```
dcl 1 node based (np),
      2 next ptr,
      2 refcount fixed bin (7),
      2 action fixed bin (7),
      2 value ptr,
      2 description ptr,        ← if constant, points at value to copy
      2 narg (1),               ← if bif, points at desc w/out nodelist
        3 nodep ptr,
        3 groupp ptr,
        3 value ptr,
        3 index fixed bin;

dcl 1 desc based (dp),
      2 outargs fixed bin,
      2 inargs fixed bin,
      2 nodelist ptr,
      2 output (1),             { blank means not used
        3 name char(8),         { variable
        3 ptr ptr;              { 8th user defined
      2 input (1),              { variable
        3 name char(8);         {
      2 pattern (16),           { display pattern
        3 bit(16);              {
      2 special-action fixed bin;   ← on creation or deletion

      2 next ptr,               ┐
      2 name char(8),           ├ first on list is current ws
      2 action fixed bin,       ┘
```

arg function (for input args)

```
arg: proc (np, ep, vp);
   if np -> node.narg(1).value -> item-type ^= 1 then
                                    signal tossage;
   vp = ep -> node.narg (np -> node.narg(1).value -> value.value)
                                    .value;
end;
```

22 node desc 67 82 ;

22 30 freeit alloca function

22 47 54 67 82 68

```
dcl 1 Item based,
      2 type fixed bin(7),
      2 refcount fixed bin(7),
(1)   2 value float;
(2)   2 point,
        3 x float,
        3 y float;
(3)   2 color,
        3 i float,
        3 h float,
        3 s float;
(4)   2 shape,
        3 npoints fixed,
        3 center,
          4 x float,
          4 y float,
        3 color,
          4 i float,
          4 h float,
          4 s float,
        3 points(1),
          4 x float,
          4 y float;
(5)   2 group,
        3 first ptr;   -> 1 group_element based (gp)
                          2 next ptr,
                          2 value ptr;

(6)   2 ptrlist,
        3 nptrs fixed,
        3 ptr(1) ptr;
```

```
eval: proc (nodelist, ep)

      none = "1"b;
reiterate: do  np = nodelist  repeat  node.next  while (np^=null());
      nargs = node.nargs;
      do i = 1 to nargs;
           if node.narg(i).nodep -> node.value  = null() then do;
                                                none = "0"b;
                                                go to skip_one;
                                                end;
      end;
      group = "0"b;
      rp = null();        lp = null();
      do i = 1 to nargs;
           tp = node.narg(i).nodep -> node.value;
           if tp -> item.type = 5 then do;
                                       group = "1"b;
                                       node.narg(i).groupp = tp->group.first;
                                       node.narg(i).value = tp -> group.first
                                                          -> group_element.value;
                                       tp = tp -> group.first;
                                       end;
                else do;
                     node.narg(i).groupp = null();
                     node.narg(i).value = tp;
                     end;
           n = node.narg(i).index;
           if n ^= 0 then do;
                          if tp -> item.type ^= 6 then signal lossage;
                          node.narg(i).value = tp -> ptrlist.ptr(n);
                          end;
      end;

evloop:
      if np -> node.action > 0 then if node.action <= maxfun then do;
                                    f = functions(np->node.action);
                                    call f(np, ep, vp);
                                    if vp = null() then signal lossage;
                                    go to cons_vp;
                                    end;
```

```
else do;
    dp = node.description;
    call evaluate (desc,nodelist,np);
    n=desc.outargs;
    call allocn (desc.outargs*2+4, vp);
    vp -> item.type = 6;  vp -> item.refcount=0;
    vp -> ptrlist.nptrs = n;
    do i = 1 to n;
        vp -> ptrlist.ptr(i) = desc.output(i).ptr -> node.value;
    end;
end;

cons_vp:
    if ^group then do;
        np -> node.value = vp;
        end;
    else do;
        allocate group_element set(gp);
        group_element.next = null();
        group_element.value = vp;
        if rp=null() then do;
            allocate group set(rp);
            rp -> item.type = 5;
            rp -> item.refcount = 0;
            rp -> group.first = gp;
            end;
        else do;
            lp -> group_element.next = gp;
            end;
        lp = gp;
        done = "1"b;
        do i = 1 to nargs;
            if node.nargs(i).groupp ^= null() then do;
                gp = node.nargs(i).groupp ->
                    group_element.next;
                if gp = null() then go to noarg;
```

```
        done = "1"b;
        do i = 1 to nargs;
            tp = node.narg(i).groupp;
            if tp ^= null() then do;
                tp = tp -> group_element.next;
                node.narg(i).groupp = tp;
                node.narg(i).value = tp -> group_element.value;
                if tp ^= null() then done = "0"b;
                end;
            end;
        if ^done then go to evloop;
    end;

    node.value -> item.refcount = node.refcount;
    do i = 1 to nargs;
        tp = node.narg(i).nodepp -> node.value;
        if tp -> item.refcount = tp -> item.refcount - 1;
        if tp -> item.refcount = 0 then call freeit(tp);
    end;

skip_one:
    end;
    if ^none then go to reiterate;
    return;
end /* eval */;
```

```
line:   lineno=lineno+1;
        i= frasci(name, addr(line), 132);
        if i<=0 then go to emit;
        call askset(line);
        call askc("(", name);
          if i=0 then go to error;
          nargs=0; actno=actno+1;
        call askc(",", tok);
          if i=0 then go to error;
          if tok ^= "(" then go to createq;
anarg:  Call askc("(", tok); if i=0 then go to error;
          nargs= nargs+1;
          arg(nargs) = tok;
        call askc(",", tok); if i=0 then go to error
          if tok ^= ")" then go to anarg;
        call askc(i, tok);
          if i=0 then go to make;
createq: if tok = "/" then go to delete;
                crnum = crnum+1;
                crs(crnum) = tok;
                crno = crnum;
        call askc(i, tok);
          if i=0 then go to make
delete:   if tok ^= "/"
                then go to error;
          call askc(i, tok);
            if i=0 then go to make;
            dlnum = dlnum+1;
            dls(dlnum) = tok;
            dlno = dlnum;
make:   call alloc(10*nargs+16, dp);
          desc.next = desks;
          desks = dp;
          desc.name = name;
          desc.action = actno;
          acts(actno) = name;
          desc.fixargs = nargs;
          desc.vargs = nargs;
          desc.xargs = 0;
          desc.create = crno;
          desc.delete = dlno;
          do i=1 to nargs;
            desc.args(i).name = arg(i);
          end;
          go to line;

actno=actno+1;
crno=0; dlno=0;
```

```
$Gxxxx    DC   $Gxxxy
          DC   C'name      '
          DB   action, args, args, 0
          DC   0
          DB   create, delete
          DS   64
<         DC   C'arg       ', 0>
```

actlist (200) char (8),
(create, delete) (150) char (8)

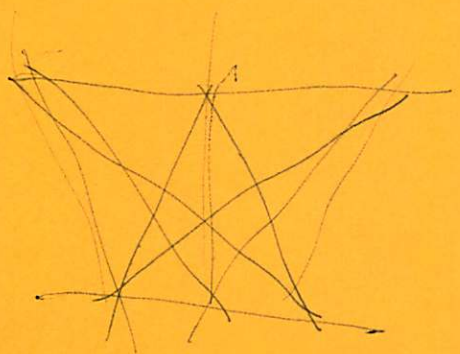
name (arg_1, ...) create, delete

Very basic nodes.

```
#       DC  o------------------> * etc
        DC  C'NUMBER'
        DB  1,0,0,1,
        DC  0
        DB  1,1
        DS  64
        DC  C'VALUE',0
```

```
[ NAME
    INARGS  (  ,  ,  )
    OUTARGS (  ,  ,  )
    ACTION  (  ,  ,  )
            —  , C=  , D=
```

node definition language



```
        DC  X<nnnnn+1>
        DC  C'name  '
        DB  action,stat,dist,out
        DC  0
        DB  cr,de
        DS  64
        DC  C'arg   ',X'FFFF'  for outarg
        DC  C'inarg ',0
```

Example II:

rotating line;

```
t:proc (s,n.);

    del 1 s,
          2 l float,
          2 theta float,
          2 cx flt.,
          2 cy flt;

    if n=0 then do;

                hx = l * cos (theta);
                hy = l * sin (theta);
                call line (x-hx, y-hy, x+hx, y+hy);
                return;
                end;

    l2 = l/2.0;
    theta2 = theta/6.2832;
                ct = cos (theta2) * l2/2.0;
                st = sin (theta2) * l2/2.0;
    cx2 = cx + ct;
    cy2 = cy + st;
    call t( s2, n-1);
    cx2 = cx - ct;
    cy2 = cy - st;
    call t(s2, n-1);

end t;
```

Example III

nested triangles

```
t:proc (S, n);
    dcl 1 S,
          2 l fixed,
          2 x fixed,
          2 y fixed;
    dcl n fixed;
    if n=0 then do;
                call line (x, y, x+l, y);
                call line (x+l, y, x+l/2, y+0.866*l);
                call line (x+l/2, y+0.866*l, x, y);
                return;
                end;
    l2 = l/2;        /* case A */
    x2 = x;
    y2 = y;
    call t(S2, n-1);
    x2 = x + l2;     /* call B */
    y2 = y + 0.866*l;
    call t(S2, n-1);
    x2 = x + l;
    y2 = y;          /* case C */
    call t(S2, n-1);
    return;
    end;
```

inductive keyframe form

discrete form, cyclically dependent on time

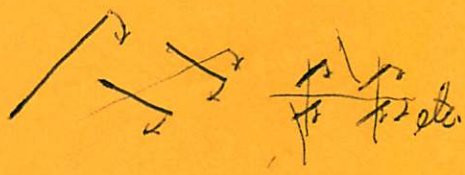
interpolation clause

$$f_A(N,t) = \bigvee_x N_x \left[A_x[\lfloor t \rfloor \bmod \hat{A}] + \{ A_x[\lfloor t+1 \rfloor \bmod \hat{A}] - A_x[\lfloor t \rfloor \bmod \hat{A}] \} \xi(t - \lfloor t \rfloor) \right]$$

f_A: keyframe table; N: state vector; t: time after inductive scaling

usually $t_i = \hat{A} t_{i-1}$

induction for adding motions



$l_i = \frac{1}{2} l_{i-1}$ lengths one cut in half

$\theta_i = \frac{6}{\pi_i}$ rotations one input

$\pi_i = \frac{\pi_{i-1}}{2\pi}$ in $n \in \mathbb{Z}$

for nice even rotations

$$Cx_i = Cx_{i-1} \pm \frac{l_i}{2} \cos\theta_i \qquad Cx_1 = 0$$
$$Cy_i = Cy_{i-1} \pm \frac{l_i}{2} \sin\theta_i \qquad Cy_1 = 0$$

} vector algebra restated

structural induction

$$l_i = \frac{l_{i-1}}{2}$$

$$x_i = A_x l_i + x_{i-1}$$
$$y_i = A_y l_i + y_{i-1}$$

A

x	y
0	0

B

x	y
$\frac{1}{2}$	$\frac{\sqrt{3}}{2}$

C

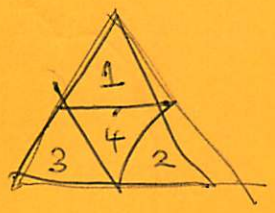
x	y
1	0

Note

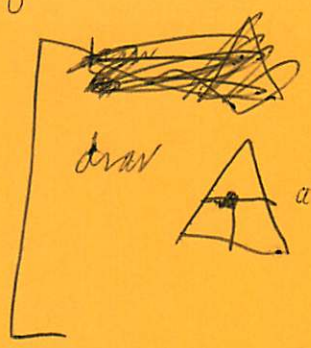
This is discrete form of keyframe form above.

Amusing side result: $\sum_{i=0}^{n} m^n = \frac{m^{n+1} - 1}{m-1}$

```
build = proc (x, y, l, n, p);

    if n <= 0 then do;
        dy = y - dl/2.0;
        dx = 0.8660 * l;   /* √3/2 */
        call line (x, y+l, x-dx, dl);
        call line (x-dx, dl, x+dx, dl);
        call line (x+dx, dl, x, y+l);
        fl.x = x;
        fl.y = y;
        xcolor = fl.color + 1;
        call flood (fl); fl.color = xcolor
        alloc node set (p);
        node.a = null();
        node.b = null();
        node.c = null();
        node.x = x;
        node.y = y;
        node.gray = fl.color;
        return;
        end;

    alloc node set (p);
    node.x = x;
    node.y = y;

    color = fl.color + 1;
    node.gray = fl.color;
    l2 = l/2.0;
    dy = y - l2;
    dx = 0.8660 * l;
    call build (x, y+l, l2, n-1, p->node.a);
    call build (x-dx, dy, l2, n-1, p->node.b);
    call build (x+dx, dy, l2, n-1, p->node.c);
    fl.color = xcolor;
    fl.x = x;
    fl.y = y;
    call flood (fl);    end;
```



if n=0



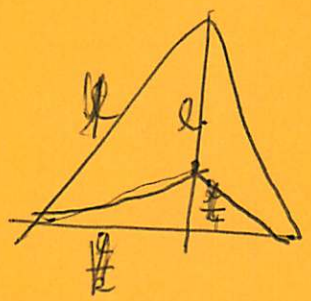
draw at cx, cy

flood w/ color

draw (A, B, C triangles)

flood w/color

build call tree



$\frac{\sqrt{3}l}{2} - \frac{l}{4}$

$2\sqrt{3}$

A 0, l

B $\frac{\sqrt{3}}{2}l, \frac{-l}{2}$

C $+\frac{\sqrt{3}}{2}l, \frac{-l}{2}$

a grammatical programming style!

State $S \rightarrow S_A\ S_B\ S_C \ldots$

where A, B, C are transforms

which are usually

- time variant — colors
- cyclic /
- constant — structure

- interpolated — colors
- discrete — structure

etc.

So

```
t: proc (S, n)
    if n=0 do drawing for S
    else   t (S_A, n-1)
           t (S_B, n-1)
           ⋮
```

Example I

nested colors

```
x: proc (A, N, t, NN);                    /* do one x-form */
    dcl A(3,0:5) flt
    dcl N(3) flt, NN(3) flt;
    dcl t flt;

    it = mod (fixed (t), 6);
    it1 = mod (it + 1, 6);
    do i = 1 to 3;
       NN(i) = N(i) * (A(i,it) + (A(i,it1) - A(i,it)) * (t - it));
    end;
end x;

r: proc (S, n);

    if n = 0 then do;
                  call putcolor (S, slot);
                  slot = slot + 1;
                  return;
                  end;
    call x (A, S, t, S2);
    call r (S2, n-1);
    call x (B, S, t, S2);
    call r (S2, n-1);
    call x (C, S, t, S2);
    call r (S2, n-1);
end r;
```

CI at 2 B

BB @ 67

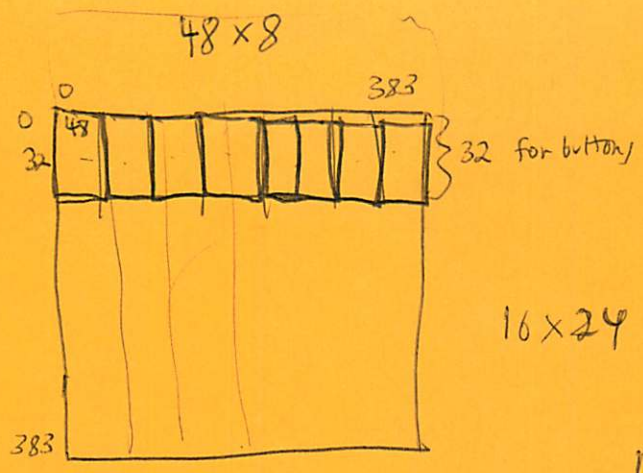
```
hittab: proc (x, y, cx, cy, mode);

        mode = 0;
loop:   call tablet (x, y, z);
        if z <= -2 then go to loop;
        call defcwa (x, y); /* move cursor */
        if z <= -1 then go to loop;
        if y < 48 then do;
                       mode = x/48;
                       return;
                       end;
        cx = x/24;
        cy = (y - 48)/24;
end hittab;
```

3

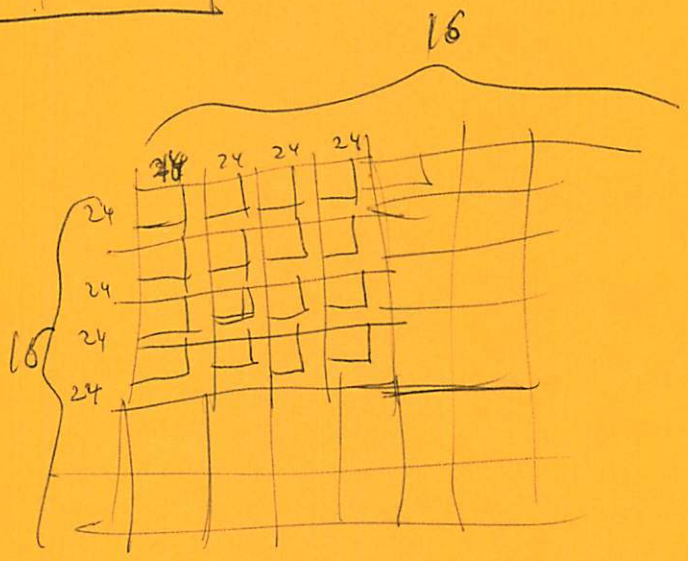
```
putnode: proc (np, cx, cy);
        snode(cx, cy) = np;
        ncol(cy) = ncol(cy) + 1;
        dp = node.description;
        do i = 0 to 15;
           bs.y = cy * 24 + i;
           do j = 1 to 16;
              bs.x = cx + 24 + j - 1;
              bs.val = substr(desc.pattern(i+1), j, 1);
              call bstore(bs);
           end;
        end;
```

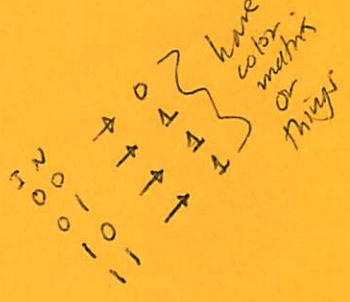
```
zapnode: proc (cx, cy);
        snode(cx, cy) = null();
        ncol(cy) = ncol(cy) - 1;

        end;
```

```
create_node:
        call ask ("Type of node?", node_type);
        if node_type="?" then go to create_help;
        do dp = descriptions repeat desc.next while (dp ^=null());
          if desc.name = node_type then go to make_it;
        end;
        call ioa("Unknown node type, type ? for list.");
        go to create_node;
create_help: do dp = descriptions repeat desc.next while (dp^=null());
          call ioa ("^a", desc.name);
        end;
        go to main_loop;
make_it:  call allocn( desc.inargs * 8+8, np);
          node.next= current -> desc.nodelist;
          current -> desc.nodelist = np;
          node.refcount = 0;
          node.action = desc.action;
          node.description =dp;
          do i = 1 to desc.inargs;
              node.narg(i).nodep = null();
              node.narg(i).index = 0;
          end;
          go to create_action (desc.special_action);
create_action (0);
create_hit: call ioa ("Where do you want it?");
        call hittab (x,y, cx,cy, mode);
        if mode ^=0 then go to create_hit;
        if sonde(cx,cy)^=null() then go to create_hit;
        call putnode (np, cx,cy);
        go to main_loop;
```



$16 \times 24 = 64 \times 6 = 384$





3 planes

1) mode pictures
2) interconnections
3) other

edit mode

> create node

» type from keyboard

it can be :

VALUE or # 2.6
* TIME
* BEFORE [n]
* AFTER [n]
* BETWEEN [n m]
* EVERY [n m]
COLOR {go into a color editor mode}
* POINT [n m] {or a tablet hit}
DRAW {go into draw mode w/ erase key}
* if no args these are regular nodes

INNODE name
OUTNODE name
any predefined node
any user defined and loaded node

> delete node
> link node point, type selector, point, type selector
> unlink node point, point
> display parent point at item to see inputs of, or at the terminal node to see where it goes
> copy node point, point
~~> [illegible]~~
> display child

run mode

> define line start end Δt
> put probe point
> tell about point
> run
> ~~continue~~ reset
> debug run
> single step

system mode

> edit mode
> run mode
> load name
> save name or all
> list user nodes

Editting

create node :
- input arg – name
- output arg – name
  } for user node definition
- time — step functions (time)
- screen
- value
- point
- color

other bifs — many

user defined — will be read from disk if not in core
← can be drawn on

delete node : all or point

copy node : point, then put it

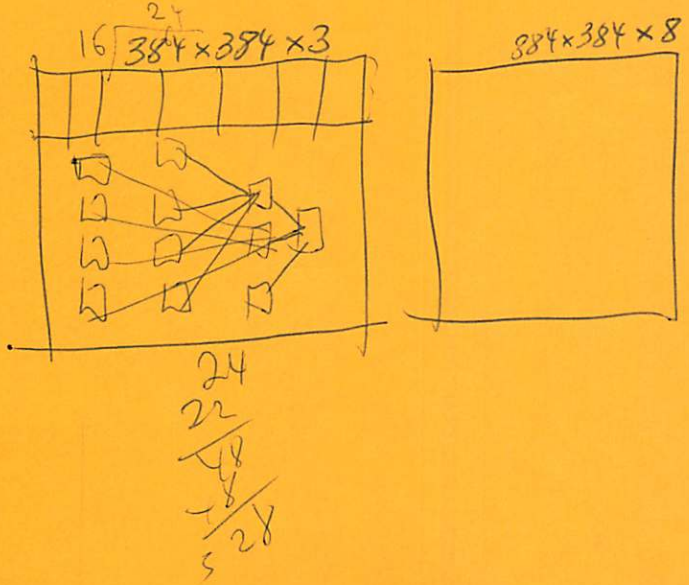
read nodes, write nodes : e.g. write this as

connect node : from → to , if more than one connector it asks

disconnect

redisplay : sources of node x
users of node x

run :
- run — display all screens
- probe — display a selected link



2724
– 6K
18K 208K
11 planes max
+ CML stuff

copynode (p1, p2)
zapnode (p1)
displaynode (given a starting node)
hit (−1 ... −7 or 1 ... 7 or 10 ... 538 squares on grid)
  super command function x, y of a node (or a node ptr)
whref (node, no, refnode)
  reads writes I/O for networks and constant values

etc, etc.

editor must maintain refcounts, or they can be recovered right before a run

7 buttons at each level

edit
1) create node — prompts for type, name, or values
2) delete node — puts node on screen and point
3) copy node — point, then point
4) connect — " "
5) disconnect — " "
6) redisplay — point, if terminal it gives where used else gives all sources
7) ~~[illegible]~~ — go to run mode

define
1) write ws to disk
2) read ws from disk
3) draw ideogram for current ws
4) list args for current ws
5) change current ws to point or type name

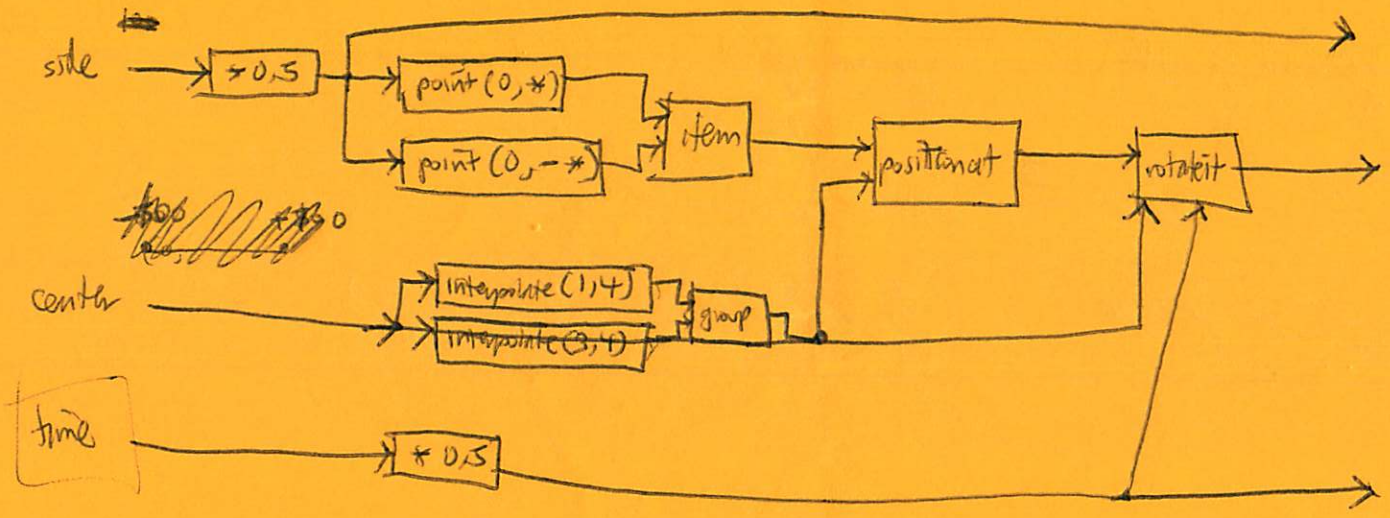
run
1) define time scale
2) set time
3) run
4) step one
5) position probe

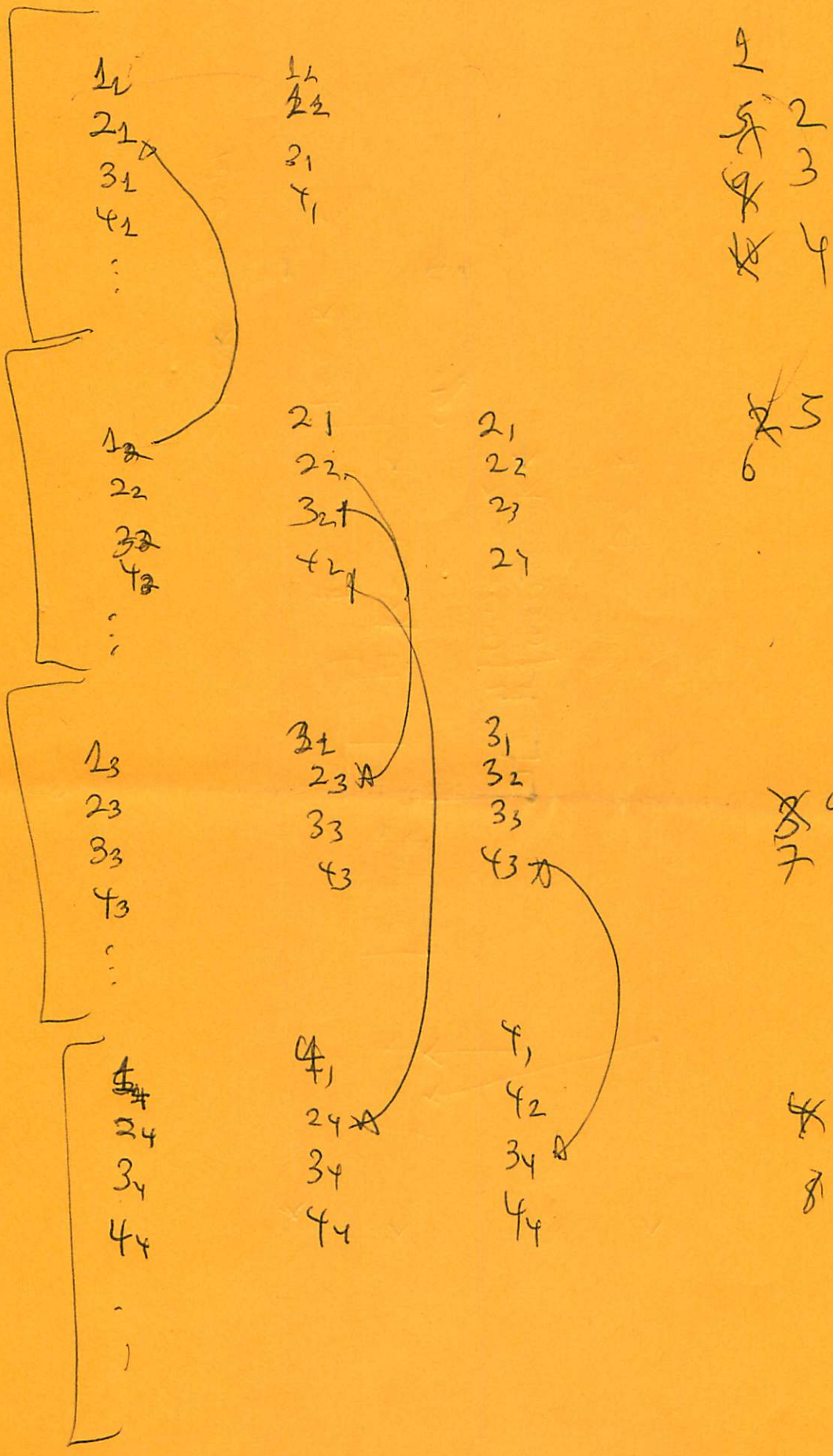
side effects of create

inputarg
outputarg
} must update current ws info

line → positionit → rotateit →

valve → *0,5





Data Format

node

- next ptr
- refcount f.b.(7)
- action f.b.(7) = type of node
- value ptr — -1 user defined
- description ptr = ptr to value w/ refcount = refcount+1
- nargs fixed bin
- narg (1)
  - nodep ptr
  - groupp ptr
  - value ptr
  - index fixed bin

desc

- next ptr
- name char(8)
- action fixed bin(7)
- totargs fixed bin(7)
- inargs fixed bin(7)
- outargs fixed bin(7)
- nodelist ptr
- special_actions
  - create fixed bin(7)
  - delete fixed bin(7)
- args (1)
  - name char(8)
  - nodep ptr — null if inarg

Display Info

- next ptr
- nodep ptr
- listid fixed bin — or 8
- nlines fixed bin
- linerd(#) fixed bin — or 8
- position
  - x fixed bin
  - y fixed bin

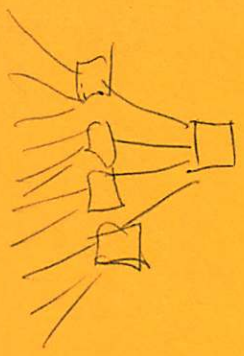
?

X6061
Get TIP

entities

I) [Valve — float — just real point
Point — float — float

II) Color — rgb or ihs — 3 floatnums



III) Graphic Item

a) flood point — 8 bytes
b) color — 12 bytes
c) line list - or - rectangle — 2 bytes + 8n bytes

line = ~~8~~ bytes 38
square = ~~8~~ bytes 54
20 sides = ~~8~~ bytes 182

IV) ~~Rectangle~~ Group

V) Screen

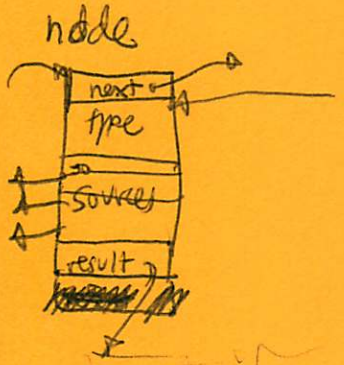
~~VI~~

operators

+ - * / ** sqrt sin cos atan exp log
> < = >= <= ≠ & | ^ Trigger (JK-FF)

merge gi, gi → gi
color gi, color → gi
scale gi, pt, val, val → gi
rotate gi, pt, val → gi
onpath gi, gi, val, val → gi
apply network, group → group
~~extract~~
delay line

extractions e.g. get x
get red
get saturation
etc



scan for eval elements until all evaled