x, y, w \rangle$:

i. 2D rotation by -30 degrees (right-handed coordinate system, positive=counterclockwise)

ii. Translation by (2,1), then scaling by (1,2), followed by Rotation by 45 degrees counterclockwise. Hint: this means the rotation is the operation farthest from the vertex itself. You may need to multiply three 3x3 matrices to get the required result.

(b) Fixed point rule in 2D: using simple symbols like $T(x, y)$, write down the matrix notation for a sequence of transformations that performs the following operations with respect to the fixed point (x_0, y_0):

- scale by 0.25 along the x-axis and by 0.3 along the y-axis
- then rotate by an angle of 60 degrees clockwise

(remember: the point (x_0, y_0) must not move)

A:

(a)

i. $$\mathbf{v'} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

ii. $$\mathbf{v'} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{2} & -\sqrt{2} & 0 \\ \frac{\sqrt{2}}{2} & \sqrt{2} & 2\sqrt{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

(b) We can recall that for scaling and rotation, the origin is fixed (not affected by the transformations); the easiest way to ensure that the point (x_0, y_0) does not move is simply to consider (x_0, y_0) as the origin. Translate so that (x_0, y_0) is at the origin, perform our transformations, then move everything back over by (x_0, y_0). So instead of doing $R(-60°) \cdot S(0.25, 0.3) \cdot \mathbf{v}$ (where $\mathbf{v}$ is a vector representing a homogeneous coordinate), do $T(x_0, y_0) \cdot R(-60°) \cdot S(0.25, 0.3) \cdot T(-x_0, -y_0) \cdot \mathbf{v}$. You can represent this sequence of operations with the following matrix multiplication expressions:

$$\mathbf{v'} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

5. **Proximity Calculations:**

Q:

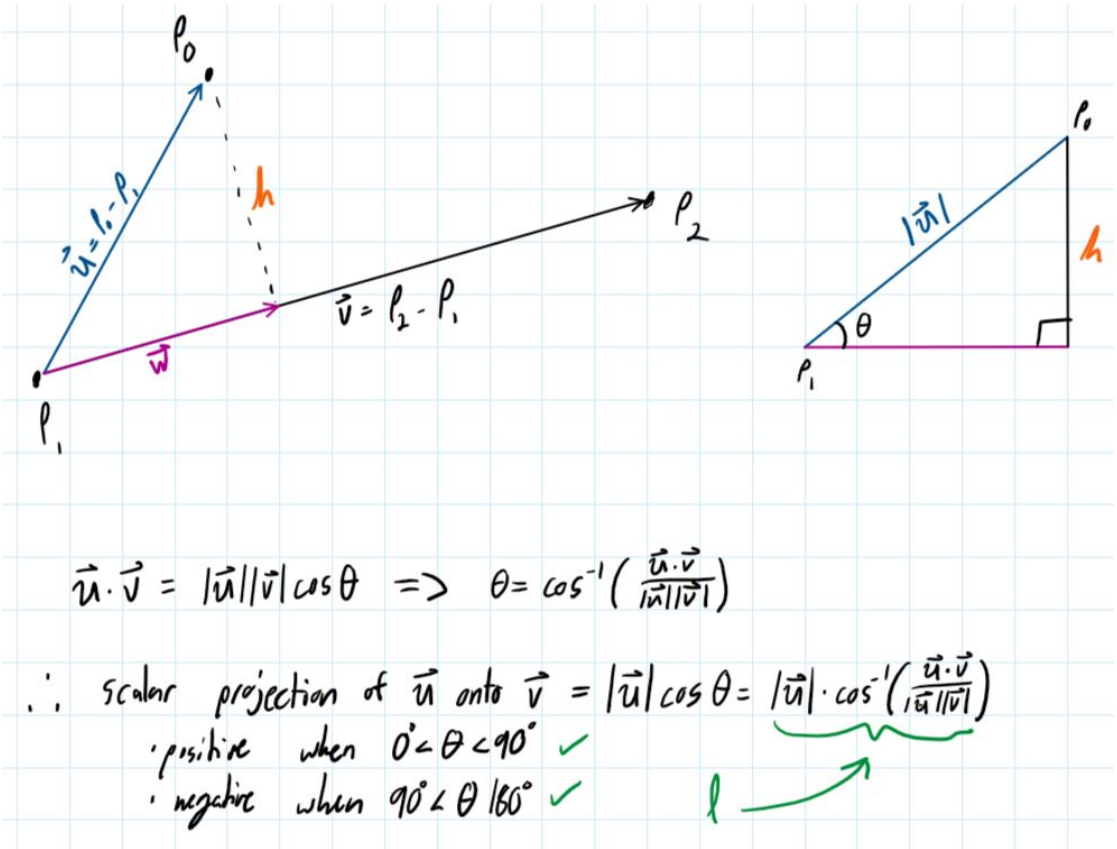
(a) What is the mathematical expression for the normalized normal vector to the subject line that goes through two 2D points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$?

(b) Given a test point $P_0 = (p_x, p_y)$, how do you compute the signed distance l between the projection of P_0 onto the subject line and one of the two points (x_1, y_1) that define the subject line?

A:

(a) The direction vector normal to the line would be given by $\mathbf{n} = \langle (y_1 - y_2), (x_2 - x_1) \rangle$, meaning the normalized (unit) normal vector is given by
$\hat{\mathbf{n}} = \frac{1}{\sqrt{(y_1-y_2)^2+(x_2-x_1)^2}} \cdot \langle (y_1 - y_2), (x_2 - x_1) \rangle$ (not expanding this for the sake of simplicity)

(b)



$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos\theta \Rightarrow \theta = \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right)$

$\therefore$ scalar projection of $\vec{u}$ onto $\vec{v} = |\vec{u}| \cos\theta = |\vec{u}| \cdot \cos^{-1}\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right)$

- positive when $0° < \theta < 90°$ ✓
- negative when $90° < \theta \leq 180°$ ✓

l

(c)

- We also know (simple trigonometry) that $h = |\vec{u}| \sin\theta$,

  giving $h = |\vec{u}| \sin\left(\cos^{-1}\left(\frac{\vec{u}\cdot\vec{v}}{|\vec{u}||\vec{v}|}\right)\right)$