CSCI 522 – Game Engine - Fall 2024 – Exam

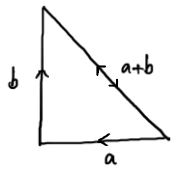
USC ID: 1154164561
NAME: Srija Madarapu
EMAIL: madarapu@usc.edu

1. How would you add and subtract two vectors A and B?

Vector Quantities: Vector Quantities refers to those physical quantities which are characterized by the presence of both magnitudes as well as direction.

Lets Vectors A = Ax î +Ay ĵ+Az k̂ & B = Bx î +By ĵ+Bz k̂

R = Rxî + Ryĵ + Rzk̂

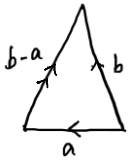


Addition:

R = A + B

Where Rx = Ax + Bx & Ry = Ay + By & Rz = Az – Bz

Substarction:



R = A – B

Where Rx = Ax – Bx & Ry = Ay – By & Rz = Az – Bz

2. How would you find the length of a vector?

Let's say we have a vector V = [x, y]. The length of V, denoted as ||V||

For a 2D vector: ||V|| = sqrt(x^2 + y^2)

For a 3D vector: ||V|| = sqrt(x^2 + y^2 + z^2)

Generalization: For an n-dimensional vector V = [x1, x2, ..., xn], the length is:

||V|| = sqrt(x1^2 + x2^2 + ... + xn^2)

3. What does normalizing a vector mean? How would you normalize a vector?

Normalizing a vector means converting the vector into a unit vector, which has a length (or magnitude) of 1. The direction of the vector remains the same, but its magnitude is scaled down (or up) to be exactly 1. To normalize a vector v=(v1,v2,…,vn).

Find the magnitude (or length) of the vector using the formula:

‖v‖=sqrt{v1^2 + v2^2 + … + vn^2}. This gives the length of the original vector.

Divide each component of the vector by its magnitude to obtain the unit vector using the formula:

vunit=v/‖v‖

4. You're developing the next Halo game. The game begins with Master Chief's ship traveling in space with a velocity V. How would you calculate how far the ship would go in time t?

To calculate how far the ship travels in time t, we can use the following formula:

Distance or Displacement = Velocity(Vx,Vy,Vz) × Time(t)

Here:

- Distance or Displacement is a vector representing the change in position of the ship.
- Velocity is the vector representing the speed and direction of the ship's movement.
- Time is the duration of the travel.

By multiplying the velocity vector by the time, we scale the vector, resulting in a new vector that represents the displacement of the ship over that time period d(t)=(Vx·t,Vy·t,Vz·t).

The magnitude of this displacement vector gives us the distance travelled |d(t)|=sqrt{(Vx·t)2+(Vy·t)2+(Vz·t)2}.

5. The ship lands on an alien planet and Master Chief steps out (at position M) only to be greeted by a grunt (at position G). How would you figure out the direction and distance his Spartan laser beam must travel to neutralize the grunt?

To determine the direction and distance of the laser beam, we need to calculate the vector pointing from Master Chief's position (M) to the Grunt's position (G).

1. Calculate the Direction Vector: Direction Vector = G – M. This vector will point from M to G, indicating the direction the laser beam should travel.

2. Calculate the Distance: Distance = ||Direction Vector||

This is the magnitude of the direction vector, representing the distance between M and G.

6. About that Spartan laser; how would you calculate the position of one laser photon P on its way to hit the grunt over time t?

If we know the velocity V the photon and its initial position P0 (the position at t=0), the position of the photon at any time t is given by the equation:

$$P(t)=P0+V\cdot t$$

- P(t): This represents the position of the laser photon at a specific time t.
- P0: This is the initial position of the photon, or the starting point of its journey.
- t: This is the time elapsed since the photon began its journey.
- V: This is the velocity vector of the photon, indicating its direction and speed.

7. Spartan's laser shoots photons with a low fire rate but high damage. Each photon has a sphere collider around it. How do you know if you have hit the grunt if grunt has a sphere collider? Note the photon moves pretty fast and covers large distance per frame

To determine if a photon shot from the Spartan's laser has hit the Grunt, given that both the photon and the Grunt have sphere colliders, we need to consider a few key points:

The photon moves quickly: The photon may travel a large distance in a single frame, which means we can't just check for a collision between the photon’s center and the Grunt’s sphere. We need to account for the photon’s trajectory over time and how far it moves per frame.

Both the photon and the Grunt have sphere colliders: Since the photon has a sphere collider and the Grunt also has a sphere collider, we are essentially checking if the photon’s sphere intersects with the Grunt’s sphere at any point along the photon's trajectory.

Direct Sphere-Sphere Intersection

Calculation: Calculate the distance between the centers of the photon's sphere and the Grunt's sphere: ||P - G||

Collision: If this distance is less than or equal to the sum of their radii (Rp + Rg), a collision occurs.

This condition ensures that the two spheres physically overlap.

Photon Overlaps Grunt's Position

Calculation: Calculate the distances from the origin (O) to both the photon (P) and the Grunt (G): ||O-P|| and ||O-G||

Collision: If the distance from the origin to the photon is greater than the distance from the origin to the Grunt, it implies that the photon has "overshot" the Grunt. This can occur if the Grunt is relatively close to the origin and the photon travels a significant distance in a single frame.

8. Master Chief sees an abandoned wraith (at position W) and gets on board. It has a machine gun mounted to the front that can rotate 90° left and right, and a bunch of grunts are swarming him. How can you find which grunts the wraith can possibly shoot at?

Given: Grunt's position(G) Wraith's position(W) Vector from Wraith to Grunt(WG=G−W) Wraith's forward direction(F)

Goal: Determine whether the grunt is within the shooting arc of the Wraith's machine gun. The machine gun has a firing arc of 90° to the left and right of the Wraith's forward direction (assuming the forward direction vector F). To do this, we will compute the dot product of the two vectors WG(the vector from the Wraith to the Grunt) and F (the Wraith's forward direction), and use it to calculate the cosine of the angle between them.

Steps:

1. Dot Product of WG & F:

$F \cdot WG = ||F|| \cdot ||WG|| \cdot \cos(\theta)$

2. Solving for cos(θ):

We can rearrange the above equation to isolate $\cos(\theta) = F.WG / ||F|| \cdot ||WG||$

3. Determine if Grunt is within the Shooting Realm:

If $\cos(\theta) \geq 0$, this means that the angle θ is less than 90 degrees, which means the Grunt lies within the shooting arc of the Wraith. Specifically:

- If $\cos(\theta) \geq 0$, the angle θ is between 0° and 90°, meaning the Grunt is in front of the Wraith and within the shooting range.
- If $\cos(\theta) < 0$, the angle θ is between 90° and 180°, meaning the Grunt is behind the Wraith and outside the shooting arc.

9. For the grunts that the wraith's the machine gun couldn't hit, Master Chief decides to try trample them by strafing left and right. How would we find the direction vectors for this?

Given: Wraith's forward direction vector F(normalized, pointing in the direction the Wraith is facing), Position of the grunt G, Vector from Wraith to Grunt WG=G−W, which points from the Wraith's position to the Grunt.

Steps:

1. Compute the dot product $A = F \cdot WG$

2. Compute the magnitude of the resultant vector sqrt{||WG||^2+A^2}

3. Check the cross product F×WG: If positive (upward), rotate F clockwise 90° to get the right strafe direction (Fy,−Fx). If negative (downward), rotate F counterclockwise 90° to get the left strafe direction (−Fy,Fx).

10. Your radar picks up a new enemy, an elite (at position E) in the distance somewhere behind you. How would you turn your wraith's plasma cannon (that can rotate 360°) which is currently aimed at a grunt (at location G) to face the elite over time t with constant angular speed?

Give: The Wraith is initially aimed at the Grunt G. The Elite is at position E. The plasma cannon can rotate 360°, and we want it to rotate with a constant angular speed to face the Elite.

Goal: We need to calculate the angle between the current direction and the direction to the Elite, then rotate the cannon at a constant speed until it reaches the Elite's position.

Steps:

1. Compute the initial direction to the Grunt: g=G−W.

2. Compute the direction to the Elite: e=E−W.

3. Calculate the angle between the current direction and the target using the dot product: θ=arccos(g·e)/||g||·||e||).

4. Set the constant angular speed ω, and calculate the time t to rotate by the angle θ: t=θ/ω.

5. Update the cannon's direction as it rotates at speed ω over time t, using rotation matrices to update the vector direction.

11. What parts of a matrix represents rotation, translation and scale? (Show using separate matrices if necessary) Does the order or multiplying transformation matrices matter? If so, how/why?

Matrix Representation of Transformations

In computer graphics, 3D transformations like rotation, translation, and scaling are often represented using 4x4 matrices. This 4x4 matrix format allows us to efficiently combine multiple transformations into a single matrix.

1. Translation Matrix:

A translation matrix shifts a point in 3D space. It's represented as:

| 1 0 0 tx |

| 0 1 0 ty |

| 0 0 1 tz |

| 0 0 0 1 |

Where tx, ty, and tz are the translation amounts along the x, y, and z axes, respectively.

2. Rotation Matrix:

A rotation matrix rotates a point around a specific axis. The exact form of the rotation matrix depends on the axis of rotation.

x-axis:

| 1 0 0 0 |

| 0 cos(θ) -sin(θ) 0 |

| 0 sin(θ) cos(θ) 0 |

| 0 0 0 1 |

y-axis:

| cos(θ) 0 sin(θ) 0 |

| 0 1 0 0 |

| -sin(θ) 0 cos(θ) 0 |

| 0 0 0 1 |

z-axis:

| cos(θ) -sin(θ) 0 0 |

| sin(θ) cos(θ) 0 0 |

| 0 0 1 0 |

| 0 0 0 1 |

Similar matrices can be derived for rotations around the x and y axes.

3. Scaling Matrix:

A scaling matrix scales a point along each axis. It's represented as:

| sx 0 0 0 |

| 0 sy 0 0 |

| 0 0 sz 0 |

| 0 0 0 1 |

Where sx, sy, and sz are the scaling factors along the x, y, and z axes, respectively.

Yes, the order of matrix multiplication matters. The order in which transformations are applied can significantly affect the final result. For example, consider a sequence of transformations: Translation by a vector T. Rotation by an angle θ around the z-axis.

If we multiply the transformation matrices in the order: Rotation × Translation, we get a different result than if we multiply them in the order Translation × Rotation.

Matrix Multiplication is Not Commutative: In general, matrix multiplication is not commutative. This means that A × B is not necessarily equal to B × A.

Transformation Order: The order in which transformations are applied affects the final result. For example, rotating a point and then translating it will yield a different result than translating it first and then rotating it.

To achieve the desired transformation, it's crucial to apply the transformations in the correct order. Typically, the order is Scale → Rotate → Translate. However, specific scenarios may require different orders**.**

12. Given a 4x4 matrix C, that represents position and orientation of a camera, 4x4 matrix A that represents position and orientation of an object. Point P is given in local space of the object. How would you calculate Pcam = point P in the camera space?

1. Transform the point P from object space to world space using the object's transformation matrix A:

$$Pworld=A \cdot P$$

2. Convert the point from world space to camera space using the inverse of the camera matrix C:

$$Pcam=C^{-1} \cdot Pworld$$

Alternatively, you can combine the two transformations into one matrix multiplication:

$$Pcam=C^{-1} \cdot A \cdot P$$

Where $C^{-1} \cdot A$ is the combined transformation matrix that converts from the object's local space directly to camera space.

13. What is the inverse of an orthonormal matrix equal to?

An orthonormal matrix is a square matrix A whose rows and columns are orthonormal vectors. This means that:

1. The rows (and columns) of A are all unit vectors (have a magnitude of 1).
2. The rows (and columns) are orthogonal to each other (i.e., their dot product is zero).

Mathematically, a matrix A is orthonormal if:

$$A^T \cdot A=I$$

From the equation $A^T \cdot A=I$, we can see that multiplying A by its transpose A^T results in the identity matrix. This implies that:

$$A^{-1}=A^T$$

So, the inverse of an orthonormal matrix is its transpose.

14. Suppose there is an object centered on the origin. Show how you would rotate it around any axis about a point P. (Show the matrices, you don’t need to provide actual numeric values)

To rotate an object around an arbitrary point P(not necessarily at the origin), you can break down the transformation into a sequence of transformations involving translation and rotation. Here's how you would do it using matrices.

Step 1: Translate the object so that P becomes the origin:

To shift the object such that the point P=(px,py,pz) is moved to the origin, we apply a translation matrix that moves the object by −P. This translation matrix is:

$T-P$:

$$\begin{vmatrix} 1 & 0 & 0 & -px \\ 0 & 1 & 0 & -py \\ 0 & 0 & 1 & -pz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Where: px,py,pz are the coordinates of point P.

Step 2: Apply the rotation around the axis of interest

For rotation, you will use a rotation matrix. The specific form of the rotation matrix depends on the axis around which you want to rotate. Commonly, we use rotation matrices for rotation about the X, Y, or Z axis.

Rotation around the X-axis by an angle θ:

| 1 0 0 0 |

| 0 cos(θ) -sin(θ) 0 |

| 0 sin(θ) cos(θ) 0 |

| 0 0 0 1 |

Rotation around the Y-axis by an angle θ:

| cos(θ) 0 sin(θ) 0 |

| 0 1 0 0 |

| -sin(θ) 0 cos(θ) 0 |

| 0 0 0 1 |

Rotation around the Z-axis by an angle θ:

| cos(θ) -sin(θ) 0 0 |

| sin(θ) cos(θ) 0 0 |

| 0 0 1 0 |

| 0 0 0 1 |

Step 3: Translate the object back by P:

After rotating the object around the origin, you need to translate the object back to its original position. This is done by applying a translation matrix T+P, which moves the object by P:

T+P:

|1 0 0 px|

|0 1 0 py|

|0 0 1 pz|

|0 0 0 1|

The final transformation matrix Mrotate that rotates the object around the point P is the product of the three matrices:

$Mrotate=T+P \cdot R \cdot T-P$

Where R is the rotation matrix for the specific axis and angle (e.g., Rx(θ), Ry(θ), Rz(θ))

15. Given an arbitrary axis A, how would you identify the two other axes (B, C) and create a matrix that represents the three axes?

Given an arbitrary axis A, we want to create two additional orthogonal axes, B and C, to form a complete orthonormal basis. This basis can then be represented as a 3x3 rotation matrix.

1. Normalize Axis A: Ensure A=(Ax,Ay,Az) is a unit vector: A=A/‖A‖.

2. Find Axis B: Choose a vector B0 that is not parallel to A. (e.g., B0=(1,0,0) if A is not aligned with the X-axis). Compute B=A×B0(cross product).

3. Normalize B: Normalize B to ensure it's a unit vector: B=B/‖B‖.

4. Find Axis C: Compute C=A×B to get the third orthogonal axis.

5. Normalize C: Normalize C: C=C/‖C‖.

6. Form Rotation Matrix: Arrange A, B, and C as the columns of a matrix

M=|Ax Bx Cx|

|Ay By Cy|

|Az Bz Cz|

This matrix represents the orthonormal basis with axes A, B, and C, and can be used to transform between coordinate systems.

16. What are Euler angles? What are some of their drawbacks?

Euler angles are a way of representing the orientation of an object in three-dimensional space with three angle. They provide a way to describe the rotation of an object using a sequence of three rotations about different axes. Typically, Euler angles are referred to as roll (φ), pitch (θ), and yaw (ψ), corresponding to rotations about the x-, y-, and z-axes, respectively.

Definition of Euler Angles:

- Roll (φ): Rotation about the x-axis.
- Pitch (θ): Rotation about the y-axis after roll is applied.
- Yaw (ψ): Rotation about the z-axis after roll and pitch are applied.

The order in which these rotations are applied is significant, and there are multiple conventions (e.g., XYZ, YZX, ZYX, etc.), where the letters denote the axes about which the rotations are performed and their order.

Drawbacks of Euler Angles:

1. Gimbal Lock: One of the most significant drawbacks is gimbal lock, which occurs when two of the three axes of rotation align. This results in the loss of one degree of freedom and can no longer represent motion about one axis because it collapses with another. In practical terms, it means that the object cannot be rotated freely in all three dimensions and certain orientations cannot be reached.
2. Ambiguity: Multiple sets of Euler angles can represent the same orientation, which can lead to ambiguity in interpreting the angles.
3. Non-Commutativity: The order of rotations is critical; changing the order of the rotation angles can lead to a completely different orientation, making the system non-commutative.
4. Complex Rotation Sequences: Interpolating between two sets of Euler angles can be complex and unintuitive because it often requires complex rotation sequences that do not correspond to a straight path in rotation space.

5. Numerical Instability: Rotations represented by Euler angles can suffer from numerical instability, particularly when rotations become small, which can introduce significant errors in calculations.

17. How would you build a plane equation from a triangle?

To build the plane equation from a triangle:

1. Find Two Edge Vectors: Given triangle vertices A, B, and C, compute two edge vectors: AB=B−A, AC=C−A

2. Compute the Normal Vector: Calculate the normal vector to the plane by taking the cross product of the two edge vectors: N=AB×AC

3. Form the Plane Equation: Use the normal vector N=(Nx,Ny,Nz) and a point on the plane (e.g., A) to form the equation: Nx(x−Ax)+Ny(y−Ay)+Nz(z−Az)+D=0

Expanding this gives the plane equation: NxX+NyY+NzZ+D=0 where D=-(NxX+NyY+NzZ). This defines the plane containing the triangle.

18. How would you find out the point at which a line intersects a plane?

To find the point at which a line intersects a plane, we need to know:

1. The equation of the plane: The general equation of the plane is: Nxx+Nyy+Nzz+D=0 where (Nx,Ny,Nz) is the normal vector to the plane, and D is the constant from the plane equation.

2. The parametric equation of the line: The line is typically defined using a point P0=(x0,y0,z0) on the line and a direction vector d=(dx,dy,dz). The parametric equation of the line is: P(t)=(x0,y0,z0)+t·(dx,dy,dz)

Where t is a parameter.

Find the Intersection Point

1. Write the Parametric Equation of the Line: The coordinates of any point on the line are given by: x(t)=x0+t·dx, y(t)=y0+t·dy, z(t)=z0+t·dz

2. Substitute the Line Equation into the Plane Equation: Substitute the parametric equations for x(t), y(t), and z(t) into the plane equation: Nx(x0+t·dx)+Ny(y0+t·dy)+Nz(z0+t·dz)+D=0

3. Solve for t:

    Expand the equation: Nxx0+t·Nxdx+Nyy0+t·Nydy+Nzz0+t·Nzdz+D=0.

    Group terms with t together: (t·(Nxdx+Nydy+Nzdz)) = -D-(Nxx0+Nyy0+Nzz0)

    Solve for t: t = -D-(Nxx0+Nyy0+Nzz0)/Nxdx+Nydy+Nzdz

4. Substitute t back into the Parametric Equation: Once you have t, substitute it back into the parametric equations for x(t), y(t), and z(t) to find the intersection point: x=x0+t·dx, y=y0+t·dy, z=z0+t·dz

Conditions for Intersection

- Intersection Exists: The line intersects the plane if Nxdx+Nydy+Nzdz≠0. This ensures that the direction vector of the line is not parallel to the plane (so the line is not parallel to the plane).
- No Intersection: If Nxdx+Nydy+Nzdz=0, then the line is either parallel to the plane or lies in the plane (if D=Nxx0+Nyy0+Nzz0).

19. How would you test if a point is inside a triangle?

To determine if a point is inside a triangle, we can use a technique called barycentric coordinates.

Barycentric Coordinates: Barycentric coordinates represent a point P within a triangle ABC as a weighted sum of the triangle's vertices. The weights (α, β, γ) are called the barycentric coordinates of P.

$$P = \alpha A + \beta B + \gamma C$$

The sum of the barycentric coordinates must equal 1:

$$\alpha + \beta + \gamma = 1$$

Testing Point Inside a Triangle:

1. Calculate Barycentric Coordinates: Use a system of linear equations to solve for α, β, and γ. There are different methods to solve this system, such as matrix inversion or Cramer's rule.

2. Check the Values: If all three barycentric coordinates (α, β, γ) are between 0 and 1, the point P lies inside the triangle. If any of the coordinates is negative or greater than 1, the point is outside the triangle.

20. [Extra Credit] You have skeleton hierarchy: root->waist->leftHip->leftKnee->leftFoot, and associated joint space matrices. You have a bind pose for the skeleton. The bind pose joint matrices are in joint space. You also have a transform of ankle bracelet in model space such that it fits the bind pose. Assuming you have some skeleton pose given to you with joint matrices in joint space, how would you calculate model space transform of ankle bracelet for the pose?

To compute the **Ankle Bracelet Transform for the Current Pose** in the world space, we

1. Compute the world space transform for the left foot in the bind pose: $W_{bind\_leftFoot}=B_{root}\cdot B_{waist}\cdot B_{leftHip}\cdot B_{leftKnee}\cdot B_{leftFoot}$

2. Compute the world space transform for the left foot in the current pose: $W_{current\_leftFoot}=C_{root}\cdot C_{waist}\cdot C_{leftHip}\cdot C_{leftKnee}\cdot C_{leftFoot}$

3. Compute the inverse of the bind pose world transform for the left foot: $W_{bind\_leftFoot\_inv}$ =inverse($W_{bind\_leftFoot}$)

Multiply the transforms to get the ankle bracelet's transform in the current pose: $A_{current}=W_{current\_leftFoot}\cdot W_{bind\_leftFoot\_inv}\cdot A_{bind}$

21. [Extra Credit] Assume triangle's points are at T0, T1, T2 where each point lies on z=0 plane and consists of integer components. That is, each point's x,y are integer, z=0. Assume we calculate A = the triangle's area. How many bits would you need to store the A's fraction part without losing any precision? Why?

To calculate how many bits are needed to store the fractional part of the area A of a triangle, we need to consider the formula used to compute the area and the potential precision required for the fractional part.

Step 1: Triangle Area Calculation

Given the vertices of the triangle in 2D space: T0=(x0,y0), T1=(x1,y1), T2=(x2,y2)

The area A of a triangle in 2D can be computed using the determinant (or cross product) formula: $A=1/2|x0(y1-y2)+x1(y2-y0)+x2(y0-y1)|$

Since the points lie on the z=0 plane and all coordinates are integers, the values of x0,y0,x1,y1,x2,y2 are all integers. Thus, the expression inside the absolute value is a sum of integer terms, so the area A itself can be expressed as: $A=1/2\times(integer)A$

The key insight here is that the area is half of an integer. This means that the area could either be an integer or a fraction with a denominator of 2. Specifically: If the area is an integer (i.e., if the numerator of the cross product is even), then A is an integer. If the area is a fraction, then the denominator will be 2, and the area will be of the form integer/2, meaning the fractional part is 0.5

Step 2: Bits Needed for Storing the Fractional Part

Now, let's focus on the fractional part of A: If A is an integer, there is no fractional part, and no bits are required for the fractional part. If A is a fraction, it will either be n+0.5(where n is an integer), and the fractional part will be 0.5. In this case, you only need 1 bit to represent the fractional part (since 0.5 can be expressed in binary as 0.1).

Step 3: Conclusion

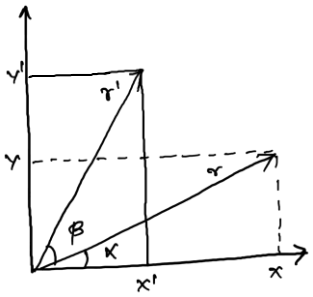
To store the fractional part of the area A: If A is an integer, no bits are required for the fractional part. If A has a fractional part (i.e., A=n+0.5), you only need 1 bit to store the fractional part (since it is just 0.5).

Thus, in the worst case (when the area has a fractional part), you will need 1 bit to store the fractional part without losing any precision.

22. [Extra Credit] Can you show us how to derive the 2D rotation matrix? $|cos\ cos\ \theta\ -sin\ sin\ \theta\ sin\ sin\ \theta\ coscos\ \theta\ |$

x' = r'cos(α+β)

y' = r'sin(α+β)



Now, find an expression for x' and y' in terms of x and y.

Using the trigonometry identities, we have

cos(α+β) = cos(α)cos(β) - sin(α)sin(β)

sin(α+β) = cos(α)sin(β) + sin(α)cos(β)

Also by rotating the vector we did not change its magnitude, so r = r'.

Now using the above information, we have

x' = r*cos(α)cos(β) - r*sin(α)sin(β)

y' = r*cos(α)sin(β) + r*sin(α)cos(β)

Simplify the above expressions using x = r*cos(α), y = r*sin(α), we get

x' = x*cos(β) - y*sin(β)

y' = x*sin(β) + y*cos(β)

We can see that the x' and y' (components of rotated vector) can be written as a function of the initial vector x and y, and then these vectors which incorporate the angle β over which we rotated our initial vector.

Now, this transformation can now be written in matrix form, we have

(x' y') = (|cos(β) -sin(β)| | sin(β) cos(β)|) (x y)

Here, the matrix (|cos(β) -sin(β)| |(sin(β) cos(β)|) is called the Rotation Matrix in 2D by an angle β.