LA-UR -89-3679

CONF-8911170--1

DEC 0 1 1989

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--89-3679

DE90 003393

TITLE A 3-D MEASUREMENT SYSTEM USING OBJECT-ORIENTED FORTH

AUTHOR(S) Kenneth B. Butterfield

SUBMITTED TO FORML Conference
Asilomar Conference Center
Pacific Grove, California

November 24-26, 1989

A 3-D MEASUREMENT SYSTEM USING OBJECT-ORIENTED FORTH

Kenneth B. Butterfield

ABSTRACT

Discussed is a system for storing 3-D measurements of points that relates the coordinate system of the measurement device to the global coordinate system. The program described here uses object-oriented FORTH to store the measured points as sons of the measuring device location. Conversion of local coordinates to absolute coordinates is performed by passing messages to the point objects. Modifications to the object-oriented FORTH system are also described.

The system described here uses 3-D spatial information obtained from a measuring device such as a transit to determine a point in a global coordinate system. The global system might be the US Geological Survey coordinates or a specific corner of a room. Most measuring devices give information relative to their current location. To find the global position for a local measurement, a Galilean transformation, consisting of a rotation and translation, must be performed. Any system for locating points in space must determine the location of the measuring device. To find the location of the measuring device, measurements must be obtained for at least two reference points. Alternatively, one measurement of a reference point plus a compass bearing could be used. All measurements are stored as objects. The location of the measuring device is also stored as an object. Measurements from a particular device location are stored as descendants of the measurement location.

The object-oriented FORTH system used here is basically the system described by Rick Hoselton.[1] This shell provides lists of dependent objects and uses standard postfix syntax. I have modified it to allow nesting of messages and to make the system more robust. For nesting messages, I store the value of 'OBJECT before changes, and I restore the value after the method has been completed (see ACT in the listing). Hoselton's system was subject to complete system failure with just simple operator mistakes. For instance, invoking a method without first specifying a valid object would have the system search a non-existent linked list. Usually, the system would never recover. To prevent this type of error, I added an object tag field to the structure of an object. ACTION checks the tag field to be sure that a valid object is present before searching the linked list. A similar tag was added to the method data structure and is used when creating new methods to check whether a unique method name already exists. This solves Hoselton's problem of methods that could not have the same name as previously defined FORTH words. These three modifications greatly enhance the usability of the object-oriented shell by eliminating many system crashes during development and by allowing one method to call a previously defined method while still remembering its own object.

The data structures for storing points and transit locations are shown in the listings. These are similar in that each contains x, y, and z data cells. Points inherit this structure from the parent object. In addition, points have an extra data cell containing a link to previous uses of the same point name. Typically this list contains all measurements of a known reference, and it is used when locating the position of the transit. The actual measurements of a point are stored in the x, y, and z cells. A method is invoked to determine the absolute coordinates of the point by transforming the local coordinates using the absolute coordinates of the father object, the station. Stations have two additional fields that are not required by point objects. These fields are the rotation angle and a flag indicating that the location is valid. Measurements are made relative to an arbitrary azimuth, and the actual azimuth must be determined as part of locating a station. Methods are provided to find the absolute position of a point and to locate a station given two or more references.

One advantage of the object-oriented paradigm is that new objects can be defined that store only the object numbers for specific points, and these objects will inherit their spatial locations from their fathers objects. For example, a triangle can be defined as having three cells. An instance of a triangle would be defined as P1 P2 P3 TRIANGLE T1, where P1, P2, and P3 are all point objects. Triangle T1 can be queried for its location using a method that first finds the vertex (P1) and then invokes another method to print the absolute position of the vertex. All three vertices can be printed because I have modified the shell to allow for nested methods. The original shell was able to print the location of the first vertex, but it was unable to find the second vertex because it lost the object (T1) when invoking the nested method.

The object-oriented FORTH shell has proved to be useful for storing transit information; the ability to calculate the absolute coordinates by passing a message to the point makes the system very easy to use. However, there are improvements that I would like to make in the future. I found developing the 3-D measurement system to be very difficult until I modified it to include the object and message tag fields. Now program development is much easier. In the future, I would like to see a simplified means of inheriting an object structure. The defining message for my STATION object adds two more fields than the POINT defining the OBJECT: message, yet the whole structure has to be replicated. I like the 'feel' of the object-oriented programming environment and the way that it enters into the FORTH language. It provides a clean way to associate related data while keeping the FORTH postfix syntax.

REFERENCE

1. R. Hoselton, "F83 Object-Oriented FORTH," FORTH Dimensions, Vol. X, 2 (July/August 1988).

```
\ OBJECT.SEQ  From Forth Dimensions, Volume 10, number 2  by Rick Hoselton
\ and modified by Ken Butterfield to
\    1) allow nesting of messages by stacking 'OBJECT
\    2) add OBJECT tag and message tag fields to make system more robust.

comment:

        Some object-oriented words slightly modified by Tom Zimmer
        for use in F-PC.


        OFFSET  #BYTES  METHOD format
        0       2       next older brother METHOD pointer
        2       2       MESSAGE number
        4       2       5AA5 stored as method marker
        6       n       method's code


        OFFSET  #BYTES  OBJECT format
        0       2       father OBJECT address
        2       2       youngest son OBJECT address + 4
        4       2       next older brother OBJECT address + 4
        6       2       youngest METHOD address
        8       2       $A55A stored as object marker
       10       n       optional local data

comment;

only forth also definitions

anew objectstuff

code (action)      ( obj msg --- )
                pop ax
                pop bx
                add bx, # 6
                begin
                        mov bx, 0 [bx]
                        cmp ax, 2 [bx]
             0= until
                add bx, # 6
                mov ax, bx
                jmp ax          end-code



: action        ( obj msg --- )
                over
                8 + @ 42330 = ( test for valid object )
                if
                   (action)
                else abort" unknown object"
                then
                ;

variable 'msg
variable 'object
```

```
: act            ( pfa msg --- )
                 2dup 'msg @ !
                 'object @ >r ( save old object )
                 'object ! action
                 r> 'object ! ( restore old object ) ;

: me             ( --- ?? )
                 'object @ ;

: >object        ( rel-addr --- addr )
                 me + ;

: >super         ( rel-addr --- addr )
                 me @ + ;

: link,          ( addr --- )
                 here over @ , swap ! ;

: object>        ( --- )
                 'object link,
                 0 ,
                 2 >super   link,
                 6 >super @   ,
                 42330 , ;

: object?        ( obj ... obj/f)
                 dup 8 + @
                 42330 = not if
                   drop 0
                 then ;

create master
                 master 'object !
                 object>
                 2 >object 6 erase

: (method)       ( --- msg )
                 create here does> act ;

: ?create        ( --- msg )
                 >in @   defined
                 if >body dup 4 + @
                         23205 =
                 else false
                 then
                 if      nip
                 else    drop >in ! (method)
                 then    ;

: (method:)      ( --- )
                 ?create
                 6 >object  link, , 23205 ,
                 ,JUMP
                 'NEST HERE - HERE 2- !  \ link into JUMP the addr of next
                 XHERE PARAGRAPH +
                 DUP XDPSEG !
```

```
                XSEG @ - ,
                XDP OFF
                !csp ] ;

(method:) anchor ." I don't understand" ;

' anchor >body 2+ 'msg !

(method:) method:       ( --- )
                (method:) ;

master method: object:  ( --- )
                create  object> ;

: .method       ( link --- )
                cr dup 6 u.r  dup @ 6 u.r
                2+ @ dup 6 u.r  2 spaces body> >name .id ;

master method: .methods ( --- )
                base @ hex  6 >object
                begin   @ ?dup
                while   dup .method
                repeat  base   ;

: .me           ( n --- )
                cr spaces me body> >name .id ;

master method: me. ( n obj ... )
                .me ;

master method: (.sons)  ( n --- )
                dup .me
                4 +
                2 >object
                begin   @ dup
                while   2dup 4 - (.sons)
                repeat  2drop
                ;

master method: .sons    ( --- )
                0  me (.sons) ;

master method: .one     ( --- )
                4 .me ;
```

```
\ TRANSIT DATA STRUCTURES                               06/28/89 10:34:12.76
\      POINT is the master class.
\      TRANSIT is a son of point.
\      Stations are sons of transit and inherit POINT structure + more.
\      Each point measured from a station is a son of that station
\            and inherits point structure
\            plus an UNCLE which is a link to any previous point using
\            the same name.

DECIMAL
MASTER OBJECT: POINT

\ methods to fetch point variables to floating point stack
POINT METHOD: Z 10 >OBJECT F@ ;  ( OBJ ... |F: ... FN )
POINT METHOD: Y 18 >OBJECT F@ ;  ( OBJ ... |F: ... FN )
POINT METHOD: X 26 >OBJECT F@ ;  ( OBJ ... |F: ... FN )

POINT METHOD: POSITION. ( OBJ ... ) \ print position of a specified object
   ME X 3 11 f.R ME Y 3 11 f.R ME Z 3 11 f.R ;

POINT METHOD: POSITION! ( OBJ ... |F: X Y Z ...  ) \ assign  position
   10 >OBJECT F! 18 >OBJECT F! 26 >OBJECT F!
;

POINT METHOD: POSITION@ ( OBJ ... |F:  ... X Y Z )

POINT METHOD: OBJECT: ( OBJ ... |F: X Y Z ... )
\ define point using value on FP stack for initialization
   CREATE OBJECT>
   f,    \ 10 >OBJECT IS Z VALUE
   f,    \ 18 >OBJECT IS Y VALUE
   f,    \ 26 >OBJECT IS X VALUE
;

f0.0 f0.0 f0.0 POINT OBJECT: TRANSIT  f0.0 f, -1 ,
( TRANSIT looks like a station )
( and can be used to store reference points. )

TRANSIT  METHOD: STATION: ( OBJ ... )
  CREATE OBJECT>
  F0.0 F, \ 10 >OBJECT IS Z VALUE
  F0.0 F, \ 18 >OBJECT IS Y VALUE
  F0.0 F, \ 26 >OBJECT IS X VALUE
  F0.0 F, \ 34 >OBJECT IS ALPHA
   0 ,    \ 42 >OBJECT IS FLAG
;

TRANSIT METHOD:  STATION! ( F: ALPHA X Y Z ... ) \ ASSIGN A STATION VALUE
   ME POSITION! 34 >OBJECT F  TRUE 42 >OBJECT !
;

TRANSIT METHOD: ALPHA ( OBJ ... |F: ... FN )
   34 >OBJECT F@ ;

TRANSIT METHOD: FLAG ( OBJ ... N ) \ fetch flag to stack
   42 >OBJECT @ ;
```

```
TRANSIT METHOD: RP. ( OBJ ... ) \ print relative position
   ME X 3 11 f.R ME Y 3 11 f.R ME Z 3 11 f.R ;

TRANSIT METHOD: POINT: ( OBJ ... |F: X Y Z ... ) \ define a transit point
   >in @ defined
   if >body object? else drop 0 then >r
   >in !
   CREATE OBJECT>
   f,    \ 10 >OBJECT IS Z VALUE
   f,    \ 18 >OBJECT IS Y VALUE
   f,    \ 26 >OBJECT IS X VALUE
   r> ,  \ 34 >object is pointer to previous use of this name (uncle)
;

TRANSIT METHOD: GET_STATION ME  @ ; ( OBJ ... OBJ ) \ fetch related station object

TRANSIT METHOD: UNCLE ( OBJ ... OBJ )
   34 >OBJECT  @
;

: DOT ( F: A B X Y ... FDOT ) \ Calculate dot product of ( a b ) * (x y)
   FROT F* FROT FROT F* F+
;

TRANSIT METHOD: AP ( OBJ ... |F: ...X Y Z )
\ calculate absolute position of point
   ME GET_STATION FLAG IF \ has station been located?
      ME GET_STATION ALPHA   ( r: alpha)       \ find rotation and translation
      FDUP
      FSIN FSWAP FCOS        ( SIN, COS ) \ rotation in X
      FOVER FOVER            ( SIN COS SIN COS )
      ME X ME Y              ( SIN COS SIN COS X Y )
      DOT                    ( SIN COS X' )
      ME GET_STATION X F+    \ TRANSLATION IN X
      FROT FROT              ( APX SIN COS )  \ rotation in Y
      FNEGATE ME Y ME X DOT  ( APX Y' )
      ME GET_STATION Y F+    ( APX APY)  \ translation in Y
      ME Z ME GET_STATION Z f+ ( APX APY APZ )  \ Z is translated, no rotation
   ELSE
      ME GET_STATION .ONE ."  NOT LOCATED"
      ABORT
   THEN
;

TRANSIT METHOD: me. ( N OBJ ... ) \ print absolute position with indentation N
   .ME ME AP
   FROT 3 11 f.R fSWAP 3 11 f.R 3 11 f.R
;

TRANSIT method: POSITION. ( OBJ ... ) \ Print absolute position
   4 ME ME. ;

TRANSIT METHOD: .ALL ( OBJ ... \ PRINT ALL namesakes ( uncles )
           ME .ONE ME
           BEGIN
               34 + @ ?DUP
```

```
          WHILE
              dup Position.
          REPEAT
;

TRANSIT METHOD: .TREE  ( N --- )
                me .one
                2 >object
                begin   @ dup
                while   dup 4 - .all
                repeat  drop
                ;


\ TRANSIT DATA     06/28/89 13:29:51.86

TRANSIT STATION: ST1

10.0e0 15.0e0 20.0e0 ST1 POINT: P1
10.0e0 30.0e0 5.0e0  ST1 POINT: P2
10.0e0 15.0e0 20.0e0 ST1 POINT: P3

TRANSIT STATION: ST2

10.0e0 20.0e0  50.0e0 ST2 POINT: P4
15.e0 25e0 10e0 ST2 POINT: P1
25e0 40e0 -5e0 ST2 POINT: P2

0e0 10e0 10e0 10e0 ST1 STATION!

\ Station two has not been located in this example.
\ Station one is just offset from the global origin and has no rotation.
```