Visualization Math Library User's Guide

for

VIZMATH Rev. 1.00

Bob Haimes & Doug Sondak

April 5, 1995

License

Acknowledgements

This is work primarily done at United Technologies Research Center in support of **RAVEN**. **RAVEN** has a standard Motif GUI built upon **Visual3**. Other added functionallity includes a field calculator. The routines documented in this guide are the math functions used in that calculator. Doug Sondak (UTRC), with assistance from Rodger Davis (UTRC), are the authors of this library.

Contents

1 Introduction

The library documented in this manual has been designed to be used with **Visual3**. The input to many of the routines are the basic internal structures that have been exposed via **Visual3**'s advanced programming constructs. In particular, see the routine V3_GETSTRUC in the Advanced Programmer's Guide for **Visual3**.

This library can also be used with the client-side of **pV3**. This is because the data structures that are used are the same in both packages. If the domain has been partitioned, some of the results will be incorrect at the partition boundaries. This can be fixed up by explicitly moving the boundary data between the partitions. The 'patching' can be performed because all node information has been generated by sums, and the number of contributions for the node is also returned. For partition boundaries (or places that have been *equivalenced*), multiply the value by the number of times the node has been touched, then sum the portions from all contributions (to the node) and redivide by the sum of the number of touches.

In describing the arguments of the routines in this manual, the following notation has been used. The variable name is first followed by 'i', 'o' or 'w', indicating input, output or work-space, respectively, showing whether or not the subroutine sets the variable. The variable is then followed by an expression indicating the variable type (I=integer or R=real) and its dimensionality (e.g. R(2,MNODE) indicates a two-dimensional real array with the first dimension being 2 and the second MNODE, where MNODE is an integer). This is FORTRANs natural notation.

2 The Use of Pointers with FORTRAN

Some of the data that gets returned from V3_GETSTRUC (and pV_GETSTRUC) is in the form of pointers to blocks of memory. There are two modes that the pointers may be used:

- %val

  A mechanism exists on all major workstation's FORTRAN (SGI, DEC, IBM, HP and SUN) to allow the pointer to be passed to a SUBROUTINE or FUNCTION and then have the memory treated as a normally declared vector or array. This is done by the VAX extension '%val(pointer)' used in the CALL or function invokation. When the pointer is passed to the sub-program by 'value', it is equivalent to passing a variable by 'reference' (the FORTRAN method). That is, in both cases, the address of the memory of interest is placed in the stack!

  In this case, the pointers are treated as INTEGER variables. This causes a problem in portability. The pointers must be treated as INTEGER*4 variables on all 32 bit machines. On 64 bit machines (DEC ALPHAs and SGIs running IRIX 6.0, or higher, in native mode) the pointers are INTEGER*8.

- POINTER statment

  The POINTER statement supported by most FORTRANs removes the portability problem. This syntax allows the coupling of a pointer with an array.

Using either mechanism, sophisticated **Visual3** and **pV3** enhancements may be performed using FORTRAN. To complete this picture, three additional entries to the librarys have been added to allow the FORTRAN programmer to allocate and free up blocks of memory. See **pV3**'s Advanced Programmer's Guide for the appropriate **pV3** routines.

2.1 MALLOCV3

PTR = MALLOCV3(NBYTES)

This function is equivalent to the C routine 'malloc'. It allocates a block of memory and returns the pointer to the block. On 64 bit machines, this function may need to be declared INTEGER*8.

NBYTES:i: I	The number of bytes to allocate.
PTR:o: I	The address of the block (0 is an error indicator). This must be declared INTEGER*8 or POINTER on 64 bit machines.

2.2 FREEV3

FREEV3(PTR)

This function is equivalent to the C routine 'free'. It deallocates a block of memory. NOTE: Only free up blocks of memory that YOU allocate!

PTR:i: I	The address of the block.

2.3 REALLOCV3

PTRN = REALLOCV3(PTRO, NBYTES)

This function is equivalent to the C routine 'realloc'. It re-allocates a block of memory and returns the pointer to the new block. On 64 bit machines, this function may need to be declared INTEGER*8.

PTRO:i: I	The address of the block. This must be declared INTEGER*8 or POINTER on 64 bit machines.
NBYTES:i: I	The number of bytes to allocate.
PTRN:o: I	The address of the new block (0 is an error indicator). This must be declared INTEGER*8 or POINTER on 64 bit machines.

3 Structures

3.1 3D Data Structures

In general, this data is the information returned from the programmer-supplied routines, though the connectivity information is internally generated. The structures are exposed so that multiple copies of the data need not be kept.

3.1.1 Node based

The following data is based on the 3D node space (NNODE is the number of nodes in the domain):

XYZ: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
S: R(NNODE)	Scalar function values.
V: R(3,NNODE)	Vector function values (Vx, Vy, Vz).
IBLANK: I(*)	IBLANK values for structured blocks. The length is the number of nodes in the structured blocks. For a case that is only structured blocks the length is NNODE. If the pointer to this array is 0, then no IBLANKing is active.

3.1.2 Cell based

The following data is based on the 3D cell space:

CEL1: I(4,KCEL1)	Node pointers for tetrahedral cells.
CEL2: I(5,KCEL2)	Node pointers for pyramid cells.
CEL3: I(6,KCEL3)	Node pointers for prism cells.
CEL4: I(8,KCEL4)	Node pointers for hexahedral cells.
NPTET: I(8,KNPTET)	Poly-Tetrahedra strip header:

NPTET(1,n) = the pointer to the end of the strip n, i.e. it points to the last entry in PTET for the poly-tetrahedral strip

NPTET(2,n) = the first node in the poly-tetrahedra

NPTET(3,n) = the second node in the poly-tetrahedra

NPTET(4,n) = the third node in the poly-tetrahedra

NPTET(5,n) = the fourth node in the poly-tetrahedra

NPTET(6,n) = the last node in the poly-tetrahedra

NPTET(7,n) = first connection (face 1 of the first tetra)

NPTET(8,n) = last connection (face 2 of the last tetra)

PTET: I(KPTET) — The rest of each poly-tetrahedra, 1 node per cell. The first and last node numbers in PTET are replaced by a flag to indicate the start and end of the strip. The flag is the negative of the strip number so that if you are in a strip, you can find which strip it is.

BLOCKS: I(6,KNBLCK) — Structured block definitions:

BLOCKS(1,m) = NI

BLOCKS(2,m) = NJ

BLOCKS(3,m) = NK

BLOCKS(4,m) = cell number that terminates the block

BLOCKS(5,m) = node number that terminates the block

BLOCKS(6,m) = index into CBLOCK that starts the connectivity table for the block.

4 Supplied Routines

4.1 V3Curl

**V3CURL(CEL1,KCEL1,CEL2,KCEL2,CEL3,KCEL3,CEL4,KCEL4,
NPTET,KNPTET,PTET,BLOCKS,KNBLCK,IBLANK,IBLNKF,
XYZ,V,NNODE,CURL,SWORK)**

Returns the curl of the vector field.

CEL1:i: I(4,KCEL1)	Node pointers for tetrahedral cells.
KCEL1:i: I	Number of tetrahedral cells.
CEL2:i: I(5,KCEL2)	Node pointers for pyramids.
KCEL2:i: I	Number of pyramid cells.
CEL3:i: I(6,KCEL3)	Node pointers for prism cells.
KCEL3:i: I	Number of prism cells.
CEL4:i: I(8,KCEL4)	Node pointers for disjoint hexahedra.
KCEL4:i: I	Number of hexahedral cells.
NPTET:i: I(8,KNPTET)	Poly-Tetrahedral strip headers.
KNPTET:i: I	Number of poly-tetrahedral strips.
PTET:i: I(*)	The poly-tetrahedral strips.
BLOCKS:i: I(6,KNBLCK)	Structured block definitions.
KNBLCK:i: I	The number of structured blocks.
IBLANK:i: I(*)	IBLANK values for structured blocks.
IBLNKF:i: I	0 – No IBLANKing, 1 – IBLANKing.
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
V:i: R(3,NNODE)	Vector function values (Vx, Vy, Vz).
NNODE:i: I	Number of nodes.
CURL:o: R(3,NNODE)	Curl of V.
SWORK:w: R(NNODE)	Scalar field work array (returns the number of times the node was touched).

4.2 V3Divg

V3DIVG(CEL1,KCEL1,CEL2,KCEL2,CEL3,KCEL3,CEL4,KCEL4, NPTET,KNPTET,PTET,BLOCKS,KNBLCK,IBLANK,IBLNKF, XYZ,V,NNODE,DIVG,SWORK)

Returns the divergence of the vector field.

CEL1:i: I(4,KCEL1)	Node pointers for tetrahedral cells.
KCEL1:i: I	Number of tetrahedral cells.
CEL2:i: I(5,KCEL2)	Node pointers for pyramids.
KCEL2:i: I	Number of pyramid cells.
CEL3:i: I(6,KCEL3)	Node pointers for prism cells.
KCEL3:i: I	Number of prism cells.
CEL4:i: I(8,KCEL4)	Node pointers for disjoint hexahedra.
KCEL4:i: I	Number of hexahedral cells.
NPTET:i: I(8,KNPTET)	Poly-Tetrahedral strip headers.
KNPTET:i: I	Number of poly-tetrahedral strips.
PTET:i: I(*)	The poly-tetrahedral strips.
BLOCKS:i: I(6,KNBLCK)	Structured block definitions.
KNBLCK:i: I	The number of structured blocks.
IBLANK:i: I(*)	IBLANK values for structured blocks.
IBLNKF:i: I	0 – No IBLANKing, 1 – IBLANKing.
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
V:i: R(3,NNODE)	Vector function values (Vx, Vy, Vz).
NNODE:i: I	Number of nodes.
DIVG:o: R(NNODE)	Divergence of V.
SWORK:w: R(NNODE)	Scalar field work array (returns the number of times the node was touched).

4.3 V3Grad

**V3GRAD(CEL1,KCEL1,CEL2,KCEL2,CEL3,KCEL3,CEL4,KCEL4,
NPTET,KNPTET,PTET,BLOCKS,KNBLCK,IBLANK,IBLNKF,
XYZ,S,NNODE,GRAD,SWORK)**

Returns the pseudo-gradient of the scalar field.

CEL1:i: I(4,KCEL1)	Node pointers for tetrahedral cells.
KCEL1:i: I	Number of tetrahedral cells.
CEL2:i: I(5,KCEL2)	Node pointers for pyramids.
KCEL2:i: I	Number of pyramid cells.
CEL3:i: I(6,KCEL3)	Node pointers for prism cells.
KCEL3:i: I	Number of prism cells.
CEL4:i: I(8,KCEL4)	Node pointers for disjoint hexahedra.
KCEL4:i: I	Number of hexahedral cells.
NPTET:i: I(8,KNPTET)	Poly-Tetrahedral strip headers.
KNPTET:i: I	Number of poly-tetrahedral strips.
PTET:i: I(*)	The poly-tetrahedral strips.
BLOCKS:i: I(6,KNBLCK)	Structured block definitions.
KNBLCK:i: I	The number of structured blocks.
IBLANK:i: I(*)	IBLANK values for structured blocks.
IBLNKF:i: I	0 – No IBLANKing, 1 – IBLANKing.
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
S:i: R(NNODE)	Scalar function values.
NNODE:i: I	Number of nodes.
GRAD:o: R(3,NNODE)	Pseudo-gradient of S.
SWORK:w: R(NNODE)	Scalar field work array (returns the number of times the node was touched).

4.4 V3Lapl

V3LAPL(CEL1,KCEL1,CEL2,KCEL2,CEL3,KCEL3,CEL4,KCEL4, NPTET,KNPTET,PTET,BLOCKS,KNBLCK,IBLANK,IBLNKF, XYZ,S,GRAD,NNODE,LAPL,SWORK)

Returns the Laplacian of the scalar field. Note: V3GRAD must be executed prior to this call to fill GRAD.

CEL1:i: I(4,KCEL1)	Node pointers for tetrahedral cells.
KCEL1:i: I	Number of tetrahedral cells.
CEL2:i: I(5,KCEL2)	Node pointers for pyramids.
KCEL2:i: I	Number of pyramid cells.
CEL3:i: I(6,KCEL3)	Node pointers for prism cells.
KCEL3:i: I	Number of prism cells.
CEL4:i: I(8,KCEL4)	Node pointers for disjoint hexahedra.
KCEL4:i: I	Number of hexahedral cells.
NPTET:i: I(8,KNPTET)	Poly-Tetrahedral strip headers.
KNPTET:i: I	Number of poly-tetrahedral strips.
PTET:i: I(*)	The poly-tetrahedral strips.
BLOCKS:i: I(6,KNBLCK)	Structured block definitions.
KNBLCK:i: I	The number of structured blocks.
IBLANK:i: I(*)	IBLANK values for structured blocks.
IBLNKF:i: I	0 – No IBLANKing, 1 – IBLANKing.
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
S:i: R(NNODE)	Scalar function values.
GRAD:i: R(3,NNODE)	Pseudo-gradient of S as returned by V3GRAD.
NNODE:i: I	Number of nodes.
LAPL:o: R(NNODE)	The Laplacian of S.
SWORK:w: R(NNODE)	Scalar field work array (returns the number of times the node was touched).

5 Portability

5.1 FORTRAN Programming

For the most part FORTRAN source that runs on one workstation implementation of **Visual3** or **pV3** will work on others.

NOTES:

(1) Some FORTRANs have their own MALLOC and FREE calls. For portability it is advised to use **Visual3**'s routines (documented in this manual). This is because the calling sequences differ from machine to machine.

(2) Getting command line arguments is different on HP machines (IGETARG) and the other workstations (GETARG).

5.2 C Programming

Unfortunately **Visual3** is not source code compatible for C across all workstations. This has to do with supporting FORTRAN with the same API.

The C programmer that wishes to have their application running on many platforms should be aware that:

- main program

  On IBM and HP workstations, normal C conventions apply. For DEC and SGI ports the name of the main program must be 'MAIN__' and on all others the name is 'MAIN_'.

- getting command line arguments

  On IBM and HP the normal C mechanism to get command line arguments may be used. On the other vendors machines, the FORTRAN method must be used.

- routine names

  For IBM and HP ports, all entry points are the FORTRAN names in lower-case. On all other platforms, the external entries are lower-case with an underscore ('_') appended to the end.