Article

Working with Matrices

Solve simultaneous equations and transform points in space.

Overview

A matrix is a 2D array of values arranged in rows and columns. The simd library provides support for matrices of up to four rows and four columns, containing 16 elements. It uses a *column major* naming convention; for example, a `simd_double4x2` is a matrix containing four columns and two rows.

The simd library provides initializers that include options for creating matrices from either rows or columns from the appropriately sized vectors. For example, the following code uses two vectors of four elements to create a 2 x 4 matrix and a 4 x 2 matrix:

```
let x = simd_double4(x: 10, y: 20, z: 30, w: 40)
let y = simd_double4(x: 1, y: 2, z: 3, w: 4)

/*
 A matrix of two columns and four rows:

     10  1
     20  2
     30  3
     40  4
 */
let a = simd_double2x4([x, y]) // columns

/*
 A matrix of four columns and two rows:

    10  20  30  40
    1   2   3   4
*/
let b = simd_double4x2(rows: [x, y])
```

The following examples show a few common uses of matrices.

Solve Simultaneous Equations

You can use matrices to solve simultaneous equations of the form *AX* = *B*; for example, to find *x* and *y* in the following equations:

```
 2x + 4y = 2
-4x + 2y = 14
```

You first create a 2 x 2 matrix containing the left-side coefficients:

```
let a = simd_double2x2(rows: [
    simd_double2( 2, 4),
    simd_double2(-4, 2)
    ])
```

Then create a vector containing the right-side values:

```
let b = simd_double2(2, 14)
```

To find the values of *x* and *y*, multiply the inverse of the matrix `a` with the vector `b`:

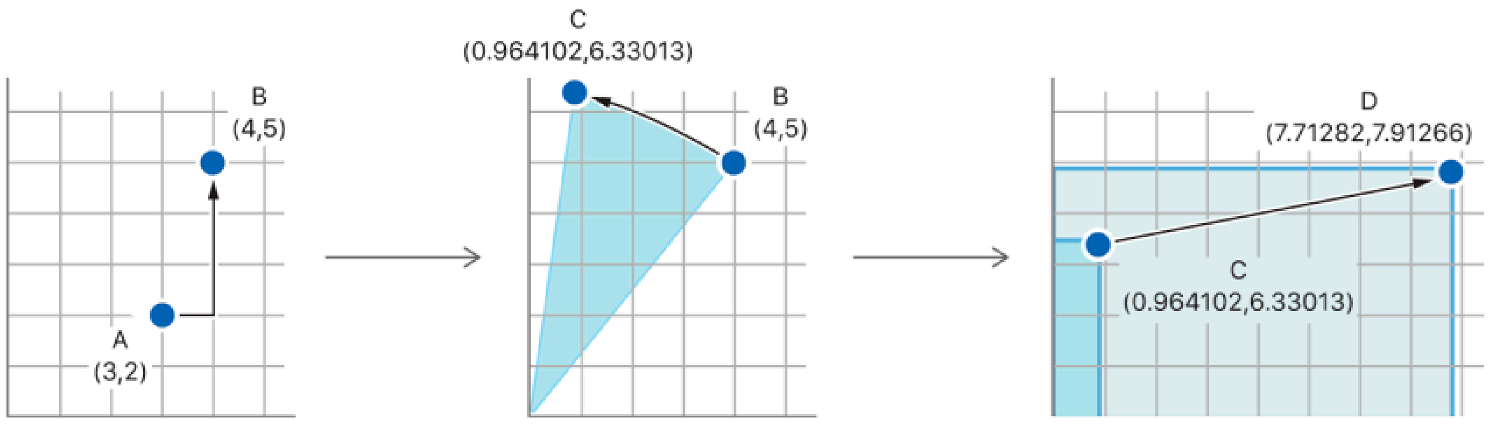
```
let x = simd_mul(a.inverse, b)
```

The result, `x`, is a two-element vector containing `(x = −2.6, y = 1.8)`.

Transform Vectors with Matrix Multiplication

Matrices provide a convenient way to transform (translate, rotate, and scale) points in 2D and 3D space.

The following image shows point *A* translated to *B*, rotated to *C*, and scaled to *D*:



1. Translate 2. Rotate 3. Scale

By representing 2D coordinates as a three-element vector, you can transform points using matrix multiplication. Typically, the third component of the vector, `z`, is set to 1, which indicates that the vector represents a position in space.

For example, the vector shown as **A** in the preceding illustration is defined as a `simd_float3` with the following code:

```
let positionVector = simd_float3(x: 3, y: 2, z: 1)
```

Transform matrices for 2D coordinates are represented by 3 x 3 matrices.

Translate

A translate matrix takes the following form:

1	**0**	**tx**
0	1	ty
0	0	1

The simd library provides constants for identity matrices (matrices with ones along the diagonal, and zeros elsewhere). The 3 x 3 `Float` identity matrix is `matrix_identity_float3x3`.

The following function returns a `simd_float3x3` matrix using the specified `tx` and `ty` translate values by setting the elements in an identity matrix:

```
func makeTranslationMatrix(tx: Float, ty: Float) -> simd_float3x3 {
    var matrix = matrix_identity_float3x3

    matrix[2, 0] = tx
    matrix[2, 1] = ty

    return matrix
}
```

To apply a translate to the position vector, you multiply the pair together:

```
let translationMatrix = makeTranslationMatrix(tx: 1, ty: 3)
let translatedVector = translationMatrix * positionVector
```

The resulting `translatedVector` has the values `(x: 4.0, y: 5.0, z: 1.0)`, shown as **B** in the illustration above.

Rotate

A rotation matrix around the z-axis (that is, on the xy plane) takes the following form:

`cos(angle)`	`-sin(angle)`	`0`
`sin(angle)`	`cos(angle)`	`0`
`0`	`0`	`1`

The following function returns a `simd_float3x3` matrix using the specified rotation angle in radians:

```
func makeRotationMatrix(angle: Float) -> simd_float3x3 {
    let rows = [
        simd_float3(cos(angle), -sin(angle), 0),
        simd_float3(sin(angle), cos(angle), 0),
        simd_float3(0,          0,          1)
    ]

    return float3x3(rows: rows)
}
```

To apply a rotation to the previously translated vector, you multiply the pair together:

```
let angle = Measurement(value: 30,
                        unit: UnitAngle.degrees)
```

```
let radians = Float(angle.converted(to: .radians).value)

let rotationMatrix = makeRotationMatrix(angle: radians)
let rotatedVector = rotationMatrix * translatedVector
```

The resulting `rotatedVector` has the values `(x: 0.964102, y: 6.33013, z: 1.0)`, shown as **C** in the illustration above.

Scale

A scale matrix takes the following form:

xScale	0	0
0	yScale	0
0	0	1

The following function returns a `simd_float3x3` matrix using the specified x and y scale values:

```
func makeScaleMatrix(xScale: Float, yScale: Float) -> simd_float3x3 {
    let rows = [
        simd_float3(xScale,      0, 0),
        simd_float3(     0, yScale, 0),
        simd_float3(     0,      0, 1)
    ]

    return float3x3(rows: rows)
}
```

To apply a scale to the previously rotated vector, you multiply the pair together:

```
let scaleMatrix = makeScaleMatrix(xScale: 8, yScale: 1.25)
let scaledVector = scaleMatrix * rotatedVector
```

The resulting `scaledVector` has the values `(x: 7.71282, y: 7.91266, z: 1.0)`, shown as **D** in the illustration above.

The three transform matrices can be multiplied together and the product multiplied with the position vector to get the same result:

```swift
let transformMatrix = scaleMatrix * rotationMatrix * translationMatrix
let transformedVector = transformMatrix * positionVector
```

See Also

Vectors, Matrices, and Quaternions

Working with Vectors

Use vectors to calculate geometric values, calculate dot products and cross products, and interpolate between values.

Working with Quaternions

Rotate points around the surface of a sphere, and interpolate between them.

Rotating a cube by transforming its vertices

Rotate a cube through a series of keyframes using quaternion interpolation to transition between them.

simd

Perform computations on small vectors and matrices.

vForce

Perform transcendental and trigonometric functions on vectors of any length.