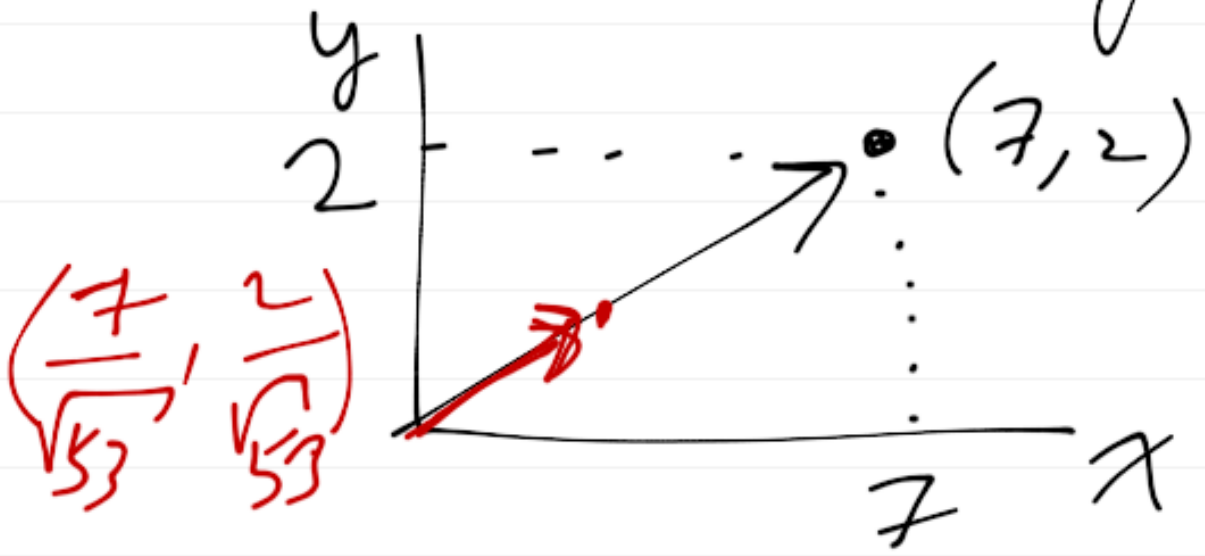
vec_unit(vec_t v1, vec_t v2)

// Compute v2 (output), a
// unit vector in dir of v1



V = (7, 2) is a point, a location, with coords (7, 2)

Unit vector $\equiv \frac{V}{\|V\|}$ direction

Keep in mind "duality" of vector semantics : position or direction

```
Vec_unit(Vec_t vin, Vec_t vout)
{
    vout[0] = vin[0] / Vec_len(vin);
    vout[1] = vin[1] / Vec_len(vin);
    vout[2] = vin[2] / Vec_len(vin);
}
```

aliasing

Suppose A call Vec_unit:

Vec_unit(v1, v1), intend

to normalize v1

---

the fix: use temp var.

for Vec_len(v1)

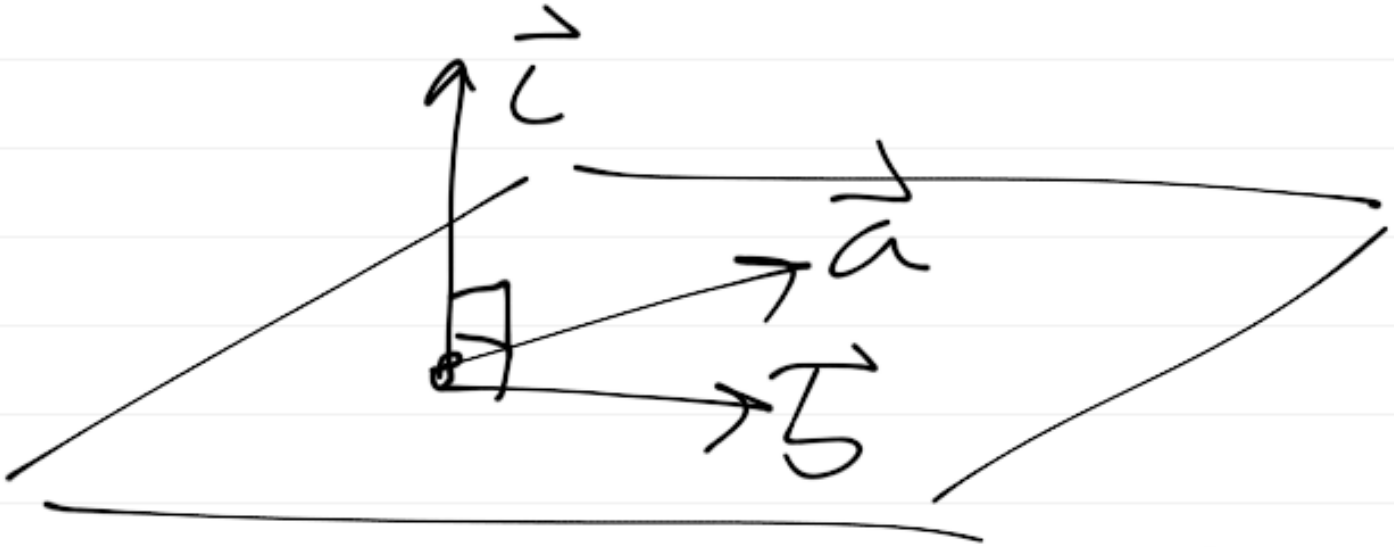
len = Vec_len(v1)

v2[0] = v1[0] / len

⋮

Cross product

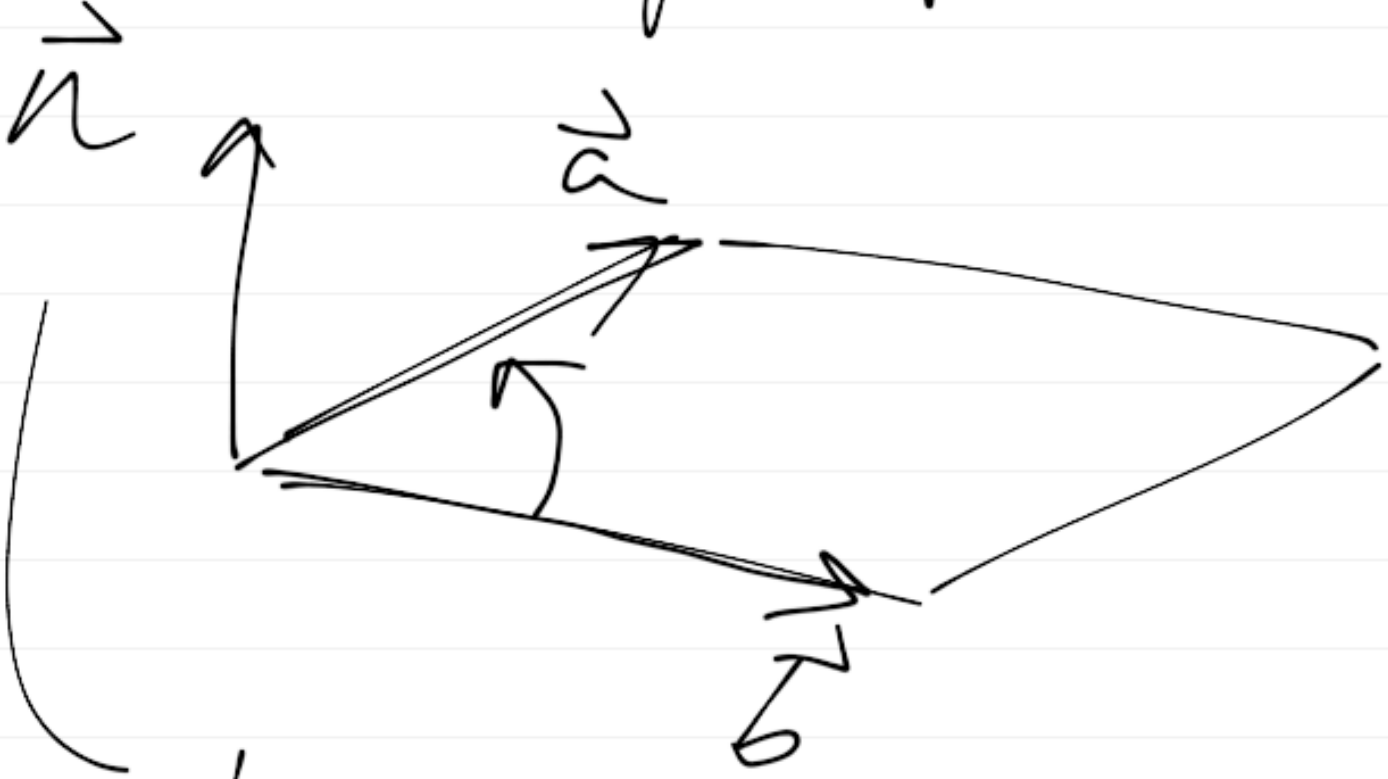
$$\vec{c} = \vec{b} \times \vec{a}$$



$\vec{c}$ is perpendicular ($\perp$) to both $\vec{a}$ & $\vec{b}$ ("orthogonal")

(above, $\vec{b} \times \vec{a}$ uses right-hand rule)

We need this to determine orientation of a plane



plane normal: vec. in dir of plane

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = c_x$$

$a_y b_z - a_z b_y$

$a[1] * b[2] - a[2] * b[1]$

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = c_y$$

$a_z b_x - a_x b_z$

$a[2] * b[0] - a[0] * b[2]$

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = c_z$$

$a_x b_y - a_y b_x$

$a[0]b[1] - a[1]b[0]$

Vec_cross(v1, v2, v3)

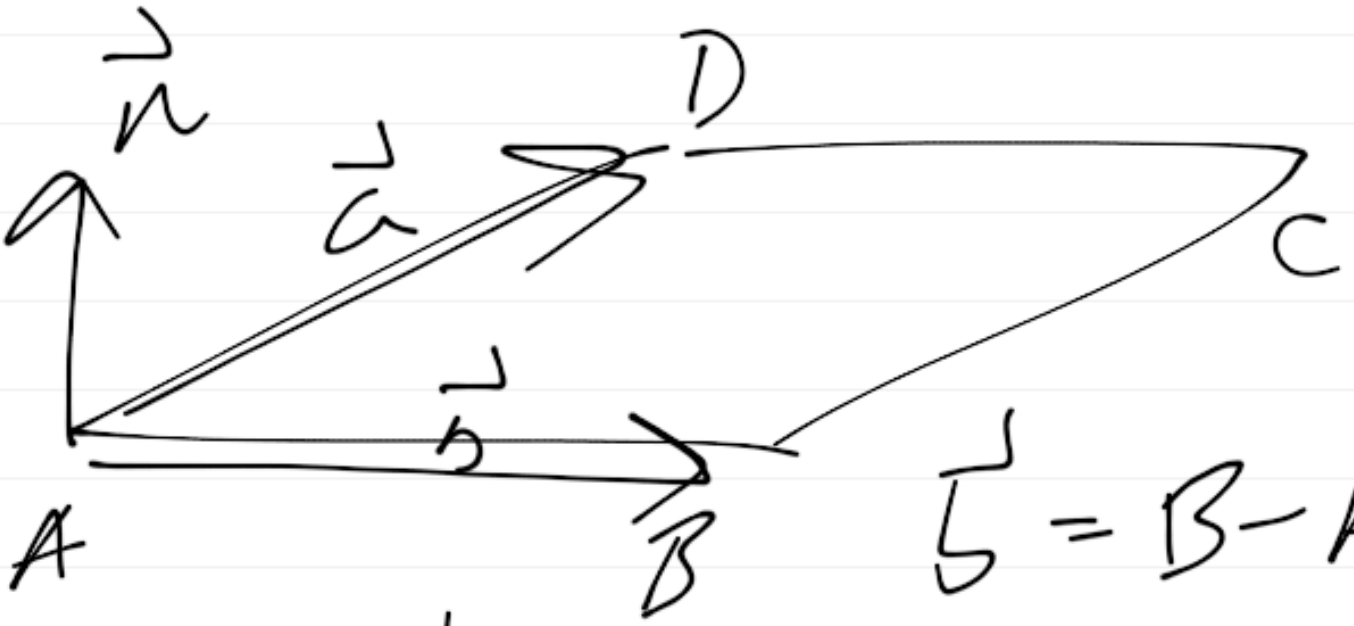
v1 × v2 = result

$v3[0] = v1[1]v2[2] - v1[2]v2[1]$

$v3[1] = v1[2]v2[0] - v1[0]v2[2]$

$v3[2] = v1[0]v2[1] - v1[1]v2[0]$

```
int i
Vec_t wj
for (i=0; i<3; i++)
  w[i] =
    v1[(3+i+1)%3]vt[(3+(i-1))%3] -
    v1[(3+(i-1))%3]v2[(3+i+1)%3]
```



$\vec{b} = B - A$

$$\vec{n} = \vec{b} \times \vec{a}$$

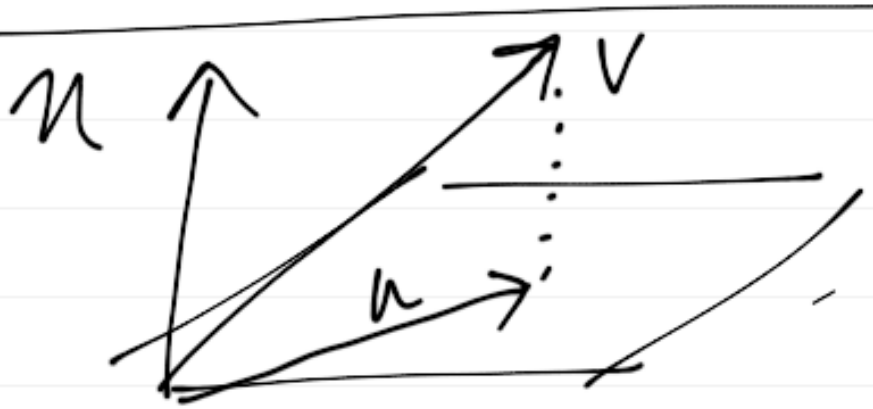
$$= \frac{(B-A)}{\|B-A\|} \times \frac{(D-A)}{\|D-A\|}$$

vec-project (n, v, u)

↑ result:

the projection u of vector V onto plane defined by n (normal)

$$u = V - (v \cdot n)n$$



```
vec_t w; //work vec

vec_scale(-vec_dot(v,n),
          n,
          w)

vec_sum(v, w, u)

vec_unit(u, u) //to
               normalize u
```

in C++ : use operators

code looks almost

like math eg.

in Obj C :

[u sum: v : w];

something similar ...

- Building a linked list
- want a generic list
  (in C++ parlance, a
  container)
- We'll use it for our
  scene objects
- the list is composed of
  2 data types:
  list
  (header) / link
  (node)

```
typedef struct _link
{
    struct _link *next;
    void         *data;
} link_t;
```

void* data;

or

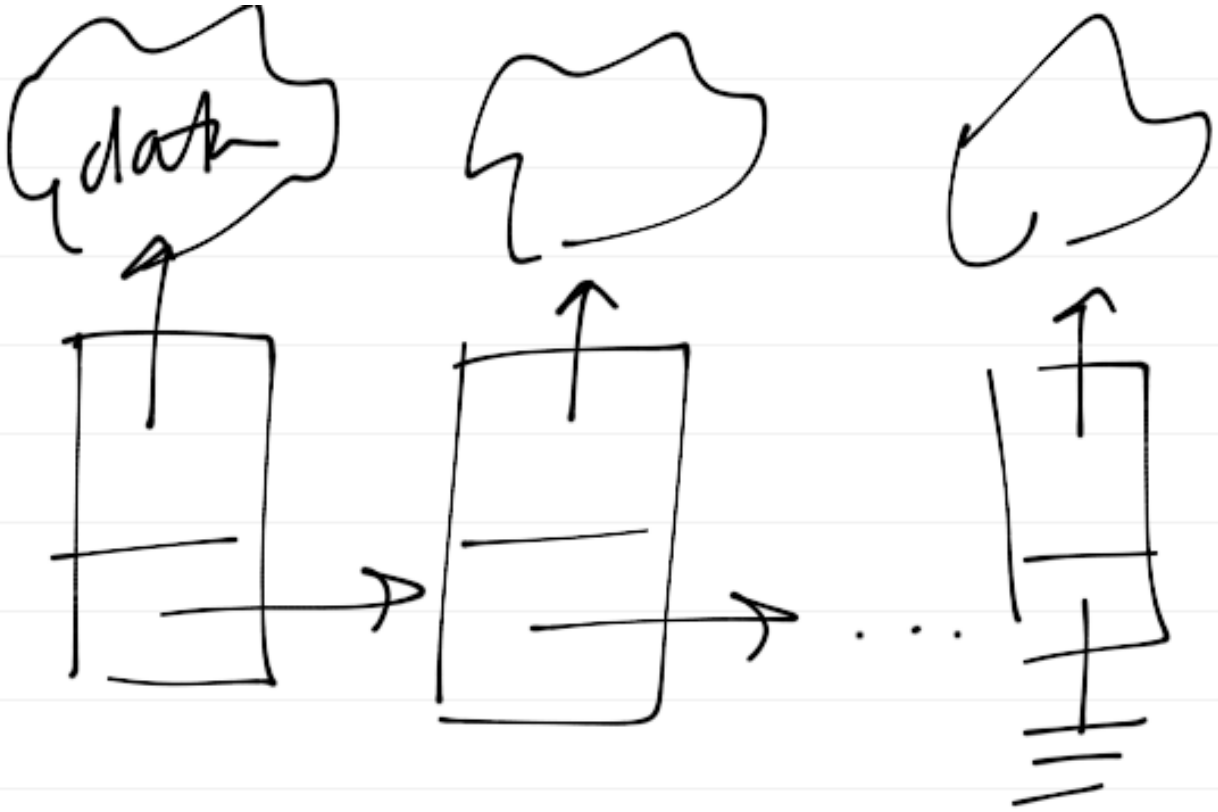
void *data;

?

---

void* data1, data2;

void *data1; void data2;



- for every data type
(like link_t)
write "object constructor"
(memory alloc & init)

- Usage:

```
link_t *link =
    link_init(datap);
```

↑ ptr to data

```
#include <assert.h>
#include "list.h"
link_t link_init(void *data)
{
  link_t *link =
    (link_t *) malloc(
                sizeof(link_t));
  assert(link != NULL);
```

— OR —

```
link_t *link = NULL;

if (!(link = (link_t *)
        malloc(sizeof(link_t)))
        == NULL)
{
    fprintf(stderr, "out
        of memory\n");
    exit(1);
}
— or — // assert
```

```
    link->next = NULL;
    link->data = data;
    return (link);
} // link Constructor
```

list header

one of these per list

```
typedef struct _list
{
    link_t *first, *last;
    link_t *current;
} list_t
```

usage:

```
list_t   *elist=NULL;

elist=list_init();

list_t *list_init(void)
    list_t *hdr=NULL;
    hdr=(list_t *)malloc(
        sizeof(list_t));
    assert(hdr!=NULL);
```

```
hdr->first =
hdr->last =
hdr->current = NULL;
return (hdr);
}
```

```
int list_empty(list_t
               *list)
{
    if(list->first == NULL)
        return 1;
```

— or —

```
    return(list->first == NULL ?
                1 : 0);
}
```