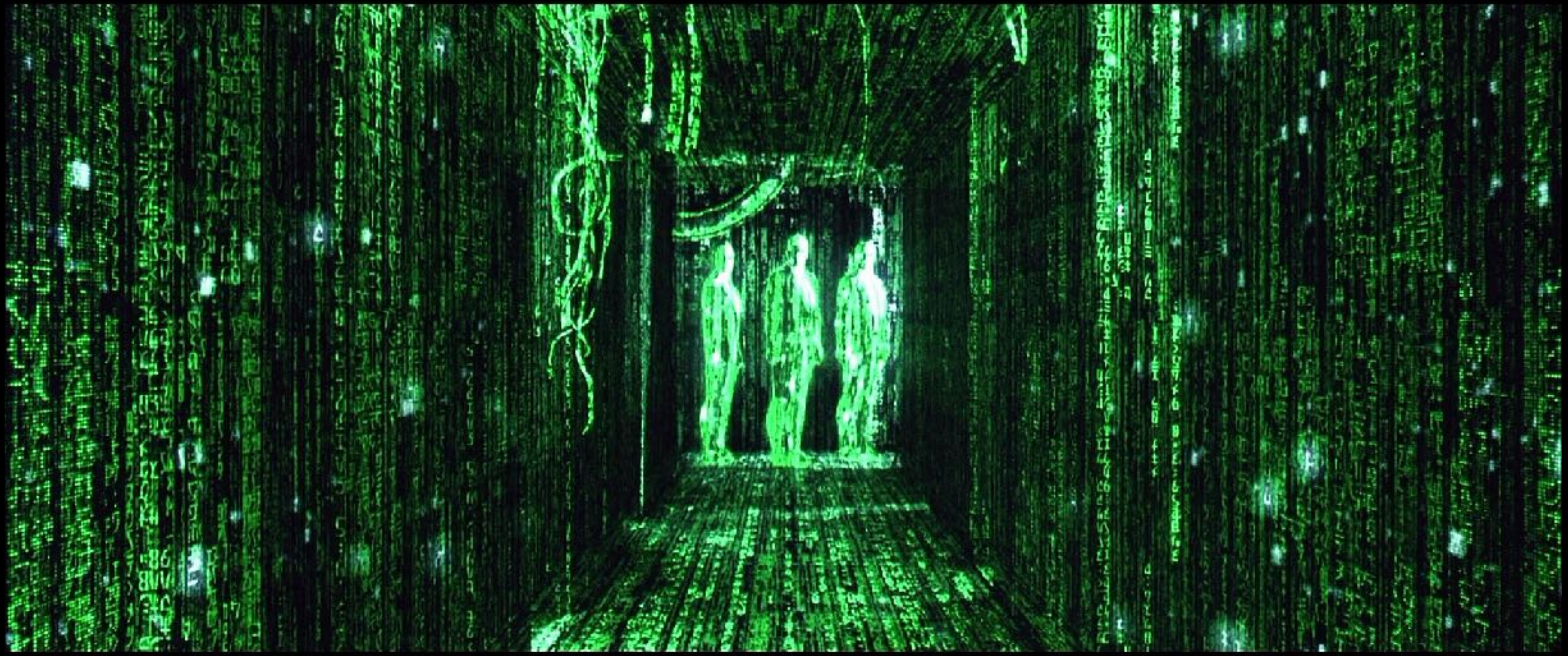
Matrix transformations.

Part 2



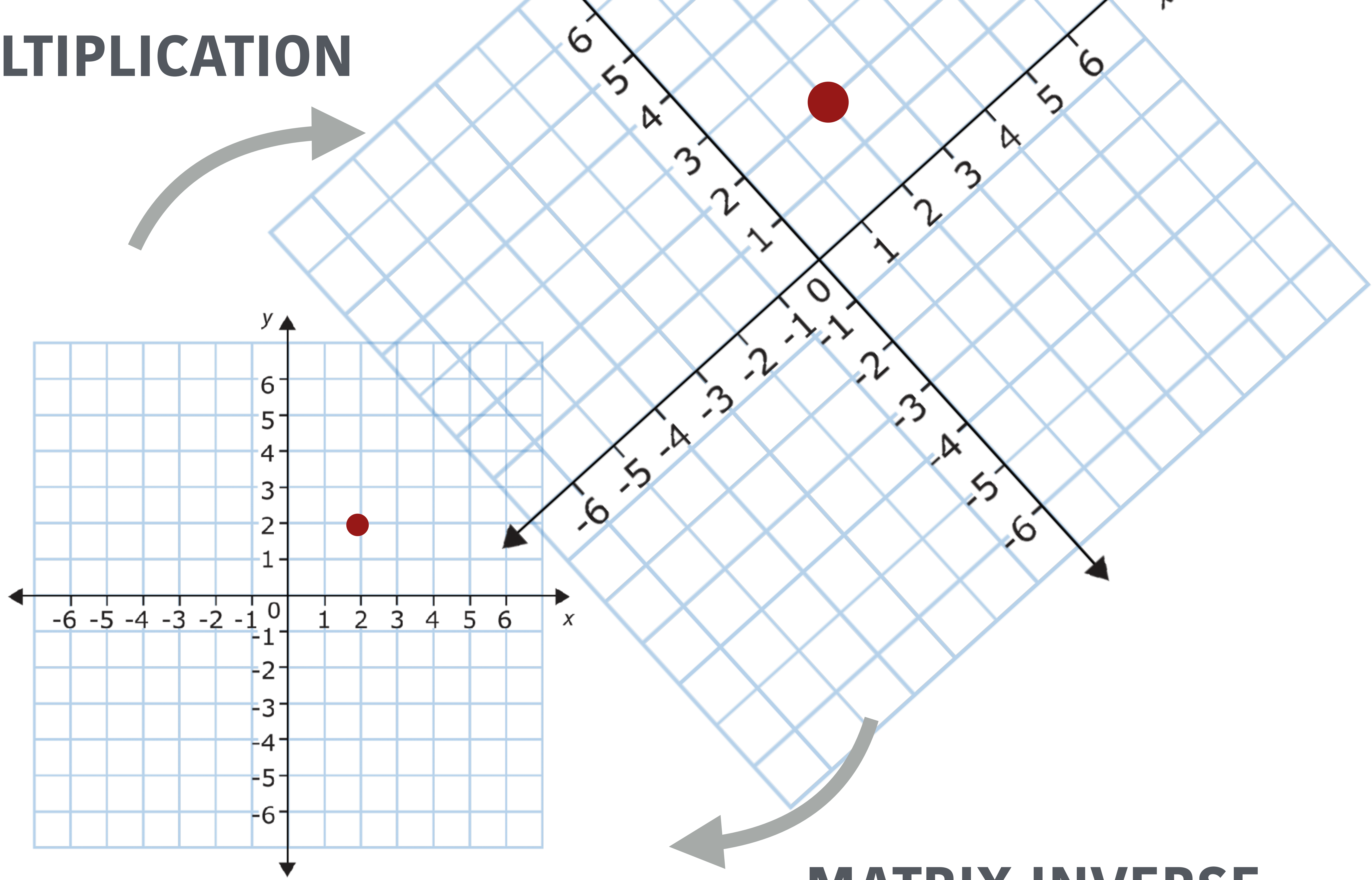
Inverse of a matrix.

Inverse of a matrix.

=

A matrix that “undoes” the transformation of the original matrix.

MATRIX MULTIPLICATION



MATRIX INVERSE

Effects of the inverse of a matrix.

- Scaled by 1/scale

Effects of the inverse of a matrix.

- Scaled by 1/scale

- Rotated by the transpose of the linear part of the matrix.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Effects of the inverse of a matrix.

- Scaled by 1/scale

- Rotated by the transpose of the rotation.

- Translated by the translation * -1

The Inverse

```
Matrix Matrix::inverse() {
    float m00 = m[0][0], m01 = m[0][1], m02 = m[0][2], m03 = m[0][3];
    float m10 = m[1][0], m11 = m[1][1], m12 = m[1][2], m13 = m[1][3];
    float m20 = m[2][0], m21 = m[2][1], m22 = m[2][2], m23 = m[2][3];
    float m30 = m[3][0], m31 = m[3][1], m32 = m[3][2], m33 = m[3][3];

    float v0 = m20 * m31 - m21 * m30;
    float v1 = m20 * m32 - m22 * m30;
    float v2 = m20 * m33 - m23 * m30;
    float v3 = m21 * m32 - m22 * m31;
    float v4 = m21 * m33 - m23 * m31;
    float v5 = m22 * m33 - m23 * m32;

    float t00 = + (v5 * m11 - v4 * m12 + v3 * m13);
    float t10 = - (v5 * m10 - v2 * m12 + v1 * m13);
    float t20 = + (v4 * m10 - v2 * m11 + v0 * m13);
    float t30 = - (v3 * m10 - v1 * m11 + v0 * m12);

    float invDet = 1 / (t00 * m00 + t10 * m01 + t20 * m02 + t30 * m03);

    float d00 = t00 * invDet;
    float d10 = t10 * invDet;
    float d20 = t20 * invDet;
    float d30 = t30 * invDet;

    float d01 = - (v5 * m01 - v4 * m02 + v3 * m03) * invDet;
    float d11 = + (v5 * m00 - v2 * m02 + v1 * m03) * invDet;
    float d21 = - (v4 * m00 - v2 * m01 + v0 * m03) * invDet;
    float d31 = + (v3 * m00 - v1 * m01 + v0 * m02) * invDet;

    v0 = m10 * m31 - m11 * m30;
    v1 = m10 * m32 - m12 * m30;
    v2 = m10 * m33 - m13 * m30;
    v3 = m11 * m32 - m12 * m31;
    v4 = m11 * m33 - m13 * m31;
    v5 = m12 * m33 - m13 * m32;

    float d02 = + (v5 * m01 - v4 * m02 + v3 * m03) * invDet;
    float d12 = - (v5 * m00 - v2 * m02 + v1 * m03) * invDet;
    float d22 = + (v4 * m00 - v2 * m01 + v0 * m03) * invDet;
    float d32 = - (v3 * m00 - v1 * m01 + v0 * m02) * invDet;

    v0 = m21 * m10 - m20 * m11;
    v1 = m22 * m10 - m20 * m12;
    v2 = m23 * m10 - m20 * m13;
    v3 = m22 * m11 - m21 * m12;
    v4 = m23 * m11 - m21 * m13;
    v5 = m23 * m12 - m22 * m13;

    float d03 = - (v5 * m01 - v4 * m02 + v3 * m03) * invDet;
    float d13 = + (v5 * m00 - v2 * m02 + v1 * m03) * invDet;
    float d23 = - (v4 * m00 - v2 * m01 + v0 * m03) * invDet;
    float d33 = + (v3 * m00 - v1 * m01 + v0 * m02) * invDet;

    Matrix m2;

    m2.m[0][0] = d00;
    m2.m[0][1] = d01;
    m2.m[0][2] = d02;
    m2.m[0][3] = d03;
    m2.m[1][0] = d10;
    m2.m[1][1] = d11;
    m2.m[1][2] = d12;
    m2.m[1][3] = d13;
    m2.m[2][0] = d20;
    m2.m[2][1] = d21;
    m2.m[2][2] = d22;
    m2.m[2][3] = d23;
    m2.m[3][0] = d30;
    m2.m[3][1] = d31;
    m2.m[3][2] = d32;
    m2.m[3][3] = d33;

    return m2;
}
```

The matrix class.

Row major vs. column major.

$$\begin{array}{cc} \text{Row major} & \text{Column major} \\ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} & \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \end{array}$$

We will use column major order as our standard.

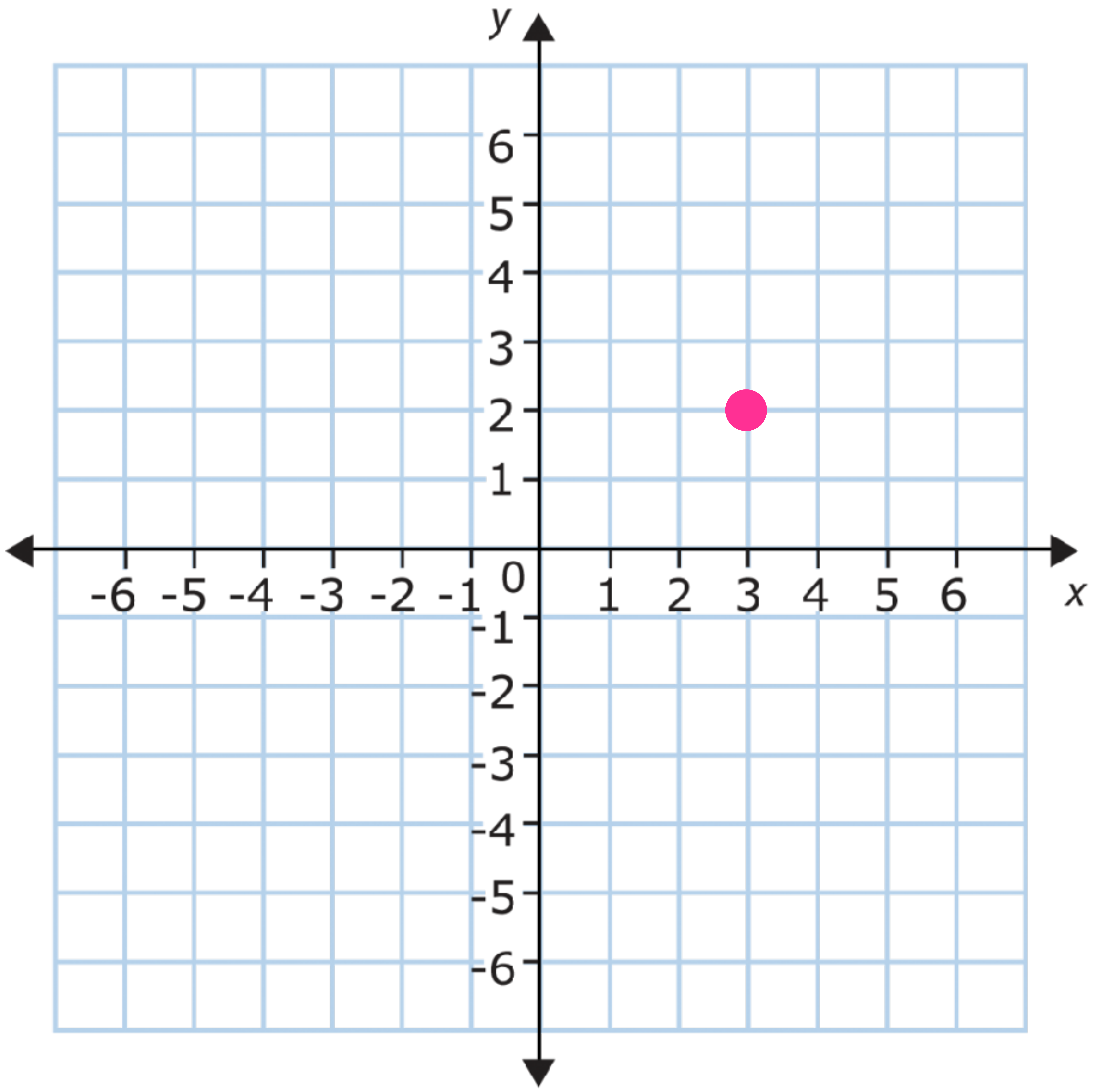
A vector class.

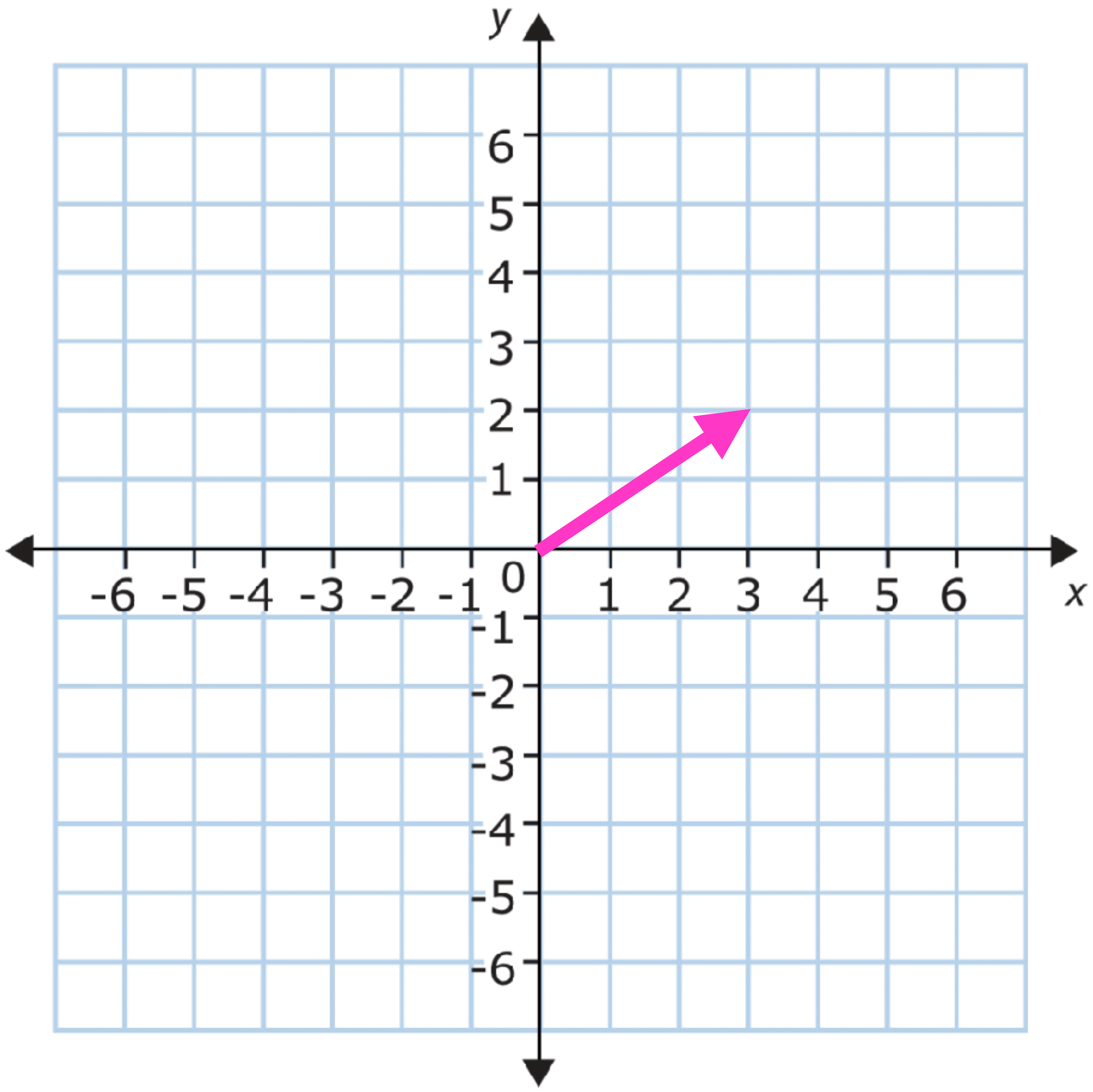
```cpp
class Vector {
    public:

        Vector();
        Vector(float x, float y, float z);

        float length();
        void normalize();

        float x;
        float y;
        float z;
};
```

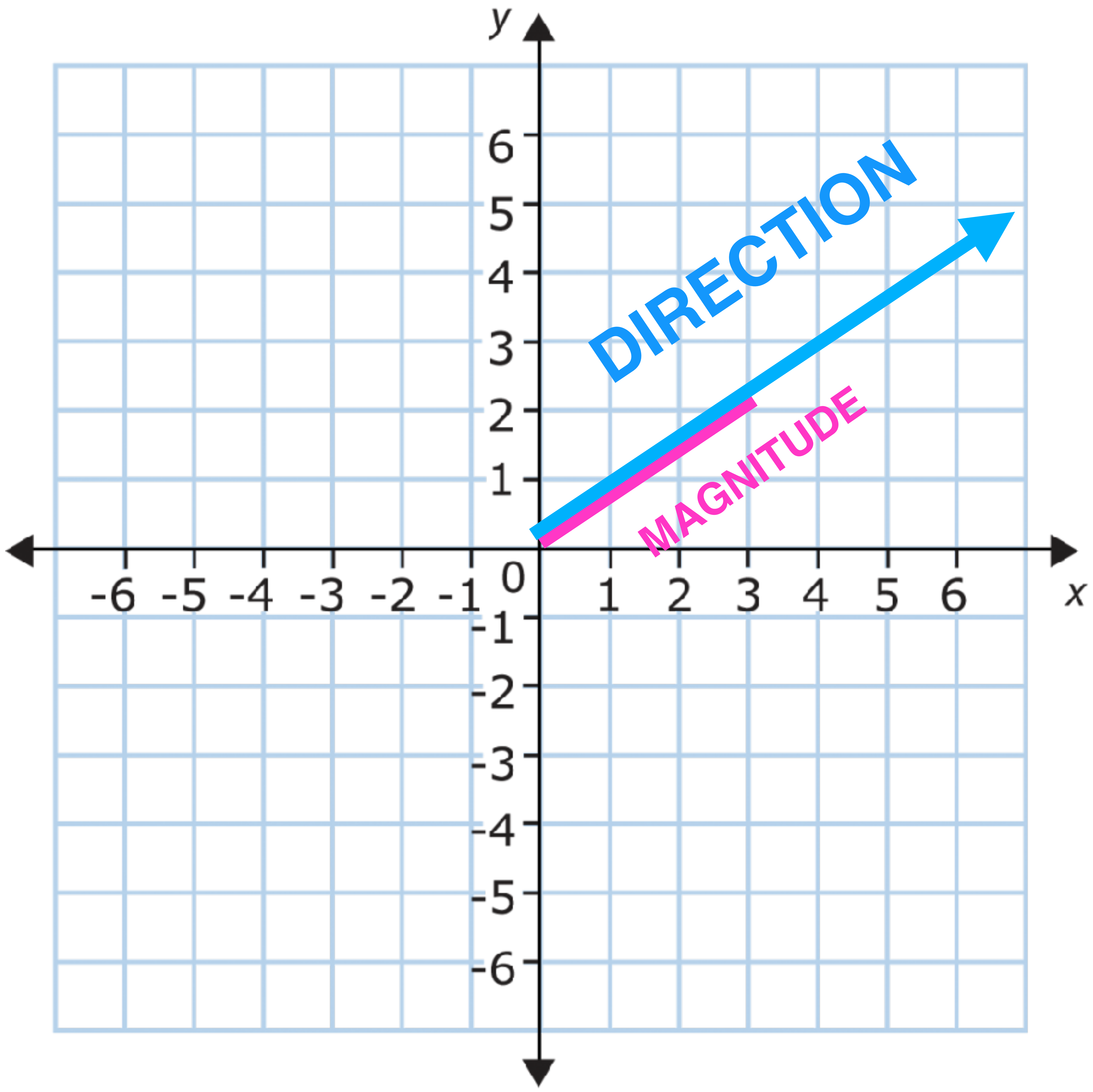
Vector length.

Use Pythagorean Theorem to get the vector length.

Normalizing a vector.

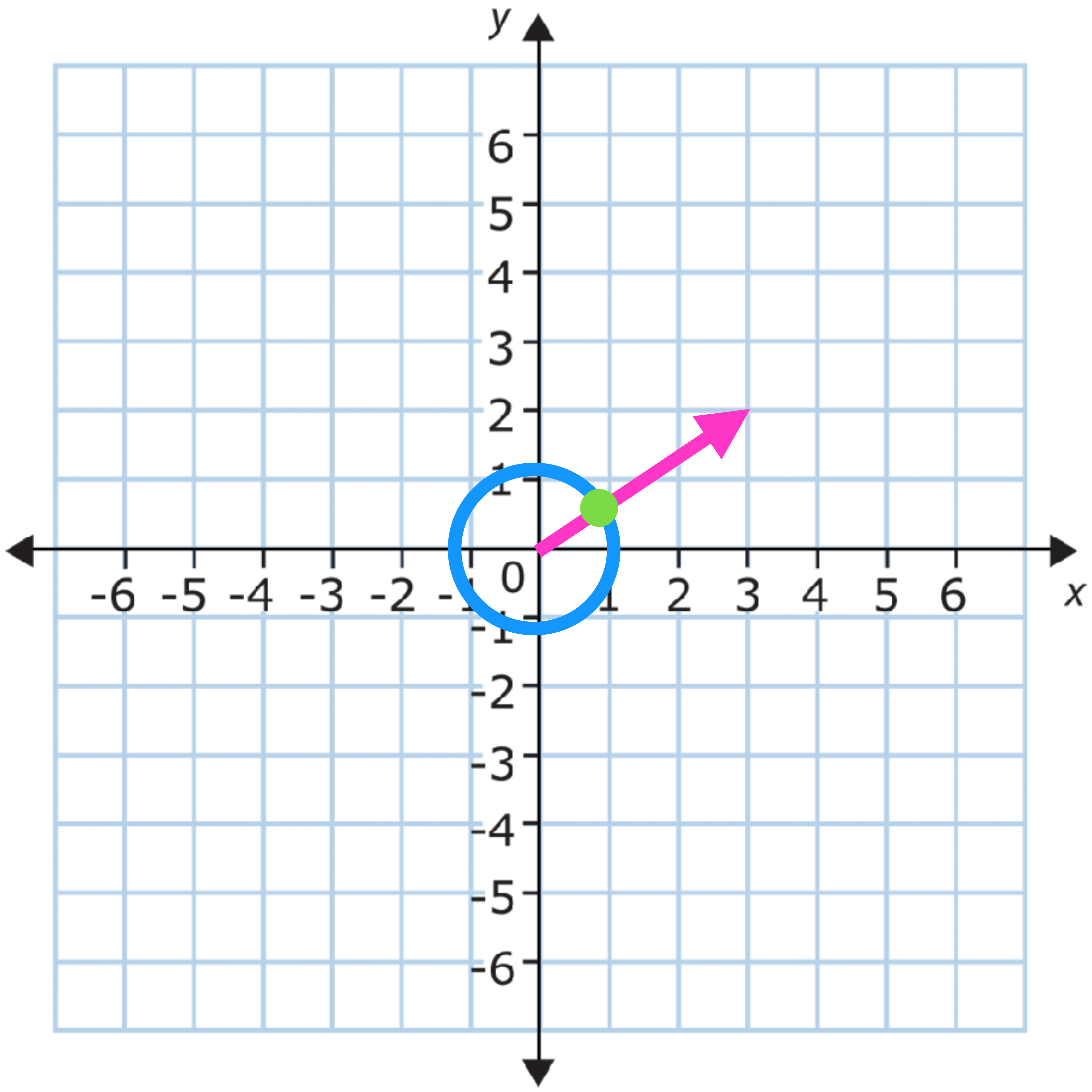

y
x
0
-6 -5 -4 -3 -2 -1
1 2 3 4 5 6
6 5 4 3 2 1
-1 -2 -3 -4 -5 -6


y
x
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1
1 2 3 4 5 6



y
x
DIRECTION
MAGNITUDE
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
6 5 4 3 2 1 -1 -2 -3 -4 -5 -6

Divide each vector component by the vector length (just be careful if the length is 0!).


y
x
0
-6 -5 -4 -3 -2 -1 1 2 3 4 5 6
6 5 4 3 2 1 -1 -2 -3 -4 -5 -6

Multiplying matrices and vectors.

```
Vector operator * (const Vector &v);
```

Creating final entity model matrix.

Identity matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale matrix.

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translate matrix.

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-axis rotation matrix.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Building the final matrix.

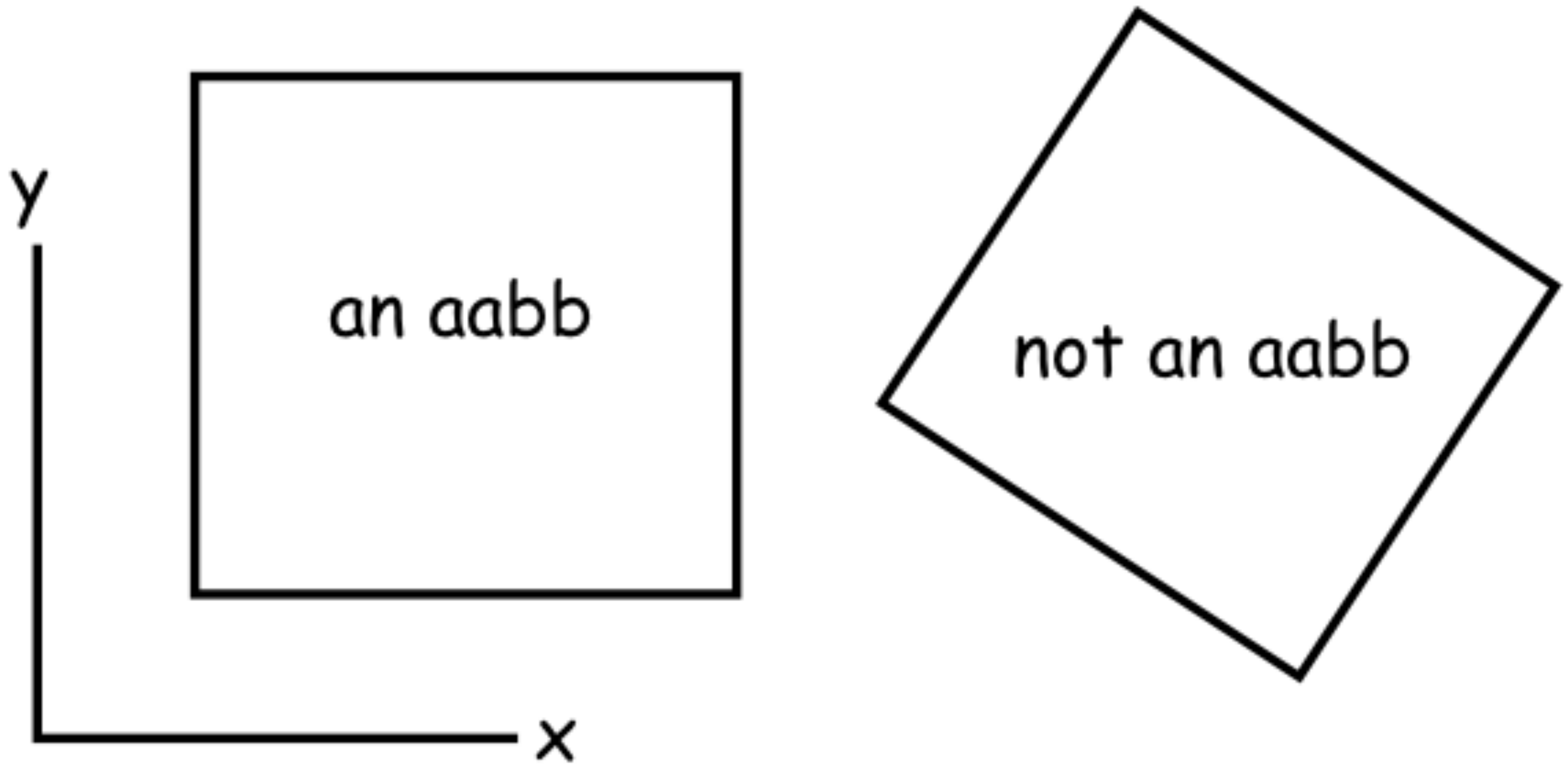
FINAL MATRIX = SCALE MATRIX * ROTATE MATRIX * TRANSLATE MATRIX

Remember that the order of matrix transform multiplication matters!

```
class Entity {
public:

    Matrix matrix;

    Vector position;
    Vector scale;
    float rotation;


};
```

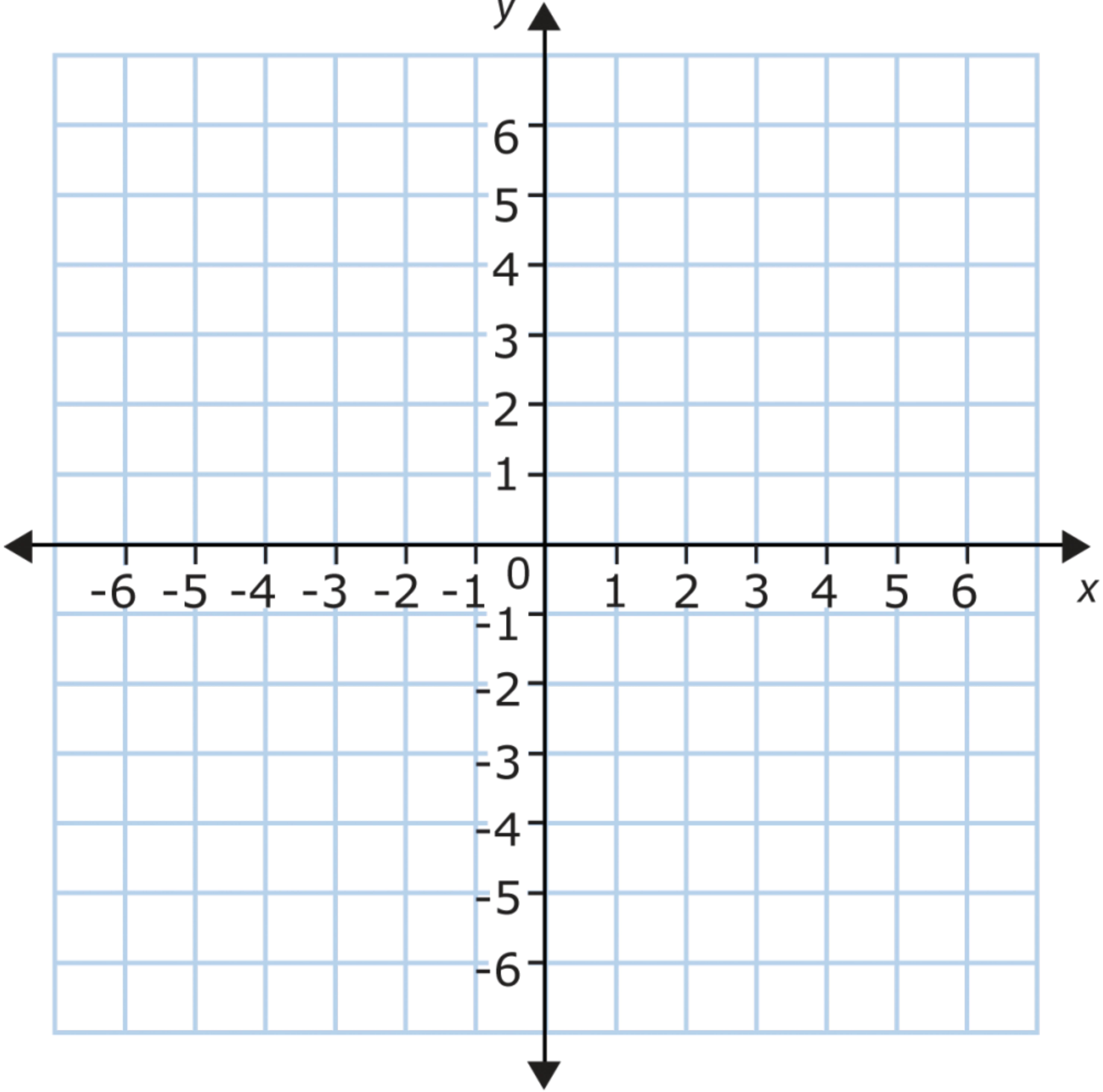
Benefits of storing the transformation matrix.

Non-axis aligned bounding boxes.

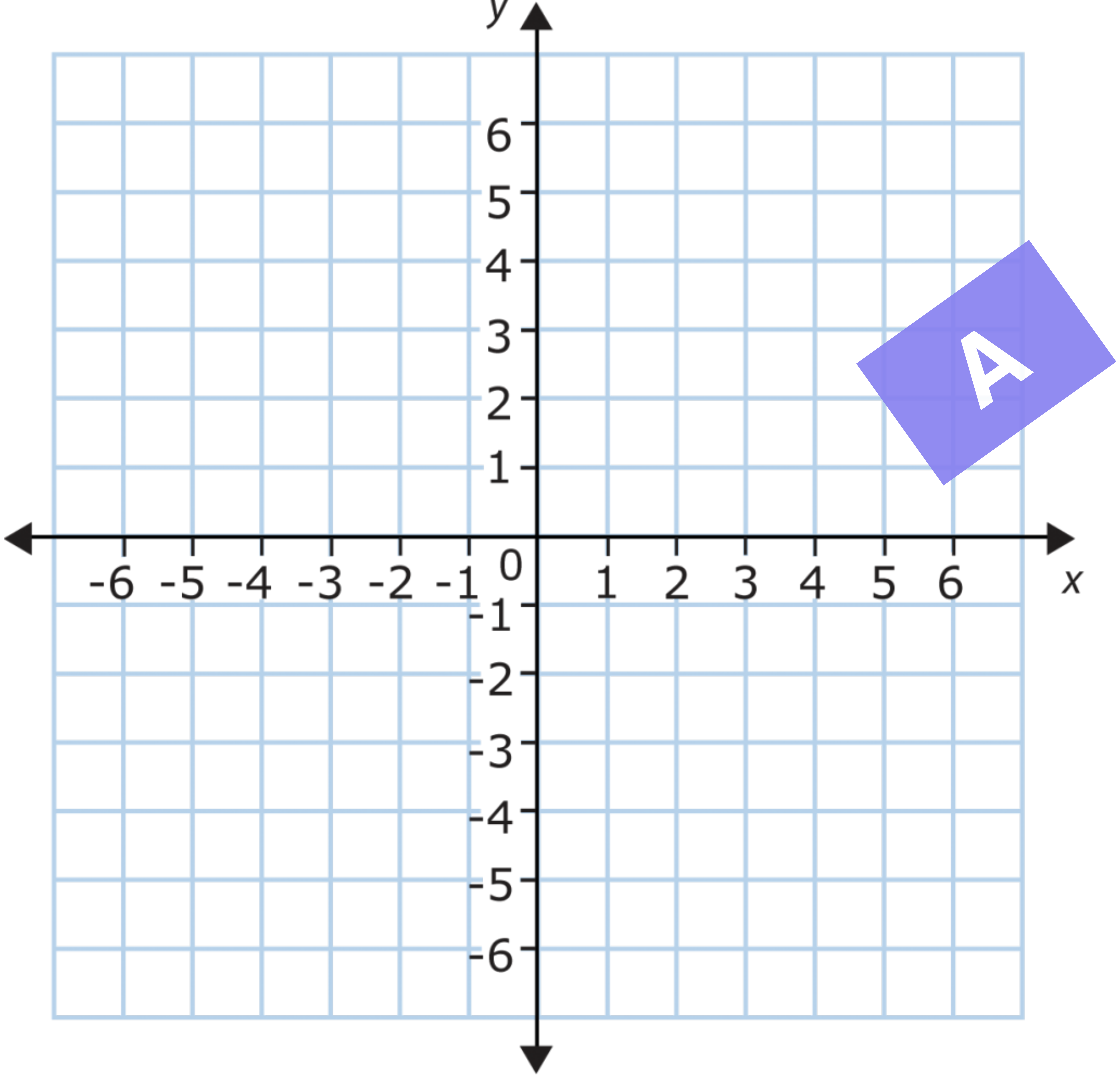

y
an aabb
x
not an aabb

Transforming between coordinate spaces.

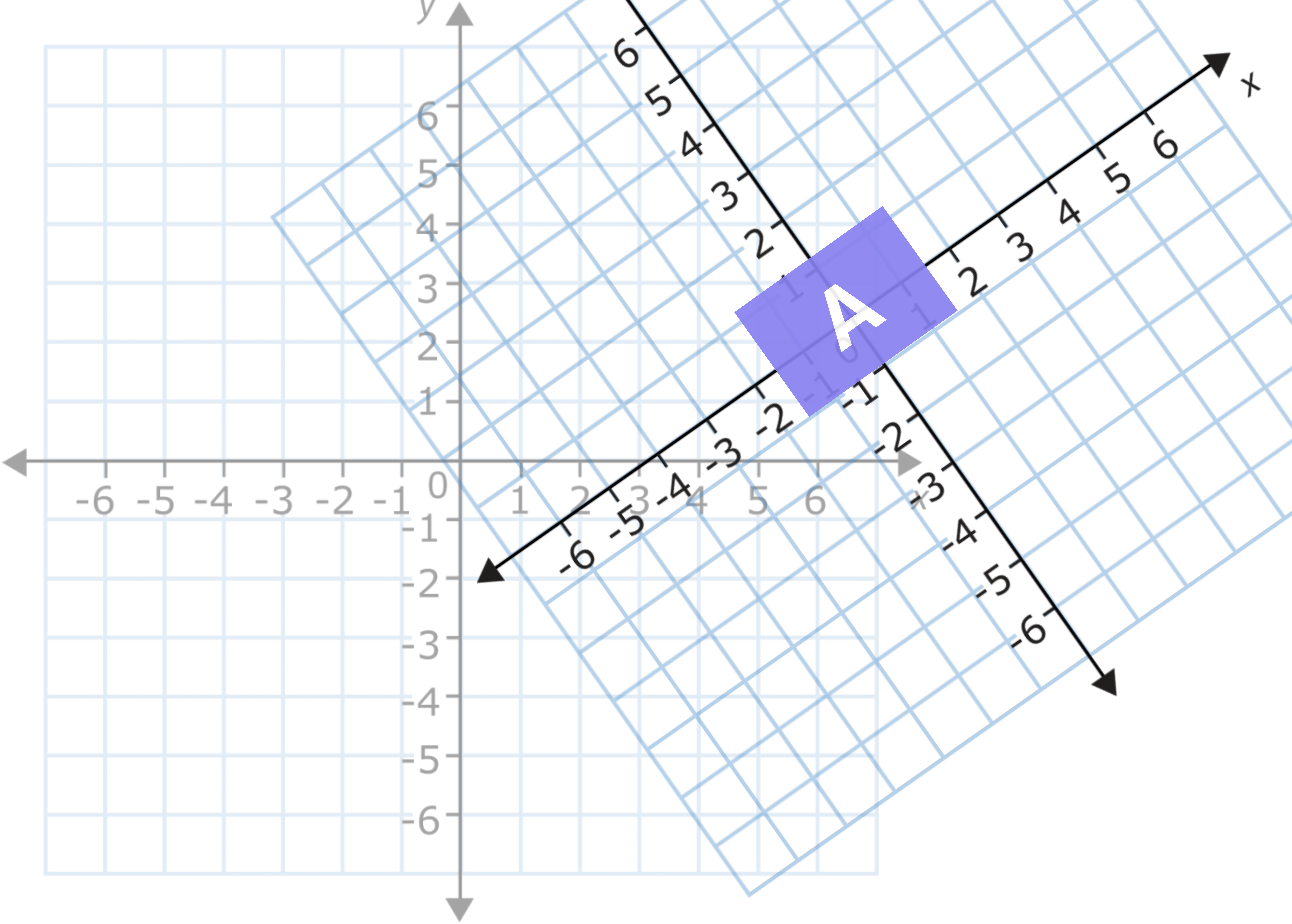
World space.



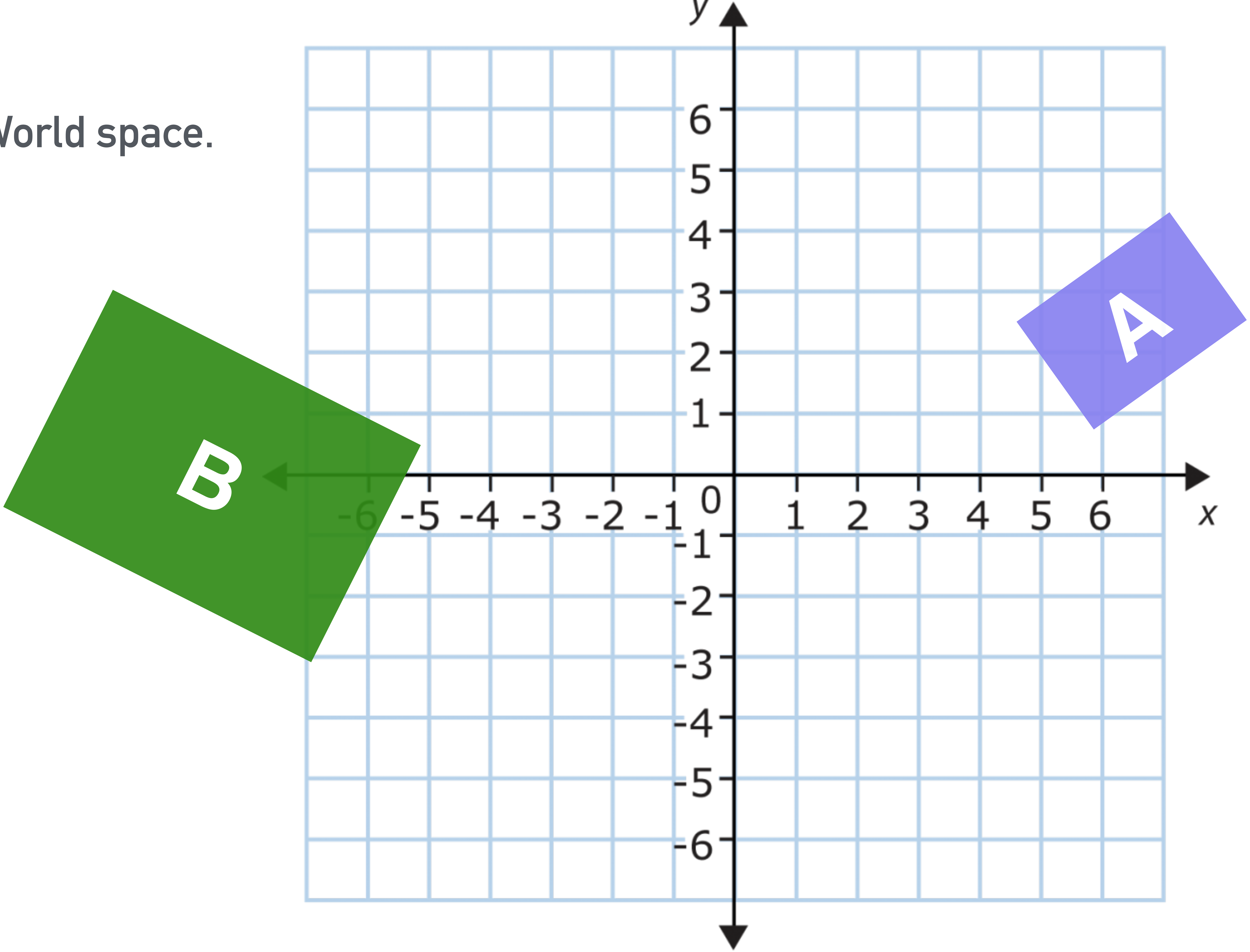
World space.

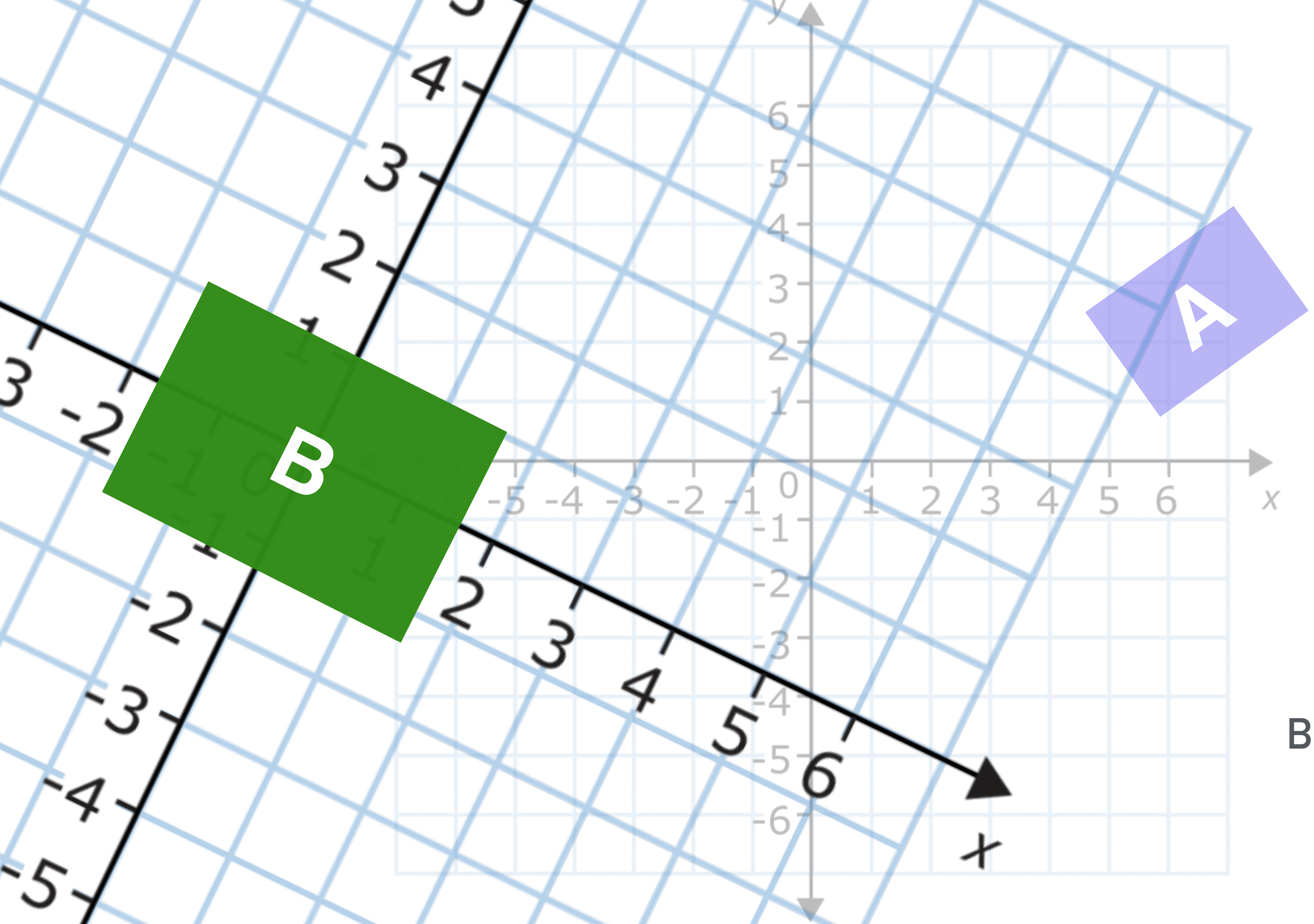


A's object space.



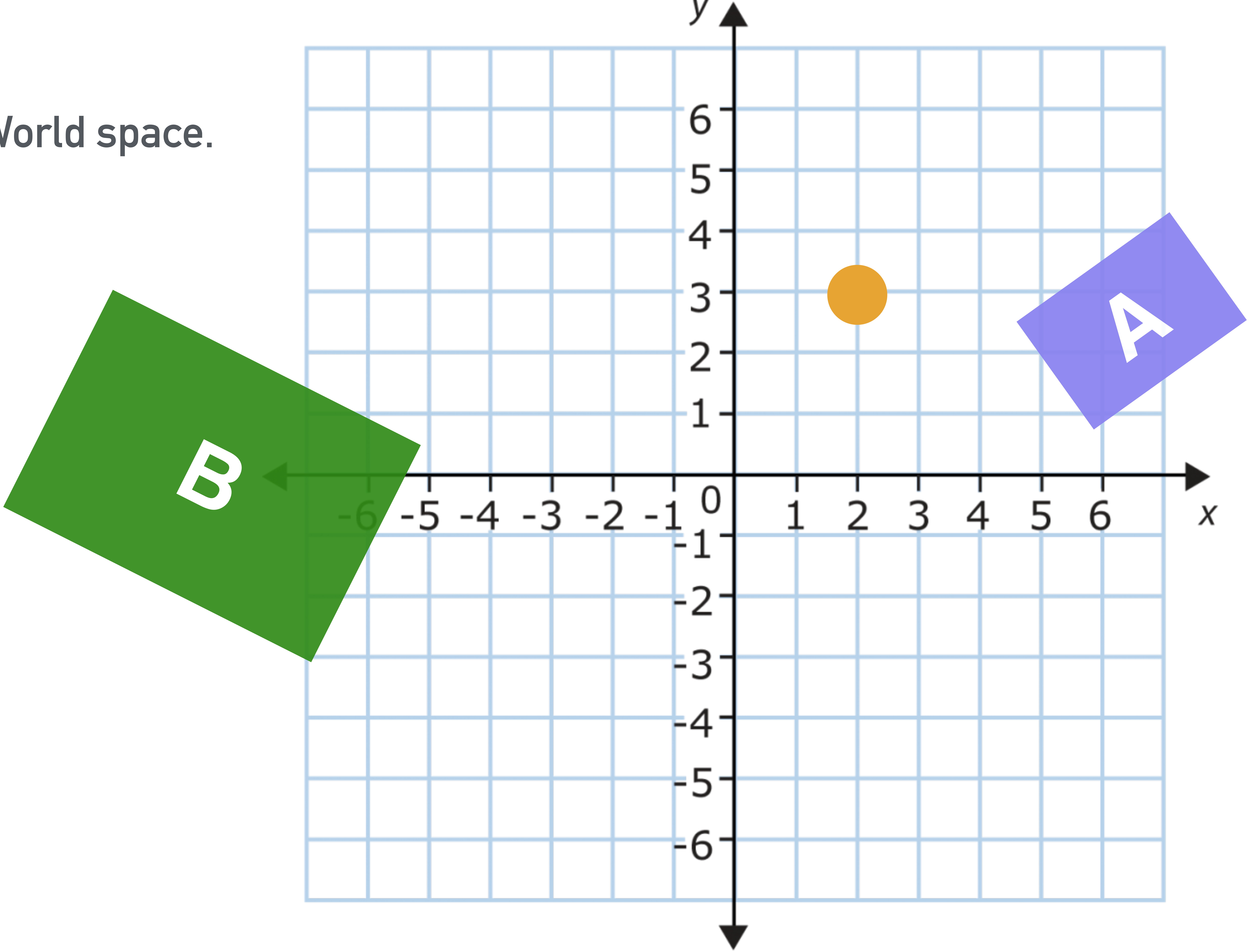
World space.



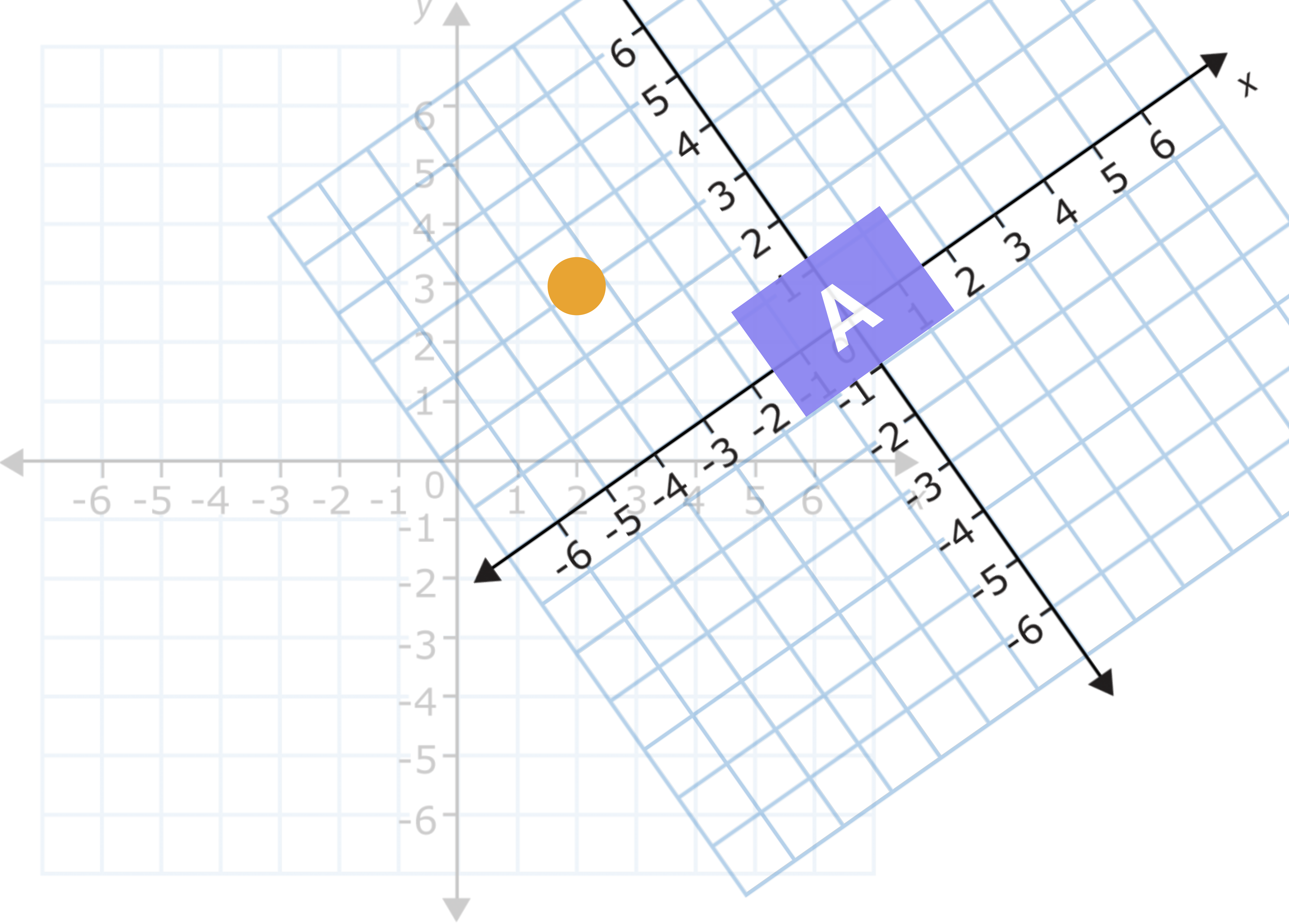


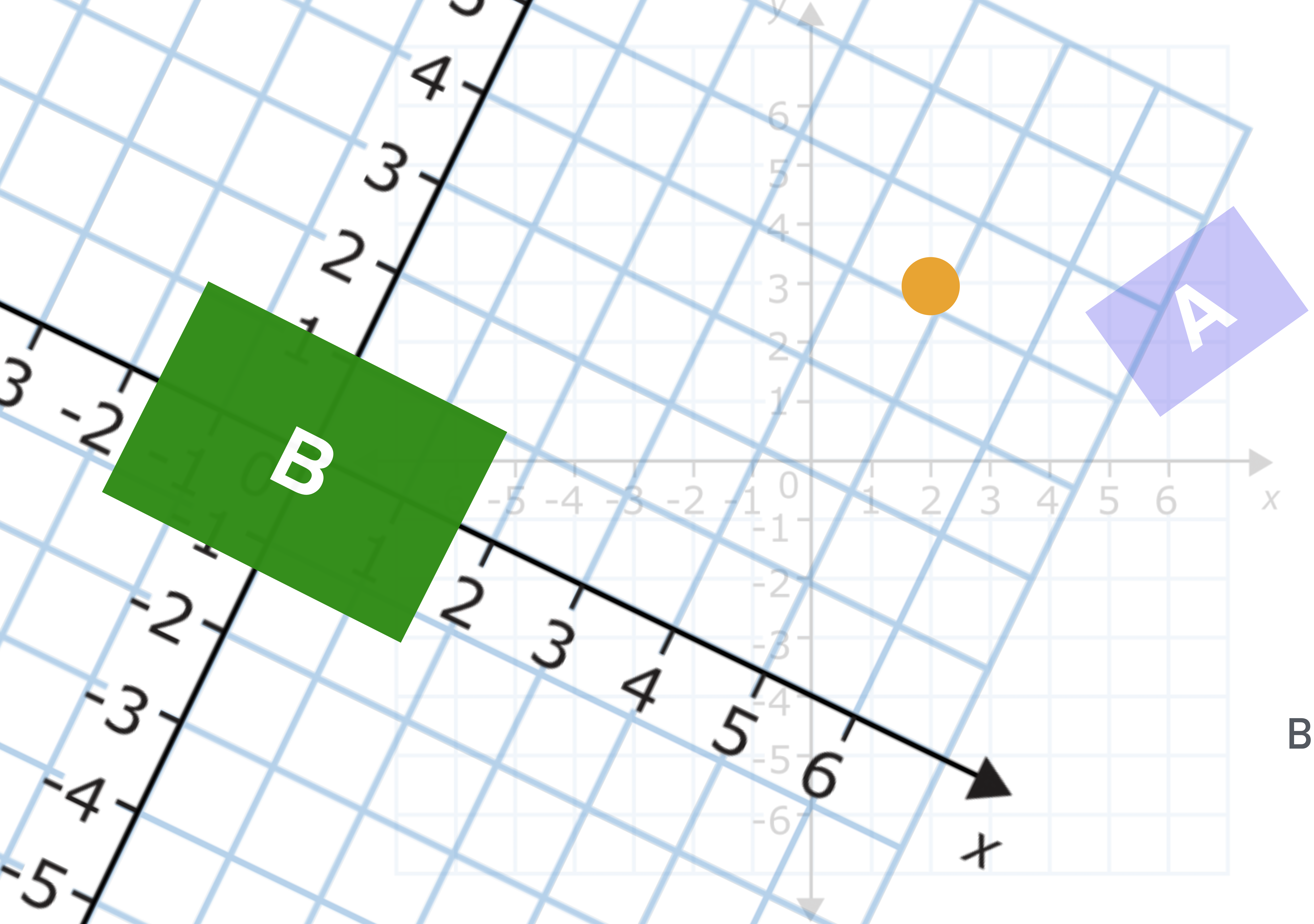
B’s object space.

World space.

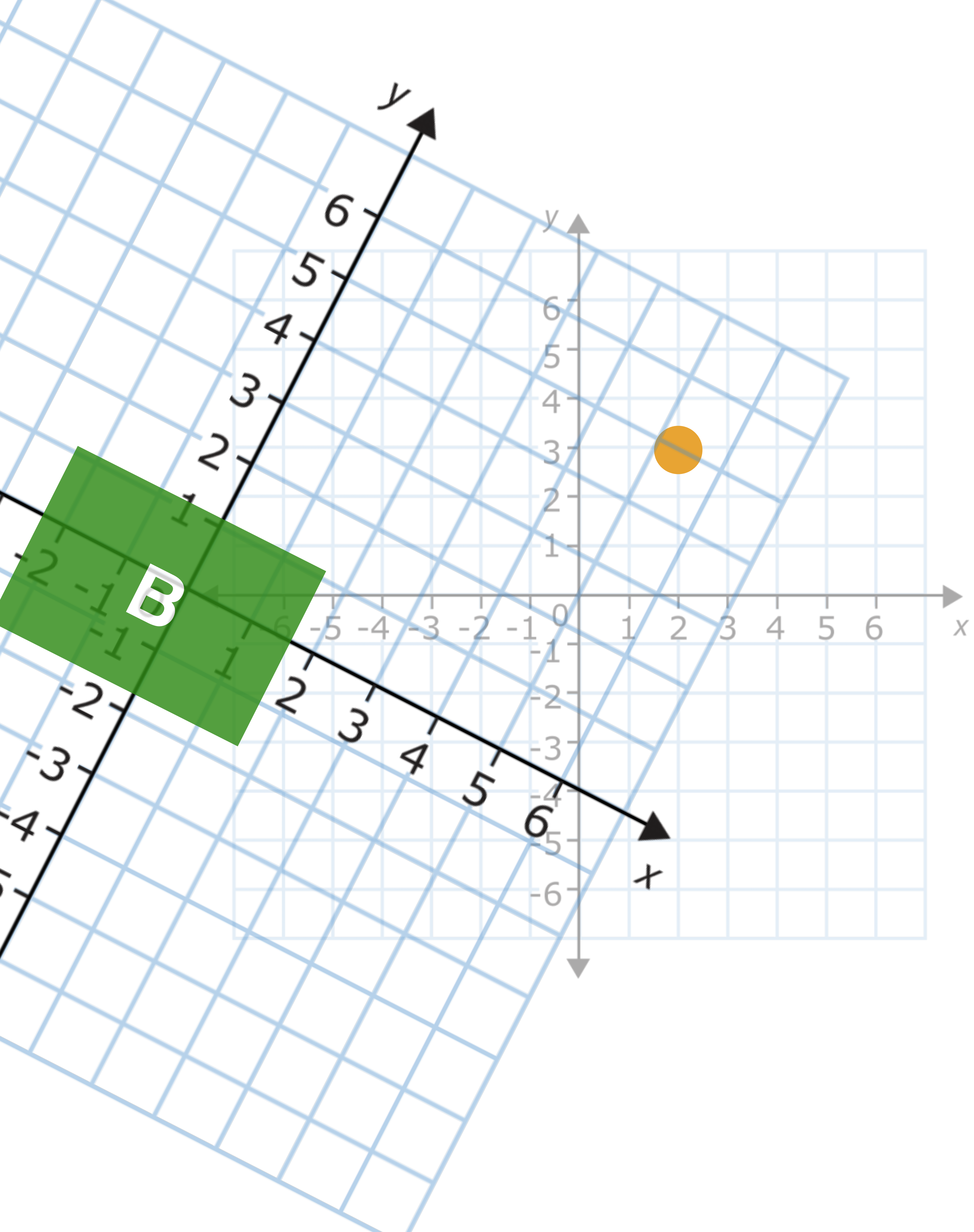


A's object space.

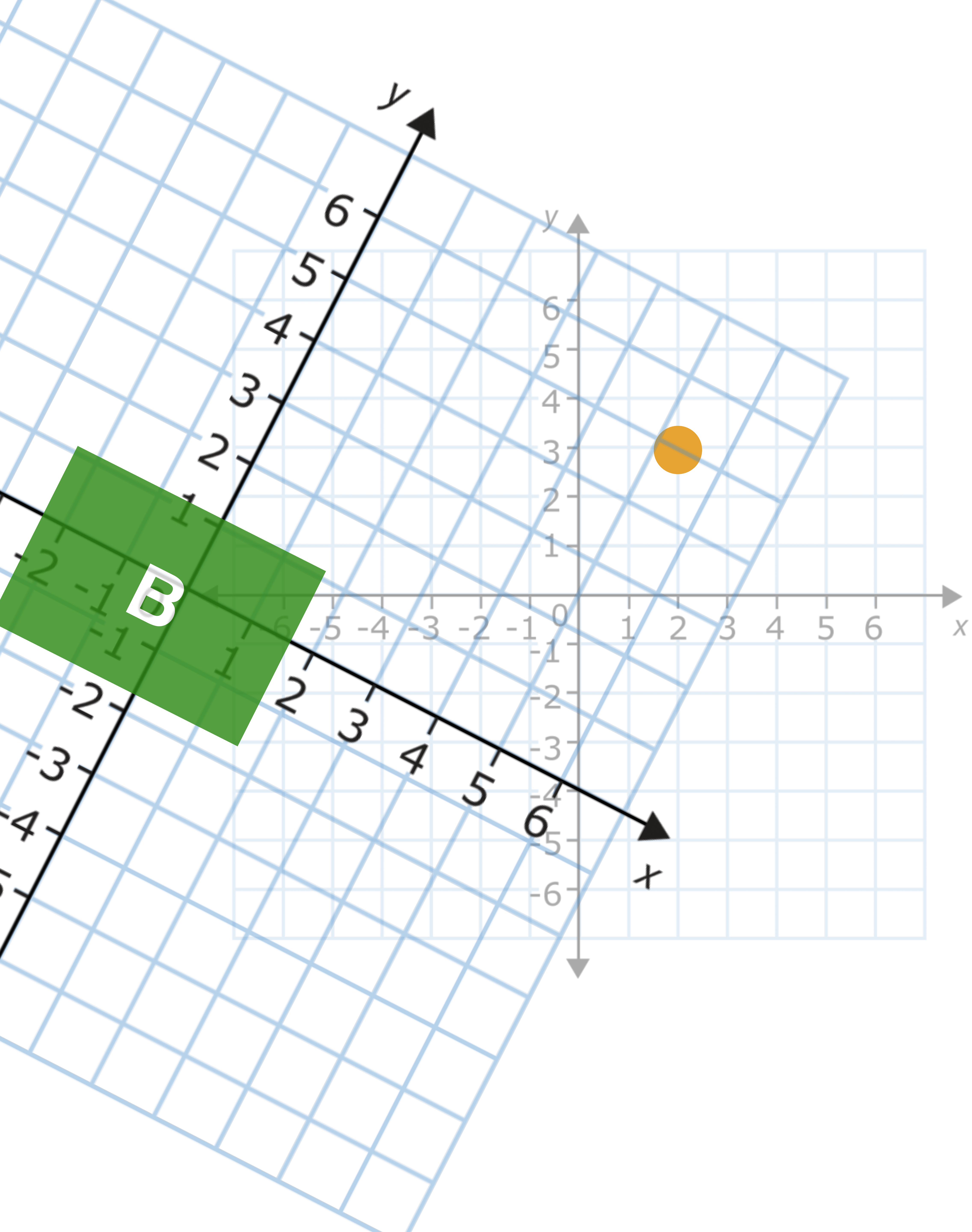




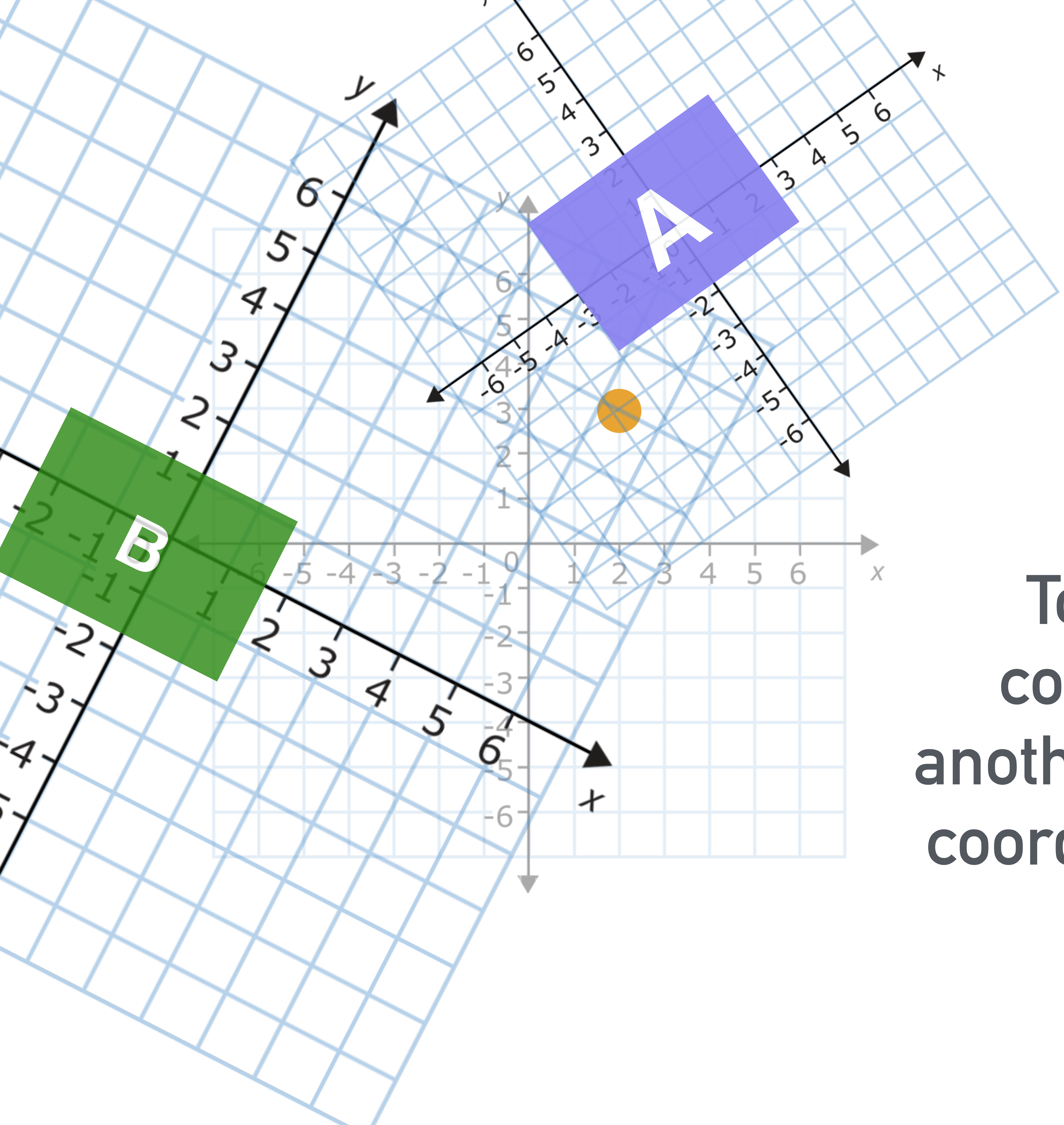
B's object space.



To express a world space coordinate vector in terms of an entity's object space, multiply the vector by the inverse of that entity's transform matrix.

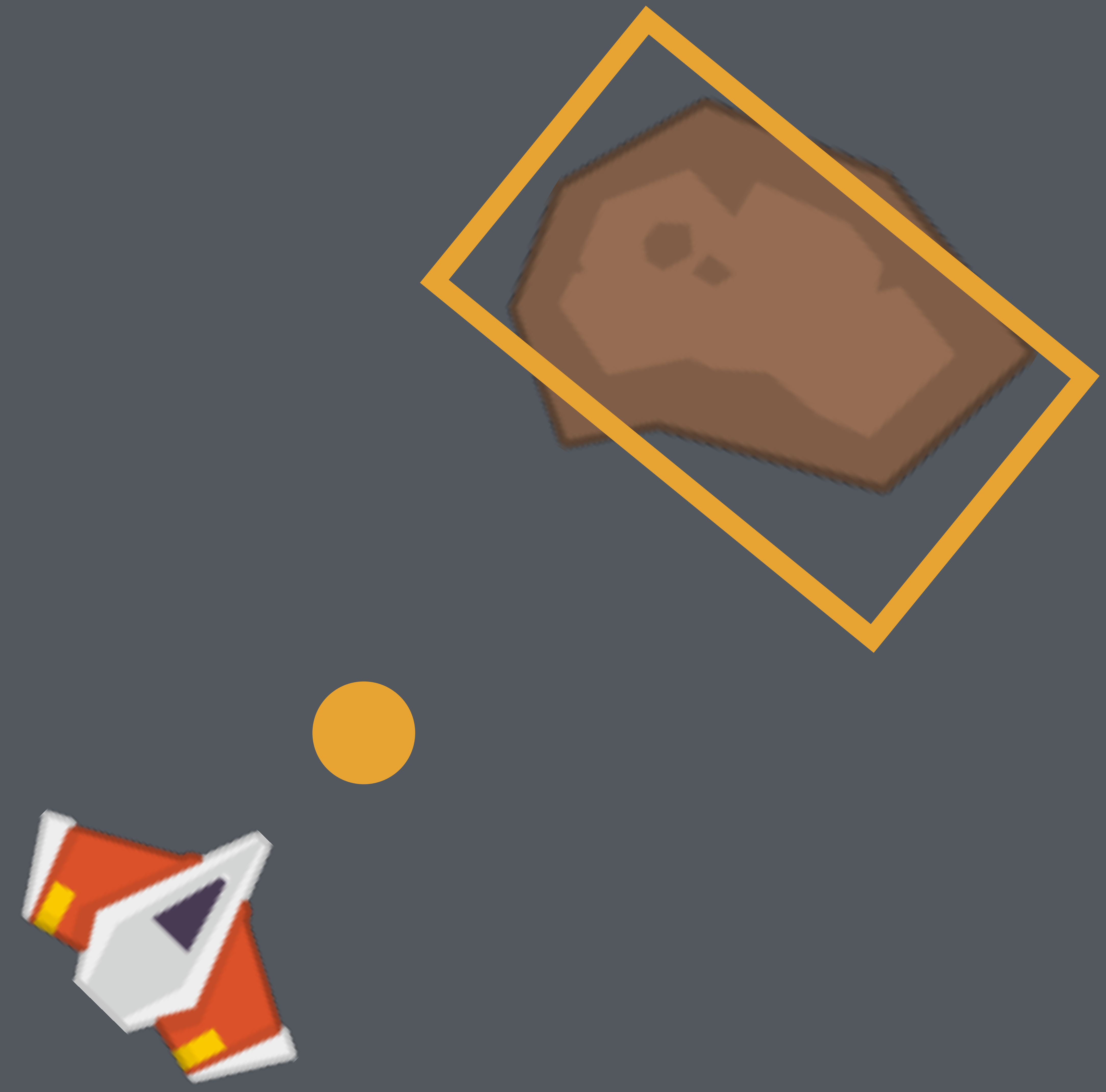


To express an object space coordinate vector in terms of world space, multiply the vector by that entity's transform matrix.

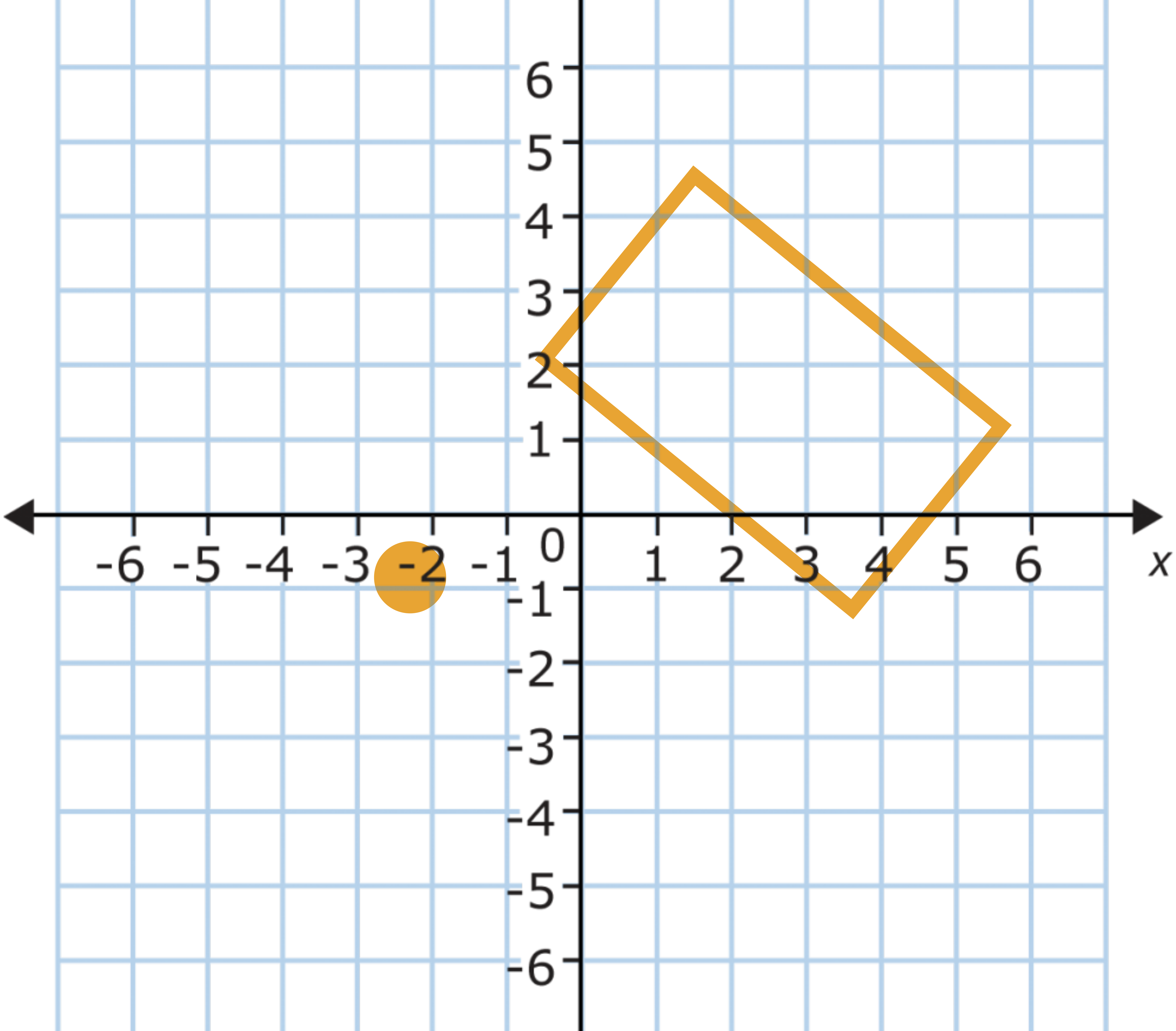


To express an object space coordinate vector in terms of another object's space, convert the coordinate to world space, then to the other object's space.

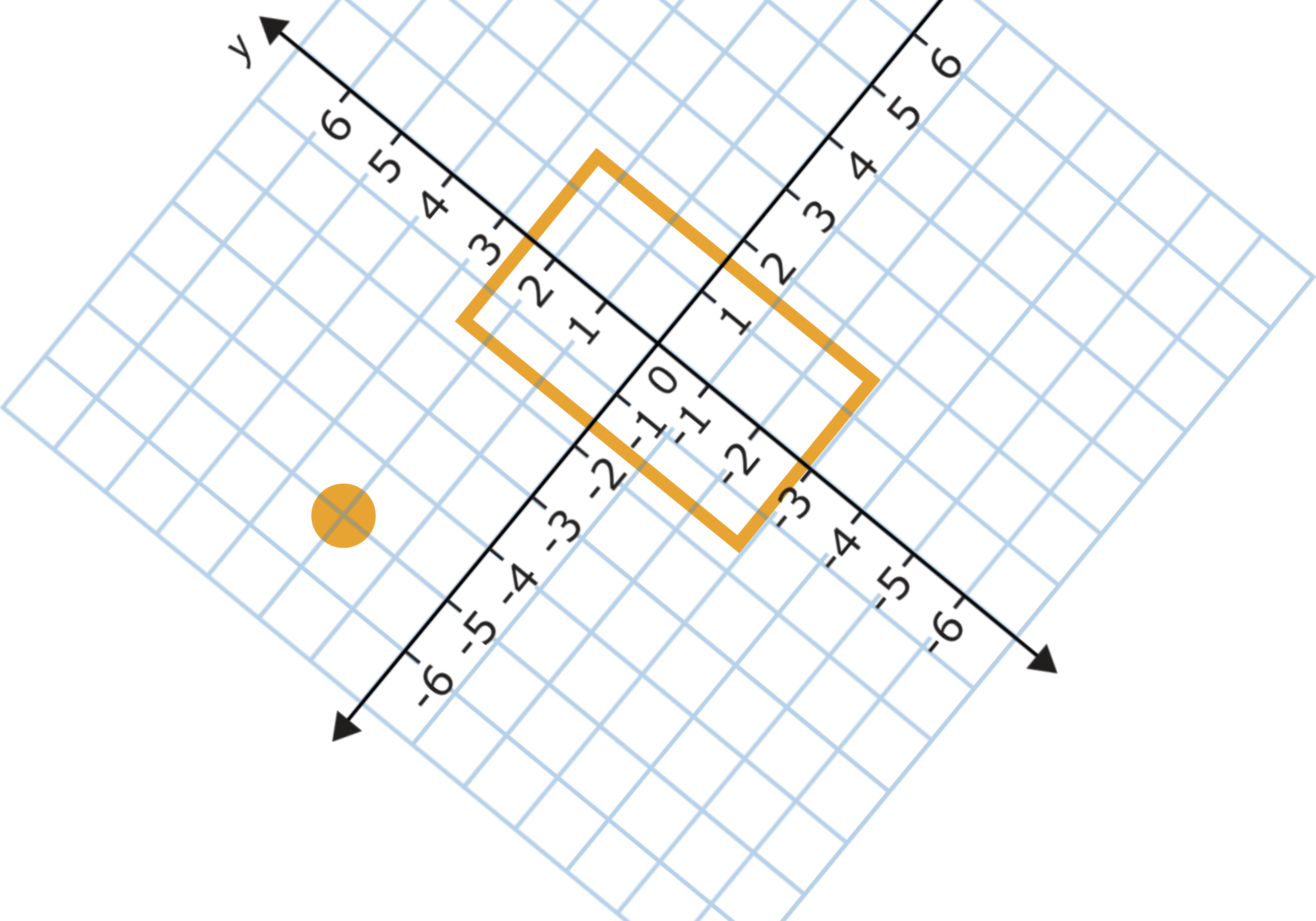
Point / rotated rectangle collision.

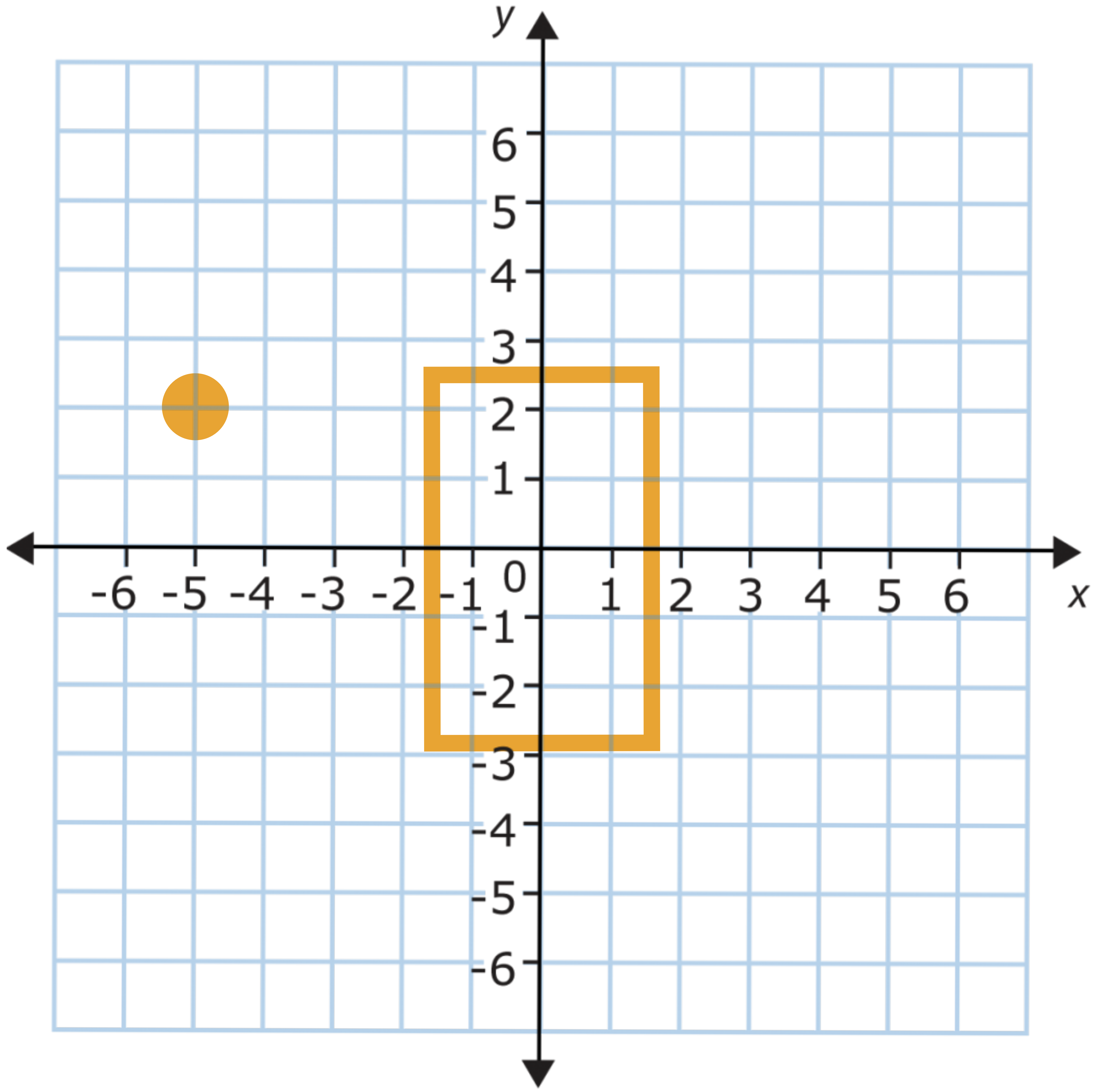


???



6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1
1 2 3 4 5 6
x


y
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6


y
x
0
-6 -5 -4 -3 -2 -1
1 2 3 4 5 6
6 5 4 3 2 1
-1 -2 -3 -4 -5 -6

Point / rotated rectangle collision.

1. Multiply the point vector by the inverse of the entity's transform matrix to bring it into the object's space.

2. Do regular point / rectangle check with the resulting coordinates.

Rotated rectangle / rotated rectangle collision.

Separating axis theorem collision.