```
import Spatial.Base
import Spatial.SPAffineTransform3D
import Spatial.SPAngle
import Spatial.SPPoint3D
import Spatial.SPPose3D
import Spatial.SPProjectiveTransform3D
import Spatial.SPRay3D
import Spatial.SPRect3D
import Spatial.SPRotation3D
import Spatial.SPRotationAxis3D
import Spatial.SPScaledPose3D
import Spatial.SPSize3D
import Spatial.SPSphericalCoordinates3D
import Spatial.SPVector3D
import Spatial.Structures
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/// Returns the `Rotatable3D` entity
rotated by the specified rotation.
///
/// - Parameter lhs: The rotation.
/// - Parameter rhs: The `Rotatable3D`
entity .
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@inlinable public func * <T>(lhs:
Rotation3D, rhs: T) -> T where T :
Rotatable3D

/// The axis of a shear transform.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public enum AxisWithFactors {

    /// The shear is on the _x_ axis
using the _y_ and _z_ shear factors.
    case xAxis(yShearFactor: Double,
zShearFactor: Double)

    /// The shear is on the _y_ axis
using the _x_ and _z_ shear factors.
    case yAxis(xShearFactor: Double,
zShearFactor: Double)

    /// The shear is on the _z_ axis
using the _x_ and _y_ shear factors.
    case zAxis(xShearFactor: Double,
yShearFactor: Double)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public struct Dimension3DSet : OptionSet
{

    /// The corresponding value of the
raw type.
    public let rawValue: Int

    /// The x dimension.
    public static let x: Dimension3DSet

    /// The y dimension.
    public static let y: Dimension3DSet
    /// The z dimension.
    public static let z: Dimension3DSet

    /// All dimensions.
    public static let all: Dimension3DSet

    /// Returns a newly initialized
dimension set using the provided integer.
    public init(rawValue: Int)

    /// The type of the elements of an
array literal.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias ArrayLiteralElement
= Dimension3DSet

    /// The element type of the option
set.
    ///
    /// To inherit all the default
implementations from the `OptionSet`
protocol,
    /// the `Element` type must be
`Self`, the default.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Element =
Dimension3DSet

    /// The raw type that can be used to
represent all values of the conforming
```

```
    /// type.
    ///
    /// Every distinct value of the
conforming type has a corresponding
unique
    /// value of the `RawValue` type, but
there may be values of the `RawValue`
    /// type that don't have a
corresponding value of the conforming
type.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias RawValue = Int
}

/// A set of methods common to Spatial
primitives.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Primitive3D : Decodable,
Encodable, Equatable {

    /// A primitive with zero values.
    static var zero: Self { get }

    /// A primitive with infinite values.
    static var infinity: Self { get }

    /// Returns whether the primitive is
zero.
    var isZero: Bool { get }

    /// Returns whether the primitive is
```

```
finite.
    var isFinite: Bool { get }

    /// Returns whether the primitive
contains any `NaN` values.
    var isNaN: Bool { get }

    /// Returns the primitive resulting
from applying an affine transform to the
primitive.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed
primitive.
    func applying(_ transform:
AffineTransform3D) -> Self

    /// Returns the primitive resulting
from applying a projective transform to
the primitive.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed
primitive.
    func applying(_ transform:
ProjectiveTransform3D) -> Self

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    func applying(_ pose: Pose3D) -> Self

    /// Applies an affine transform.
    ///
    /// - Parameter transform: The affine
transform.
    mutating func apply(_ transform:
AffineTransform3D)

    /// Applies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    mutating func apply(_ transform:
ProjectiveTransform3D)

    /// Applies a pose.
    ///
    /// - Parameter pose: The pose.
    mutating func apply(_ pose: Pose3D)

    /// Returns the primitive resulting
from unapplying an affine transform to
the primitive.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed
primitive.
    func unapplying(_ transform:
AffineTransform3D) -> Self
```

```
    /// Unapplies an affine transform.
    mutating func unapply(_ transform:
AffineTransform3D)

    /// Returns the primitive resulting
from unapplying a projective transform to
the primitive.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed
primitive.
    func unapplying(_ transform:
ProjectiveTransform3D) -> Self

    /// Returns the primitive resulting
from unapplying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    func unapplying(_ pose: Pose3D) ->
Self

    /// Unapplies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    mutating func unapply(_ transform:
ProjectiveTransform3D)

    /// Unapplies a pose.
    ///
    /// - Parameter pose: The pose.
    mutating func unapply(_ pose: Pose3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Primitive3D {

    /// Applies an affine transform.
    ///
    /// - Parameter transform: The affine
transform.
    public mutating func apply(_
transform: AffineTransform3D)

    /// Applies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    public mutating func apply(_
transform: ProjectiveTransform3D)

    /// Applies a pose.
    ///
    /// - Parameter pose: The pose.
    public mutating func apply(_ pose:
Pose3D)

    /// Unapplies an affine transform.
    ///
    /// - Parameter transform: The affine
transform.
```

```
    /// - Note: When unapplying
transforms on rectangle structures, the
transform
    /// must be rectilinear, otherwise
this function has no effect.
    public mutating func unapply(_
transform: AffineTransform3D)

    /// Unapplies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Note: When unapplying
transforms on rectangle structures, the
transform
    /// must be rectilinear, otherwise
this function has no effect.
    public mutating func unapply(_
transform: ProjectiveTransform3D)

    /// Unapplies a pose.
    ///
    /// - Parameter pose: The projective
transform.
    /// - Note: When unapplying
transforms on size and rectangle
structures, the transform
    /// must be rectilinear, otherwise
this function has no effect.
    public mutating func unapply(_ pose:
Pose3D)
}
```

```
/// A set of methods that defines the
interface for Spatial entities that can
rotate.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Rotatable3D {

    /// Returns the entity rotated by the
specified rotation around the origin.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns The rotated primitive.
    func rotated(by rotation: Rotation3D)
-> Self

    /// Rotates the entity rotated by the
specified rotation around the origin.
    ///
    /// - Parameter rotation: The
rotation.
    mutating func rotate(by rotation:
Rotation3D)

    /// Returns the entity rotated by the
specified quaternion around the origin.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns The rotated primitive.
    func rotated(by quaternion:
simd_quatd) -> Self
```

```
    /// Rotates the entity rotated by the
specified quaternion around the origin.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    mutating func rotate(by quaternion:
simd_quatd)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotatable3D {

    /// Rotates the entity by the
specified rotation around the origin.
    ///
    /// - Parameter rotation: The
rotation.
    public mutating func rotate(by
rotation: Rotation3D)

    /// Rotates the entity by the
specified quaternion around the origin.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    public mutating func rotate(by
rotation: simd_quatd)
}

/// A set of methods that defines the
interface for Spatial entities that can
scale.
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Scalable3D {

    /// Returns the entity scaled by the
specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled entity.
    func scaledBy(x: Double, y: Double,
z: Double) -> Self

    /// Scales the entity by the
specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    mutating func scaleBy(x: Double, y:
Double, z: Double)

    /// Returns the entity scaled by the
specified size.
    ///
    /// - Parameter size: The size
```

```
structure that defines that scale.
    /// - Returns The scaled entity.
    func scaled(by size: Size3D) -> Self

    /// Scales the entity by the
specified size.
    ///
    /// - Parameter size: The size
structure that defines that scale.
    mutating func scale(by size: Size3D)

    /// Returns the entity uniformly
scaled by the specified scalar value.
    ///
    /// - Parameter scale: The value that
defines that scale.
    /// - Returns The scaled entity.
    func uniformlyScaled(by scale:
Double) -> Self

    /// Uniformly scales the entity by
the specified scalar value.
    ///
    /// - Parameter scale: The value that
defines that scale.
    mutating func uniformlyScale(by
scale: Double)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Scalable3D {
    /// Scales the entity by the
specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    public mutating func scaleBy(x:
Double = 1, y: Double = 1, z: Double = 1)

    /// Scales the entity by the
specified size.
    ///
    /// - Parameter size: The size
structure that defines that scale.
    public mutating func scale(by scale:
Size3D)

    /// Uniformly scales the entity by
the specified scalar value.
    ///
    /// - Parameter scale: The value that
defines that scale.
    public mutating func
uniformlyScale(by scale: Double)
}

/// A set of methods that defines the
interface for Spatial entities that can
shear.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Shearable3D {

    /// Returns a sheared entity.
    ///
    /// - Parameter shear: The axis and
shear factors.
    /// - Returns The sheared entity.
    func sheared(_ shear:
AxisWithFactors) -> Self

    /// Shears the entity.
    ///
    /// - Parameter shear: The axis and
shear factors.
    mutating func shear(_ shear:
AxisWithFactors)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Shearable3D {

    /// Shears the entity.
    ///
    /// - Parameter shear: The axis and
shear factors.
    public mutating func shear(_ shear:
AxisWithFactors)
}

/// A set of methods that defines the
interface for Spatial entities that can
```

```
translate.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Translatable3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    func translated(by size: Size3D) ->
Self

    /// Translates the entity by the
specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    mutating func translate(by size:
Size3D)

    /// Returns the entity translated by
the specified vector.
    ///
    /// - Parameter vector: The vector
that defines that translation.
    func translated(by vector: Vector3D)
-> Self
```

```
    /// Translates the entity by the
specified vector.
    ///
    /// - Parameter vector: The vector
that defines that translation.
    mutating func translate(by vector:
Vector3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Translatable3D {

    /// Translates the entity by the
specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public mutating func
translate(by size: Size3D)

    /// Returns the entity translated by
the specified vector.
    ///
    /// - Parameter vector: The vector
that defines that translation.
    @inlinable public func translated(by
vector: Vector3D) -> Self

    /// Translates the entity by the
specified vector.
```

```
    ///
    /// - Parameter vector: The vector
that defines that translation.
    @inlinable public mutating func
translate(by vector: Vector3D)
}

/// A set of methods for working with
Spatial primitives with volume.
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
public protocol Volumetric {

    /// The size of the volume.
    var size: Size3D { get }

    /// Returns a Boolean value that
indicates whether the entity contains the
specified volumetric entity.
    ///
    /// - Parameter other: The second
primitive.
    /// - Returns A Boolean value that
indicates whether the entity contains the
specified volumetric entity.
    func contains(_ other: Self) -> Bool

    /// Returns a Boolean value that
indicates whether this volume contains
the specified point.
    ///
    /// - Parameter point: The point.
    /// - Returns A Boolean value that
```

```
indicates whether the entity contains the
point.
    func contains(point: Point3D) -> Bool

    /// Returns the smallest volumetric
entity that contains the two source
entities.
    ///
    /// - Parameter other: The second
primitive.
    /// - Returns The union of `self` and
`other`.
    func union(_ other: Self) -> Self

    /// Returns the intersection of two
volumetric entities.
    ///
    /// - Parameter other: The second
primitive.
    /// - Returns The intersection of
`self` and `other`.
    func intersection(_ other: Self) ->
Self?

    /// Sets the primitive to the
intersection of itself and the specified
primitive.
    ///
    /// - Parameter other: The second
primitive.
    mutating func formIntersection(_
other: Self)
```

```
    /// Sets the primitive to the union
of itself and the specified primitive.
    ///
    /// - Parameter other: The second
primitive.
    mutating func formUnion(_ other:
Self)

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    @available(*, deprecated, renamed:
"contains(anyOf:)")
    func containsAny(of points:
[Point3D]) -> Bool

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    func contains(anyOf points:
[Point3D]) -> Bool
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Volumetric {

    /// Sets the primitive to the
intersection of itself and the specified
primitive.
    ///
    /// - Parameter other: The second
primitive.
    public mutating func
formIntersection(_ other: Self)

    /// Sets the primitive to the union
of itself and the specified primitive.
    ///
    /// - Parameter other: The second
primitive.
    public mutating func formUnion(_
other: Self)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func cos(_ angle:
Angle2D) -> Double

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func cosh(_ angle:
Angle2D) -> Double
```

```
@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func sin(_ angle:
Angle2D) -> Double

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func sinh(_ angle:
Angle2D) -> Double

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func tan(_ angle:
Angle2D) -> Double

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
@inlinable public func tanh(_ angle:
Angle2D) -> Double

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.x")
public let x: Axis3D

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.y")
public let y: Axis3D
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.z")
public let z: Axis3D

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D {

    @available(*, deprecated, renamed:
"Rotation3D.init(angle:axis:)")
    public init(axis: RotationAxis3D,
angle: Angle2D)

    @available(*, deprecated, renamed:
"Rotation3D.init()")
    public init(quaternion: simd_quatf)

    @available(*, deprecated, renamed:
"Rotation3D.init()")
    public init(quaternion: simd_quatd)

    @available(*, deprecated, renamed:
"Rotation3D.init(position:target:up:)")
    public init(eye: Point3D, target:
Point3D, up: Vector3D = Vector3D(x: 0, y:
1, z: 0))
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D {
    @available(*, deprecated, renamed:
"vector")
    @inlinable public var simd:
simd_double3
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    @available(*, deprecated, renamed:
"vector")
    @inlinable public var simd:
simd_double3

    @available(*, deprecated, message:
"This property is deprecated")
    @inlinable public var origin: Point3D
{ get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D {

    @available(*, deprecated, renamed:
"vector")
    @inlinable public var simd:
simd_double3
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D {

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    @available(*, deprecated, renamed:
"contains(anyOf:)")
    @inlinable public func containsAny(of
points: [Point3D]) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D {

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    @available(*, deprecated, renamed:
"contains(anyOf:)")
```

```
    @inlinable public func containsAny(of
points: [Point3D]) -> Bool

    /// Returns the distance between the
origins of two rectangle.
    @available(*, deprecated, message:
"This function is deprecated")
    @inlinable public func distance(to
other: Rect3D) -> Double

    /// Returns the rotation around @p
(0,0,0)  from the first rectangle to the
second rectangle.
    @available(*, deprecated, message:
"This function is deprecated")
    @inlinable public func rotation(to
other: Rect3D) -> Rotation3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> Point3D
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> Rect3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> Ray3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
```

```
extension Pose3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> Pose3D

    /// Creates a pose with the specified
double-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    /// - note: if the upper-left 3 x 3
submatrix is not a rotation matrix (that
is, orthogonal with a determinant of
`+1`), the function returns `nil`.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init?(matrix:
simd_double4x4)

    /// Creates a pose with the specified
single-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    /// - note
    /// If the upper-left 3 x 3 submatrix
```

```
is not a rotation matrix (that is,
orthogonal with a determinant of `+1`),
the function returns `nil`.
    /// If the bottom row of the matrix
is not `[0,0,0,1]`, the functions
returns `SPPose3DInvalid`.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init?(matrix:
simd_float4x4)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D {

    /// Returns a new affine transform
from a projective transform.
    ///
    /// - Parameter transform: The source
projective transform.
    /// - Note The projective transform
must be affine, that is, its last row
must be `0, 0, 0, 1`,
    /// otherwise this initialiser
returns `nil`.
    @available(*, deprecated, message:
"Use
`AffineTransform3D.init(truncating:)`
instead.")
    @inlinable public init?
(projectiveTransform transform:
ProjectiveTransform3D)
```

```
    /// Returns a new affine transform
from a double-precision 4 x 4 matrix.
    ///
    /// - Parameter matrix: The source
matrix.
    /// - Note The the last row of the
source matrix must be `0, 0, 0, 1`,
    /// otherwise this initialiser
returns `nil`.
    @available(*, deprecated, message:
"Use
`AffineTransform3D.init(truncating:)`
instead.")
    @inlinable public init?(_ matrix:
simd_double4x4)

    @available(*, deprecated, renamed:
"init(offset:)", message: "Use `Vector3D`
variant.")
    @inlinable public init(translation:
Size3D)

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> AffineTransform3D
```

```
    @available(*, deprecated, renamed:
"translation")
    @inlinable public var offset:
Vector3D

    @available(*, deprecated, message:
"Use `AffineTransform3D.getter:inverse`
instead.")
    @inlinable public func inverted() ->
AffineTransform3D?

    /// Returns a new affine transform
structure from the specified single-
precision 4 x 3 matrix.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init(matrix:
simd_float4x3)

    /// Returns a new affine transform
from a single-precision 4 x 4 matrix.
    ///
    /// - Parameter matrix: The source
matrix.
    /// - Note The the last row of the
source matrix must be `0, 0, 0, 1`,
    /// otherwise this initialiser
returns `nil`.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init?(matrix:
simd_float4x4)

    /// Returns a new affine transform
from a double-precision 4 x 4 matrix.
    ///
    /// - Parameter matrix: The source
matrix.
    /// - Note The the last row of the
source matrix must be `0, 0, 0, 1`,
    /// otherwise this initialiser
returns `nil`.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init?(matrix:
simd_double4x4)

    /// Returns a new affine transform
from a single-precision 4 x 4 matrix.
    ///
    /// - Parameter matrix: The source
matrix.
    /// - Note The the last row of the
source matrix must be `0, 0, 0, 1`,
    /// otherwise this initialiser
returns `nil`.
    ///
    /// - note: This function is provided
```

```
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, message:
"Use
`AffineTransform3D.init(truncating:)`
instead.")
    @inlinable public init?(_ matrix:
simd_float4x4)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D {

    @available(*, deprecated, renamed:
"init(offset:)", message: "Use `Vector3D`
variant.")
    @inlinable public init(translation:
Size3D)

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> ProjectiveTransform3D

    @available(*, deprecated, renamed:
"translation")
    @inlinable public var offset:
```

```
Vector3D

    @available(*, deprecated, message:
"Use
`ProjectiveTransform3D.getter:inverse`
instead.")
    @inlinable public func inverted() ->
ProjectiveTransform3D?

    /// Returns a new projective
transform structure from the specified
single-precision 4 x 3 matrix.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"init(_:)")
    @inlinable public init(matrix:
simd_float4x4)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension SIMD3 where Scalar == Float {

    /// Returns a new vector from a
Spatial rotation axis.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
```

```
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"SIMD3.init(_:)")
    @inlinable public init(rotationAxis:
RotationAxis3D)

    /// Returns a new vector from a
Spatial point.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"SIMD3.init(_:)")
    @inlinable public init(point:
Point3D)

    /// Returns a new vector from a
Spatial point.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"SIMD3.init(_:)")
    @inlinable public init(vector:
```

```
Vector3D)

    /// Returns a new vector from a
Spatial size.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"SIMD3.init(_:)")
    @inlinable public init(size: Size3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension simd_quatf {

    /// Returns a new quaternion from a
Spatial rotation.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"simd_quatf.init(_:)")
    @inlinable public init(rotation:
Rotation3D)
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension simd_float4x3 {

    /// Returns a new matrix from a
Spatial affine transform.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"simd_float4x3.init(_:)")
    @inlinable public
init(affineTransform: AffineTransform3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension simd_float4x4 {

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial
projective transform.
    ///
    /// Values are rounded to a
representable value, if necessary.
    ///
    /// - note: This function is provided
```

```
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"simd_float4x4.init(_:)")
    @inlinable public
init(projectiveTransform:
ProjectiveTransform3D)

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial affine
transform.
    ///
    /// Values are rounded to a
representable value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
"simd_float4x4.init(_:)")
    @inlinable public
init(affineTransform: AffineTransform3D)

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial pose.
    ///
    /// Values are rounded to a
representable value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @available(*, deprecated, renamed:
```

```
"simd_float4x4.init(_:)")
    @inlinable public init(pose: Pose3D)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension EulerAngles {

    /// Returns a new Euler angles
structure from three angle structures.
    ///
    /// - Parameter x The first angle.
    /// - Parameter y The second angle.
    /// - Parameter z The third angle.
    /// - Parameter order The Euler angle
ordering.
    @available(*, deprecated, renamed:
"init(x:y:z:order:)")
    public init(_ x: Angle2D, _ y:
Angle2D, _ z: Angle2D, order:
EulerAngles.Order)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Rect3D {

    /// Returns the smallest value for
the x-coordinate of the rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.min.x`.")
    public var minX: Double { get }
    /// Returns the smallest value for
the y-coordinate of the rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.min.y`.")
    public var minY: Double { get }

    /// Returns the smallest value for
the z-coordinate of the rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.min.z`.")
    public var minZ: Double { get }

    /// Returns the x-coordinate that
establishes the center of a rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.center.x`.")
    public var midX: Double { get }

    /// Returns the y-coordinate that
establishes the center of a rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.center.y`.")
    public var midY: Double { get }

    /// Returns the z-coordinate that
establishes the center of a rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.center.z`.")
    public var midZ: Double { get }

    /// Returns the largest value of the
x-coordinate for the rectangle.
    @available(*, deprecated, message:
```

```
"Use `Rect3D.max.x`.")
    public var maxX: Double { get }

    /// Returns the largest value of the
y-coordinate for the rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.max.y`.")
    public var maxY: Double { get }

    /// Returns the largest value of the
z-coordinate for the rectangle.
    @available(*, deprecated, message:
"Use `Rect3D.max.z`.")
    public var maxZ: Double { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Returns a point from the
specified values.
    ///
    /// - Parameter x: The x component of
the point.
    /// - Parameter y: The y component of
the point.
    /// - Parameter z: The z component of
```

```
the point.
    @inlinable public init(x: Double = 0,
y: Double = 0, z: Double = 0)

    /// Returns a new point from a
double-precision vector.
    ///
    /// - Parameter xyz: The source
vector.
    @inlinable public init(_ xyz:
simd_double3)

    /// Returns a new point from a
Spatial vector.
    ///
    /// - Parameter xyz: The source
vector.
    @inlinable public init(_ xyz:
Vector3D)

    /// Returns a new point from a size.
    ///
    /// - Parameter size: The source
size.
    @inlinable public init(_ size:
Size3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Clamps the mutable point to the
```

```
specified rectangle.
    ///
    /// - Parameter rect: The rectangle
that defines the clamp volume.
    @inlinable public mutating func
clamp(to rect: Rect3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Primitive3D {

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func applying(_ pose: Pose3D)
-> Point3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ pose:
Pose3D) -> Point3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
```

```
extension Point3D {

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func applying(_ scaledPose:
ScaledPose3D) -> Point3D

    /// Returns the primitive resulting
from unapplying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ pose:
ScaledPose3D) -> Point3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Rotatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Translatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Equatable {

    /// Returns a Boolean value
indicating whether two points are equal.
    ///
    /// - Parameter lhs: The first point
to compare.
    /// - Parameter rhs: The second point
to compare.
    @inlinable public static func ==
(lhs: Point3D, rhs: Point3D) -> Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Point3D {

    /// Returns a Boolean value
indicating whether two points are equal
within a specified tolerance.
    ///
    /// - Parameter other: The second
point.
    /// - Parameter tolerance: The
tolerance of the comparison.
    @inlinable public func
isApproximatelyEqual(to other: Point3D,
tolerance: Double = sqrt(.ulpOfOne)) ->
Bool
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
```

```
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Point3D : CustomReflectable {
    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Returns a point that's the
element-wise sum of a point's `x`, `y`,
and `z` and a size's
    /// `width`, `height`, and `depth`.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Point3D, rhs: Size3D) -> Point3D

    /// Calculates the element-wise sum
of a point's `x`, `y`, and `z` and a
size's
    /// `width`, `height`, and `depth`
and stores the result in the left-hand-
side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func +=
(lhs: inout Point3D, rhs: Size3D)
```

```
    /// Returns a point that's the
element-wise sum of a size's
    /// `width`, `height`, and `depth`
and a point's `x`, `y`, and `z` .
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Size3D, rhs: Point3D) -> Point3D

    /// Returns a point that's the
element-wise difference of a point's `x`,
`y`, and `z` and a size's
    /// `width`, `height`, and `depth`.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Point3D, rhs: Size3D) -> Point3D

    /// Calculates the element-wise
difference of a point's `x`, `y`, and `z`
and a size's
    /// `width`, `height`, and `depth`
and stores the result in the left-hand-
side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
```

```
    @inlinable public static func -=
(lhs: inout Point3D, rhs: Size3D)

    /// Returns a point that's the
element-wise difference of a size's
    /// `width`, `height`, and `depth`
and a point's `x`, `y`, and `z`.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Size3D, rhs: Point3D) -> Point3D

    /// Returns a point that's the
element-wise negation of the point.
    ///
    /// - Parameter point: The value.
    @inlinable prefix public static func
- (point: Point3D) -> Point3D

    /// Returns a point with each element
mulitplied by a scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Double, rhs: Point3D) -> Point3D

    /// Calculates the product of each
element of a point and a scalar value and
stores the result in the left-hand-side
```

```
variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout Point3D, rhs: Double)

    /// Returns a point with each element
mulitplied by a scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Point3D, rhs: Double) -> Point3D

    /// Returns a point with each element
divided by a scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func / (lhs:
Point3D, rhs: Double) -> Point3D

    /// Calculates the division of each
element of a point and a scalar value and
stores the result in the left-hand-side
variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
```

```
value.
    @inlinable public static func /=
(lhs: inout Point3D, rhs: Double)

    /// Returns the point that results
from applying the transform to the point.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
AffineTransform3D, rhs: Point3D) ->
Point3D

    /// Returns the point that results
from applying the transform to the point.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ProjectiveTransform3D, rhs: Point3D) ->
Point3D

    /// Returns the point that results
from applying the pose to the point.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Pose3D, rhs: Point3D) -> Point3D
}
```

```
@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Size3D : AdditiveArithmetic {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D {

    /// Returns a size that's the
element-wise negation of the size.
    ///
    /// - Parameter size: The value.
    @inlinable prefix public static func
- (size: Size3D) -> Size3D

    /// Returns a size that's the
element-wise sum of two sizes.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Size3D, rhs: Size3D) -> Size3D

    /// Calculates the element-wise sum
of two sizes and stores the result in the
left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
```

```
    @inlinable public static func +=
(lhs: inout Size3D, rhs: Size3D)

    /// Returns a size that's the
element-wise difference of two sizes.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Size3D, rhs: Size3D) -> Size3D

    /// Calculates the element-wise
difference of two sizes and stores the
result in the left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func -=
(lhs: inout Size3D, rhs: Size3D)

    /// Returns a size with each element
mulitplied by a scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Double, rhs: Size3D) -> Size3D

    /// Returns a size with each element
mulitplied by a scalar value.
```

```
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Size3D, rhs: Double) -> Size3D

    /// Calculates the element-wise
product of a size and a scalar value and
stores the result in the left-hand-side
variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout Size3D, rhs: Double)

    /// Returns a size with each element
divided by a scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func / (lhs:
Size3D, rhs: Double) -> Size3D

    /// Calculates the element-wise
division of a size and a scalar value and
stores the result in the left-hand-side
variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func /=
(lhs: inout Size3D, rhs: Double)

    /// Returns the size that results
from applying the transform to the size.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
AffineTransform3D, rhs: Size3D) -> Size3D

    /// Returns the size that results
from applying the transform to the size.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ProjectiveTransform3D, rhs: Size3D) ->
Size3D

    /// Returns the size that results
from applying the pose to the size.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Pose3D, rhs: Size3D) -> Size3D
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D {

    /// Returns the rectangle that
results from applying the transform to
the rectangle.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
AffineTransform3D, rhs: Rect3D) -> Rect3D

    /// Returns the rectangle that
results from applying the transform to
the rectangle.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ProjectiveTransform3D, rhs: Rect3D) ->
Rect3D

    /// Returns the rectangle that
results from applying the pose to the
rectangle.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
```

```
    @inlinable public static func * (lhs:
Pose3D, rhs: Rect3D) -> Rect3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D {

    /// Returns the concatenation of two
affine transforms.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
AffineTransform3D, rhs:
AffineTransform3D) -> AffineTransform3D

    /// Calculates the concatenation of
two affine transforms and stores the
result in the left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout AffineTransform3D, rhs:
AffineTransform3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D {
    /// Returns the product of two
projective transforms.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ProjectiveTransform3D, rhs:
ProjectiveTransform3D) ->
ProjectiveTransform3D

    /// Calculates the concatenation of
two projective transforms and stores the
result in the left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout ProjectiveTransform3D, rhs:
ProjectiveTransform3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Returns a vector that's the
element-wise difference of two points.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
```

```
value.
    @inlinable public static func - (lhs:
Point3D, rhs: Point3D) -> Vector3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D {

    /// Returns the point that's the
element-wise sum of a point and a vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Point3D, rhs: Vector3D) -> Point3D

    /// Returns the point that's the
element-wise sum of a vector and a point.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Vector3D, rhs: Point3D) -> Point3D

    /// Returns the point that's the
element-wise difference of a point and a
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
```

```
value.
    @inlinable public static func - (lhs:
Point3D, rhs: Vector3D) -> Point3D

    /// Returns the point that's the
element-wise difference of a vector and a
point.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Vector3D, rhs: Point3D) -> Point3D

    /// Returns the vector that's the
element-wise negation of the vector.
    ///
    /// - Parameter point: The value.
    @inlinable prefix public static func
- (vector: Vector3D) -> Vector3D

    /// Returns the vector that's the
element-wise sum of a vector and a
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Vector3D, rhs: Vector3D) -> Vector3D

    /// Calculates the vector that's the
element-wise sum of a vector and a vector
```

```
and stores the result in the left-hand-
side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func +=
(lhs: inout Vector3D, rhs: Vector3D)

    /// Returns the vector that's the
element-wise difference of a vector and a
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Vector3D, rhs: Vector3D) -> Vector3D

    /// Calculates the vector that's the
element-wise difference of a vector and a
vector and stores the result in the left-
hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func -=
(lhs: inout Vector3D, rhs: Vector3D)

    /// Returns the vector that's the
element-wise product of a scalar value
and a vector.
```

```
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Double, rhs: Vector3D) -> Vector3D

    /// Returns the vector that's the
element-wise product of a vector and a
scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Vector3D, rhs: Double) -> Vector3D

    /// Calculates the vector that's the
element-wise product of a vector and a
scalar value and stores the result in the
left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout Vector3D, rhs: Double)

    /// Returns the vector that's the
element-wise division of a vector and a
scalar value.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func / (lhs:
Vector3D, rhs: Double) -> Vector3D

    /// Calculates the vector that's the
element-wise division of a vector and a
scalar value and stores the result in the
left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func /=
(lhs: inout Vector3D, rhs: Double)

    /// Returns the vector that results
from applying the transform to the
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
AffineTransform3D, rhs: Vector3D) ->
Vector3D

    /// Returns the vector that results
from applying the transform to the
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
```

```
value.
    @inlinable public static func * (lhs:
ProjectiveTransform3D, rhs: Vector3D) ->
Vector3D

    /// Returns the vector that results
from applying the pose to the vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Pose3D, rhs: Vector3D) -> Vector3D

    /// Returns the size that's the
element-wise sum of a size and a vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Size3D, rhs: Vector3D) -> Size3D

    /// Returns the size that's the
element-wise sum of a vector and a size.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func + (lhs:
Vector3D, rhs: Size3D) -> Size3D

    /// Returns the size that's the
```

```
element-wise difference of a size and a
vector.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Size3D, rhs: Vector3D) -> Size3D

    /// Returns the size that's the
element-wise difference of a vector and a
size.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func - (lhs:
Vector3D, rhs: Size3D) -> Size3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D {

    /// Returns the point that results
from applying the pose to the ray.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Pose3D, rhs: Ray3D) -> Ray3D
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS 
16.0, watchOS 9.0, *)
extension Point3D {

    /// Calculates the point that's the 
element-wise sum of a point and a vector 
and stores the result in the left-hand-
side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second 
value.
    @inlinable public static func +=
(lhs: inout Point3D, rhs: Vector3D)

    /// Calculates the point that's the 
element-wise difference of a point and a 
vector and stores the result in the left-
hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second 
value.
    @inlinable public static func -=
(lhs: inout Point3D, rhs: Vector3D)
}

@available(macOS 13.0, iOS 16.0, tvOS 
16.0, watchOS 9.0, *)
extension Size3D {

    /// Calculates the size that's the
```

```
element-wise sum of a size and a vector
and stores the result in the left-hand-
side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func +=
(lhs: inout Size3D, rhs: Vector3D)

    /// Calculates the size that's the
element-wise difference of a size and a
vector and stores the result in the left-
hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func -=
(lhs: inout Size3D, rhs: Vector3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D {

    /// Returns a new pose that's
constructed by concatenating two existing
poses.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
```

```
    @inlinable public static func * (lhs:
Pose3D, rhs: Pose3D) -> Pose3D

    /// Calculates the concatenation of
poses and stores the result in the left-
hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout Pose3D, rhs: Pose3D)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Rotation3D {

    /// Calculates the spherical linear
interpolation between the identity
rotation and the LHS rotation
    /// at the RHS interpolation
parameter.
    ///
    /// For example, multiplying an angle
of 90° by `0.5`, returns an angle of 45°:
    ///
    ///      let rotation =
Rotation3D(angle: Angle2D(degrees: 90),
    ///
axis: .init(x: 0, y: 1, z: 0))
    ///      let rotated = (rotation *
0.5).angle.degrees
```

```
    ///      print(rotated) // prints
"45.0"
    ///
    /// - Note This function returns the
longest path where the angle is greater
than 180°,
    /// - Parameter lhs: The rotation
    /// - Parameter rhs: The
interpolation parameter.
    @inlinable public static func * (lhs:
Rotation3D, rhs: Double) -> Rotation3D

    /// Calculates the spherical linear
interpolation between the identity
rotation and the RHS rotation
    /// at the LHS interpolation
parameter.
    ///
    /// For example, multiplying an angle
of 90° by `0.5`, returns an angle of 45°:
    ///
    ///      let rotation =
Rotation3D(angle: Angle2D(degrees: 90),
    ///
axis: .init(x: 0, y: 1, z: 0))
    ///      let rotated = ( 0.5 *
rotation).angle.degrees
    ///      print(rotated) // prints
"45.0"
    ///
    /// - Note This function returns the
longest path where the angle is greater
than 180°,
    /// - Parameter lhs: The
interpolation parameter
    /// - Parameter rhs: The rotation.
    @inlinable public static func * (lhs:
Double, rhs: Rotation3D) -> Rotation3D

    /// Calculates the product of two
rotations.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Rotation3D, rhs: Rotation3D) ->
Rotation3D

    /// Calculates the product of two
rotations and stores the result in the
left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout Rotation3D, rhs: Rotation3D)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Angle2D : AdditiveArithmetic {

    /// Returns the  additive inverse of
the specified angle.
```

```
    @inlinable prefix public static func
- (angle: Angle2D) -> Angle2D

    /// Returns the given angle
unchanged.
    @inlinable prefix public static func
+ (angle: Angle2D) -> Angle2D

    /// Adds two angles and produces
their sum.
    @inlinable public static func + (lhs:
Angle2D, rhs: Angle2D) -> Angle2D

    /// Adds two angles and stores the
result in the left-hand-side variable.
    @inlinable public static func +=
(lhs: inout Angle2D, rhs: Angle2D)

    /// Subtracts one angle from another
and produces their difference.
    @inlinable public static func - (lhs:
Angle2D, rhs: Angle2D) -> Angle2D

    /// Subtracts the second angle from
the first and stores the difference in
the left-hand-side variable.
    @inlinable public static func -=
(lhs: inout Angle2D, rhs: Angle2D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D {
    /// Returns a new pose that's
constructed by concatenating two existing
poses.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ScaledPose3D, rhs: Pose3D) ->
ScaledPose3D

    /// Returns a new scaled pose that's
constructed by concatenating two existing
poses.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
ScaledPose3D, rhs: ScaledPose3D) ->
ScaledPose3D

    /// Returns a new scaled pose that's
constructed by concatenating a pose and a
scaled pose.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func * (lhs:
Pose3D, rhs: ScaledPose3D) ->
ScaledPose3D
```

```
    /// Calculates the concatenation of
scaled poses and stores the result in the
left-hand-side variable.
    ///
    /// - Parameter lhs: The first value.
    /// - Parameter rhs: The second
value.
    @inlinable public static func *=
(lhs: inout ScaledPose3D, rhs:
ScaledPose3D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D {

    /// Creates a scaled  pose from
Spatial primitives that describe the
position, rotation, and scale.
    ///
    /// - Parameter position: The
position of the scaled pose.
    /// - Parameter rotation: The
rotation of the pose.
    /// - Parameter scale: The uniform
scale of the scaled  pose.
    @inlinable public init(position:
Point3D = .zero, rotation: Rotation3D,
scale: Double = 1)

    /// Creates a pose from double-
precision simd primitives that describe
```

```
the position, rotation, and scale.
    ///
    /// - Parameter position: The
position of the scaled pose.
    /// - Parameter rotation: The
rotation of the scaled pose.
    /// - Parameter scale: The uniform
scale of the scaled pose.
    @inlinable public init(position:
simd_double3 = .zero, rotation:
simd_quatd, scale: Double = 1)

    /// Creates a pose from single-
precision simd primitives that describe
the position, rotation, and scale.
    ///
    /// - Parameter position: The
position of the scaled pose.
    /// - Parameter rotation: The
rotation of the scaled pose.
    /// - Parameter scale: The uniform
scale of the scaled pose.
    @inlinable public init(position:
simd_float3 = .zero, rotation:
simd_quatf, scale: Float = 1)

    /// Creates a scaled  pose at the
specified position that's oriented
towards a look at target.
    ///
    /// - Parameter position: The
position of the scaled pose.
    /// - Parameter target: The point
```

```
that the scaled pose looks at.
    /// - Parameter scale: The uniform
scale of the scaled  pose.
    /// - Parameter up: The up direction.
    ///
    /// - note: This function creates a
scaled pose where `+z` is forward.
    @inlinable public init(position:
Point3D = .zero, target: Point3D, scale:
Double = 1, up: Vector3D = Vector3D(x: 0,
y: 1, z: 0))

    /// Creates a scaled pose with the
specified forward and up vectors.
    ///
    /// - Parameter forward: The forward
direction.
    /// - Parameter scale: The uniform
scale of the scaled  pose.
    /// - Parameter up: The up direction.
    ///
    /// - note: This function creates a
scaled pose where `+z` is forward.
    @inlinable public init(forward:
Vector3D, scale: Double = 1, up: Vector3D
= Vector3D(x: 0, y: 1, z: 0))

    /// Creates a scaled pose with with a
position, rotation, and scale that are
defined by an affine transform.
    ///
    /// - Parameter transform: The source
transform. The function only considers
```

```
the transform's
    /// rotation and translation
components.
    /// - Returns: A pose with a
position, rotation, and scale that are
defined by an affine transform.
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`. If
the specified  ``AffineTransform3D``
doesn't have uniform
    /// scale, the function returns
`nil`.
    @inlinable public init?(transform:
AffineTransform3D)

    /// Creates a pose with with a
position, rotation, and scale that are
defined by a projective transform.
    ///
    /// - Parameter transform: The source
transform. The function only considers
the transform's
    /// rotation and translation
components.
    /// - Returns: A pose with a
position, rotation, and scale that are
defined by a projective transform.
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
```

```
translate affine transform. In that case,
    /// the function returns `nil`.  If
the specified  ``ProjectiveTransform3D``
doesn't have uniform
    /// scale, the function returns
`nil`.
    @inlinable public init?(transform:
ProjectiveTransform3D)

    /// Creates a scaled pose with the
specified double-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.  If
the specified matrix doesn't have uniform
scale the function returns `nil`.
    @inlinable public init?(_ matrix:
simd_double4x4)

    /// Creates a scaled pose with the
specified single-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`. If
the specified matrix doesn't have uniform
scale the function returns `nil`.
    @inlinable public init?(_ matrix:
simd_float4x4)

    /// A 4 x 4 matrix that represents
the scaled  pose's scale, rotation, and
translation.
    @inlinable public var matrix:
simd_double4x4 { get }

    /// The inverse of the scaled pose's
underlying matrix.
    @inlinable public var inverse:
ScaledPose3D { get }

    /// Returns a transform constructed
by concatenating two existing scaled
poses.
    ///
    /// - Parameter transform: The second
scaled pose.
    @inlinable public func
concatenating(_ transform: ScaledPose3D)
-> ScaledPose3D

    /// Returns a transform constructed
by concatenating two a scaled pose and a
pose.
    ///
    /// - Parameter transform: The second
pose.
```

```
    @inlinable public func
concatenating(_ transform: Pose3D) ->
ScaledPose3D

    /// The identity scaled pose.
    @inlinable public static var
identity: ScaledPose3D { get }

    /// Returns the scaled pose flipped
along the specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    @inlinable public func flipped(along
axis: Axis3D) -> ScaledPose3D

    /// Flips the scaled pose along the
specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    public mutating func flip(along axis:
Axis3D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : Translatable3D {

    /// Returns the entity translated by
the specified size.
    ///
    /// - Parameter size: The size
```

```
structure that defines that translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public func translated(by
size: Size3D) -> ScaledPose3D

    /// Returns the entity translated by
the specified vector.
    ///
    /// - Parameter vector: The vector
that defines that translation.
    @inlinable public func translated(by
offset: Vector3D) -> ScaledPose3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : Rotatable3D {

    /// Returns a pose with a rotation
that's rotated by the specified rotation.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns A scaled pose with a
rotation that's rotated by the specified
rotation.
    @inlinable public func rotated(by
rotation: Rotation3D) -> ScaledPose3D

    /// Returns a scaled pose with a
rotation that's rotated by the specified
quaternion.
```

```
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns A scaled pose with a
rotation that's rotated by the specified
quaternion.
    @inlinable public func rotated(by
quaternion: simd_quatd) -> ScaledPose3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : @unchecked
Sendable {
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : Equatable {

    /// Returns a Boolean value
indicating whether two poses are equal.
    ///
    /// - Parameter lhs: The first pose
to compare.
    /// - Parameter rhs: The second pose
to compare.
    @inlinable public static func ==
(lhs: ScaledPose3D, rhs: ScaledPose3D) ->
Bool

    /// Returns a Boolean value
indicating whether two scaled poses are
```

```
equal within a specified tolerance.
    ///
    /// - Parameter other: The second
scaled  pose.
    /// - Parameter tolerance: The
tolerance of the comparison.
    @inlinable public func
isApproximatelyEqual(to other:
ScaledPose3D, tolerance: Double =
sqrt(.ulpOfOne)) -> Bool

    /// Returns true if the scaled pose
is the identity pose.
    @inlinable public var isIdentity:
Bool { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
```

```
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ScaledPose3D :
CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Pose3D {

    /// Returns a transform constructed
by concatenating two a pose and a scaled
pose.
    ///
    /// - Parameter transform: The second
scaled pose.
    @inlinable public func
```

```
concatenating(_ transform: ScaledPose3D)
-> ScaledPose3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D {

    /// Creates a pose from Spatial
primitives that describe the position and
rotation.
    ///
    /// - Parameter position: The
position of the pose.
    /// - Parameter rotation: The
rotation of the pose.
    @inlinable public init(position:
Point3D = .zero, rotation: Rotation3D)

    /// Creates a pose from double-
precision simd primitives that describe
the position and rotation.
    ///
    /// - Parameter position: The
position of the pose.
    /// - Parameter rotation: The
rotation of the pose.
    @inlinable public init(position:
simd_double3 = .zero, rotation:
simd_quatd)

    /// Creates a pose from single-
precision simd primitives that describe
```

```
the position and rotation.
    ///
    /// - Parameter position: The
position of the pose.
    /// - Parameter rotation: The
rotation of the pose.
    @inlinable public init(position:
simd_float3 = .zero, rotation:
simd_quatf)

    /// Creates a pose at the specified
position that's oriented towards a look
at target.
    ///
    /// - Parameter position: The
position of the pose.
    /// - Parameter target: The point
that the ray looks at.
    /// - Parameter up: The up direction.
    ///
    /// - note: This function creates a
pose where `+z` is forward.
    @inlinable public init(position:
Point3D = .zero, target: Point3D, up:
Vector3D = Vector3D(x: 0, y: 1, z: 0))

    /// Creates a pose with the specified
forward and up vectors.
    ///
    /// - Parameter forward: The forward
direction.
    /// - Parameter up: The up direction.
    ///
    /// - note: This function creates a
pose where `+z` is forward.
    @inlinable public init(forward:
Vector3D, up: Vector3D = Vector3D(x: 0,
y: 1, z: 0))

    /// Creates a pose with with a
position and rotation that are defined by
an affine transform.
    ///
    /// - Parameter transform: The source
transform. The function only considers
the transform's
    /// rotation and translation
components.
    /// - Returns: A pose with a position
and rotation that are defined by an
affine transform.
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.
    @inlinable public init?(transform:
AffineTransform3D)

    /// Creates a pose with with a
position and rotation that are defined by
a projective transform.
    ///
    /// - Parameter transform: The source
transform. The function only considers
the transform's
```

```
    /// rotation and translation
components.
    /// - Returns: A pose with a position
and rotation that are defined by a
projective transform.
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.
    @inlinable public init?(transform:
ProjectiveTransform3D)

    /// Creates a pose with the specified
double-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    ///
    /// - note: This function can't
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.
    @inlinable public init?(_ matrix:
simd_double4x4)

    /// Creates a pose with the specified
single-precision 4 x 4 matrix
    ///
    /// - Parameter matrix: The source
matrix
    ///
    /// - note: This function can't
```

```
extract rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.
    @inlinable public init?(_ matrix:
simd_float4x4)

    /// A 4 x 4 matrix that represents
the pose's translation and rotation.
    @inlinable public var matrix:
simd_double4x4 { get }

    /// The inverse of the pose's
underlying matrix.
    @inlinable public var inverse: Pose3D
{ get }

    /// Returns a transform constructed
by concatenating two existing poses.
    ///
    /// - Parameter transform: The second
pose.
    @inlinable public func
concatenating(_ transform: Pose3D) ->
Pose3D

    /// The identity pose.
    @inlinable public static var
identity: Pose3D { get }

    /// Returns the pose flipped along
the specified axis.
    ///
    /// - Parameter axis: The axis of the
```

```
flip operation.
    @inlinable public func flipped(along
axis: Axis3D) -> Pose3D

    /// Flips the pose along the
specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    public mutating func flip(along axis:
Axis3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : Translatable3D {

    /// Returns the entity translated by
the specified vector.
    ///
    /// - Parameter vector: The vector
that defines that translation.
    @inlinable public func translated(by
offset: Vector3D) -> Pose3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : Rotatable3D {

    /// Returns a pose with a rotation
that's rotated by the specified rotation.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns A pose with a rotation
that's rotated by the specified rotation.
    @inlinable public func rotated(by
rotation: Rotation3D) -> Pose3D

    /// Returns a pose with a rotation
that's rotated by the specified
quaternion.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns A pose with a rotation
that's rotated by the specified
quaternion.
    @inlinable public func rotated(by
quaternion: simd_quatd) -> Pose3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : Equatable {

    /// Returns a Boolean value
indicating whether two poses are equal.
    ///
    /// - Parameter lhs: The first pose
```

```
to compare.
    /// - Parameter rhs: The second pose
to compare.
    @inlinable public static func ==
(lhs: Pose3D, rhs: Pose3D) -> Bool

    /// Returns a Boolean value
indicating whether two poses are equal
within a specified tolerance.
    ///
    /// - Parameter other: The second
pose.
    /// - Parameter tolerance: The
tolerance of the comparison.
    @inlinable public func
isApproximatelyEqual(to other: Pose3D,
tolerance: Double = sqrt(.ulpOfOne)) ->
Bool

    /// Returns true if the pose is the
identity transform.
    @inlinable public var isIdentity:
Bool { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Pose3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Pose3D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
@unchecked Sendable {
    /// Creates a new spherical
coordinates structure from the specified
Cartesian coordinates.
    ///
    /// - Parameter x: The `x` Cartesian
coordinate.
    /// - Parameter y: The `y` Cartesian
coordinate.
    /// - Parameter z: The `z` Cartesian
coordinate.
    public init(x: Double, y: Double, z:
Double)

    /// Creates a new spherical
coordinates structure from the specified
Cartesian coordinates.
    ///
    /// - Parameter vector: The Cartesian
coordinates.
    public init(_ vector: simd_double3)

    /// Creates a new spherical
coordinates structure from the specified
Cartesian coordinates represented by a
Spatial point.
    ///
    /// - Parameter point: The Cartesian
coordinates.
    public init(_ point: Point3D)

    /// Creates a new spherical
coordinates structure from the specified
```

```
Cartesian coordinates represented by a
Spatial vector.
    ///
    /// - Parameter vector: The Cartesian
coordinates.
    public init(_ vector: Vector3D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Point3D {

    /// Creates a new point structure
that contains the spherical coordinates
converted to Cartesian coordinates.
    ///
    /// - Parameter coords: The spherical
coordinates.
    public init(_ coords:
SphericalCoordinates3D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Vector3D {

    /// Creates a new vector structure
that contains the spherical coordinates
converted to Cartesian coordinates.
    ///
    /// - Parameter coords: The spherical
coordinates.
    public init(_ coords:
```

```
SphericalCoordinates3D)
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
Equatable {

    /// Returns a Boolean value
indicating whether two points are equal.
    ///
    /// - Parameter lhs: The first point
to compare.
    /// - Parameter rhs: The second point
to compare.
    @inlinable public static func ==
(lhs: SphericalCoordinates3D, rhs:
SphericalCoordinates3D) -> Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws
```

```
    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension SphericalCoordinates3D :
CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Angle2D {

    @inlinable public static func
radians(_ radians: Double) -> Angle2D
```

```
    @inlinable public static func
degrees(_ degrees: Double) -> Angle2D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D : Equatable {

    /// Returns a Boolean value
indicating whether two angles are equal.
    ///
    /// - Parameter lhs: The first angle
to compare.
    /// - Parameter rhs: The second angle
to compare.
    @inlinable public static func ==
(lhs: Angle2D, rhs: Angle2D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Angle2D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Angle2D {
    /// Returns the specified angle
normalized to `(-π, π]` radians (`(-180°,
180.0°]`).
    @inlinable public var normalized:
Angle2D { get }

    @inlinable public static func acos(_
x: Double) -> Angle2D

    @inlinable public static func asin(_
x: Double) -> Angle2D

    @inlinable public static func atan(_
x: Double) -> Angle2D

    @inlinable public static func acosh(_
x: Double) -> Angle2D

    @inlinable public static func asinh(_
x: Double) -> Angle2D

    @inlinable public static func atanh(_
x: Double) -> Angle2D

    @inlinable public static func
atan2(y: Double, x: Double) -> Angle2D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
@unchecked Sendable {
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D {

    /// The translation component
    @inlinable public var translation:
Vector3D

    /// Returns a new identity projective
transform.
    @inlinable public init()

    /// Returns a new scale, rotate,
translate transform.
    ///
    /// - Parameter scale: The scale.
    /// - Parameter rotation: The
rotation.
    /// - Parameter translation: The
translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public init(scale: Size3D
= Size3D(width: 1.0, height: 1, depth:
1), rotation: Rotation3D = .zero,
translation: Size3D)

    /// Returns a new scale, rotate,
translate transform.
    ///
    /// - Parameter scale: The scale.
    /// - Parameter rotation: The
```

```
rotation.
    /// - Parameter translation: The
translation.
    @inlinable public init(scale: Size3D
= Size3D(width: 1.0, height: 1, depth:
1), rotation: Rotation3D = .zero,
translation: Vector3D = .zero)

    /// Returns a new transform from the
specified pose.
    ///
    /// - Parameter pose: The source pose
    @inlinable public init(pose: Pose3D)

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
    @inlinable public init(scaledPose:
ScaledPose3D)

    /// Returns a Boolean value
indicating whether two transforms are
equal within a specified tolerance.
    ///
    /// - Parameter transform: The second
transform.
    /// - Parameter tolerance: The
tolerance of the comparison.
    @inlinable public func
isApproximatelyEqual(to other:
ProjectiveTransform3D, tolerance: Double
= sqrt(.ulpOfOne)) -> Bool

    /// Returns the transform flipped
```

```
along the specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    public func flipped(along axis:
Axis3D) -> ProjectiveTransform3D

    /// Flips the transform along the
specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    public mutating func flip(along axis:
Axis3D)

    /// Returns the transform scaled by
the specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled transform.
    @inlinable public func scaledBy(x:
Double, y: Double, z: Double) ->
ProjectiveTransform3D

    /// Returns true if the transform is
affine and uniform over the specified
dimensions.
    ///
    /// - Parameter overDimensions: The
dimensions that the function tests are
uniform.
    ///
    /// If you specify `overDimensions`
as `Dimension3DSet.all`, the function
returns the same
    /// result as
`Transform3D.getter:isUniform`. If you
specify as `overDimensions` with
    /// zero or one dimension, the
function returns the same result as
`Transform3D.getter:isRectilinear`.
    @inlinable public func
isUniform(overDimensions: Dimension3DSet)
-> Bool

    /// Returns a new shear transform
    ///
    /// - Parameter shear: The shear.
    @inlinable public init(shear:
AxisWithFactors)

    /// Returns the transform sheared by
the specified shear.
    ///
    /// - Parameter shear: The axis and
shear factors.
    /// - Returns The sheared transform.
    @inlinable public func sheared(_
shear: AxisWithFactors) ->
ProjectiveTransform3D
```

```
    /// The projective transform's
rotation.
    ///
    /// This function computes the
rotation from the first three rows of the
transform matrix and ignores the fourth
row.
    /// This function can't extract
rotation from a non-scale-rotate-
translate projective transform. In that
case,
    /// the function returns `nil`.
    @inlinable public var rotation:
Rotation3D? { get }

    /// The projective transform's scale.
    ///
    /// This function computes the scale
from the first three rows of the
transform matrix and ignores the fourth
row.
    /// If the projective transform isn't
an affine scale-rotate-translate
transform, this value is `nil`.
    @available(*, deprecated, renamed:
"scaleComponent")
    @inlinable public var scale: Size3D?
{ get }

    /// The projective transform's
inverse.
    ///
    /// If the source transform isn't
```

```
invertible, this value is `nil`.
    @inlinable public var inverse:
ProjectiveTransform3D? { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
Equatable {

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    @inlinable public static func ==
(lhs: ProjectiveTransform3D, rhs:
ProjectiveTransform3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
Hashable {

    /// Hashes the essential components
```

```
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D : Codable
{

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
```

```
decoding from the given decoder.
    ///
    /// This initializer throws an error
if reading from the decoder fails, or
    /// if the data read is corrupted or
otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension ProjectiveTransform3D :
CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
```

```
extension ProjectiveTransform3D {

    /// The projective transform's scale
component.
    ///
    /// This function computes the scale
from the first three rows of the
transform matrix and ignores the fourth
row.
    @inlinable public var scaleComponent:
Size3D { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Primitive3D {

    /// Unapplies an affine transform.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Note: The transform must be
rectilinear otherwise this function
returns `self`.
    @inlinable public func unapplying(_
transform: AffineTransform3D) -> Rect3D

    /// Unapplies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Note: The transform must be
rectilinear otherwise this function
returns `self`.
    @inlinable public func unapplying(_
transform: ProjectiveTransform3D) ->
Rect3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func applying(_ pose: Pose3D)
-> Rect3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    /// - Note: The pose's rotation angle
must be zero, otherwise this function
returns `self`.
    public func unapplying(_ pose:
Pose3D) -> Rect3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rect3D {

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func applying(_ scaledPose:
ScaledPose3D) -> Rect3D

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    /// - Note: The pose's rotation angle
must be zero, otherwise this function
returns `self`.
    public func unapplying(_ scaledPose:
ScaledPose3D) -> Rect3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Translatable3D {
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Scalable3D {

    /// Returns the rectangle scaled by
the specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled rectangle.
    @inlinable public func scaledBy(x:
Double = 1, y: Double = 1, z: Double = 1)
-> Rect3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Rotatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Shearable3D {

    /// Returns a sheared rectangle.
    ///
    /// - Parameter shear: The axis and
shear factors.
```

```
    /// - Returns The sheared rectangle.
    ///
    ///  Because affine transforms do not
preserve rectangles in general, this
function returns the
    ///  smallest rectangle that contains
the transformed corner points of the rect
parameter.
    @inlinable public func sheared(_
shear: AxisWithFactors) -> Rect3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Volumetric {

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    @inlinable public func contains(anyOf
points: [Point3D]) -> Bool

    /// Insets the rectangle by the
specified size.
    ///
    /// - Parameter dXYZ: The inset
amount.
```

```
    ///
    /// This function insets the
rectangle by the specified values on each
axis. The origin value is offset by
    /// the distance specified by the
`dXYZ` parameter. The size is adjusted by
`(2*dXYZ)`, relative to
    /// the source rectangle.
    ///
    /// If the width, height, or depth
are positive values, then the rectangle’s
size is decreased on that
    /// dimension.  If width, height, or
depth are negative values, the
rectangle’s size is increased on
    /// that dimension.
    @inlinable public mutating func
formInset(by dXYZ: Size3D)

    /// Returns the intersection of two
rectangles.
    ///
    /// - Parameter other: The second
rectangle.
    /// - Returns The intersection of
`self` and `other`.
    @inlinable public func intersection(_
other: Rect3D) -> Rect3D?
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Equatable {
    /// Returns a Boolean value
indicating whether two angles are equal.
    ///
    /// - Parameter lhs: The first angle
to compare.
    /// - Parameter rhs: The second angle
to compare.
    @inlinable public static func ==
(lhs: Rect3D, rhs: Rect3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D {

    /// Creates a rectangle from simd
vectors that describe the origin and
size.
    ///
    /// - Parameter origin: The origin of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
    public init(origin: simd_double3
= .zero, size: simd_double3)

    /// Creates a rectangle from Spatial
vectors that describe the origin and
size.
    ///
    /// - Parameter origin: The origin of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
    public init(origin: Vector3D = .zero,
size: Vector3D)

    ///  Creates a rectangle from simd
vectors that describe the center and
size.
    ///
    /// - Parameter center: The center of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
```

```
    public init(center: simd_double3
= .zero, size: simd_double3)

    ///  Creates a rectangle from Spatial
vectors that describe the center and
size.
    ///
    /// - Parameter center: The center of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
    public init(center: Vector3D = .zero,
size: Vector3D)

    /// Creates a rectangle that's the
bounding box of the specified points.
    ///
    /// - Parameter points: The array of
points.
    /// - Precondition: The `points`
array must contain at least one element.
    public init(points: [Point3D])

    /// Returns the corner points of the
rectangle.
    ///
    /// This function returns the
vertices in a clockwise direction,
starting from the origin:
    ///
    ///                    5-----6
    ///                    |     |
    ///                 1-----2  |
    ///                 |  |  |  |        y
z
    ///                 |  4--|--7        | /
    ///                 |     |           |/
    ///                 0-----3           +--
x
    ///
    /// For example, `points[0]` equals
`rect.origin`, and `points[6]` is at
`rect.origin`
    /// offset by `rect.size`.
    public var cornerPoints: [Point3D] {
get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rect3D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Angle2D {

    /// Returns a new angle from
floating-point radians.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(radians: T)
where T : BinaryFloatingPoint

    /// Returns a new angle from
floating-point degrees.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(degrees: T)
where T : BinaryFloatingPoint
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D {

    /// Returns a new rotation axis from
a single-precision vector.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ xyz:
```

```
simd_float3)

    /// Returns a new rotation axis from
the floating-point values.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(x: T, y: T,
z: T) where T : BinaryFloatingPoint
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension SIMD3 where Scalar == Float {

    /// Returns a new vector from a
Spatial rotation axis.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_
rotationAxis: RotationAxis3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D {
    /// Returns a new rotation from a
single-precision quaternion.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ quaternion:
simd_quatf)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension simd_quatf {

    /// Returns a new quaternion from a
Spatial rotation.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ rotation:
Rotation3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Point3D {

    /// Returns a new point from a
single-precision vector.
```

```
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ xyz:
simd_float3)

    /// Returns a new point from the
floating-point values.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(x: T, y: T,
z: T) where T : BinaryFloatingPoint
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension SIMD3 where Scalar == Float {

    /// Returns a new vector from a
Spatial point.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ point:
Point3D)
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D {

    /// Returns a vector point from a
single-precision vector.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ xyz:
simd_float3)

    /// Returns a new vector from the
floating-point values.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(x: T, y: T,
z: T) where T : BinaryFloatingPoint
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension SIMD3 where Scalar == Float {

    /// Returns a new vector from a
Spatial point.
    ///
    /// Values rounded to a representable
value, if necessary.
```

```
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ vector:
Vector3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D {

    /// Returns a new point from a
single-precision vector.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ xyz:
simd_float3)

    /// Returns a new point from the
floating-point values.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init<T>(width: T,
height: T, depth: T) where T :
BinaryFloatingPoint
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension SIMD3 where Scalar == Float {

    /// Returns a new vector from a
Spatial size.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ size:
Size3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rect3D {

    /// Creates a rectangle from single-
precision vectors that describe the
origin and size.
    ///
    /// - Parameter origin: The origin of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(origin:
```

```
simd_float3 = .zero, size: simd_float3)

    ///  Creates a rectangle from single-
precision vectors that describe the
center and size.
    ///
    /// - Parameter center: The center of
the rectangle.
    /// - Parameter size: The size of the
rectangle.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(center:
simd_float3 = .zero, size: simd_float3)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D {

    /// Returns a new affine transform
structure from the specified single-
precision 4 x 3 matrix.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ matrix:
simd_float4x3)
}
```

```
@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension AffineTransform3D {

    /// Returns a new affine transform
from a single-precision 4 x 4 matrix
truncated to a  4 x 3 matrix.
    ///
    /// - Parameter matrix: The source
matrix.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(truncating
matrix: simd_float4x4)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension simd_float4x3 {

    /// Returns a new matrix from a
Spatial affine transform.
    ///
    /// Values rounded to a representable
value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_
affineTransform: AffineTransform3D)
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS 
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D {

    /// Returns a new affine transform 
structure from the specified single-
precision 4 x 3 matrix.
    ///
    /// - note: This function is provided 
as a convenience. All Spatial storage and 
calculations are double-precision.
    @inlinable public init(_ matrix: 
simd_float4x4)
}

@available(macOS 13.0, iOS 16.0, tvOS 
16.0, watchOS 9.0, *)
extension simd_float4x4 {

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial 
projective transform.
    ///
    /// Values are rounded to a 
representable value, if necessary.
    ///
    /// - note: This function is provided 
as a convenience. All Spatial storage and 
calculations are double-precision.
    @inlinable public init(_ 
projectiveTransform: 
```

```
ProjectiveTransform3D)

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial affine
transform.
    ///
    /// Values are rounded to a
representable value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_
affineTransform: AffineTransform3D)

    /// Returns a new 4 x 4 single-
precision matrix from a Spatial pose.
    ///
    /// Values are rounded to a
representable value, if necessary.
    ///
    /// - note: This function is provided
as a convenience. All Spatial storage and
calculations are double-precision.
    @inlinable public init(_ pose:
Pose3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D {

    /// Creates a ray from Spatial
```

```
primitives that describe the origin and
direction.
    ///
    /// - Parameter origin: The origin of
the ray.
    /// - Parameter direction: The
direction of the ray.
    /// - note: This function normalizes
the direction vector.
    public init(origin: Point3D = .zero,
direction: Vector3D)

    /// Creates a ray from double-
precision simd vectors that describe the
origin and direction.
    ///
    /// - Parameter origin: The origin of
the ray.
    /// - Parameter direction: The
direction of the ray.
    /// - note: This function normalizes
the direction vector.
    public init(origin: simd_double3
= .zero, direction: simd_double3)

    /// Creates a ray from single-
precision simd vectors that describe the
origin and direction.
    ///
    /// - Parameter origin: The origin of
the ray.
    /// - Parameter direction: direction
of the rectangle.
```

```swift
    /// - note: This function normalizes
the direction vector.
    public init(origin: simd_float3
= .zero, direction: simd_float3)

    /// Returns `true` if the the ray
intersects the specified rectangle.
    ///
    /// - Parameter rect: The second
primitive.
    public func intersects(_ rect:
Rect3D) -> Bool

    /// Returns a ray that's transformed
by the specified pose.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed ray.
    /// This function rotates the ray's
direction by the pose's rotation and
offsets the ray's origin by the pose's
position.
    public func applying(_ pose: Pose3D)
-> Ray3D

    /// Applies a pose.
    ///
    /// - Parameter pose: The pose.
    /// This function rotate's the ray's
direction by the pose's rotation and sets
the ray's origin to the pose's position.
    public mutating func apply(_ pose:
Pose3D)
```

```
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Ray3D {

    /// Returns a ray that's transformed
by the inverse of the specified scaled
pose.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed ray.
    /// This function rotates the ray's
direction by the pose's rotation and
offsets the ray's origin by the pose's
position.
    public func unapplying(_ scaledPose:
ScaledPose3D) -> Ray3D

    /// Returns a ray that's transformed
by the specified scaled pose.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed ray.
    /// This function rotates the ray's
direction by the pose's rotation and
offsets the ray's origin by the pose's
position.
    public func applying(_ scaledPose:
ScaledPose3D) -> Ray3D
}
```

```
@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Ray3D : Primitive3D {

    /// Returns a ray that's transformed
by the specified affine transform.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed ray.
    ///
    /// This function applies the
transform to the ray's origin and
direction.
    public func applying(_ transform:
AffineTransform3D) -> Ray3D

    /// Returns a ray that's transformed
by the inverse of the specified affine
transform.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed ray.
    ///
    /// This function applies the
transform to the ray's origin and
direction.
    public func unapplying(_ transform:
AffineTransform3D) -> Ray3D

    /// Returns a ray that's transformed
```

```
by the specified projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed ray.
    ///
    /// This function applies the
transform to the ray's origin and
direction.
    public func applying(_ transform:
ProjectiveTransform3D) -> Ray3D

    /// Returns a ray that's transformed
by the inverse of the specified
projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed ray.
    ///
    /// This function applies the
transform to the ray's origin and
direction.
    public func unapplying(_ transform:
ProjectiveTransform3D) -> Ray3D

    /// Returns a ray that's transformed
by the inverse of the specified pose.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed ray.
    /// This function rotates the ray's
direction by the pose's rotation and
```

```
offsets the ray's origin by the pose's
position.
    public func unapplying(_ pose:
Pose3D) -> Ray3D

    /// Returns a ray that's rotated by
the specified rotation around a specified
pivot.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Parameter pivot: The center of
rotation.
    public func rotated(by rotation:
Rotation3D, around pivot: Point3D) ->
Ray3D

    /// Returns a ray that's rotated by
the specified rotation around a specified
pivot.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Parameter pivot: The center of
rotation.
    public func rotated(by quaternion:
simd_quatd, around pivot: Point3D) ->
Ray3D

    /// A Boolean value that indicates
whether all of the values of the ray are
finite.
    public var isNaN: Bool { get }
    /// A Boolean value that indicates
whether all of the values of the ray are
finite.
    public var isFinite: Bool { get }

    /// A Boolean value that indicates
whether all of the values of the ray are
zero.
    public var isZero: Bool { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : Translatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : Rotatable3D {

    /// Returns a ray that's rotated by
the specified rotation.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns A ray with a direction
that's rotated by the specified rotation.
    public func rotated(by rotation:
Rotation3D) -> Ray3D

    /// Returns a ray that's rotated by
the specified quaternion.
```

```
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns A ray with a direction
that's rotated by the specified
quaternion.
    public func rotated(by quaternion:
simd_quatd) -> Ray3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : Equatable {

    /// Returns a Boolean value
indicating whether two rays are equal.
    ///
    /// - Parameter lhs: The first angle
to compare.
    /// - Parameter rhs: The second angle
to compare.
    @inlinable public static func ==
(lhs: Ray3D, rhs: Ray3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : Hashable {
    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : Codable {
    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Ray3D : CustomStringConvertible
{

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
```

```
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Ray3D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : @unchecked Sendable
{

    /// A vector with values `[1, 0, 0]`.
    public static let right: Vector3D

    /// A vector with values `[0, 1, 0]`.
    public static let up: Vector3D

    /// A vector with values `[0, 0, 1]`.
    public static let forward: Vector3D

    /// Returns a vector from the
specified values.
    ///
    /// - Parameter x: The first element
of the vector.
    /// - Parameter y: The second element
of the vector.
    /// - Parameter z: The third element
of the vector.
    @inlinable public init(x: Double = 0,
y: Double = 0, z: Double = 0)

    /// Returns a new vector from a
double-precision vector.
    ///
    /// - Parameter xyz: The source
vector.
    @inlinable public init(_ xyz:
```

```
simd_double3)

    /// Returns a new vector from a size.
    ///
    /// - Parameter size: The source
size.
    @inlinable public init(_ size:
Size3D)

    /// Returns a new vector from a
point.
    ///
    /// - Parameter point: The source
point.
    @inlinable public init(_ point:
Point3D)

    /// Returns a new vector from a
rotation axis.
    ///
    /// - Parameter axis: The source
rotation axis.
    @inlinable public init(_ axis:
RotationAxis3D)

    /// Returns the dot product of this
vector and the specified vector.
    @inlinable public func dot(_ other:
Vector3D) -> Double

    /// Returns the cross product of this
vector and the specified vector.
    @inlinable public func cross(_ other:
```

```
Vector3D) -> Vector3D

    /// Returns this vector with a length
of `1`.
    @inlinable public var normalized:
Vector3D { get }

    /// Normalizes the vector.
    @inlinable public mutating func
normalize()

    /// Returns this vector projected
onto the specified vector.
    @inlinable public func projected(_
other: Vector3D) -> Vector3D

    /// Returns the reflection direction
of this incident vector and the specified
unit normal vector.
    @inlinable public func reflected(_
normal: Vector3D) -> Vector3D

    @inlinable public var length: Double
{ get }

    @inlinable public var lengthSquared:
Double { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Primitive3D {
    /// Returns the vector resulting from
applying an affine transform to the
vector.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed
primitive.
    @inlinable public func applying(_
transform: AffineTransform3D) -> Vector3D

    /// Returns the vector resulting from
applying a projective transform to the
vector.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed
primitive.
    @inlinable public func applying(_
transform: ProjectiveTransform3D) ->
Vector3D

    /// Returns the vector resulting from
unapplying an affine transform to the
vector.
    ///
    /// - Parameter transform: The affine
transform.
    /// - Returns The transformed
primitive.
    @inlinable public func unapplying(_
transform: AffineTransform3D) -> Vector3D
```

```
    /// Returns the vector resulting from
unapplying a projective transform to the
vector.
    ///
    /// - Parameter transform: The
projective transform.
    /// - Returns The transformed
primitive.
    @inlinable public func unapplying(_
transform: ProjectiveTransform3D) ->
Vector3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func applying(_ pose: Pose3D)
-> Vector3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ pose:
Pose3D) -> Vector3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Vector3D {

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func applying(_ scaledPose:
ScaledPose3D) -> Vector3D

    /// Returns the primitive resulting
from unapplying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ scaledPose:
ScaledPose3D) -> Vector3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Rotatable3D {

    /// Returns the vector rotated by the
specified rotation around the origin.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns The rotated vector.
    @inlinable public func rotated(by
rotation: Rotation3D) -> Vector3D

    /// Returns the vector rotated by the
specified quaternion around the origin.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns The rotated vector.
    @inlinable public func rotated(by
quaternion: simd_quatd) -> Vector3D

    /// Returns the rotation from the
normalized first vector to the normalized
second vector.
    @inlinable public func rotation(to
other: Vector3D) -> Rotation3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Scalable3D {

    /// Returns the vector scaled by the
specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
```

```
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled vector.
    @inlinable public func scaledBy(x:
Double = 1, y: Double = 1, z: Double = 1)
-> Vector3D

    /// Returns the vector scaled by the
specified size.
    ///
    /// - Parameter size: The size
structure that defines that scale.
    /// - Returns The scaled vector.
    @inlinable public func scaled(by
size: Size3D) -> Vector3D

    /// Returns the entity uniformly
scaled by the specified scalar value.
    ///
    /// - Parameter scale: The value that
defines that scale.
    /// - Returns The scaled vector.
    @inlinable public func
uniformlyScaled(by scale: Double) ->
Vector3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Shearable3D {

    /// Returns a sheared vector.
    ///
    /// - Parameter shear: The axis and
shear factors.
    /// - Returns The sheared vector.
    @inlinable public func sheared(_
shear: AxisWithFactors) -> Vector3D
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Vector3D : AdditiveArithmetic {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Equatable {

    /// Returns a Boolean value
indicating whether two points are equal.
    ///
    /// - Parameter lhs: The first point
to compare.
    /// - Parameter rhs: The second point
to compare.
    @inlinable public static func ==
(lhs: Vector3D, rhs: Vector3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
```

```
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Vector3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
```

```
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Vector3D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
```

```
extension Size3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D {

    /// A size with a width, height, and
depth of `1`.
    public static let one: Size3D

    /// Returns a size structure from the
specified values.
    ///
    /// - Parameter width: The width of
the size.
    /// - Parameter height: The height of
the size.
    /// - Parameter depth: The width of
the depth.
    @inlinable public init(width: Double
= 0, height: Double = 0, depth: Double =
0)

    /// Returns a size structure from the
specified 3-element vector.
    ///
    /// - Parameter xyz: The source
vector.
    @inlinable public init(_ xyz:
simd_double3)

    /// Returns a size structure from the
```

```
specified point.
    ///
    /// - Parameter point: The source
point.
    @inlinable public init(_ point:
Point3D)

    /// Returns a size structure from the
specified Spatial vector.
    ///
    /// - Parameter xyz: The source
vector.
    @inlinable public init(_ xyz:
Vector3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Primitive3D {

    /// Unapplies an affine transform.
    ///
    /// - Parameter transform: The affine
transform.
    @inlinable public func unapplying(_
transform: AffineTransform3D) -> Size3D

    /// Unapplies a projective transform.
    ///
    /// - Parameter transform: The
projective transform.
    @inlinable public func unapplying(_
transform: ProjectiveTransform3D) ->
```

```
Size3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func applying(_ pose: Pose3D)
-> Size3D

    /// Returns the primitive resulting
from applying a pose to the primitive.
    ///
    /// - Parameter pose: The pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ pose:
Pose3D) -> Size3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Size3D {

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
```

```
    public func applying(_ scaledPose:
ScaledPose3D) -> Size3D

    /// Returns the primitive resulting
from applying a scaled pose to the
primitive.
    ///
    /// - Parameter pose: The scaled
pose.
    /// - Returns The transformed
primitive.
    public func unapplying(_ scaledPose:
ScaledPose3D) -> Size3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Scalable3D {

    /// Returns the size scaled by the
specified values.
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled size.
    @inlinable public func scaledBy(x:
Double = 1, y: Double = 1, z: Double = 1)
-> Size3D
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Rotatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Shearable3D {

    /// Returns a sheared size.
    ///
    /// - Parameter shear: The axis and
shear factors.
    /// - Returns The sheared size.
    @inlinable public func sheared(_
shear: AxisWithFactors) -> Size3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Volumetric {

    /// The size of the volume.
    public var size: Size3D { get }

    /// Returns a Boolean value that
indicates whether this volume contains
any of the specified points.
    ///
    /// - Parameter points: An array of
points.
    /// - Returns A Boolean value that
indicates whether `self` contains any of
the specified points.
    @inlinable public func contains(anyOf
points: [Point3D]) -> Bool

    /// Returns the intersection of two
volumetric entities.
    ///
    /// - Parameter other: The second
primitive.
    /// - Returns The intersection of
`self` and `other`.
    @inlinable public func intersection(_
other: Size3D) -> Size3D?
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Equatable {

    /// Returns a Boolean value
indicating whether two sizes are equal.
    ///
    /// - Parameter lhs: The first size
to compare.
    /// - Parameter rhs: The second size
to compare.
    @inlinable public static func ==
(lhs: Size3D, rhs: Size3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
```

```
extension Size3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D : Codable {
    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Size3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
```

```
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Size3D : CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D : @unchecked
Sendable {

    public init(_ quaternion: simd_quatd)

    /// The angle of the rotation.
    public var angle: Angle2D

    /// The axis of the rotation.
    public var axis: RotationAxis3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D {

    /// Returns a rotation that's the
look at direction from the eye position
to the target.
    ///
    /// - Parameter position: The eye
position.
    /// - Parameter target: The point
that the rotation looks at.
    /// - Parameter up: The up direction.
    ///
    /// - note: This function creates a
rotation where `+z` is forward.
    public init(position: Point3D =
Point3D(x: 0, y: 0, z: 0), target:
Point3D, up: Vector3D = Vector3D(x: 0, y:
1, z: 0))
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rotation3D {

    /// Returns a rotation with the
specified forward vector.
    ///
    /// - Parameter forward The forward
direction.
    ///
    /// - note: This function creates a
rotation with an up vector that's
    /// `Vector3D(x: 0, y: 1, z: 0)` and
where `+z` is forward.
    public init(forward: Vector3D)
}

extension Rotation3D {

    @available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
    @backDeployed(before: macOS 14.0, iOS
17.0, tvOS 17.0, watchOS 10.0)
    public init()
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
```

```
extension Rotation3D : Rotatable3D {

    /// Returns the entity rotated by the
specified rotation around the origin.
    ///
    /// - Parameter rotation: The
rotation.
    /// - Returns The rotated primitive.
    public func rotated(by rotation:
Rotation3D) -> Rotation3D

    /// Returns the entity rotated by the
specified quaternion around the origin.
    ///
    /// - Parameter quaternion: The
quaternion that defines the rotation.
    /// - Returns The rotated primitive.
    public func rotated(by quaternion:
simd_quatd) -> Rotation3D

    /// An enumeration that specifies
which path a slerp operation should take.
    public enum SlerpPath {

        /// Spherical linear
interpolation along the shortest arc
between two rotations.
        case shortest

        /// Spherical linear
interpolation along the longest arc
between two rotations.
        case longest
        /// For less than or equal to
180°, the spherical linear interpolation
along the shortest arc between
        /// two rotations, otherwise
along the longest arc.
        /// - note: This option returns
results that are identical to
        /// `* (lhs: Rotation3D, rhs:
Double) -> Rotation3D` where `from` is
equal
        /// to `Rotation3D.identity`.
        case automatic

        /// Returns a Boolean value
indicating whether two values are equal.
        ///
        /// Equality is the inverse of
inequality. For any values `a` and `b`,
        /// `a == b` implies that `a !=
b` is `false`.
        ///
        /// - Parameters:
        ///   - lhs: A value to compare.
        ///   - rhs: Another value to
compare.
        public static func == (a:
Rotation3D.SlerpPath, b:
Rotation3D.SlerpPath) -> Bool

        /// Hashes the essential
components of this value by feeding them
into the
```

```
        /// given hasher.
        ///
        /// Implement this method to
conform to the `Hashable` protocol. The
        /// components used for hashing
must be the same as the components
compared
        /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
        /// with each of these
components.
        ///
        /// - Important: In your
implementation of `hash(into:)`,
        ///   don't call `finalize()` on
the `hasher` instance provided,
        ///   or replace it with a
different instance.
        ///   Doing so may become a
compile-time error in the future.
        ///
        /// - Parameter hasher: The
hasher to use when combining the
components
        ///   of this instance.
        public func hash(into hasher:
inout Hasher)

        /// The hash value.
        ///
        /// Hash values are not
guaranteed to be equal across different
executions of
```

```
        /// your program. Do not save
hash values to use during a future
execution.
        ///
        /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
        ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
        ///   The compiler provides an
implementation for `hashValue` for you.
        public var hashValue: Int { get }
    }

    /// Returns the spherical linear
interpolation along the either the
shortest or longest arc
    /// between two rotations.
    public static func slerp(from:
Rotation3D, to: Rotation3D, t: Double,
along path: Rotation3D.SlerpPath
= .shortest) -> Rotation3D
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rotation3D {

    /// Returns an interpolated value
between two rotations along a spherical
cubic spline.
    ///
    /// - Parameter leftEndpoint: The
left endpoint of the previous interval.
    /// - Parameter from: The starting
rotation.
    /// - Parameter to: The ending
rotation.
    /// - Parameter rightEndpoint: The
right endpoint of the previous interval.
    ///
    /// Use this function to smoothly
interpolate between a sequence of
rotations.
    ///
    ///- returns: A new rotation that's
the interpolated value between the two
rotations along a spherical cubic spline.
    public static func
spline(leftEndpoint r0: Rotation3D, from
r1: Rotation3D, to r2: Rotation3D,
rightEndpoint r3: Rotation3D, t: Double)
-> Rotation3D
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D : Equatable {

    /// Returns a Boolean value
indicating whether two rotations are
equal.
    ///
    /// - Parameter lhs: The first
rotation to compare.
```

```
    /// - Parameter rhs: The second
rotation to compare.
    @inlinable public static func ==
(lhs: Rotation3D, rhs: Rotation3D) ->
Bool
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rotation3D {

    /// Returns a Boolean value
indicating whether two rotations are
equal within a specified tolerance.
    ///
    /// - Parameter other: The second
rotation.
    /// - Parameter tolerance: The
tolerance of the comparison.
    @inlinable public func
isApproximatelyEqual(to other:
Rotation3D, tolerance: Double =
sqrt(.ulpOfOne)) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
```

```
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Rotation3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension Rotation3D : CustomReflectable
{

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension EulerAngles {
    public typealias Order =
__SPEulerAngleOrder

    /// Returns a new Euler angles
structure from the single-precision
angles.
    ///
    /// - Parameter angles A vector that
specifies the angles, in radians.
    /// - Parameter order The Euler angle
ordering.
    public init(angles: simd_float3,
order: EulerAngles.Order)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension EulerAngles {

    /// Returns a new Euler angles
structure from three angle structures.
    ///
    /// - Parameter x The first angle.
    /// - Parameter y The second angle.
    /// - Parameter z The third angle.
    /// - Parameter order The Euler angle
ordering.
    public init(x: Angle2D, y: Angle2D,
z: Angle2D, order: EulerAngles.Order)
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension Rotation3D {

    ///  Returns the rotation's swing-
twist decomposition for a given twist
axis.
    ///
    ///  - Parameter twistAxis The twist
axis.
    ///  - Returns: A tuple that contains
the swing rotation and the twist
rotation.
    public func swingTwist(twistAxis:
RotationAxis3D) -> (swing: Rotation3D,
twist: Rotation3D)
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension __SPEulerAngleOrder {

    @available(*, deprecated, renamed:
"EulerAngles.Order.xyz")
    public static let pitchYawRoll:
__SPEulerAngleOrder

    @available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
    public static let xyz:
__SPEulerAngleOrder

    @available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
```

```
    public static let zxy:
__SPEulerAngleOrder
}

@available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
extension __SPEulerAngleOrder :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : @unchecked
Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D {

    /// The translation component
    @inlinable public var translation:
Vector3D

    /// Returns a Boolean value
indicating whether two transforms are
equal within a specified tolerance.
    ///
    /// - Parameter transform: The second
transform.
    /// - Parameter tolerance: The
tolerance of the comparison.
```

```
    @inlinable public func
isApproximatelyEqual(to other:
AffineTransform3D, tolerance: Double =
sqrt(.ulpOfOne)) -> Bool

    /// Returns a new identity affine
transform.
    @inlinable public init()

    /// Returns a new scale, rotate,
translate transform.
    ///
    /// - Parameter scale: The scale.
    /// - Parameter rotation: The
rotation.
    /// - Parameter translation: The
translation.
    @available(*, deprecated, message:
"Use `Vector3D` variant.")
    @inlinable public init(scale: Size3D
= Size3D(width: 1.0, height: 1, depth:
1), rotation: Rotation3D = .zero,
translation: Size3D)

    /// Returns a new scale, rotate,
translate transform.
    ///
    /// - Parameter scale: The scale.
    /// - Parameter rotation: The
rotation.
    /// - Parameter translation: The
translation.
    @inlinable public init(scale: Size3D
= Size3D(width: 1.0, height: 1, depth:
1), rotation: Rotation3D = .zero,
translation: Vector3D = .zero)

    /// Returns a new shear transform
    ///
    /// - Parameter shear: The shear.
    @inlinable public init(shear:
AxisWithFactors)

    /// Returns a new transform from the
specified pose.
    ///
    /// - Parameter pose: The source pose
    @inlinable public init(pose: Pose3D)

    @available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
    @inlinable public init(scaledPose:
ScaledPose3D)

    /// Returns the transform sheared by
the specified shear.
    ///
    /// - Parameter shear: The axis and
shear factors.
    /// - Returns The sheared transform.
    @inlinable public func sheared(_
shear: AxisWithFactors) ->
AffineTransform3D

    /// Returns the transform scaled by
the specified values.
```

```
    ///
    /// - Parameter x: The scale factor
on the `x` dimension.
    /// - Parameter y: The scale factor
on the `y` dimension.
    /// - Parameter z: The scale factor
on the `z` dimension.
    /// - Returns The scaled transform.
    @inlinable public func scaledBy(x:
Double = 1, y: Double = 1, z: Double = 1)
-> AffineTransform3D

    /// Returns true if the transform is
uniform over the specified dimensions.
    ///
    /// - Parameter overDimensions: The
dimensions that the function tests are
uniform.
    ///
    /// If you specify `overDimensions`
as `Dimension3DSet.all`, the function
returns the same
    /// result as
`Transform3D.getter:isUniform`. If you
specify as `overDimensions` with
    /// zero or one dimension, the
function returns the same result as
`Transform3D.getter:isRectilinear`.
    @inlinable public func
isUniform(overDimensions: Dimension3DSet)
-> Bool

    /// Returns the transform flipped
```

```
along the specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    @inlinable public func flipped(along
axis: Axis3D) -> AffineTransform3D

    /// Flips the transform along the
specified axis.
    ///
    /// - Parameter axis: The axis of the
flip operation.
    public mutating func flip(along axis:
Axis3D)

    /// The affine transform's rotation.
    ///
    /// This function can't extract
rotation from a non-scale-rotate-
translate affine transform. In that case,
    /// the function returns `nil`.
    @inlinable public var rotation:
Rotation3D? { get }

    /// The affine transform's inverse.
    ///
    /// If the source transform isn't
invertible, this value is `nil`.
    @inlinable public var inverse:
AffineTransform3D? { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Equatable {

    /// Returns a Boolean value
indicating whether two transforms are
equal.
    ///
    /// - Parameter lhs: The first
transform to compare.
    /// - Parameter rhs: The second
transform to compare.
    @inlinable public static func ==
(lhs: AffineTransform3D, rhs:
AffineTransform3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension AffineTransform3D {

    /// Returns a new affine transform
structure by applying a change-of-basis.
    ///
    /// - Parameter from: The old basis.
    /// - Parameter to: The new basis.
    /// - returns: A new affine transform
structure.
    public func changeBasis(from:
AffineTransform3D = .identity, to:
AffineTransform3D) -> AffineTransform3D?
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
```

```
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension AffineTransform3D :
CustomReflectable {

    /// The custom mirror for this
instance.
```

```
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D : @unchecked
Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Axis3D : @unchecked Sendable {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension Axis3D {

    /// The operation is along the x-
axis.
    @inlinable public static var x:
Axis3D { get }

    /// The operation is along the y-
axis.
    @inlinable public static var y:
Axis3D { get }

    /// The operation is along the z-
axis.
    @inlinable public static var z:
Axis3D { get }
```

```
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D {

    /// The x-axis, expressed in unit
coordinates.
    public static let x: RotationAxis3D

    /// The y-axis, expressed in unit
coordinates.
    public static let y: RotationAxis3D

    /// The z-axis, expressed in unit
coordinates.
    public static let z: RotationAxis3D

    /// The xy-axis, expressed in unit
coordinates.
    public static let xy: RotationAxis3D

    /// The yz-axis, expressed in unit
coordinates.
    public static let yz: RotationAxis3D

    /// The xz-axis, expressed in unit
coordinates.
    public static let xz: RotationAxis3D

    /// The xyz-axis, expressed in unit
coordinates.
    public static let xyz: RotationAxis3D
    /// Returns a rotation axis from the
specified values.
    ///
    /// - Parameter x: The rotation value
on the x-axis.
    /// - Parameter y: The rotation value
on the y-axis.
    /// - Parameter z: The rotation value
on the z-axis.
    @inlinable public init(x: Double = 0,
y: Double = 0, z: Double = 0)

    /// The x-axis value.
    @inlinable public var x: Double

    /// The y-axis value.
    @inlinable public var y: Double

    /// The z-axis value.
    @inlinable public var z: Double

    /// Returns a new rotation axis from
a single-precision vector.
    @inlinable public init(_ xyz:
simd_double3)

    /// Returns a new rotation axis from
a Spatial vector.
    @inlinable public init(_ xyz:
Vector3D)
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D : Equatable {

    /// Returns a Boolean value
indicating whether two rotation axes are
equal.
    ///
    /// - Parameter lhs: The first
rotation axis to compare.
    /// - Parameter rhs: The second
rotation axis to compare.
    @inlinable public static func ==
(lhs: RotationAxis3D, rhs:
RotationAxis3D) -> Bool
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D : Hashable {

    /// Hashes the essential components
of this value by feeding them into the
given hasher.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components of this
instance.
    public func hash(into hasher: inout
Hasher)

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D : Codable {

    /// Encodes this value into the given
encoder.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// Creates a new instance by
decoding from the given decoder.
    ///
    /// - Parameter decoder: The decoder
to read data from.
```

```
    public init(from decoder: any
Decoder) throws
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension RotationAxis3D :
CustomStringConvertible {

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
    public var description: String {
get }
}

@available(macOS 15.0, iOS 18.0, tvOS
18.0, watchOS 11.0, *)
extension RotationAxis3D :
CustomReflectable {

    /// The custom mirror for this
instance.
    public var customMirror: Mirror { get
}
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Shearable3D
{
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Scalable3D
{
}
```

```
@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D : Rotatable3D
{
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension AffineTransform3D :
Translatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
Shearable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
Scalable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
extension ProjectiveTransform3D :
Rotatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
```

```
extension ProjectiveTransform3D :
Translatable3D {
}

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.x")
public var x: Axis3D

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.y")
public var y: Axis3D

@available(macOS 13.0, iOS 16.0, tvOS
16.0, watchOS 9.0, *)
@available(*, deprecated, renamed:
"Axis3D.z")
public var z: Axis3D
```