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

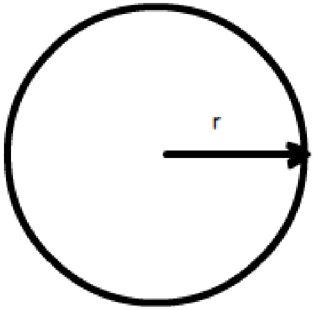
```
clamp(low, n, high) {
  return min(max(low, n), high);
}
```

Round

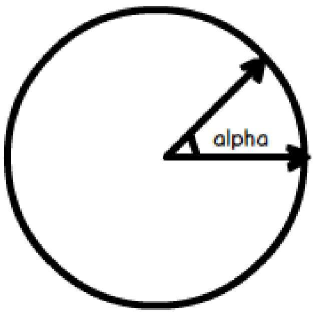
```
math.round(n, deci = 0) {
  deci = 10^deci;
  return floor(n*deci + 0.5)/deci;
}
```

Circles

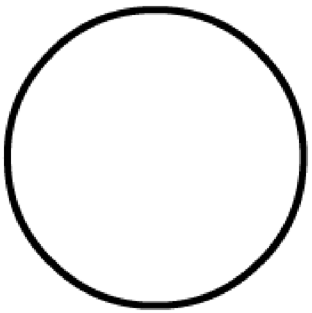
Radius



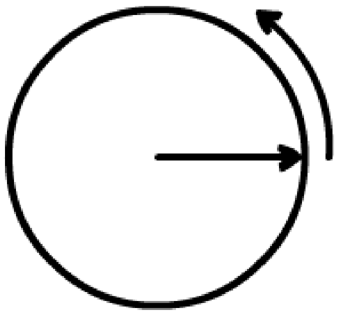
Degrees, radians and perimeter

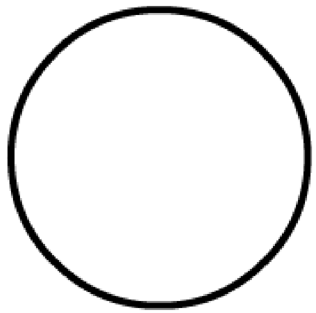


360 degrees



2Pi Radian





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

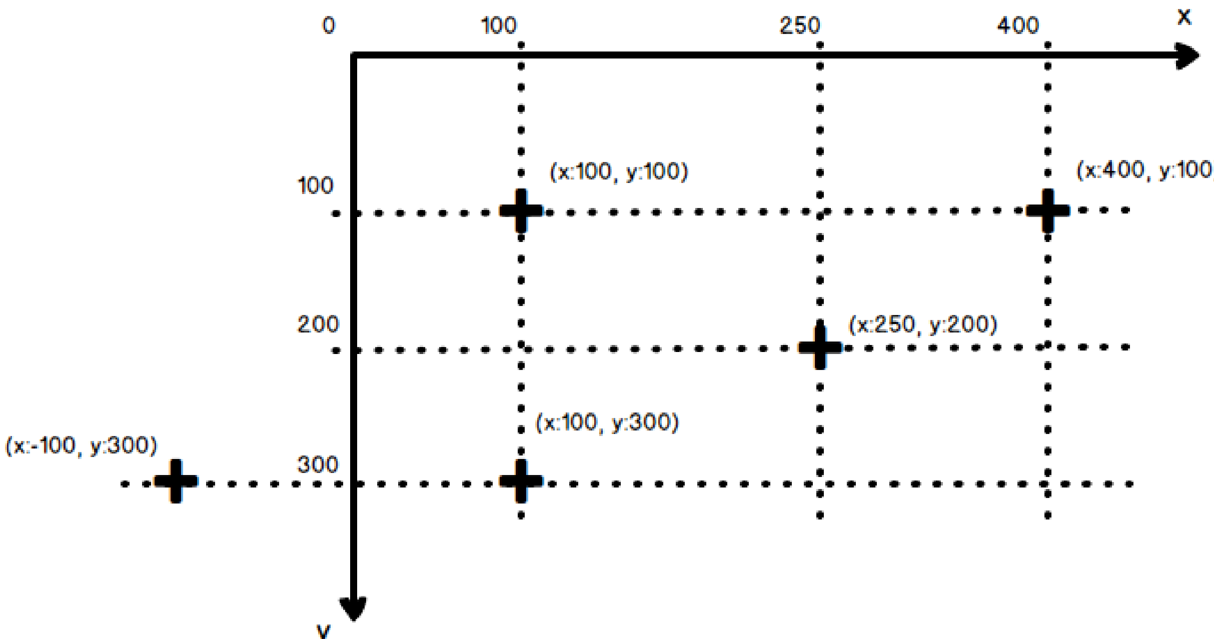
Perimeter = 2Pi * radius

1 Radian = (180 / Pi) degrees

Area

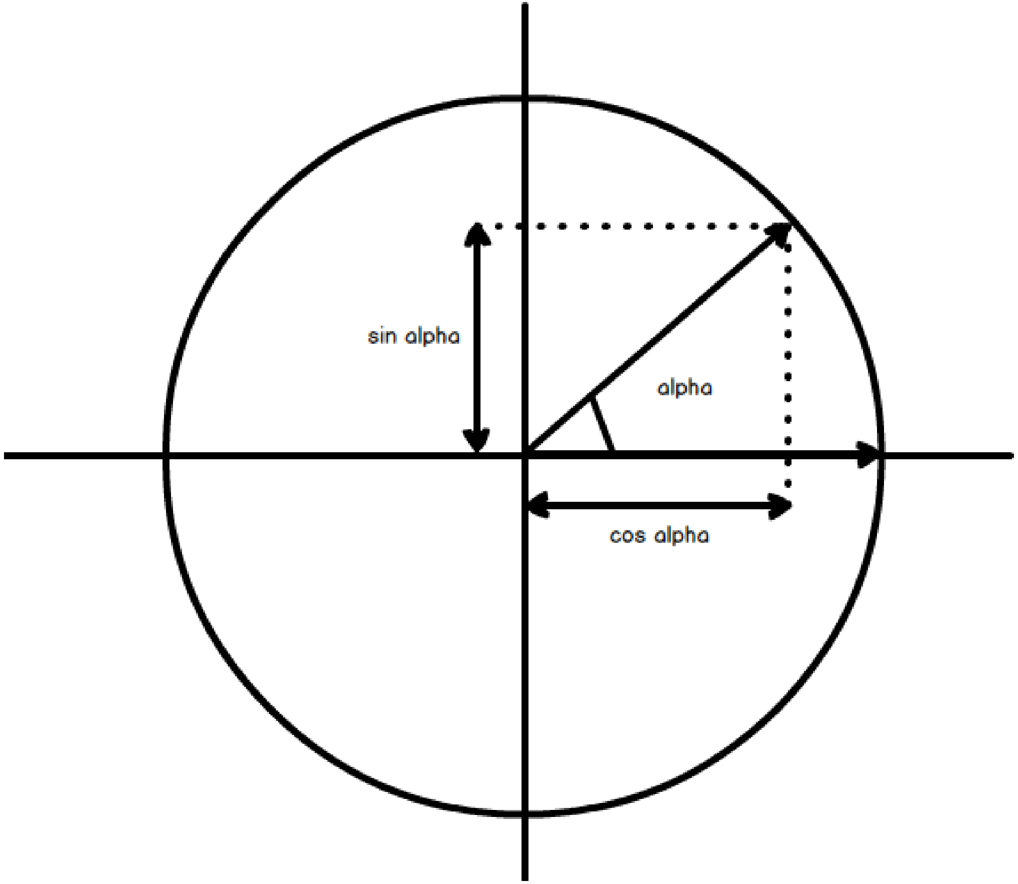
Area = 2 Pi * radius * radius

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

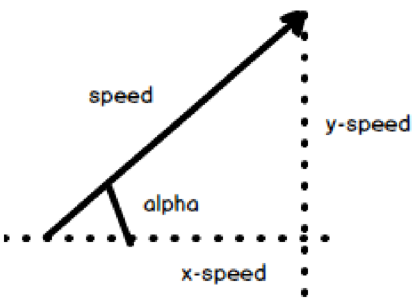
https://www.raywenderlich.com/2736-trigonometry-for-game-programming-part-1-2
https://www.raywenderlich.com/2737-trigonometry-for-game-programming-part-2-2

Move a 2d entity at constant speed:

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

arctan2

arctan2



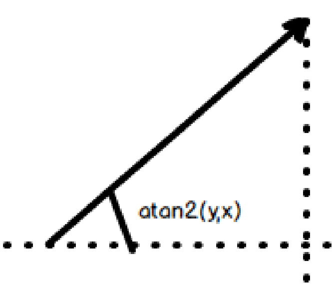
If you want to get an angle alpha, you can use :

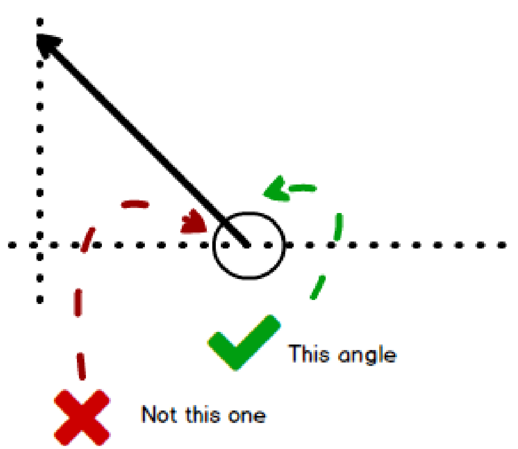
atan(opposite/adjacent) = alpha

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 :



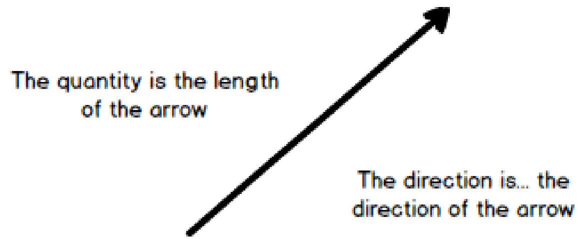


Vectors

Vectors, dimensions and coordinates

Definition

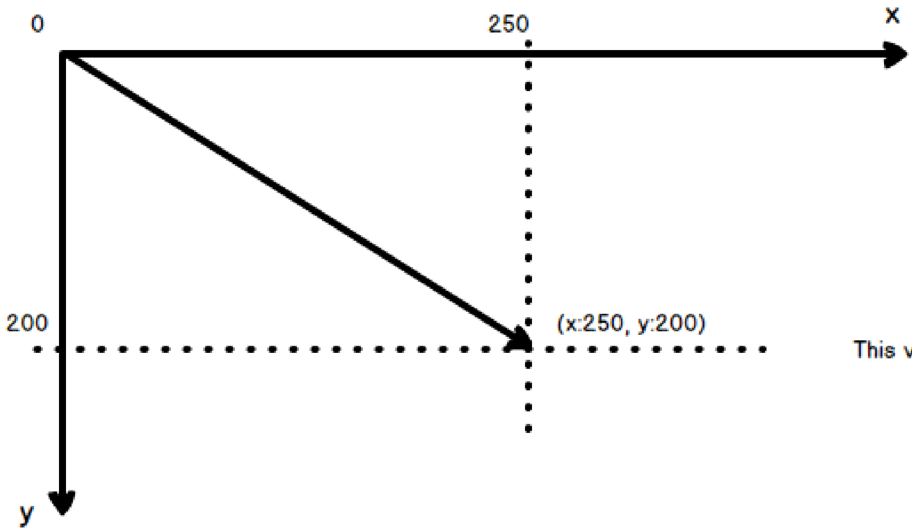
A vector is a **quantity** with a **direction**



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.



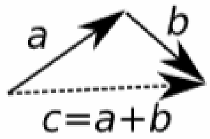
This vector will be written as :

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

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

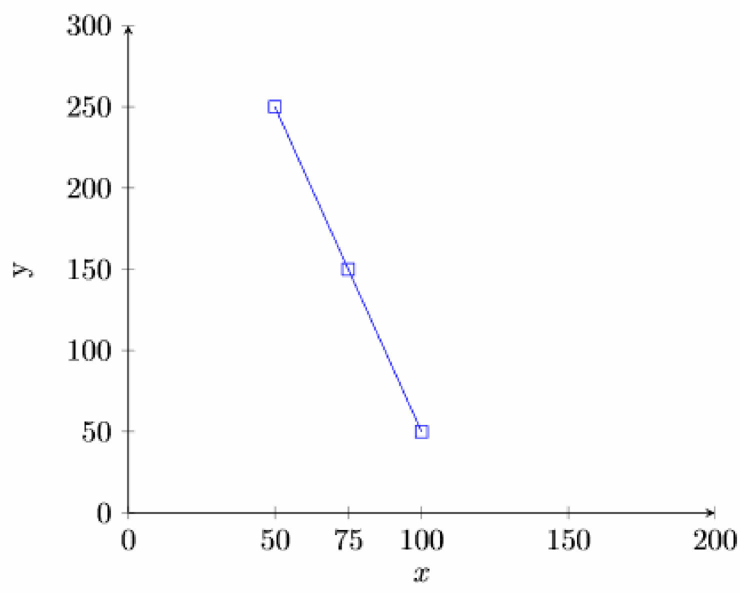
Vectors can be multiplied by scalars (real numbers). You just have to multiply the coordinates.

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

Medium point

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

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



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 :

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

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

Colinearity

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

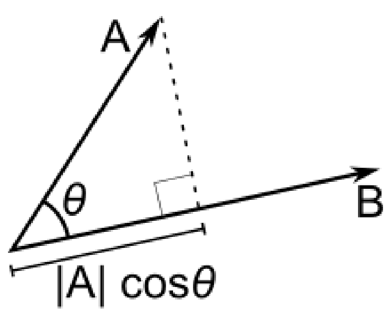
If A . B = 0, then A and B are perpendicular.
if A . B > 0, A and B "go into the same direction".
if A . B < 0, A and B "go into opposite directions".
if A . B = 1 or -1, A and B are parallel, respectively with same and opposite directions.

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.

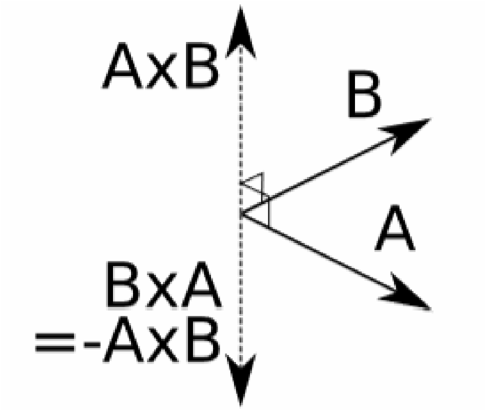
```
proj(Vector2 a, Vector2 b) {
    return dot(a, b) / magnitude(b);
}
```



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

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 :

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

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

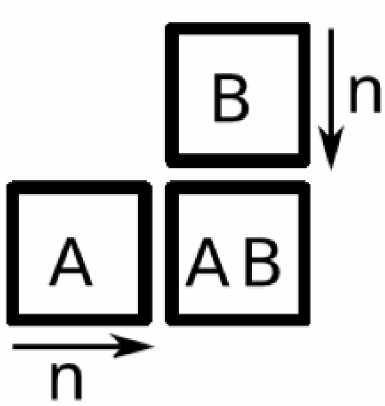
Matrix * Vector

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

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)

Matrices special values

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`

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: $n' = n(M^{-1})^T$

Matrices and transformations

Principle

We can use 4 x 4 matrices to store in one matrix the position, rotation and scale of a 3D entity. This matrix gives the transform of the entity.

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

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

Quaternions

Intersections

2D intersections

Lines

Rectangles

Circles

Ray-Plane intersection

Ray-Sphere intersection

Triangles