Coordinate Mathematics for CAD

2024-10-30

This reference and textbook on vectors and coordinates focuses on mathematics and programming solutions. As an easy-to-read work, it forms a practical collection on CAD faces, vectors, angles, normals, projections, rotations and more. Code fragments are written in CAD Lisp and therefore usable in programmes such as BricsCAD and AutoCAD.

About and credits ... is what you may want to read but for a quick start, please use the Table of Contents for navigation.

Table of Contents

Trekwei 7, 8711GR Workum, The Netherlands.

Definitions

Face

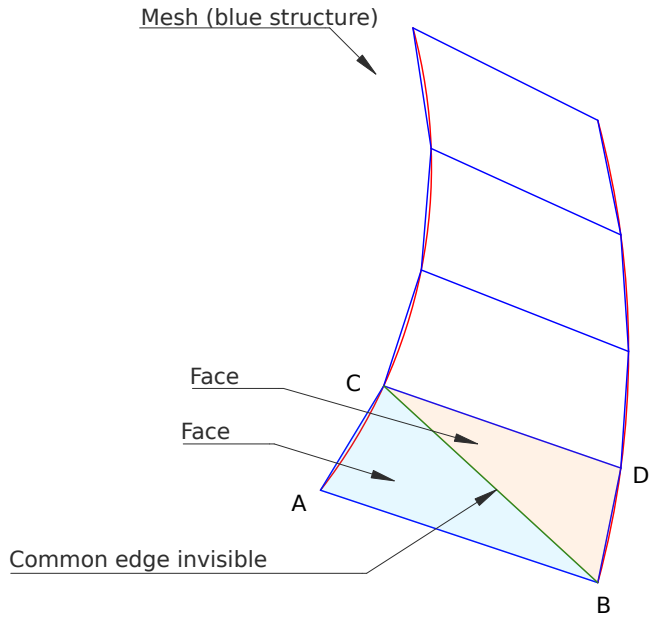
A face is, by mathematical definition, a **flat** polygon and almost always a **triangle** in a CAD environment. A (CAD) face has three **edges** and **vertices**. Complex 3D surfaces can be simulated by using flat triangular faces. Flat faces with four vertices could be suitable for orthogonal based surfaces (like a cube) but are almost never suitable for complex 3D surfaces.

3DFace

This is both the entity name and command name in CAD systems like BricsCAD and AutoCAD. A 3DFace entity can be visible as a tetragon or a triangle. The use of tetragons has the advantage of clarity. On the other hand, in 3D, the four corners of each tetragon are (almost) never in one plane. To fix that, for a better visual, each tetragon consists of two adjacent triangles i.e. **faces**. The common edge of these two faces forms an invisible diagonal of a visible tetragon - the visible sides of the two triangles give the impression of a tetragon but, again, upon examination, the corners (almost) never lie in a plane.

Mesh

A mesh is a collection of triangular and or tetragonal **3Dface** entities. A mesh forms a **surface** object.



Vector

A vector is a **direction** with a size or **magnitude**.
A vector in space consists of three coordinates. Nothing more and nothing less. Those three coordinates only say something about the direction and length of the vector. Let go of the idea that the vector is attached to something. A normal vector is a good example of this. Suppose we have three points with an imaginary plane through them. A normal vector is perpendicular to that plane. If you know just one point through which that plane passes and you know the normal vector, then you have that plane defined in space. Where the normal vector is situated is totally irrelevant. So again, a vector is only a direction with a magnitude.

- A vector V is represented as $\vec{V}$
  - with a **direction** x, y, z
  - and a **length** or **magnitude** $\|V\|$
- The edges of faces can be considered as vectors like $\vec{AC}$ and $\vec{AB}$.
- Knowing the coordinates of the vertices, we can do calculations on these vectors, on these faces. That is where our journey in mathematics start!

Basic mathematical solutions

Basic operators

Let's start simple. With coordinates of vectors, we can do basic things like addition and subtraction. As functions are also defined, here follows an enumeration based on examples.

For convenience, integers (1,2,3,...) are used with occasional real numbers (1.0,2.0,1.88,...). Be aware of the differences, for example: $5/2 = 2$ and $5/2.0 = 2.5$.

Add

Add, symbol $+$

$(1, 2, 3) + (6, 5, 4) = (7, 7, 7)$

Subtract

Subtract, symbol - or $-$

$$(1,2,3)-(6,5,4)=(-5,-3,-1)$$

Divide

Divide, symbol / or —

$$\frac{(10,5,5.0)}{2}=(5,2,2.5)$$

Please note: $5/2=2$ (integers) and $5.0/2=2.5$ (reals)

Multiply

Multiply, symbol * or $\times$ or $\cdot$

$$(1,2,3)\times 2.0=(2.0,4.0,6.0)$$

Minus

Minus, change sign or multiply with -1

$$-(1,-2,2.0)=(-1,2,-2.0)$$

Sum

Sum, Σ(list of coordinates)

$$\Sigma((1,2,3)(4,5,6)(7,8,9))=(12,15,18)$$

Length

Length, cardinality, |(list of coordinates)|

$$|((1,2,3)(4,5,6)(7,8,9))|=3$$

Average

Average , $\overline{(list\,of\,coordinates)}$

$$\overline{((1,2,3)(4,5,6)(7,8,9))}=\frac{\Sigma((1,2,3)(4,5,6)(7,8,9))}{|((1,2,3)(4,5,6)(7,8,9))|}=\frac{(12,15,18)}{3}=(4,5,6)$$

Essential operations

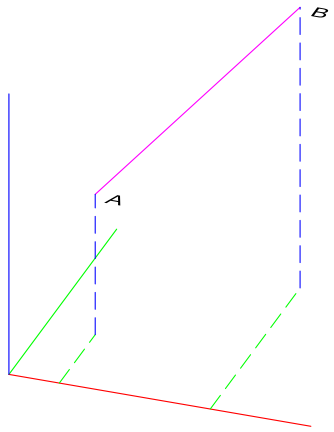
Magnitude of a vector

The **magnitude** is the **size**, or more precise, the exact **length** of a vector. For vector $\vec{v}$, magnitude is noted as $\|v\|$.

For vector $\vec{A}$ with value x, y, z, the **magnitude** or **length** $\|A\|$ is $\sqrt{x^2 + y^2 + z^2}$ according to the Pythagorean theorem.

For vector $\vec{AB}$ with coordinates $A = Ax, Ay, Az$ and $B = Bx, By, Bz$, the magnitude $\|AB\|$ is:

$\sqrt{(Bx - Ax)^2 + (By - Ay)^2 + (Bz - Az)^2}$.



Notation $\|v\|$ is equivalent to **`|v|`**, but **`|number|`** is also used for absolute values.

Dot product of two vectors

The **dot product** is an operation on two vectors $\vec{a}$ and $\vec{b}$ with angle θ in between, resulting in **a scalar** (number, not a vector).

The dot product by itself has little meaning but is important as a value for other formulas. For example, the law of cosines has the dot product as its basis.

Dot product is written as: $\vec{a} \cdot \vec{b}$

There are two common ways to calculate the dot product:

- $\vec{a} \cdot \vec{b} = \|a\|\|b\| \cos(\theta)$
- $\vec{a} \cdot \vec{b} = xa * xb + ya * yb + za * zb$

This also means that:

- $\theta = \arccos\left(\frac{xa * xb + ya * yb + za * zb}{\|a\|\|b\|}\right)$
- See "Calculate the angle between two vectors" for more and caveats.

More...

Cross product of two vectors

The **cross product** is an algebraic operation on two vectors $\vec{a}$ and $\vec{b}$ resulting in **a normal vector $\vec{c}$ perpendicular to both** $\vec{a}$ and $\vec{b}$.

Cross product is written as: $\vec{a} \times \vec{b} = \vec{c}$

Calculation:

- If vectors $\vec{a}$ and $\vec{b}$ consist of coordinates (xa, ya, za) and (xb, yb, zb), then the cross product is:
- $\vec{a} \times \vec{b} = \vec{c} =$
  $(ya * zb - za * yb, za * xb - xa * zb, xa * yb - ya * xb)$

Valuable for solving equations: The magnitude of the resulting vector, $\|\vec{c}\|$, equals the area of the parallelogram of $\vec{a}$ and $\vec{b}$.

Normal: A line, ray or vector perpendicular to something. In this case we mean a vector perpendicular to a plane. The side of the plane on which the normal starts is determined by the right hand rule. Therefore the order of a and b is important, $a \times b \neq b \times a$ and $a \times b = -(b \times a)$. More...

When not using the right hand rule, the sign $(+, -)$ of the coordinates of $\vec{c}$ should be inverted.

More...

Derived operations

Unit vector of a vector

A unit vector of a vector has the same direction but has a length or magnitude of 1. It can be retrieved by dividing each x, y and z of a vector by the magnitude of that vector, resulting in a new vector.

Angle between two vectors

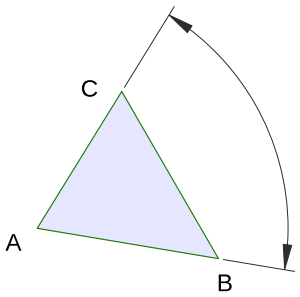
In 2D, it is easy to understand. 3D is more difficult.

For example, take a pencil in each hand and place and point them randomly in front of you. The centre lines do not cross each other by a long shot. You see a projected angle but if you move your head that angle also changes. How is it possible to determine an angle in that situation?

Remember that a vector is nothing more than a direction and a magnitude. So where the vector starts is totally irrelevant. In other words, you can connect the starting points of the vector.

At that point, you can imagine a plane through the three points and it becomes clear that there is an angle in that plane.

That leaves the simple question, do you want the angle from pencil A to pencil B or vice versa? Finally, you can look at the plane from two sides, which also changes the answers. So there is complexity but that can be solved with the right-hand rule and vector order.



Using arccosine

The angle α between $\vec{AB}$ and $\vec{AC}$...

$$\alpha = \arccos\left(\frac{\vec{AB} \cdot \vec{AC}}{\|AB\|\|AC\|}\right)$$

Example in sheet notation:

```
AB = 1,2,3
AC = 4,5,6
|AB| =SQRT(1^2+2^2+3^2) = 3.74
|AC| =SQRT(4^2+5^2+6^2) = 8.77
AB.AC =1*4+2*5+3*6 = 32
acos((AB.AC)/(|AB|*|AC|)) = acos(32/(3.74*8.77)) = 0.226 rad = 12.9
degrees
```

Here, the angle range is 0 to π. Although technically correct, that may not always be what you want - in cases with small angles there is a lot of cumulative error. What else is there?

Using arctangent

The same angle again but by using function **atan2**.

Atan2 is arctan, arctangent, inverse tangent, of value x and y. But be careful, **syntaxis, x and y order**, can be different. For example, spreadsheet: **atan2(x,y)**, C language: **atan2(y,x)** and CAD Lisp language: **(atan y x)**. Also keep in mind that **angles range** from 0 to π radians.

How do we get x and y?

- y is the magnitude of the cross product of vector AB and AC
  $y = \|\vec{AB} \times \vec{AC}\|$
- x is the dot product of vector AB and AC.
  $x = \vec{AB} \cdot \vec{AC}$

Again, the angle α between $\vec{AB}$ and $\vec{AC}$...

$$\alpha = atan2(y, x) = atan2(\|\vec{AB} \times \vec{AC}\|, \vec{AB} \cdot \vec{AC})$$

In addition written out in sheet notation for vector AB and AC and angle $alpha$ in radians:

```
AB = (x1,y1,z1)
AC = (x2,y2,z2)
x =x1*x2+y1*y2+z1*z2
y =sqrt((y1*z2-z1*y2)^2+(z1*x2-x1*z2)^2+(x1*y2-y1*x2)^2)
alpha =atan2(x,y)
```

Vector angle: obtuse, perpendicular or acute?

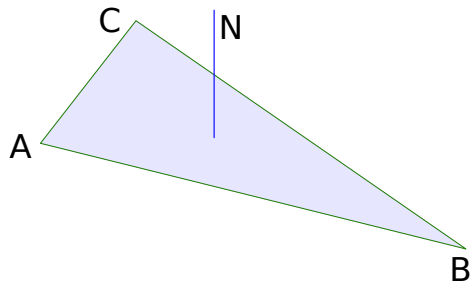
Continuing with the previous... Instead of, or in addition to, calculating an angle, the dot product is enough to say whether the angle is obtuse, perpendicular or acute. It is depending on the sign of $m = \vec{AB} \cdot \vec{AC}$. More specific:

- If m is positive, the angle is acute.
- If m is zero, the angle is right, perpendicular.

- If m is negative, the angle is obtuse.

A rectangle contains two adjacent faces. We can say something about the angle between the faces by first calculating the normal vectors of each face.

Normal of a face



The normal vector is the cross product of vector AB times vector AC. According to the right hand rule, it is written as $\vec{AB} \times \vec{AC}$.

Example:

```
AB = (1,2,3)
AC = (4,5,6)
AB×AC = (2*6−3*5,1*6−3*4,1*5−2*4) = (−3,6,−3)
```

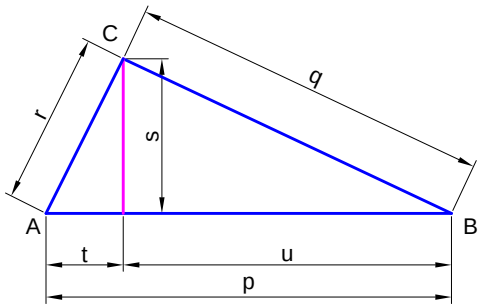
The opposite normal vector is $AC \times AB$, being $AB \times AC$ with opposite signs, i.e. $(3, -6, 3)$.

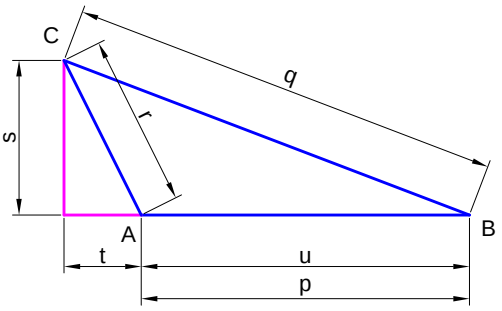
The magnitude of normal $AB \times AC$ is area $\|AB\|\|AC\| \cos(\angle CAB)$. So you can tell something about angle CAB if you have the magnitudes of all three vectors.

2D projection of a face - Pythagorean

With three coordinates A, B and C known of a 3D face, we can calculate a projection in 2D. Alternative, law of cosines can be used here too.

Calculating C based on A is $(0,0)$ and B is $(p,0)$





Summary

Known:

Distances $p,\ q,\ r$

Constraints:

Point $A = (0,0)$

Point $B = (p,0)$

Point C is above axis AB

First Cy is positive

z coordinates always 0

Requested:

Coordinates of C.

Answer:

$$C = \frac{((p^2 + r^2 - q^2)}{(2 * p)}, \sqrt{(r^2 - t^2))}$$

Subsequent Cy values can be negative too, i.e. $-\sqrt{(r^2 - t^2)}$

Rationale

A: (x,y,z)
C: (x',y',z')

Variant 1 $\vee$ Variant 2 ($\vee$ is OR)

Lets start with Cy, basically **s**:

```
r^2=t^2+s^2
s^2=r^2-t^2
```

Next focus on **q**:

```
∠ CAB <= pi()/2 ∨ ∠ CAB > pi()/2 ⇒
x <= x' ∨ x > x'

q^2 =
(p-t)^2+s^2 ∨ (p+t)^2+s^2  ⇒
(p-t)^2+r^2-t^2 ∨ (p+t)^2+r^2-t^2  ⇒
p^2-2pt+t^2+r^2-t^2 ∨ p^2+2pt+t^2+r^2-t^2  ⇒
p^2-2pt+r^2 ∨ p^2+2pt+r^2
```

So we can tell something about **t** or Cx:

```
2pt = p^2+r^2-q^2 ∨ -p^2-r^2+q^2  ⇒
t = (p^2+r^2-q^2)/(2*p) ∨ (p^2+r^2-q^2)/(-2*p)
```

And we can substitute **t**, Cy, to get **s**, Cy:

```
s^2=r^2-t^2
s=sqrt(r^2-t^2) ∨ s=-sqrt(r^2-t^2)
```

With two answers for **t** it is tempting to just code both values. Annoying and itching is the fact I cannot say anything beforehand about CAB.

Hmm, is that so? Climbing from the bottom - the resulting formula's - up, there is fragment **p^2+r^2-q^2**. Wait, Pythagoras! That should be 0 when $\angle CAB$ is $90°$!

More specific:

- $p^2 + r^2 - q^2 < 0 \Rightarrow \angle CAB > 90°$
- $p^2 + r^2 - q^2 = 0 \Rightarrow \angle CAB = 90°$
- $p^2 + r^2 - q^2 > 0 \Rightarrow \angle CAB < 90°$

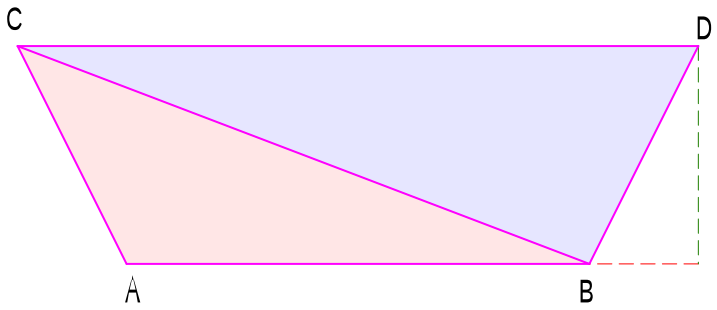
One problem solved, one problem created! Let's see, If **p^2+r^2-q^2 < 0** then **t = (p^2+r^2-q^2)/(-2*p)** and else **t = (p^2+r^2-q^2)/(2*p)**. So **t** is always a positive value and whether it is on the left or right side of **x=0** (point A) depends on the value of **p^2+r^2-q^2**.

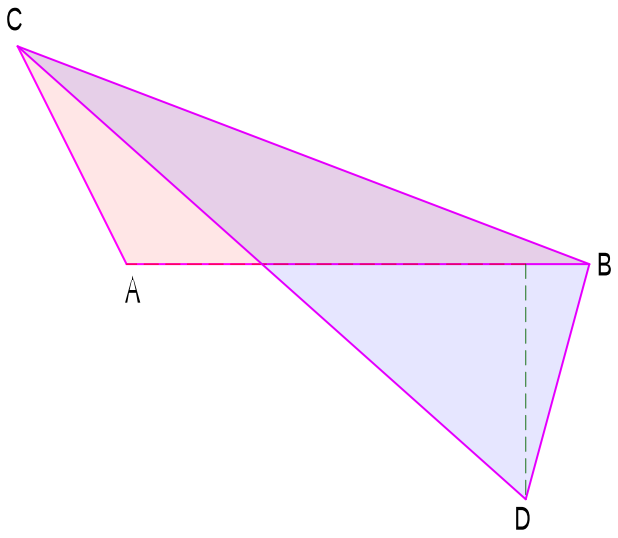
More specific, as an x-coordinate, **t = (p^2+r^2-q^2)/(2*p)**. And that is always true! Great, and even my old brains survived this.

A final note about **s**. It is a square root so the argument can be both positive and negative. It won't bother us for the assumed first C, but it will bother us for subsequent C's and all D's.

Calculate 2D projection of an adjacent face

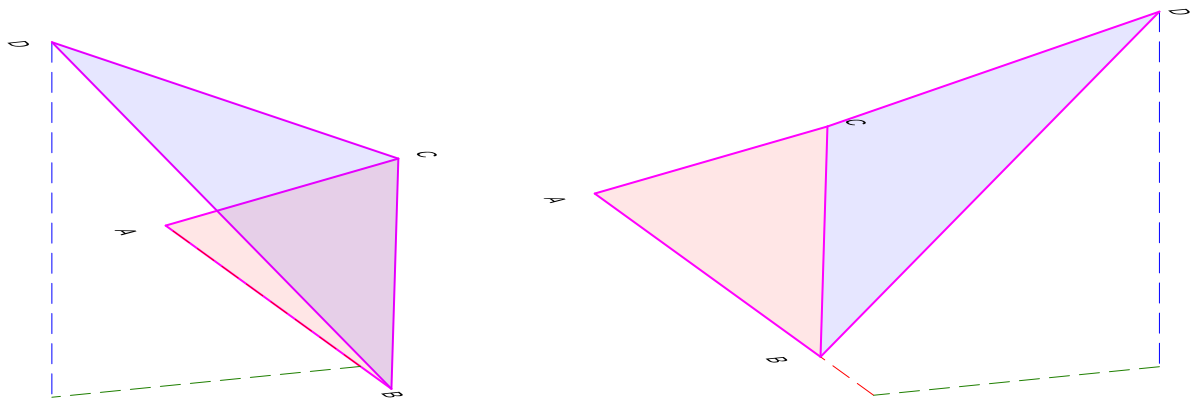
Getting the coordinate D is similar to finding C.





One segment of the **RuleSurf** command result is build out of two triangles or faces. Scanning a mesh in CAD Lisp means in most cases: Start with A and B and calculate C and D, then treat those resulting C and D as a new set of A and B and start all-over, until the complete mesh is processed. In most cases, the first picture is in play. However, sometimes the second picture can be in play. In addition, even C can be under axis AB but focus is on D for proper explanation.

A,B,C and D are 3D points. There is a plane ABC. D is somewhere relative to this plane. We need to know where D lies. More specific, in a planar view to ABC, is D under or above axis BC? To get a better impression, look at it from a different angle:

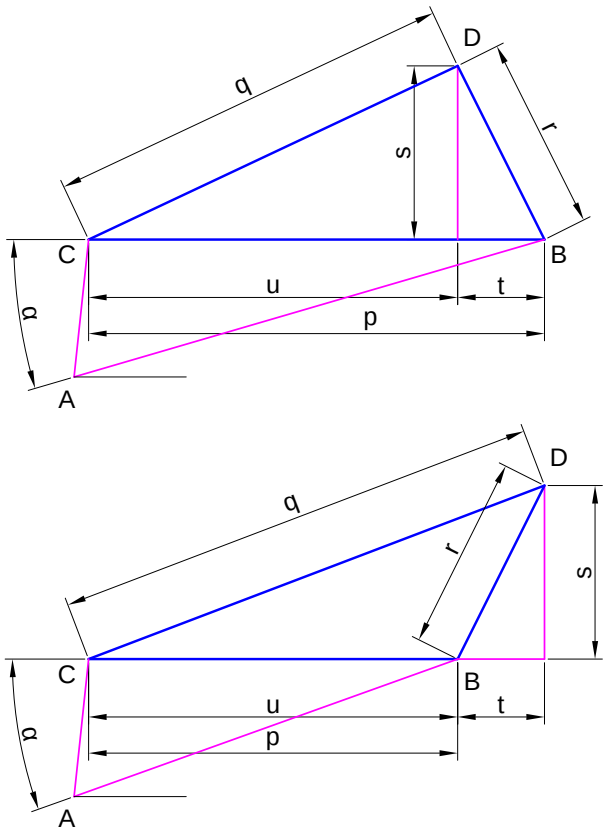


This question seems not easy to answer, but, as long as we talk about angles, we can say that:

- If the angle between the face normals is acute, D lies above BC.
- If the angle between the face normals is obtuse, D lies under BC.

After calculating the normals as explained before, the dot product is all we need to answer the question. Just remember to use the right hand rule.

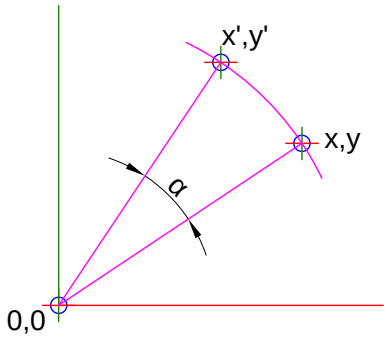
That was actually the hard part. D is more or less calculated the same way as C.



However, we need to use the axis CB (not AB) as the base, as the mirror line for when D gets under CB. Finally, C and B were rotated around A, D should also be rotated properly. That is where α starts to play a role. Rotating...

Calculate rotation of a coordinate in 2D

Seriously written for mathematicians, so don't read https://en.wikipedia.org/wiki/Rotation_(mathematics). Pff... Okay, it boils down to this:



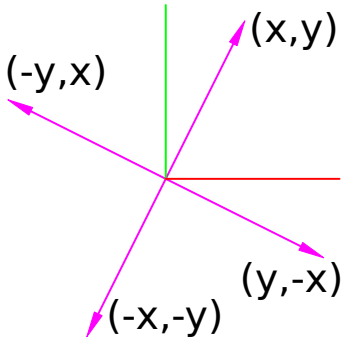
Given is coordinate pair x, y and angle α. Get x', y' by using:

```
x'=x*cos(a)-y*sin(a)
y'=y*cos(a)+x*sin(a)
```

In order not to get unexpected results, angle α is positive when counter-clockwise – in this example it is a positive value.

Tip!:For calculating a rotation in 3D you can consider two rotations, one in the xy-plane and one in the yz-plane.

When dealing with angles of 90 degrees another, easier, approach means changing signs and positions of coordinate values.



Value	CCW	Description
x,y	0	Equal
-y,x	90	90 (or -270) Normal (CCW)
-x,-y	180	180 Opposite
y,-x	270	270 (or -90) Normal (CW)

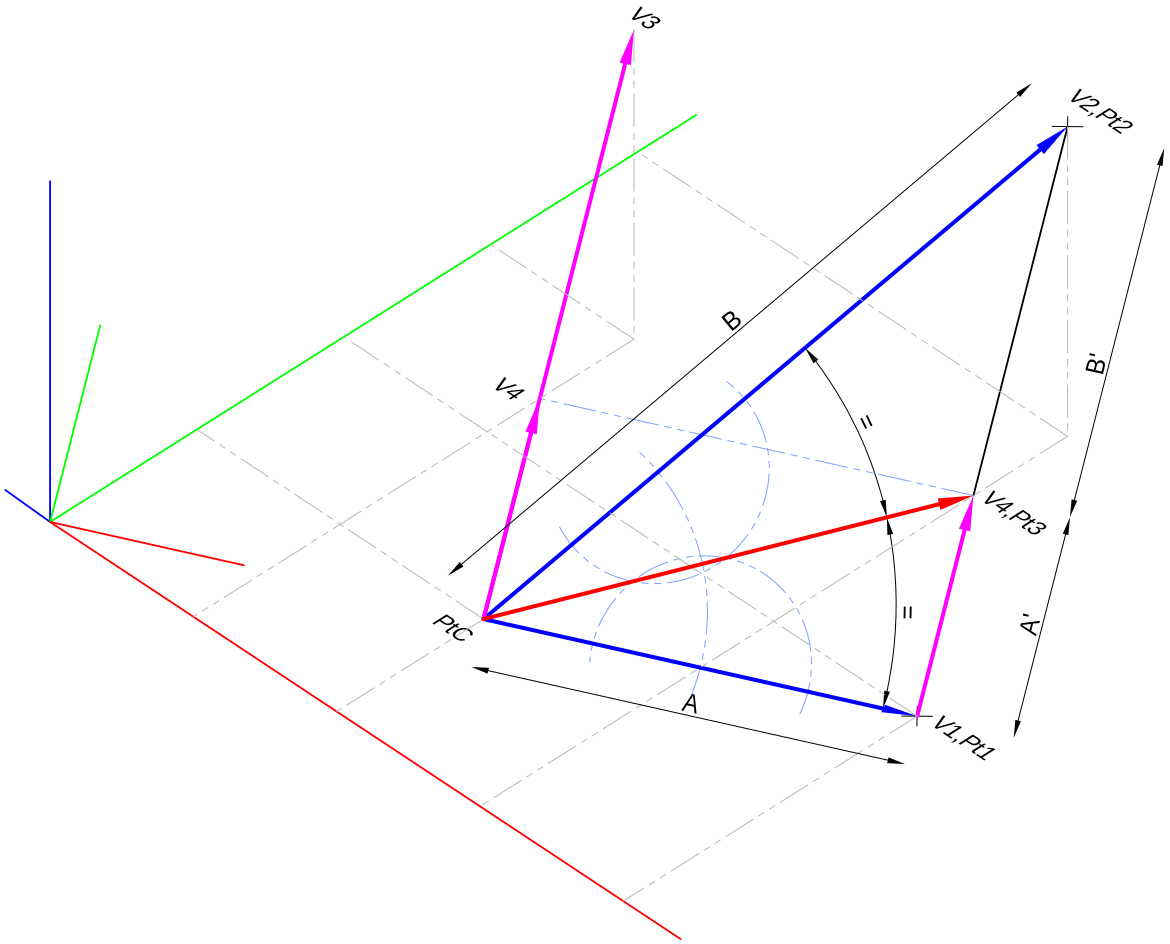
Translating and scaling

This one is easy, translations, like command **Move**, is just adding or subtracting static values to x, y and z values. Since we are talking about vectors, scaling means multiplying or dividing the x,y,z values individually with a constant factor. There is not much more to say about this, see the programming solutions below for more about **lv:add lv:subtract lv:multiply lv:divide**.

Determining bisector point

The image below:

- Two blue vectors $\vec{V1}$ and $\vec{V2}$ in space and the bisector as red vector $\vec{V4}$.
- $\vec{V1}$ and $\vec{V2}$ start from corner point PtC.
- Angle bisector theorem: $A/B = A'/B'$.
  - Magnitude, length, $\|V1\|$ and $\|V2\|$ are A and B and can be calculated.
  - $V3 = V2 - V1$.
    - Now we know $\|V3\|$ too and that is equal to $(A' + B')$.
  - By substitution we can now determine $V4$.
- Bisector point $Pt3$ is the sum of vectors PtC, $V1$ and $V4$, or, $Pt3 = PtC + V1 + V4$.



Trivia

- Constructing the bisector can be done with circles (see picture).
- A common mistake is to think that the bisector goes through the middle of line $Pt1 - Pt2$.

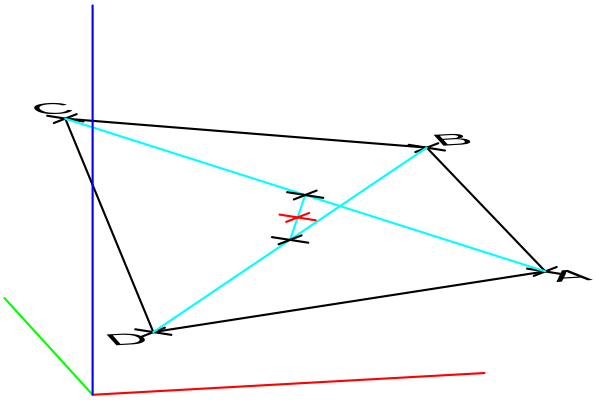
Practical Mathematical Solutions

With the above, we can solve some practical challenges.

Plane through four 3D coordinates

Basics

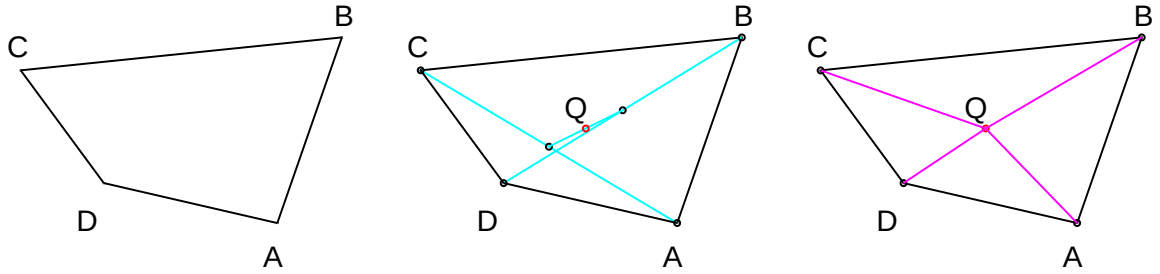
There are four 3D points, lying more or less in one plane. How can we define that plane so that the deviation, the distances of the points to that plane, is as small as possible?



The process is as follows:

- A quadrilateral has spatial coordinates $ABCD$.
- This example can be used for four measurement points of a total-station in one plane. In such a situation, it is important to realise that although those points are approximately in one plane, in practice they are never exactly in one plane.
- So we are looking for a method where that plane is defined such that the deviation, the distances of the points from that plane, is minimal.
- Several mathematical methods can be devised to solve this problem. The choice is as follows:

- The plane passes exactly through the mean of all coordinates (fig. right). This is point Q. Q can also be formulated as the midpoint between the midpoints of the diagonals (fig. middle).



- The angle of the plane is determined by averaging the normals of the new triangles, all of which have a vertex Q.

With this, we have the methodology to hand and this is arguably a proper method to minimise margins. It is good to realise that the plane lies unambiguously in space by point Q and the direction of the final normal vector.

- Point Q is determined. This results in four triangles: AQB, BQC and so on.
- Normals of all triangles can be determined according to the right-hand rule:
- $\vec{QA} \times \vec{QB}$ then $\vec{QB} \times \vec{QC}$ and so on. These normals are close in direction.
- An average normal $\vec{N}$ is then determined.

Hands-on

Given coordinates: A=(10.0 ,0.0,5.0) B=(30.0,10.0,10.0) C=(30.0,45.0,35.0) D=(10.0,35.0,30.0)

Coordinates $ABCD$: to variables **abcd**:

```
(setq a '(10.0 10.0 5.0) b '(30.0 10.0 10.0) c '(30.0 45.0 35.0) d '(10.0 35.0
30.0))
(10.0 35.0 30.0)
```

First step: get Q. Make list **qls**:

```
(setq qls (list a b c d))
((10.0 10.0 5.0) (30.0 10.0 10.0) (30.0 45.0 35.0) (10.0 35.0 30.0))
```

Get Q as variable **q**:

```
(setq q (lv:average qls))
(20.0 25.0 20.0)
```

So: $Q = (20.0,25.0,20.0)$, meaning we have part of the plane defined, the plane runs through Q. Next we need the normals. Involved are vectors QA, QB, QC and QD.

```
(setq qa (lv:subtract a q) qb (lv:subtract b q) qc (lv:subtract c q) qd
(lv:subtract d q))
(-10.0 10.0 10.0)
```

Cross products for these triangles, our normals are: $N\vec{a}qb = Q\vec{A} \times Q\vec{B}$, $N\vec{b}qc = Q\vec{B} \times Q\vec{C}$, $N\vec{c}qd = Q\vec{C} \times Q\vec{D}$ and $N\vec{d}qa = Q\vec{D} \times Q\vec{A}$.

Nice, but we want the average value of these four vectors. In fact, the magnitude of the average normal defines in a way the area of $ABCD$ and we might need to use that later on.

The magnitude of the cross product of each triangle is the area of the parallelogram based on that triangle. So dividing that magnitude by 2.0 equals the area of a triangle. We have four triangles. So the area, the magnitude of the averaged resulting normal should be multiplied by 4.0 and divided by 2.0. That equals to multiplying the average normal by $4.0/2.0{=}2.0$. Next: Determine $\vec{N}$ in Lisp:

```
(setq n
  (lv:divide
    (lv:average
      (list
        (lv:cross-product qa qb)
        (lv:cross-product qb qc)
        (lv:cross-product qc qd)
        (lv:cross-product qd qa)
      )
    )
    2.0
  )
)
(-6.25 -125.0 150.0)
```

About the magnitude $\|N\|$:

```
(setq mn (lv:magnitude '(-6.25 -125.0 150.0)))
195.356245101097
```

And that is it. The conclusion is that we have defined a plane in space based on these data:

The resulting plane goes through point Q with coordinate $(20.0,25.0,20.0)$ and has a normal vector $\vec{N}$ with coordinate (-6.25 -125.0 150.0), where the area $ABCD$ ~equals $\frac{\|N\|}{2.0} \approx 195.356$.

Final considerations

- This approach assumes that the points are somewhat regularly arranged around Q and that the points $ABCD$ run counterclockwise.
- Of course, such an approximation is also feasible with more than four points.
- In this approximation, the weight of all normal vectors is equal. You might consider basing the average value for the normal vector on the influence of each triangle. This can be achieved by including the magnitude of each normal vector as a weight index.
- The resulting point Q and vector $\vec{N}$ can be used as input parameters for command **UCS** with option Z-Axis or **ZA**, in order to create a drafting plane with z-coordinates of $ABCD$ close to zero.

Programming Solutions

The purpose of this code is educational. It works and aims to show the structure. Unless you experience slow processing, it should be sufficient.

In order to avoid interference with functions from third parties, functions below are prefixed with "**lv:**" as in **Lib Vector**.

Datatypes

Datatype will often be *REAL* but please be aware of the consequence of using integers (*INT*) as input. Some examples you can paste on the command line, divide 3 by 2 as in 3/2:

```
(/ 3 2) > 1 INT ... May not be what you expect
(/ 3.0 2) > 1.5 REAL
(/ 3 2.0) > 1.5 REAL
(/ 3.0 2.0) > 1.5 REAL
```

Also "divide by zero" is not handled.

Functions **lv:realp** and **lv:intp** can check input, protect you from wrong input. For example, force real input:

```
LV:REALP⏎
: (setq a 5)⏎
5
: (lv:realp a)⏎
nil
: (setq a 5.0)⏎
5.0
: (lv:realp a)⏎
5.0
```

So only when **a** is a *real*, value **a** is returned, otherwise **nil**.

It is not within the scope of this booklet but it is important enough to shortly illustrate datatypes. A bit more general, CAD Lisp knows many datatypes and many functions work only with one datatype. Function **(type ...)**⏎ shows the datatype. Some examples for the CLI:

(type "Hello")⏎	*STR*	Just a string
(type 100)⏎	*INT*	An integer
(type 100.0)⏎	*REAL*	A real
(type abc)⏎	*nil*	A not existing and therefore not assigned variable
(type (setq abc pi))⏎	*REAL*	π assigned to abc, so:
(type abc)⏎	*REAL*	... As expected
(type (oson))⏎	*SYM*	Symbol
(type c:c3)⏎	*SUBR*	Subroutine like this Lisp code command

You get the gist. If not, search the net.

SYM stands for symbol and is a basic ingredient of Lisp. In the system where parenthesis form the structure, symbols form a crucial part for definitions of functions, variables and more. Then what is a variable? A variable is also a symbol, but used for storing program data, like **(setq pr-data "Hello World")**⏎.

All output of the previous **(type...)** commands are symbols too. Check: **(type "Hello")**⏎ *STR* and **(type (type "Hello")**⏎ *SYM*. Yes, Lisp is an interesting language. *STR* turns out to be a symbol. How about this then: **(type (type 'type))**⏎. Also interesting is examining if a symbol is in use, with the **(boundp ...)** function. Some final examples:

(type (entlast))⏎	*ENAME*	Entity name of last entity
(type (ssadd (entlast)))⏎	*PICKSET*	Selection set
(setq selection (ssadd (entlast)))⏎		symbol *SELECTION*...
(type selection)⏎	*PICKSET*	... is a selection set too
(type "selection")⏎	*STR*	Just a string
(type 'selection)⏎	*SYM*	*SELECTION* is a symbol
(boundp 'selection)⏎	*T*	Is *SELECTION* assigned?

`(boundp 'boundp)⏎`	*T*	Is *BOUNDP* assigned?
`(boundp 'boundpositive)⏎`	*nil*	Is *BOUNDPOSITIVE* assigned?
`(type '(selection))⏎`	*LIST*	List with symbol
`(type (setq abc pi))⏎`	*REAL*	*ABC* is a real
`(boundp 'type)⏎`	*T*	Is *TYPE* assigned?
`(boundp 'types)⏎`	*nil*	Is *TYPES* assigned?
`(atoms-family 0 '("tYPe" "types"))⏎`	*(TYPE NIL)*	As `(boundp ...)`, but...

If a symbol does not exist, `(boundp 'symbolname)` creates the symbol with setting *nil*. `(atoms-family 0 '("symbolname"))` does not create the symbol.

Last but not least, an atom is the smallest possible part, symbol, of a list. You might want to say, everything that is not a list is an atom. The symbol of that 1000++ lines of code function *FOO* is the atom.

Fixed variables

Multiplying and dividing pi or π is daunting, some variables are added:

- **pim2** = π*2 ≈ 6.28, **PI** **M**ultiplied by **2**, i.e. 360°
- **pidN** = π/N, **PI** **D**ivided by **2**, with N = 2, 3, 4, 6 and 12, for resp. 90°, 60°, 45°, 30° and 15°. So pid2 = 90° ≈ 1.57 radians.
- **fr2d** = multiply **F**actor for **R**adians **2** **D**egrees ≈ 57.29. Function `(lv:r2d radians)` is an option too.
- **fd2r** = multiply **F**actor for **D**egrees **2** **R**adians ≈ 0.01745. Function `(lv:d2r degrees)` is an option too.

Functions

acos

Calculate arccosine in AutoCAD. BricsCAD has a native function `(acos ...)`.

Returns the arccosine of a number in radians. This code can be used for `(acos ...)`:

```
(acos
  num1
)
```

Arguments

Num1 is an integer or real.

Return Values

The arccosine of num1, in radians.

Examples

```
(acos (/ pi 4))
0.667457216028384
```

asin

Calculate arcsine in AutoCAD. BricsCAD has a native function **(asin...)**.

Returns the arcsine of a number in radians. This code can be used for **(asin ...)**:

```
(asin
  num1
)
```

Arguments

Num1 is an integer or real.

Return Values

The arcsine of num1, in radians.

Examples

```
(asin (/ pi 4))
0.903339110766513
```

lv:add

Returns the sum of two coordinate lists.

This functions is useful for vectors and or points.

Say, we have point1 and vector1, point1 is **x,y,z** and vector1 is **x',y',z'**. The result in spreadsheet notation is:

```
x+x',y+y',z+z'
```

```
(lv:add
  list1 list2
)
```

Arguments

List1 and list2 are lists containing x,y,z coordinates.

Return Values

A list with coordinates.

Examples

```
(setq pt1 (list 2 2 2))
(2 2 2)
(setq vec1 (list -2 -2 0))
(-2 -2 0)
(lv:add pt1 vec1)
(0 0 2)
```

lv:substract

Returns the difference of two coordinate lists.

This function subtracts the second argument from the first argument. This functions is useful for vectors and or points.

With point1 and vector1, point1 is **x,y,z** and vector1 is **x',y',z'**. The result in spreadsheet notation is:
x-x',y-y',z-z'

```
(lv:substract
  list1 list2
)
```

Arguments

List1 and list2 are lists containing x,y,z coordinates.

Return Values

A list with coordinates.

Examples

```
(setq pt1 (list 2 2 2))
(2 2 2)
(setq vec1 (list -2 -2 0))
(-2 -2 0)
(lv:subtract pt1 vec1)
(4 4 2)
```

lv:multiply

Multiplies coordinate values with a fixed value.

With coordinate list **x,y,z** and factor **f**, the result in spreadsheet notation is:
x*f,y*f,z*f

```
(lv:multiply
  list factor
)
```

Arguments

List contains x,y,z coordinates. Factor is a value.

Return Values

A list with coordinates.

Examples

```
(setq vec (list 2 3 4))
(2 3 4)
(setq fac 2.0)
2.0
(lv:multiply vec fac)
(4 6 8)
```

lv:divide

Divides coordinate values with a fixed value.

With coordinate list **x,y,z** and factor **f**, the result in spreadsheet notation is: **x/f,y/f,z/f**

```
(lv:divide
  list factor
)
```

Arguments

List contains x,y,z coordinates. Factor is a value, real or integer, see example for consequences.

Return Values

A list with coordinates.

Examples

```
(setq vec (list 2 3 4))
(2 3 4)
(setq fac 2.0)
2.0
(lv:divide vec fac)
(1.0 1.5 2.0)
(setq fac 2)
2
(lv:divide vec fac)
(1 1 2)
```

lv:minus

Additional inverses a list with values.

Changes the sign of a vector, creating the opposite vector.

Arguments

List contains values, for example x,y,z coordinates.

Return Values

A list with coordinates or values.

Examples

```
(setq vec (list 2 3 4.0))
(2 3 4)
(lv:minus vec )
(-2 -3 -4.0))
```

lv:sum

Calculate sum of coordinates.

Arguments

A list containing values, for example x,y,z coordinates.

Return Values

A list with coordinates or values.

Examples

```
(setq lst (list '(1 2 3) '(4 5 6) '(7 9 10.0)))
((1 2 3) (4 5 6) (7 9 10.0))
(lv:sum lst)
(12 16 19.0)
```

lv:average

Calculate average of coordinates.

Arguments

A list with coordinates or values.

Return Values

A list with coordinates or values.

Examples

```
(setq lst (list '(1 2 3) '(4 5 6) '(7 9 10.0)))
((1 2 3) (4 5 6) (7 9 10.0))
(lv:average lst)
(4 5 6.33333333333333)
```

lv:dist3d

3D distance between point 1 and 2.

In Lisp there is a function **(distance a b)** but it is tempting to use your own function because **(distance a b)** treats all points 2D when it encounters one 2D point. An alternative function **(lv:dist3d point1 point2)** works always 3D and calculates "the square root of the sum of the squares of the delta x, y and z values".

With the distance between point1 and point2, point1 is **x,y,z** and point2 is **x',y',z'**, the formula in spreadsheet notation is:
dist3d=sqrt(((x'-x)^2)+((y'-y)^2)+((z'-z)^2))

```
(lv:dist3d
  point1 point2
)
```

Arguments

Point1 and point2 are lists containing x,y,z coordinates.

Return Values

The 3D distance between point1 and point2 as a real.

Examples

```
(setq pt2 (list 4 0 0))
(4 0 0)
(setq pt1 (list 0 3 0))
(0 3 0)
(lv:dist3d pt1 pt2)
5.0
```

lv:vector

Vector between point 1 and point 2

For the vector coordinates, point 2 is subtracted from point 1.

```
(lv:dist3d
  point1 point2
)
```

Arguments

Point1 and point2 are lists containing x,y,z coordinates.

Return Values

Resulting vector as a list of reals and or integers.

Examples

```
(setq point1 (list 5.0 0.0 0.0))
(5.0 0.0 0.0)
(setq point2 (list 0.0 3.0 0.0))
(0.0 3.0 0.0)
(lv:vector point1 point2)
```

```
(-5.0 3.0 0.0)
```

lv:cross-product

Cross product of vector1 and vector2

Take care of the order of vector1 and vector2, it affects the sign of the coordinates.

```
(lv:cross-product
  vector1 vector2
)
```

Arguments

Vector1 and vector2 are lists containing x,y,z coordinates.

Return Values

The cross product as a list of reals and or integers.

Examples

```
(setq v1 (list 0.0 3 0))
(0.0 3 0)
(setq v2 (list 4 0 0))
(4 0 0)
(lv:cross-product v1 v2)
(0 0.0 -12.0)
```

lv:dot-product

Dot product of vector1 and vector2

```
(lv:dot-product
  vector1 vector2
)
```

Arguments

Vector1 and vector2 are lists containing x,y,z coordinates.

Return Values

The dot product as a real or integer.

Examples

```
(setq v1 (list 1 3 0))
(1 3 0)
(setq v2 (list 4 0 0))
(4 0 0)
(lv:dot-product v1 v2)
4
```

lv:magnitude

Magnitude of vector vc

```
(lv:magnitude
  vc
)
```

Arguments

Vector is a list containing x,y,z coordinates.

Return Values

The magnitude, vector length, as a real.

Examples

```
(setq vc (list 0.0 4.0 3.0))
(0.0 4.0 3.0)
(lv:magnitude vc)
5.0
```

lv:unit-vector

Unit vector of a vector vc.

```
(lv:unit-vector
  vc
)
```

Arguments

Vector is a list containing x,y,z coordinates.

Return Values

The unit vector of vector vc as a list of reals.

Examples

```
(setq vc (list 4 0 0))
(4 0 0)
(lv:unit-vector vc)
(1.0 0.0 0.0)
```

lv:acute

Check if angle between vector a and b is acute, according to right hand rule.

See also `(lv:arop ...)`.

```
(lv:acute
  vector1 vector2
)
```

Arguments

Vector1 and vector2 are lists containing x,y,z coordinates.

Return Values

T (true) is returned if angles are obtuse, else nil is returned.

Examples

```
(setq vector1 (list 1.0 0.0 0.0))
(1.0 0.0 0.0)
(setq vector2 (list 0.0 1.0 0.0))
(0.0 1.0 0.0)
(not (lv:acute vector1 vector2))
T
```

lv:arop

Determine if angle between vector a and b is **a**cute, **r**ight or **o**btuse (aro).

```
(lv:arop
  vector1 vector2
)
```

Arguments

Vector1 and vector2 are lists containing x,y,z coordinates.

Return Values

Function **lv:arop** returns string "**a**", "**r**" and "**o**" for resp. an **acute**, **right** or **obtuse** angle between vector **a** and **b**.

Examples

```
(setq vector1 (list 1.0 0.0 0.0))
(1.0 0.0 0.0)
(setq vector2 (list 0.0 1.0 0.0))
(0.0 1.0 0.0)
(lv:arop vector1 vector2)
"r"
```

lv:vector-ang

Angle between vector1 and vector2.

Calculation is based on right hand rule and function **acos**.

```
(lv:vector-ang
  vector1 vector2
)
```

Arguments

Vector1 and vector2 are lists containing x,y,z coordinates.

Return Values

Angle in radians as a real.

Examples

```
(setq vector1 (list 1.0 0.0 0.0))
(1.0 0.0 0.0)
(setq vector2 (list 0.0 1.0 0.0))
(0.0 1.0 0.0)
(/ (lv:vector-ang vector1 vector2) pi 0.5)
1.0
```

lv:ang2d

Projected angle of a vector as a list in radians, similar to function `(angle ...)`.

lv:r2d

Translate a value of radians to degrees.

lv:d2r

Translate a value of degrees to radians.

lv:rot-pt

Rotation of point **pt** over angle **ang**

Angle **ang** in radians, sign is positive when counter clock wise. Point **pt**: only **x** and **y** are evaluated.

```
(lv:rot-pt
  pt ang
)
```

Arguments

Point **pt** is a list containing x,y or x,y,z coordinates and Angle **ang** is a real or integer.

Return Values

A list containing resulting x,y coordinates

Examples

```
(setq pt (list 6.5 0.0 0.0))
(6.5 0.0 0.0)
(setq ang pi)
3.14159265358979
(lv:rot-pt pt ang)
(-6.5 0.0)
```

lv:rot-q-pt

Quadrant rotation of point or vector **pt** in steps (**1**, **2** or **3**) of 90 degrees CCW.

Quadrant **q** as a number, Point **pt**: only x and y are evaluated.

```
(lv:rot-q-pt
  q pt
)
```

Shortcuts without quadrant number argument are functions **lv:rot-q1**, **lv:rot-q2** and **lv:rot-q3**. See examples.

Arguments

Point **pt** is a list containing x,y or x,y,z coordinates and quadrant **q** is an integer **1**, **2** or **3**, i.e. 1 is 90, 2 is 180 and 3 is 270 degrees CCW.

Return Values

A list containing resulting x,y coordinates.

Examples

```
(setq pt '(1.0 2.0 3.0))
(1.0 2.0 3.0)
(setq q 3)
3
(lv:rot-q-pt pt q)
(2.0 -1.0)
(lv:rot-q3  pt)
(2.0 -1.0)
(lv:rot-q1  pt)
(-2.0 1.0)
```

lv:bisect

Determine bisector point

Two lines, under a certain angle, share one endpoint **C**. The other endpoints are **L1** and **L2**.

```
(lv:bisect
  C L1 L2
)
```

Arguments

Arguments are point lists, in order: corner point, point on leg 1 and point on leg 2.

Return Values

A list containing resulting x,y,z coordinates.

Examples

```
(setq C '(2.0 1.0 0.0) L1 '(4.0 2.0 0.0) L2'(3.0 4.0 2.0))
(3.0 4.0 2.0)
(lv:bisect C L1 L2)
(3.6259 2.7481 0.7481)
```

About and credits ...

Why does the chicken cross the road?

(Bricsys... London... 2018...)
For my work at NedCAD, I occasionally need maths. I devise CAD solutions and solve problems that are CAD-related. And so a problem exists for me: I am not a mathematician.

For that reason, I gather information to quickly refresh my memory. The article on WordPress, a single internet page, became too big, I couldn't quite figure out where I had made a likely mistake and so the idea arose to dump all into LibreOffice Writer.

"If you are using it as a reminder to yourself, why are you publishing it?"

I believe that everyone in this world has a moral obligation to share information. Wherever you live in this world, you will agree with me that future generations will have big problems to solve. Progress, innovation, is one of the most important conditions for solving them. Education, knowledge, is a key condition and so it makes sense to publish. I don't make my money by publishing, I make it by helping organisations with innovative solutions on a commercial basis.

The advice: put this booklet in your digital bookshelf and when you think, ‘how was it again?’, grab it. Occasionally I add some information but I think the basics are pretty complete - you can always check for updates.

If you think something is wrong, missing or could be better, please comment. Contact information see NedCAD. By the way, my name is Wiebe van der Worp.

Some statistics and information

This document is created with LibreOffice Writer, running on Linux desktop. It relies heavily on templating (nedcad.ott). This document contains unmodified $\LaTeX$-code using TexMath. More about $\LaTeX$... Several illustrations are exports from BricsCAD, post processed by Inkscape and embedded in this document as pure SVG - no jaggy bitmaps.

Statistics: 6827 words, 38352 characters in file Coordinate_Mathematics_for_CAD.odm.

Typesetting

The text you are reading at this very moment in time, is styled using font MLMroman9, a slightly bolder and better readable variant of Latin Modern, on its turn a typical LaTeX-font. For formulas, made with TexMath, a font is used named "TexMath Symbols". As you noticed, sometimes "sheet notation" is used for mathematics. Let's examine these:

- Mathematic notation in LaTeX
  $AC = 4, 5, 6$
  $\|AB\| = \sqrt{(1^2 + 2^2 + 3^2)} = 3.74$
- Spreadsheet notation in "MLMroman9"
  AC = 4,5,6
  ||AB|| =SQRT(1^2+2^2+3^2) = 3.74
- *Spreadsheet notation in "TexMaths Symbols"*
  AC = 4,5,6
  ||AB|| =SQRT(1^2+2^2+3^2) = 3.74
- **Spreadsheet notation in "FreeMono"**
  AC = 4,5,6
  ||AB|| =SQRT(1^2+2^2+3^2) = 3.74

A document like this should have a harmonious style rather than a chaotic collection of totally different styles. It takes some tinkering to pull that off. After all, there has to be a clear difference between the reading texts, the mathematics and the formulas. Spreadsheet notation in 'TexMaths Symbols' is tempting, but 'FreeMono' is much clearer and more distinguishable. By the way, **`commands`** and *`(their) output`* are formatted too.

License

Lisp code as one file

The following is an attachment of this document: *`lib-vector.odt`*. You may want to open it separately and paste the contents in a text file *`lib-vector.lsp.`*

In Notepad++ syntax highlighting may be very helpful. We've created a language file, you can find and download it here.

lib-vector.lsp can be loaded by dragging into the drawing area, or better, by proper loading in BricsCAD (and AutoCAD).

```
;; Functions and commands for vector mathematics.
;; License: Creative Commons By (NedCAD) and SA, Share Alike
;; Put all of this in a file called lib-vector.lsp in the search path or drag
file to the dwg.
;; "How it works...":
;; See https://nedcad.nl/coordinate-mathematics-for-cad/ or search the net for
"lib-vector.lsp"
;; Functions: lv:function-name and lv: means Library Vector.
(setq lib-vector-loaded T) ; for testing, (if (not lib-vector-loaded) (load
"lib-vector.lsp"))
;; Adding functionality for AutoCAD: function acos and asin
(if (not acos)(defun acos (r / ) (atan (sqrt (- 1 (expt r 2))) r)))
(if (not asin)(defun asin (r / ) (atan r (sqrt (- 1 (expt r 2))))))
;; Some additional variables pim2, pid2...
(setq pim2 (* pi 2) fr2d (/ 180.0 pi) fd2r (/ 1 fr2d) nx '(1 0 0) ny '(0 1 0) nz
'(0 0 1))
(mapcar '(lambda (a) (set (read (strcat "pid" (itoa a))) (/ pi a))) '(2 3 4 6
12))
;; Functions...
(defun lv:intp (a / ) (if (= (vl-symbol-name (type a)) "INT") a ))
(defun lv:realp (a / ) (if (= (vl-symbol-name (type a)) "REAL") a ))
(defun lv:dist3d (a b / )
  (sqrt (apply '+ (mapcar '(lambda (k l) (expt (- l k) 2)) a b)))
)
(defun lv:vector (a b / ) (mapcar '(lambda (k l) (- l k)) a b))
(defun lv:cross-product (a b / lsa lsb lsc lsd xa xb ya yb za zb)
  (setq
    lsa (list (setq ya (cadr a)) (setq za (caddr a)) (setq xa (car a)))
    lsb (list (setq zb (caddr b)) (setq xb (car b)) (setq yb (cadr b)))
    lsc (list yb zb xb) lsd (list za xa ya)
  )
  (mapcar '(lambda (o p q r) (- (* o p) (* q r))) lsa lsb lsc lsd)
)
(defun lv:dot-product (a b / ) (apply '+ (mapcar '(lambda (k l) (* k l)) a b)))
(defun lv:magnitude (a / ) (sqrt (apply '+ (mapcar '(lambda (b) (expt b 2.0))
a))))
(defun lv:add (a b / ) (mapcar '(lambda (c d) (+ c d)) a b))
(defun lv:subtract (a b / ) (mapcar '(lambda (c d) (- c d)) a b))
(defun lv:multiply (a b / ) (mapcar '(lambda (c) (* c b)) a))
(defun lv:divide (a b / ) (mapcar '(lambda (c) (/ c b)) a))
(defun lv:minus (a / ) (mapcar '(lambda (b) (- b)) a))
(defun lv:sum (a / ) (apply 'mapcar (cons '+ a)))
(defun lv:average (a / ) (lv:divide (lv:sum a) (length a)))
(defun lv:unit-vector (a / )
  (lv:divide a (lv:magnitude a))
)
(defun lv:acute (a b / ) (if (> (lv:dot-product a b) 0.0) T))
(defun lv:arop (a b / dp)
  (setq dp (lv:dot-product a b))
  (cond ((> dp 0.0) "a") ((< dp 0.0) "o") ("r"))
)
(defun lv:vector-ang (a b / )
   (acos (/ (lv:dot-product a b) (* (lv:magnitude a) (lv:magnitude b))))
)
(defun lv:ang2d (a / ) (angle '(0 0) a))
(defun lv:r2d (a / ) (* a (/ 180.0 pi)))
(defun lv:d2r (a / ) (* (/ a 180.0) pi))
(defun lv:rot-pt (a g / sg cg xa ya)
   (setq sg (sin g) cg (cos g) xa (car a) ya (cadr a))
   (list (- (* xa cg) (* ya sg)) (+ (* ya cg) (* xa sg)))
)
(defun lv:rot-q-pt (q xy / )
  (if (<= 1 q 3)
    (cond
      ((= 1 q) (list (- (cadr xy)) (car xy)))
      ((= 2 q) (list (- (car xy)) (- (cadr xy))))
      ((= 3 q) (list (cadr xy) (- (car xy))))
      T nil
    )
  )
)
(defun lv:rot-q1 (a / ) (list (- (cadr a)) (car a)))
```

```
(defun lv:rot-q2 (a / ) (list (- (car a)) (- (cadr a))))
(defun lv:rot-q3 (a / ) (list (cadr a) (- (car a))))
(defun lv:bisect (a b c / v1 v2)
  (setq v1 (lv:subtract b a) v2 (lv:subtract c a))
  (lv:add
    b
    (lv:multiply
      (lv:subtract v2 v1)
      (expt (+ 1 (/  (lv:magnitude v2) (lv:magnitude v1))) -1.0)
    )
  )
)
;; Start user functions
(defun c:ncnormal3p ( / a b c m n)
  (princ "\nDraws a normal vector as a line, based on right hand rule and 3
points. ")
  (setq c (getpoint (setq b (getpoint (setq a (getpoint "\nSelect first point: "))
"\nSelect second point: ")) "\nSelect third point: "))
  (setq m (mapcar '(lambda (d) (/ d 3)) (mapcar '+ a b c)))
  (setq n (lv:add m (lv:cross-product (lv:vector a b) (lv:vector a c))))
  (entmakex (list (cons 0 "LINE") (cons 10 m) (cons 11 n)))
  (princ)
)
(defun c:ncnormalize ( / a ed)
  (princ "\nNormalizes a vector represented by a Line entity. ")
  (while (not (= "LINE" (cdr (assoc 0 (setq ed (entget (car (entsel "\nSelect the
Line entity to normalize: "))))))))
  (setq
    a (cdr (assoc 10 ed))
    ed (subst (cons 11 (lv:add a (lv:unit-vector (lv:subtract (cdr (assoc 11 ed))
a)))) (assoc 11 ed) ed)
  )
  (entmod ed)
  (princ)
)
(princ "\nLibrary Vector (lv:function-name) loaded. ")
(princ)
```