Maths for computer graphics and games

Gaëtan Blaise-Cazalet - Head of programming section

Basics

Changing numbers

Min and Max

```
max(a, b) { if (a >= b) return a; else return b; }
```

```
min(a, b) { if (a <= b) return a; else return b; }
```

Clamp

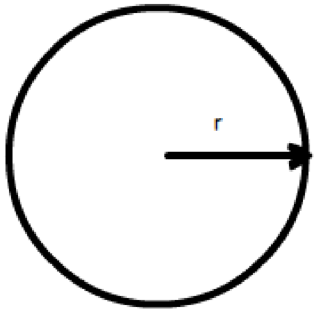
```
clamp(low, n, high) {
  return min(max(low, n), high);
}
```

Round

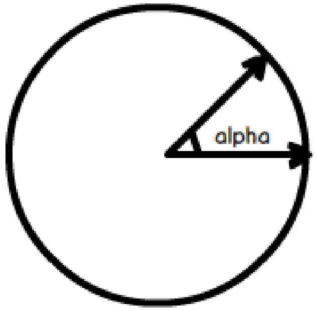
```
math.round(n, deci = 0) {
  deci = 10^deci;
  return floor(n*deci + 0.5)/deci;
}
```

Circles

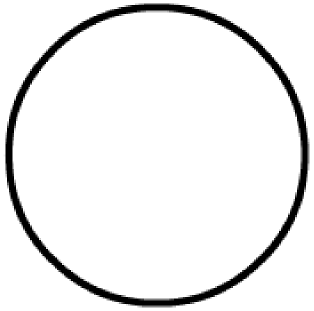
Radius



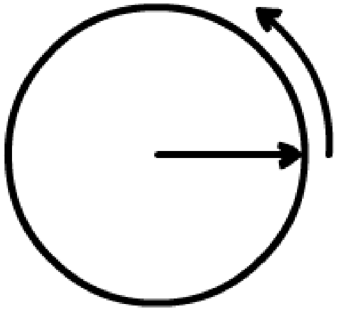
Degrees, radians and perimeter

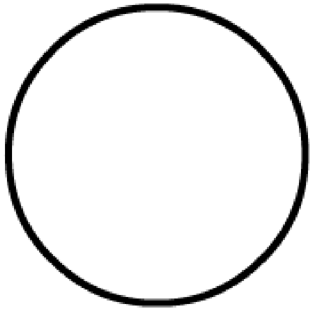


360 degrees



2Pi Radian





1 Radian is the projection of the radius on the circle perimeter (for a 1-radius circle)

$\longrightarrow$ Perimeter = 2Pi * radius

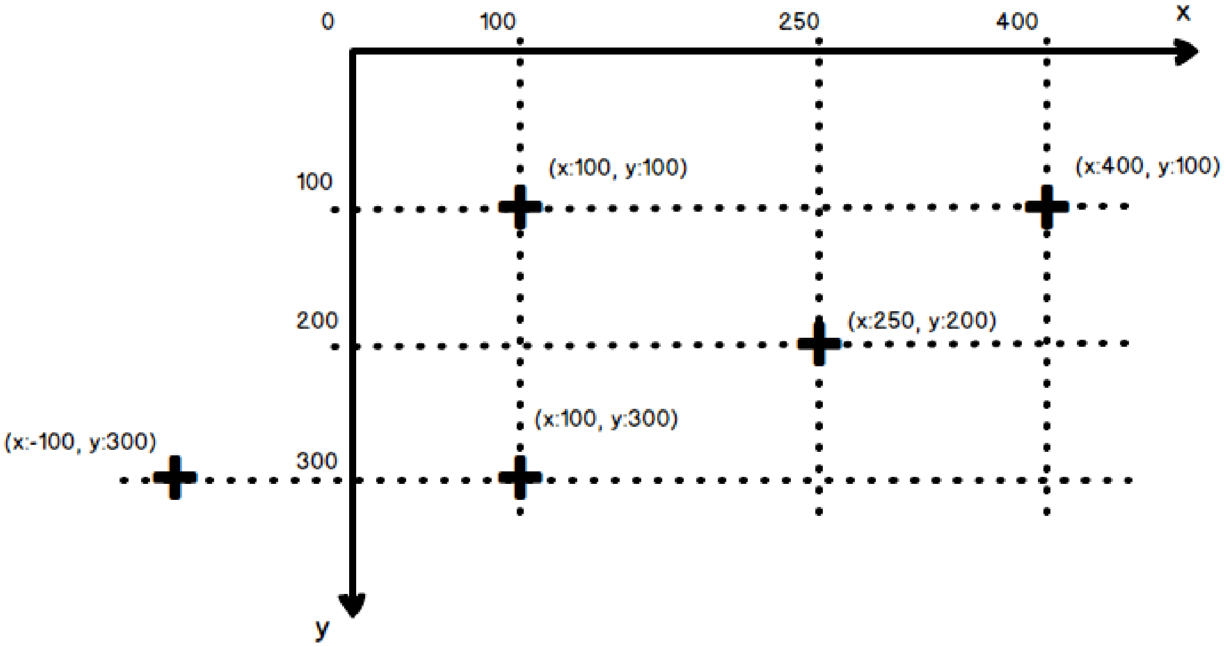
$\downarrow$

1 Radian = (180 / Pi) degrees

Area

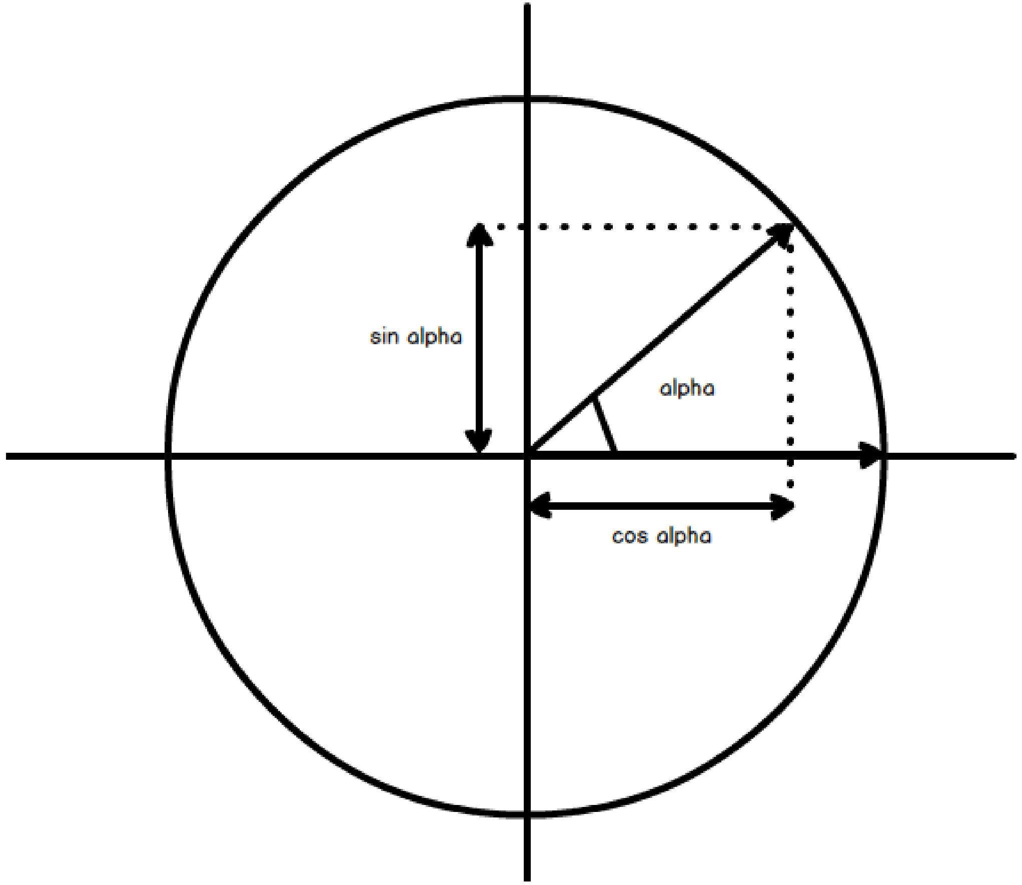
Area = 2 Pi * radius * radius

Cartesian coordinates



Trigonometry

Cosinus and sinus



Lot of trigonometry formulas

https://en.wikipedia.org/wiki/List_of_trigonometric_identities

Applied trigonometry in a 2D game

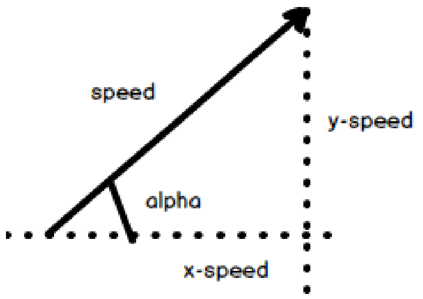
https://www.raywenderlich.com/2736-trigonometry-for-game-programming-part-1-2
https://www.raywenderlich.com/2737-trigonometry-for-game-programming-part-2-2

Move a 2d entity at constant speed:

```
x += speed.x * cos(angle) * dt;
y += speed.y * sin(angle) * dt;
```

arctan2

arctan2



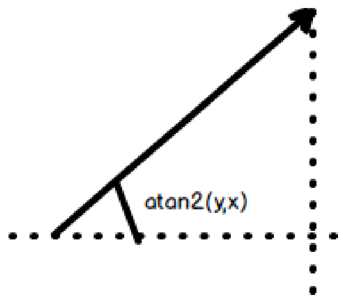
If you want to get an angle alpha, you can use :

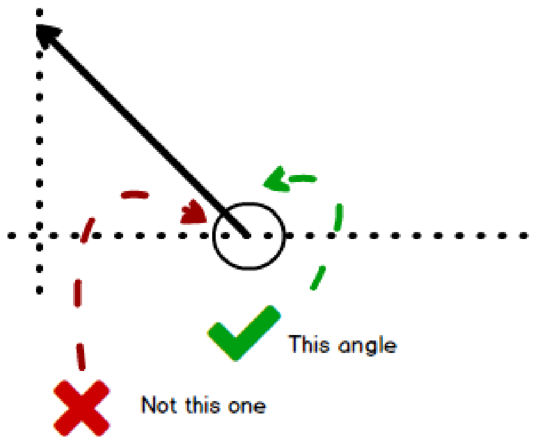
atan(opposite/adjacent) = alpha

For a 2d move, opposite will be y-speed, and adjacant will be x-speed.

Now it can happen that x-speed is 0, because the move is vertical. Your angle computing will fail with a divided-by-zero error. That is why most math librairies implement a atan2(y, x) function, that will support a zero x value.

Beware: atan2 gives the angle from the 0 degree line :



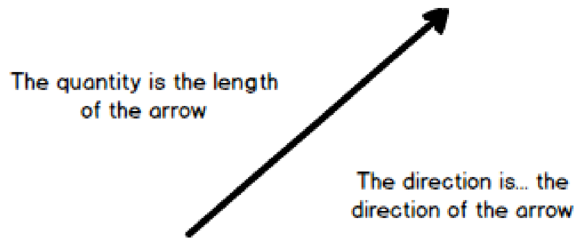


Vectors

Vectors, dimensions and coordinates

Definition

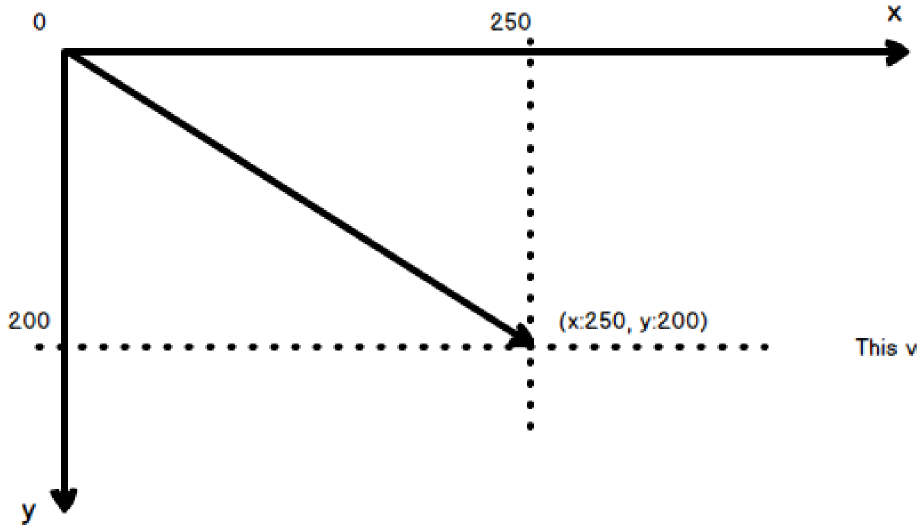
A vector is a **quantity** with a **direction**


The quantity is the length of the arrow
The direction is... the direction of the arrow

Cartesian interpretation

Because a vector is only a quandtity and a direction, the origin of the vector has no importance.

Still, we can use a n-dimension cartesian coordinate system to specify the vector. We suppose the origin to be (0, 0 ...) and the arrow to be at the coordinate that speecifies the vector.



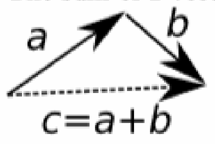
This vector will be written as :

$$\begin{pmatrix} 250 \\ 200 \end{pmatrix}$$

Vector properties

Vector addition

The sum of 2 vectors completes the triangle.



also $a = c - b$ and $b = c - a$

Position change

Positions are points. Points are n-dimensional elements in a n-dimension space. Although we use vectors to represent points in n-dimension spaces, they are different objects.

Still, using vector to represent points is handy : we can add vectors together. Thus, we can add a position and a speed vector to get future position.

Scalar multiplication

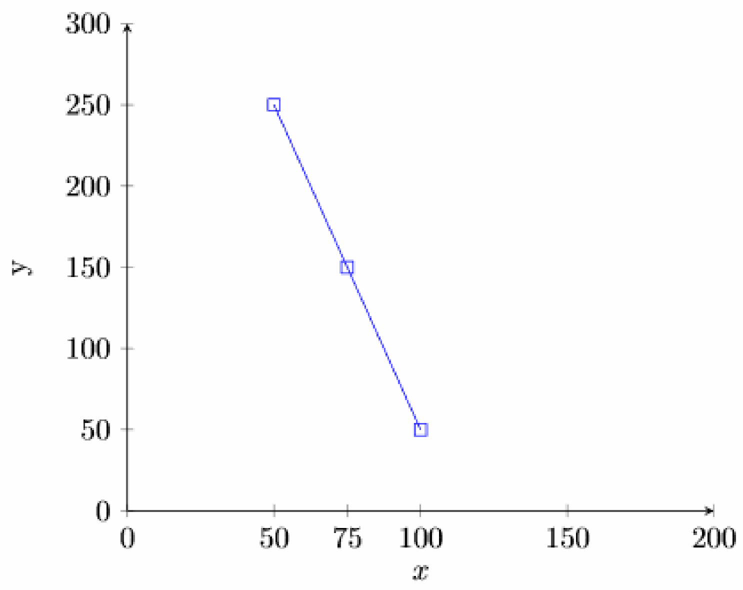
Vectors can be multiplied by scalars (real numbers). You just have to multiply the coordinates.

$$2\begin{pmatrix}250\\200\end{pmatrix} = \begin{pmatrix}500\\400\end{pmatrix}$$

Medium point

To get a point in the middle of to points, you just have to add their coordinate and divide by two.

$$(\begin{pmatrix}100\\50\end{pmatrix} + \begin{pmatrix}50\\250\end{pmatrix}) \;/\; 2 = \begin{pmatrix}75\\150\end{pmatrix}$$



Linear (lerp) interpolation between two points

If you want a point to go from one point to an other, you can use linear interpolation.

```
Vector2 pointA = Vector2(50, 250);
Vector2 pointB = Vector2(100, 50);

Vector2 point = a * pointA + (1 - a) * pointB;
```

While a go from 0 to 1, point will go from pointA to pointB.

Vector magnitude

Definition

The magnitude is the quantity, the "length" of a vector. Magnitude is also called "norm".

In a cartesian coordinate system, we can use Pythagore's theorem to get the magnitude of a vector :

$$\mathrm{magnitude}^2 = \mathrm{x}^2 + \mathrm{y}^2$$

$$\mathrm{magnitude} = \sqrt{x^2 + y^2}$$

If the vector is v, we can write the magnitude as follows :

$$\|v\| = \sqrt{x^2 + y^2}$$

Distance to target

To get the distance between two points, you can substract the two points coordinates, to get the vector between them.
Then get the magnitude of this vector to get the distance.

```
dist(Vector2 a, Vector2 b) {
    v = b - a;
    return sqrt( v.x * v.x + v.y * v.y );
}
```

Using squares instead of square roots

The computation of square roots is quite intensive. That's why we prefer working with squared distances instead of working with distances.

For instance, if you want to know if a player is in the circular field of view of an enemy, compare the squared distance between the player and the enemy with the squared size of the enemy's field of view radius.

Unit vector

Definition

A unit vector is a vector of magnitude 1.

Unit (normalized) vector

To get a unit vector from a vector, that is to say to "normalize" it, you just have to divide this vector by its magnitude.

$$\hat{A} = \frac{\vec{A}}{||\vec{A}||}$$

Dot product

Definition

The dot product of two vectors A and B is a scalar (a real number) defined by :

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$

Dot product and angle

An other dot product definition is :

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos(\theta)$$

So we can compute the angle between a and b with the dot product :

$$\theta = arccos(\frac{\vec{A} \cdot \vec{B}}{||\vec{A}||||\vec{B}||})$$

$$\theta = arccos(\hat{A} \cdot \hat{B})$$

Properties

Dot product is commutative and distributive.

Colinearity

Because of above property, the dot product can be used to mesure colinearity. A and B are unit vectors.

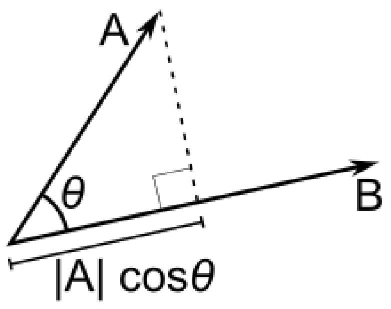
If A . B = 0, then A and B are perpendicular.
if A . B > 0, A and B "go into the same direction".
if A . B < 0, A and B "go into opposite directions".
if A . B = 1 or -1, A and B are parallel, respectively with same and opposite directions.

Projection of one vector onto an other

Because of the cosine definition, the dot product gives the projection of a vector onto the other.

The following function gives the projection a * cos(angle) of a on b.

```
proj(Vector2 a, Vector2 b) {
    return dot(a, b) / magnitude(b);
}
```



Magnitude and dot product

$|a|^2 = a \cdot a$

Dot product applications

The sneaking enemy

We have a player in a 3D space and an enemy. We have the forward vector of our player.

How can we find whether the enemy is in front of or behind the player ?

Height of a point

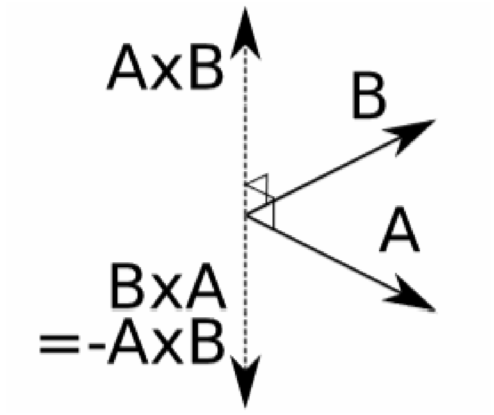
We have a plane in a 3D space, defined by a point Q and a normal vector n (perpendicular to the plane). We want to know the height of a point P which is abone the plane.

How can we compute this height ?

Cross product

Definition

The cross product of two vectors gives a third vector perpendicular to the plane that create the first ones.



There is also a numeric definition, for a 3d coordinate system created by x, y and z vectors :

A x B	=	AxBy z	–	AxBz y
	–	AyBx z	+	AyBz x
	+	AzBx y	–	AzBy x

In a right-hand coordinate system, a x b direction is given by the right-hand rule. And vice-versa for a left-hand coordinate system.

Magnitude of a cross product

I a x b I = lal lbl sin(alpha), where alpha is the angle between a and b.

The magnitude of a this cross product is the area of the parallelogram formed by a and b.

Get the normal of a vector or of a plane

When you want to determine the direction for a bouncing element, the cross product is an handy way to know the direction of the bounce. Just compute the cross product of the plane (or of the vector in a 2d space), and normalize it, to get the normal vector.

```
N = normalize (cross (A, B));
```

Link between cross product and dot product

The two operators are related by :

$$\|\mathbf{a}\times\mathbf{b}\|^2 = \|\mathbf{a}\|^2\|\mathbf{b}\|^2 - (\mathbf{a}\cdot\mathbf{b})^2.$$

Cross product applications

Finding the normal of a triangle, for shading algorithm

How would you find the normal of a triangle ?

Physics : find a torque

When you apply a forceto an object, it will give rise to a rotational motion, if applied off-center. This rotational force is called a "torque".

If F is the force and r a vector from the center of the mass to the point on which the force is applied, the torque $N = r \times F$.

Scalar triple product

Definition

$$\boxed{[\mathbf{a}, \mathbf{b}, \mathbf{c}] = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c} = (\mathbf{b} \times \mathbf{c}) \cdot \mathbf{a} = (\mathbf{c} \times \mathbf{a}) \cdot \mathbf{b}.}$$

Property : [c, b, a] = - [a, b, c]

Geometry

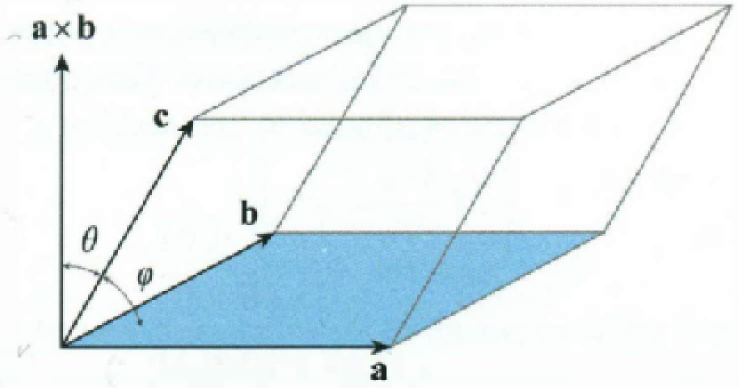


Figure 1.11. The scalar triple product $(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$ yields the volume of the parallelepiped spanned by the vectors **a**, **b**, and **c**. The area of the base is given by $\|\mathbf{a} \times \mathbf{b}\|$, and the height is given by $\|\mathbf{c}\| \sin \varphi$. Because φ and θ are complementary angles, the height is also given by $\|\mathbf{c}\| \cos \theta$, so the dot product $(\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$ is equal to the area of the base multiplied by the height.

Matrices

Matrices

Definition

A n x m matrix is a n * m table with numbers inside.. For instance, a 3 x 3 matrix :

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

A vector is a 1 * n matrix.

In a 3d spaces, matrices are useful because they can represent projection functions. We usually compute 3 and 4 dimensional matrices.

Identity matrix

All 0, except the top-left to bottom-right diagonal.

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

if $AB = I$ then A is the inverse of B and *vice versa*.

Matrix addition / substraction

Just add / substract numbers one by one.

Matrix multiplication

Matrix * Vector

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

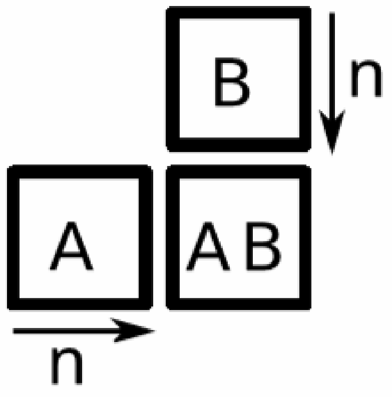
Matrix * Matrix

Each cell (row, col) in AB is:

$$\sum_{i=1}^{n} A(row, 1) * B(1, col) + \cdots + A(row, n) * B(n, col)$$

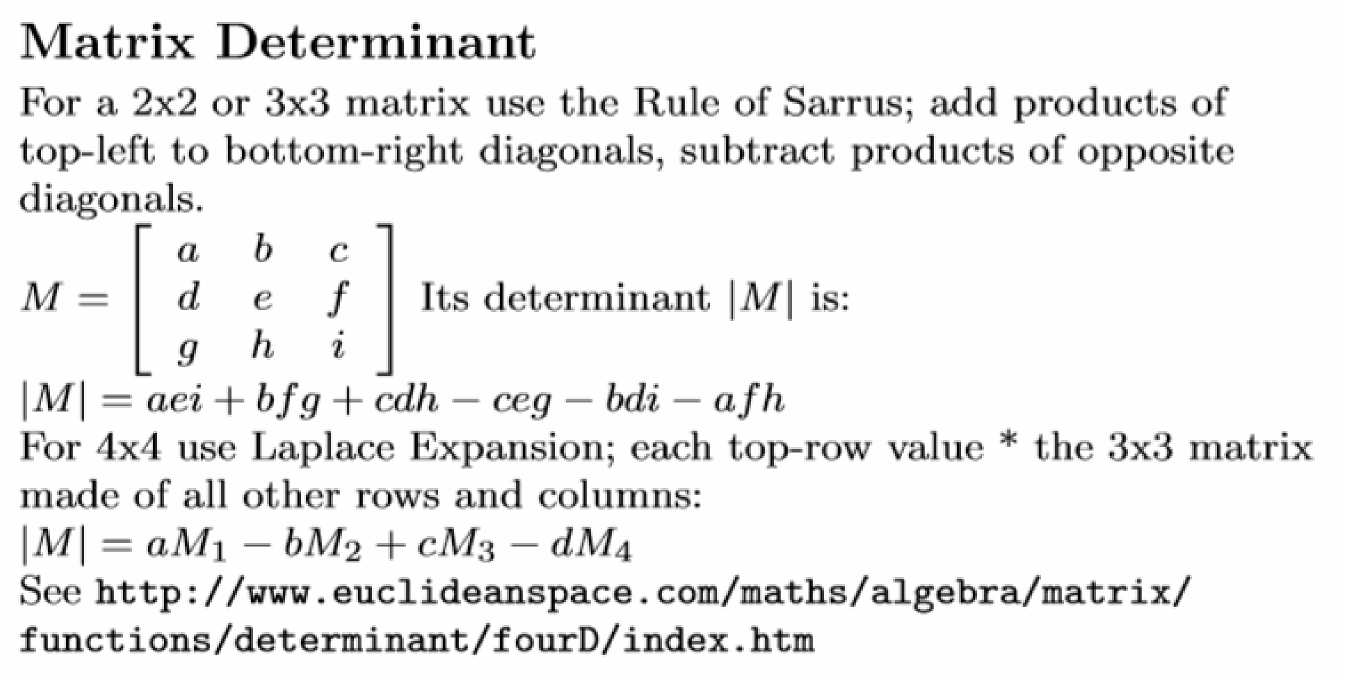
Where n is dimensionality of matrix.

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$



(rows of A with columns of B)

Determinant



Matrix Determinant

For a 2x2 or 3x3 matrix use the Rule of Sarrus; add products of top-left to bottom-right diagonals, subtract products of opposite diagonals.

$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ Its determinant $|M|$ is:

$|M| = aei + bfg + cdh - ceg - bdi - afh$

For 4x4 use Laplace Expansion; each top-row value * the 3x3 matrix made of all other rows and columns:

$|M| = aM_1 - bM_2 + cM_3 - dM_4$

See `http://www.euclideanspace.com/maths/algebra/matrix/functions/determinant/fourD/index.htm`

The matrix determinant is a sort of magnitude of the matrix. If we consider the n columns or the n rows of a matrix as vectors, then the determinant is equal to the hypervolum of the n-dimentional parallelotope formed by those vectors. It can be positive or negative depending on the orientation of the vectors.

Transpose and Inverse

Matrix Transpose

Flip matrix over its main diagonal. In special case of orthonormal xyz matrix then inverse is the transpose. Can use to **switch between row-major and column-major matrices.**

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \; M^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

Matrix Inverse

Use an inverse matrix to **reverse its transformation**, or to transform **relative to another object.**

$MM^{-1} = I$ Where I is the identity matrix.

If the determinant of a matrix is 0, then there is no inverse. The inverse can be found by multiplying the determinant with a large matrix of cofactors. For the long formula see `http://www.cg.info.hiroshima-cu.ac.jp/~miyazaki/knowledge/teche23.html`

Use the **transpose of an inverse model matrix** to transform normals: $n' = n(M^{-1})^T$

det(I) = 1
det(transpose(A)) = det(A)
det(inverse(A)) = 1 / det(A)
det(AB) = det(A).det(B)
det(tA) = t^n . det(A)

Transformations : translations and rotations

Changing coordinate systems

In 3d, we often define several coordinate systems. For instance, world space, object local spaces, camera spaces...

We need to transform the coordinates of an object into the coordinate of each one of those systems. To do that, we use transformation matrices.

Say we have a position p(A) in a A coordinate system, and we want its coordinates p(B) within the B coordinate system.

p(B) = M p(A) + t where M is a rotation matrix that rotates the axes from A to B, and t the translation that moves the A origin to the B origin

If M is invertible, we also have :

p(A) = inv(M) (p(B) - t)

Orthogonal transforms

Usual coordinate systems unit vectors in 3d spaces are orthogonal, so transformation matrices are orthogonal in this case.

Orthoganal matrix inverse is its transpose. When an objecti is reoriented of reflected in space with an orthogonal matrix, it keeps its shape. Orthogonal matrices preserves the dot product, the magnitude and the angles.

The determinant of a orthogonal pure rotation matrix is 1, or -1 if the rotation includes a reflection.

Rotation matrices

Rotation around an axis

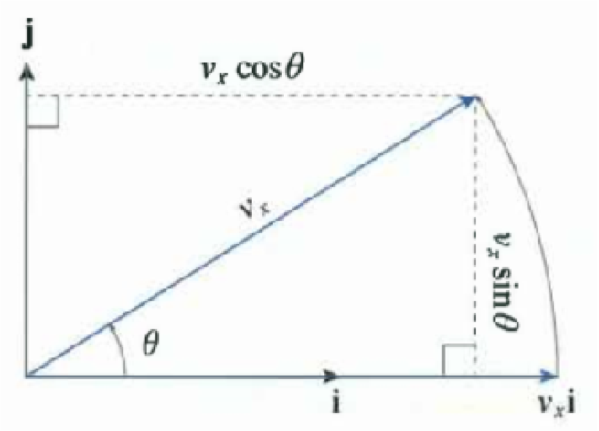
In a perpenticular coordinate system, with i, j and k as unit vectors, we can write a vector v as :

$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k}.$$

We rotate of an angle theta around k, so the z component will be unchanged. If we rotate around k, the x component will become :

$$v_x \cos\theta\,\mathbf{i} + v_x \sin\theta\,\mathbf{j}.$$

Because :



Same for the y component :

$$v_y \cos\theta\,\mathbf{j} - v_y \sin\theta\,\mathbf{i}.$$

So v becomes :

$$\mathbf{v}' = (v_x \cos\theta - v_y \sin\theta)\mathbf{i} + (v_x \sin\theta + v_y \cos\theta)\mathbf{j} + v_z\mathbf{k}.$$

Which can be expressed as with a matrice :

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

Rotation around coordinate system axes

$$\boxed{\mathbf{M}_{\text{rot}\,x}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix},}$$

$$\boxed{\mathbf{M}_{\text{rot}\,y}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix},}$$

$$\boxed{\mathbf{M}_{\text{rot}\,z}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.}$$

Rotation around an arbitrary axis a

$$\boxed{\mathbf{M}_{\text{rot}}(\theta, \mathbf{a}) = \begin{bmatrix} c+(1-c)a_x^2 & (1-c)a_x a_y - s a_z & (1-c)a_x a_z + s a_y \\ (1-c)a_x a_y + s a_z & c+(1-c)a_y^2 & (1-c)a_y a_z - s a_x \\ (1-c)a_x a_z - s a_y & (1-c)a_y a_z + s a_x & c+(1-c)a_z^2 \end{bmatrix}}$$

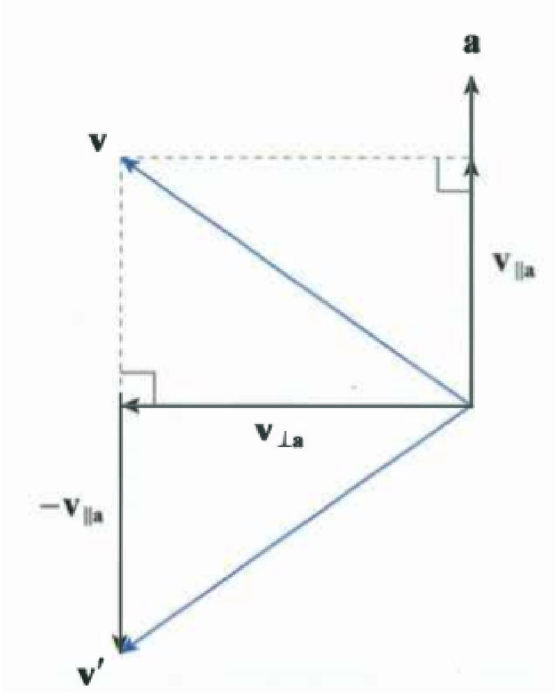
Reflections through a plane

If M is the matrix of a reflection through a plane perpendicular to a vector a :

$$\boxed{\mathbf{M}_{\text{reflect}}(\mathbf{a}) = \begin{bmatrix} 1-2a_x^2 & -2a_x a_y & -2a_x a_z \\ -2a_x a_y & 1-2a_y^2 & -2a_y a_z \\ -2a_x a_z & -2a_y a_z & 1-2a_z^2 \end{bmatrix}}$$

Its determinant is -1.

The vector v can be decomposed into a parallel to a component and a perpendicular to a component. The parallel is reflected, and the perpendicular is preserved.



$\mathbf{a}$
$\mathbf{v}$
$\mathbf{v}_{\parallel \mathbf{a}}$
$\mathbf{v}_{\perp \mathbf{a}}$
$-\mathbf{v}_{\parallel \mathbf{a}}$
$\mathbf{v}'$

Scales & skews

Scale

To scale an object, you can multiply by this matrix :

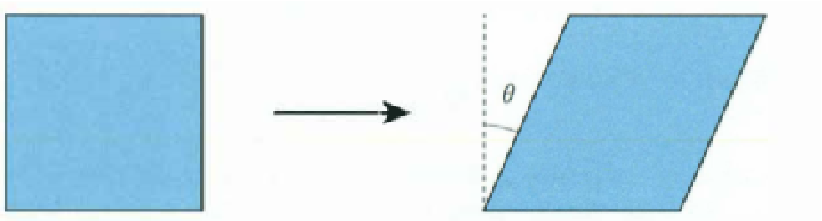
$$\mathbf{M}_{\text{scale}}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}.$$

If you want to scale by a s factor in a direction vector a :

$$\mathbf{M}_{\text{scale}}(s, \mathbf{a}) = \begin{bmatrix} (s-1)a_x^2+1 & (s-1)a_x a_y & (s-1)a_x a_z \\ (s-1)a_x a_y & (s-1)a_y^2+1 & (s-1)a_y a_z \\ (s-1)a_x a_z & (s-1)a_y a_z & (s-1)a_z^2+1 \end{bmatrix}$$

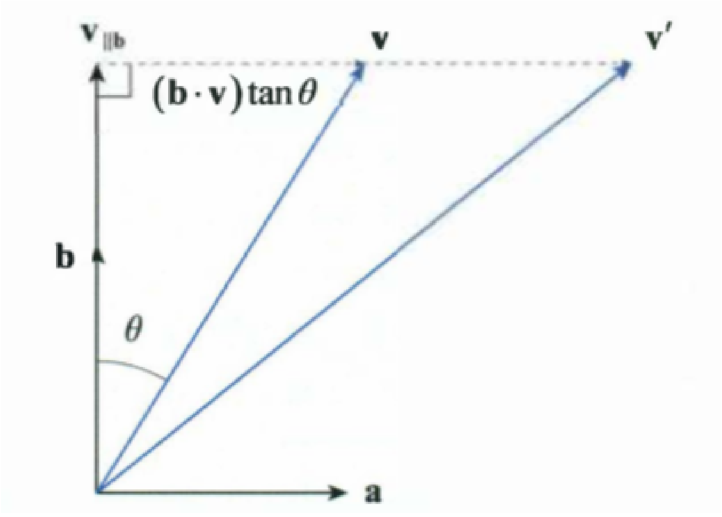
Skew

Skewing is to shearing an object along one direction a by an angle theta made with a perpendicular direction b.



The box is skewed by an angle theta to the right.

v' = v + a(b.v) tan(theta)


$\mathbf{v}_{\|\mathbf{b}}$
$\mathbf{v}$
$\mathbf{v}'$
$(\mathbf{b}\cdot\mathbf{v})\tan\theta$
$\mathbf{b}$
θ
$\mathbf{a}$

We can use this matrix to skew :

$$\mathbf{M}_{\text{skew}}(\theta, \mathbf{a}, \mathbf{b}) = \begin{bmatrix} a_x b_x \tan\theta + 1 & a_x b_y \tan\theta & a_x b_z \tan\theta \\ a_y b_x \tan\theta & a_y b_y \tan\theta + 1 & a_y b_z \tan\theta \\ a_z b_x \tan\theta & a_z b_y \tan\theta & a_z b_z \tan\theta + 1 \end{bmatrix}.$$

Homogeneous Matrix

Principle

To add information about location, we use a 4D projective space called homogeneous coordinates.

A homogeneous vector has a forth component w, that is equal to 1 when the vector represent a position (a point) and equal to 0 when it represents a direction. The vector (x, y, z, 0) is considered the point at the infinity in the direction (x, y, z) when projected into 3D space.

Homogeneous matrix

We can define the homogeneous matrix, that carries information about rotation, scale AND translation by :

$$\mathbf{H} = \begin{bmatrix} \mathbf{M} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \left[\begin{array}{ccc:c} M_{00} & M_{01} & M_{02} & t_x \\ M_{10} & M_{11} & M_{12} & t_y \\ M_{20} & M_{21} & M_{22} & t_z \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right].$$

M carries rotation and scale information, t translation info. if the bottom-right w component is equal to 0, the translation does not occur.

The homogeneous transform matric has an inverse :

$$\boxed{\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{M}^{-1} & -\mathbf{M}^{-1}\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix},}$$

OpenGL and DirectX

OpenGL and DirectX use a different order to store thode 4 x 4 matrices :

Column-Order Homogeneous Matrix

Commonly used in **OpenGL maths libraries**

$$v' = \begin{bmatrix} X_x & Y_x & Z_x & T_x \\ X_y & Y_y & Z_y & T_y \\ X_z & Y_z & Z_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \\ 1 \end{bmatrix}$$

Row-Order Homogeneous Matrix

Commonly used in **Direct3D maths libraries**

$$v' = \begin{bmatrix} V_x & V_y & V_z & 1 \end{bmatrix} \begin{bmatrix} X_x & X_y & X_z & 0 \\ Y_x & Y_y & Y_z & 0 \\ Z_x & Z_y & Z_z & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Transforms in homogeneous matrices

Scale

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{(column-order)}$$

$$R_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{(column-order)}$$

$$R_z = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{(column-order)}$$

Translate

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transform order

We first scale, then rotate, then translate the object. Thus the scaling has no influence on the rotation, and the rotation has no influence on the translation. The matrix multiplication order is inversed :

Transform matrix = T x R x S

View Projections

See Shader introduction.

Quaternions

Quaternions

Principle

Quaternions were invented by Sir Hamilton as a 4D generalization of complex numbers.

A q number from the set H (for Hamilton) is defined by :

$q = xi + yj + zk + w$ with x, y and z imaginary parts w the real part
and $i^2 = j^2 = k^2 = ijk = -1$
(or $ij = -ji = k$, $jk = -kj = i$, $ki = -ik = j$)

Quaternions can be understood as the sum of a 3D vector and a scalar.

$q = v + s$

Quaternion product

$$\begin{aligned}\mathbf{q}_1\mathbf{q}_2 = &(x_1w_2 + y_1z_2 - z_1y_2 + w_1x_2)\,i \\ &+ (y_1w_2 + z_1x_2 + w_1y_2 - x_1z_2)\,j \\ &+ (z_1w_2 + w_1z_2 + x_1y_2 - y_1x_2)\,k \\ &+ (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2).\end{aligned}$$

$$\mathbf{q}_1\mathbf{q}_2 = \mathbf{v}_1 \times \mathbf{v}_2 + s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + s_1s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2.$$

Quaternions

Conjugate

The conjugate q* of a quaternion q is :

q* = -v + s

Property : qq* = q*q = v² + s²

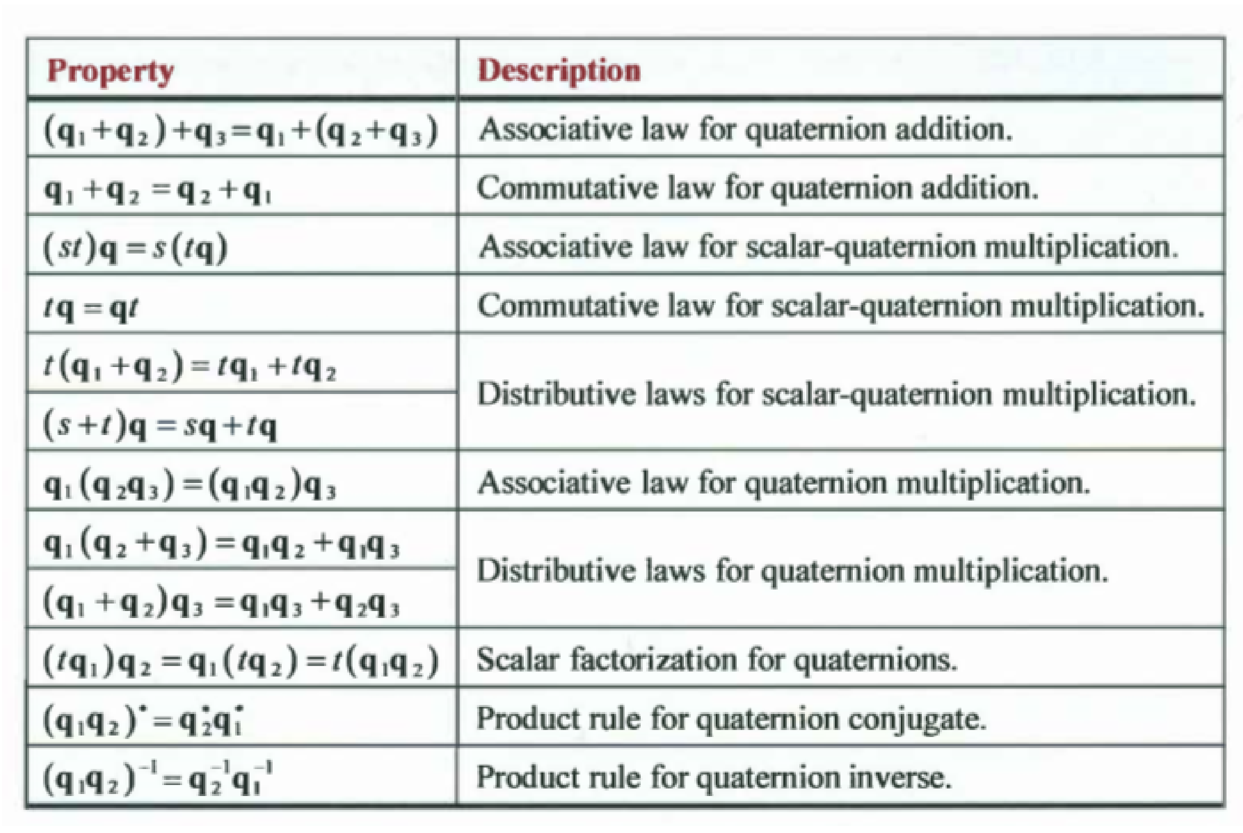
Magnitude of a quaternion :

||q|| = sqrt(qq*) = sqrt(v² + s²)

||q|| ||r|| = || q r ||

inverse(q) = q* / qq* = (-v+s) / (v² + s²)

Quaternion's other properties



Property	Description
$(\mathbf{q}_1+\mathbf{q}_2)+\mathbf{q}_3=\mathbf{q}_1+(\mathbf{q}_2+\mathbf{q}_3)$	Associative law for quaternion addition.
$\mathbf{q}_1+\mathbf{q}_2=\mathbf{q}_2+\mathbf{q}_1$	Commutative law for quaternion addition.
$(st)\mathbf{q}=s(t\mathbf{q})$	Associative law for scalar-quaternion multiplication.
$t\mathbf{q}=\mathbf{q}t$	Commutative law for scalar-quaternion multiplication.
$t(\mathbf{q}_1+\mathbf{q}_2)=t\mathbf{q}_1+t\mathbf{q}_2$	Distributive laws for scalar-quaternion multiplication.
$(s+t)\mathbf{q}=s\mathbf{q}+t\mathbf{q}$	
$\mathbf{q}_1(\mathbf{q}_2\mathbf{q}_3)=(\mathbf{q}_1\mathbf{q}_2)\mathbf{q}_3$	Associative law for quaternion multiplication.
$\mathbf{q}_1(\mathbf{q}_2+\mathbf{q}_3)=\mathbf{q}_1\mathbf{q}_2+\mathbf{q}_1\mathbf{q}_3$	Distributive laws for quaternion multiplication.
$(\mathbf{q}_1+\mathbf{q}_2)\mathbf{q}_3=\mathbf{q}_1\mathbf{q}_3+\mathbf{q}_2\mathbf{q}_3$	
$(t\mathbf{q}_1)\mathbf{q}_2=\mathbf{q}_1(t\mathbf{q}_2)=t(\mathbf{q}_1\mathbf{q}_2)$	Scalar factorization for quaternions.
$(\mathbf{q}_1\mathbf{q}_2)^*=\mathbf{q}_2^*\mathbf{q}_1^*$	Product rule for quaternion conjugate.
$(\mathbf{q}_1\mathbf{q}_2)^{-1}=\mathbf{q}_2^{-1}\mathbf{q}_1^{-1}$	Product rule for quaternion inverse.

Rotations with quaternions

How to rotate

Quaternions are used for rotations, and have several advantage above matrices :

- No gimbal lock
- Less storage (4 instead of 9)
- Rotation concatenation is cheaper (whereas point rotations are cheaper with matrices)

A rotation with quaternions can be written :

v' = q v inv(q)

If q is a unit quaternion, since qq* = 1, then :

v' = qvq*

Only unit quaternions are used to represent rotations in practice.

Rotation around an arbitrary axis a

$$\boxed{\mathbf{q} = \left(\sin\frac{\theta}{2}\right)\mathbf{a} + \cos\frac{\theta}{2}}$$

Rotation concatenation

If we want to rotate by a quaternion q then by a quaternion r :

v' = (rq) v (rq)*

16 multiplications and 12 additions, whereas matrix multiplication means 27 multiplications and 18 additions

Quaternion to matrix

When we want scaling and translations, we reconvert quaternions to 3x3 matrices :

$$\boxed{\mathbf{M}_{\text{rot}}(\mathbf{q}) = \begin{bmatrix} 1-2y^2-2z^2 & 2(xy-wz) & 2(xz+wy) \\ 2(xy+wz) & 1-2x^2-2z^2 & 2(yz-wx) \\ 2(xz-wy) & 2(yz+wx) & 1-2x^2-2y^2 \end{bmatrix}.}$$

Matrix to quaternion

http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/

Intersections

2D intersections

Lines

Rectangles

Circles

Ray-Plane intersection

Ray-Sphere intersection

Triangles