ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Mello Moraes, nº2231 CEP05508-900 São Paulo SP

Departamento Engenharia Mecatrônica e de Sistemas Mecânicos

1a. Lista de Exercícios – PMR3522

Prazo para entrega: 24 de abril de 2019.

Prof. Dr. Marcos Tsuzuki

O objetivo deste exercício é implementar rotinas que determinam o valor das seguintes propriedades integrais de sólidos poliedrais:

- área
- volume (para cálculo do volume utilize o ponto médio do sólido)
- rotação em relação a um eixo (dados os parâmetros do eixo)
- criação de primitivo barra L

Deve ser entregue um relatório contendo a listagem do programa desenvolvido, uma explicação justificando a codificação do programa, exemplos de execução do programa para cada rotina criada (pelo menos quatro casos para cada um: VOLUME, ROTAÇÃO, BARRA L).

Para isto você deverá utilizar o software USPDesigner. Tudo deverá ser implementado utilizando um compilador de linguagem C. Apesar dos arquivos estarem com a terminação .cpp; todo o programa foi desenvolvido em **linguagem C**.

O USPDesigner possui uma linguagem de programação própria, através desta linguagem é possível definir os modelos sólidos que serão estudados. No conjunto de arquivos fornecido, os arquivos com a terminação ***.cad** são exemplos desta linguagem.

Para executar a rotina que você está programando, criamos o comando **`PROP`**. Para interfacear as suas rotinas com o USPDesigner, você deverá criar uma rotina semelhante a seguinte:

```
#include <stdio.h>
#define __VIRTUAL_MEM
#define __ROV__
#include "mensagem.h"
#include "memvirtu.h"
#include "lowparam.h"
#include "lowmacro.h"
#include "lowsolid.h"
#include "eulerops.h"
#include "vectorop.h"
#include "genfunc_.h"
#include "analise_.h"
#include "mancommd.h"
```

Acima, estão declarados os arquivos de include onde todas as funções estão definidas. Os *defines* acima também estar presente para configurar a memória virtual.

A rotina de interface deve ser algo semelhante ao seguinte algoritmo:

```
void MSD_execManipulatePropriedade(void)
{
    int ip ;
    char onam[30] ;
    float area, volume, x1, x2, y1, y2, z1, z2, th, d1, d2, d3, d4, d5, d6, d7;
                                                    // definição das variáveis locais

    for (ip = 0 ; ip == 0 ; ) {
        switch(optin()) {                 // recupera as ações do comando
            // - Comando que calcula a area - recupera o nome do solido
            case 'a':
                if (1 == sscanf(restbuf, "%s", onam)) {
                    area = MSD_highNamePropriedadeArea(onam) ;
                    ip = 1 ;
                    < imprime o valor da area >
                }
                break ;
            // - Comando que calcula o volume - recupera o nome do solido
            case 'v':
                if (1 == sscanf(restbuf, "%s", onam)) {
                    volume = MSD_highNamePropriedadeVolume(onam) ;
                    ip = 1 ;
                    < imprime o valor do volume >
                }
                break ;
            // - Comando que calcula a rotação em relação a um eixo
            case 'r':
                if (8 == sscanf(restbuf, "%s %f %f %f %f %f %f %f",
                                          onam, &x1, &y1, &z1, &z2, &y2, &z2, &th)) {
                    MSD_highNamePropriedadeRotaciona(onam, x1, y1, z1, x2, y2, z2, th) ;
                    ip = 1 ;
                }
                break ;
            // - Comando para criar o primitivo barra L
            case 'l':
                if (6 == sscanf(restbuf, "%s %f %f %f %f %f",
                                        onam, &d1, &d2, &d3, &d4, &d5)) {
                    MSD_ highCreatePrimitivoL(onam, d1, d2, d3, d4, d5) ;
                    ip = 1 ;
                }
                break ;
        }
        if (ip == 0) {
            printf("-avrl nome do sólido\n") ;
            if (!lineins("? "))
                return ;
        }
    }
}
```

A rotina de interface permitirá que o acionamento das rotinas que você implementar sejam feitos através dos comandos da linguagem do USPDesigner listados abaixo:

```
prop -a casa                # determina a área do sólido casa
prop -v casa                # determina o volume do sólido casa
prop -r casa 0 0 0 0 0 1 90 # rotaciona o sólido casa em relação ao eixo de 90 graus
prop -l casa 10 3 2 5 3     # cria um primitivo L com as dimensões especificadas
```

O eixo de rotação passa pelos pontos P1 (coordenadas x1, y1, z1) e P2 (coordenadas x2, y2, z2).

Para implementar o seu programa é necessário que você tenha acesso aos ponteiros da estrutura interna do sólido. Para, a partir do *nome do sólido*, obter o *ponteiro* de sua estrutura, isto os seguintes passos são necessários:

```
float MSD_highNamePropriedadeArea(char *name)
{
    int sn ;                        // identificador do sólido

    if ((sn = MSD_getSolidIdFromName(name)) == -1) {
        fprintf(stderr, "area: nao encontrou o sólido %s!\n", name) ;
        return(ERROR) ;
    }
    return(MSD_highPropriedadeArea(sn)) ;
}

float MSD_highPropriedadeArea(Id sn)
{
    SPTYPE s ;                      // ponteiro para o sólido

    if ((s = MSD_getSolid(sn)) == SNIL) {
        fprintf(stderr, "area: nao encontrou o sólido %d!\n", sn) ;
        return(0.0) ;
    }
    return(MSD_lowPropriedadeArea(s)) ;
}
```

Estes passos são necessários, pois o USPDesigner está estruturado de maneira a possuir três níveis de programação:

1. nível superior, onde os sólidos são acessados por nomes
2. nível intermediário, onde os sólidos são acessados por identificadores
3. nível inferior, onde os sólidos são acessados pelos seus ponteiros.

Não aconselhamos acessar informações do nível 3 diretamente, a partir do nível 1.

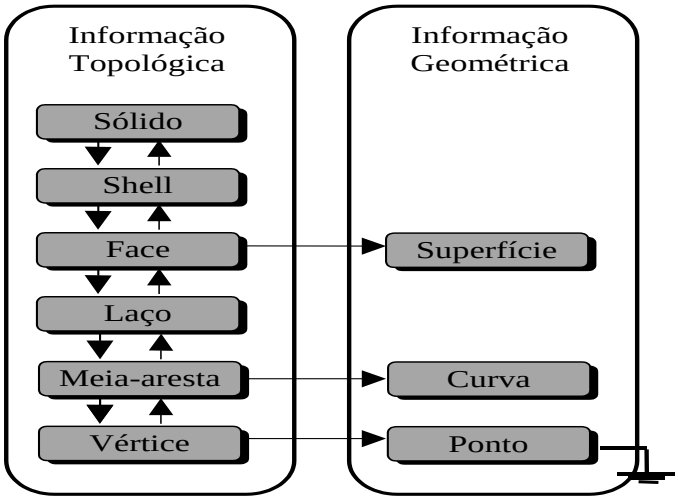


Figure 1. Estrutura de Dados do USPDesigner

Os seguintes tipos podem ser utilizados:

```
SPTYPE      // ponteiro para um sólido
DPTYPE      // ponteiro para um shell
FPTYPE      // ponteiro para uma face
LPTYPE      // ponteiro para um laço
HPTYPE      // ponteiro para uma half-edge
VPTYPE      // ponteiro para um vértice
```

As seguintes funções para manipulação de *sólidos* estão disponíveis:

SolNxt(p): Recupera o próximo sólido relativo ao sólido **p**, na lista ligada.
SolSShells(p) Recupera o primeiro shell da lista ligada ligada de shells do sólido **p**.

As seguintes funções para manipulação de *shells* estão disponíveis:

SheNextD(p): Recupera o próximo shell relativo ao shell **p**, na lista ligada.
SheSFaces(p) Recupera a primeira face da lista ligada ligada de faces do shell **p**.

As seguintes funções para manipulação de *faces* estão disponíveis:

FacNextF(p): Recupera a próxima face relativa a face **p**, na lista ligada.
FacFLOut(p): Recupera o laço externo da face **p**.
FacFLoops(p) Recupera o primeiro laço da lista ligada ligada de laços da face **p**.
FacFeq(p)[0]: coordenada x da equação da face **p** - $a \cdot x + b \cdot y + c \cdot z + d = 0$.
FacFeq(p)[1]: coordenada y da equação da face **p**.
FacFeq(p)[2]: coordenada z da equação da face **p**.
FacFeq(p)[3]: coordenada homogênea da equação da face **p**.

Para as *faces*, é importante observar a diferença entre laço interno e laço externo. A equação da face pode ser tratada como um vetor de quatro elementos - a função **`FacFeq(p)`** retorna um ponteiro para o vetor que contém a equação da face. As seguintes funções para manipulação de *laços* estão disponíveis:

`LooNextL(p)`: Recupera o próximo laço relativo ao laço **`p`**, na lista ligada.
`LooLEdg(p)`: Recupera a primeira halfedge do circuito de arestas do laço **`p`**.

As seguintes funções para manipulação de *halfedges* estão disponíveis:

`HalWLoop(p)`: Recupera o laço associado a halfedge **`p`**.
`HalEdg(p)`: Recupera a areasta associada a halfedge **`p`**.
`HalNxt(p)`: Recupera a próxima halfedge relativa a halfedge **`p`**, no laço associado.
`HalPrv(p)`: Recupera a halfedge anterior relativa a halfedge **`p`**, no laço associado.
`HalVtx(p)`: Recupera o vértice associado a halfedge **`p`**.

As seguintes funções para manipulação de *vértices* estão disponíveis:

`VerVCoord(p)[0]`: coordenada x do vértice **`p`**.
`VerVCoord(p)[1]`: coordenada y do vértice **`p`**.
`VerVCoord(p)[2]`: coordenada z do vértice **`p`**.
`VerVCoord(p)[3]`: coordenada homogênea do vértice **`p`**.

As coordenadas do vértice podem ser tratadas como um vetor de quatro elementos - a função **`VerVCoord(p)`** retorna um ponteiro para o vetor que contém as coordenadas do vértice. Estão disponíveis as seguintes rotinas para manipulação de vetores:

`void vecmult(vector v1, vector v2, matrix m)`: $v_1 = v_2 * m$
`void vecminus(vector v1, vector v2, vector v3)`: $v_1 = v_2 - v_3$
`void vecplus(vector v1, vector v2, vector v3)`: $v_1 = v_2 + v_3$
`void cross(vector v1, vector v2, vector v3)`: $v_1 = v_2 \times v_3$
`float dot(vector v1, vector v2)`: $v_1 \cdot v_2$
`void veccopy(vector v1, vector v2)`: $v_1 = v_2$
`void matzer(matrix m)`: zera a matriz **`m`**.
`void veczer(vector v)`: zera o vetor **`v`**.

Considere a rotina para cálculo de área como exemplo para o seu desenvolvimento.

```
/*** Calculo da area do laco ***/
double MSD_ lowPropriedadeAreaLaco(LPTYPE l)
{
    HPTYPE he ;
    VPTYPE v1 ;
    vector aa, bb, cc, dd, vv1 ;

    veczer(dd) ;
    he = LooLEdg(l) ;
    v1 = HalVtx(he) ;
    he = HalNxt(he) ;
    do {
        veccopy(vv1, VerVCoord(v1)) ;
        vecminus(aa, VerVCoord(HalVtx(he)), vv1) ;
        vecminus(bb, VerVCoord(HalVtx(HalNxt(he))), vv1) ;
        cross(cc, aa, bb) ;
        vecplus(dd, dd, cc) ;
    } while ((he = HalNxt(he)) != LooLEdg(l)) ;
    return(-0.5 * dot(FacFeq(LooLFace(l)), dd)) ;
}
```

Para compilar o seu sistema você deverá utilizar os arquivos **uspdesin.lib** e **ex16bit.cpp** distribuídos.

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Mello Moraes, nº2231 CEP05508-900 São Paulo SP

Departamento Engenharia Mecatrônica e de Sistemas Mecânicos

Como compilar o seu sistema.

Você deve ter recebido um arquivo em formato **zip**, descompacte-o em um diretório de seu computador. Em seguida vários arquivos serão abertos, mas você deverá utilizar em maior detalhe apenas o arquivo listado abaixo:

- **prop.cpp** → arquivo fonte com o esqueleto de programa ilustrado no exercício.

Teste o seu programa com vários modelos de sólidos, coloque as respostas no relatório a ser entregue.

Os arquivos ***.cad** contém exemplos de programas prontos, feitos na linguagem do USPDesigner.

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Mello Moraes, nº2231 CEP05508-900 São Paulo SP

Departamento Engenharia Mecatrônica e de Sistemas Mecânicos

Anexo 1 - Determinando propriedades integrais em sólidos poliedrais

As propriedades integrais de um sólido Q é definido como a integral de volume de uma função f sobre um sólido:

$$I = \int_Q f(x, y, z) \cdot dv \qquad (1)$$

A maioria dos estudos computacionais de integrais múltiplas tratam o problema em que o domínio Q é geometricamente simples mas o integrando f é complexo. Entretanto, na determinação da massa, momento de inércia, etc., nós estamos diante de um problema inverso: a função f geralmente é simples mas o domínio Q geralmente é complexo.

Neste exercício utilizaremos o teorema do divergente que fornece um método alternativo para determinar as propriedades integrais dos sólidos pela integração de seus contornos:

$$\int_Q f(x, y, z) \cdot dv = \int_Q div(\mathbf{g}) \cdot dv = \int_{\partial Q} \mathbf{g} \cdot \mathbf{n} \cdot ds \qquad (2)$$

onde $\mathbf{g}$ é uma função vetorial que satisfaz a condição $f = div(\mathbf{g})$, ∂Q é o contorno de Q, $\mathbf{n}$ é vetor normal unitário ao contorno, e ds é o diferencial de superfície.

Determinando a área do triângulo

A área de um triângulo T com vértices (v_0, v_1, v_2) é igual ao produto vetorial dos dois vetores $\mathbf{r}_1 = (v_0, v_1)$ e $\mathbf{r}_2 = (v_0, v_2)$:

$$Area(T) = 0.5 \cdot \mathbf{r}_1 \times \mathbf{r}_2 \qquad (3)$$

onde o símbolo $\times$ representa o produto vetorial de dois vetores. Desta maneira a área é definida como um vetor que possui uma quantidade e também um orientação. A área de uma região plana fechada arbitrária R é dada por:

$$Area(R) = 0.5 \cdot \oint_{\partial R} \mathbf{r} \times \mathbf{dl} \qquad (4)$$

onde ∂R é o contorno da região R e $\mathbf{dl}$ é o diferencial do vetor tangente do contorno a área do polígono planar $R = \{v_1, v_2, \Lambda, v_n\}$ com n vértices $v_i = (x_i, y_i, z_i)$ é dada por:

$$\begin{aligned} Area(R) &= 0.5 \oint_{\partial R} \mathbf{r} \times \mathbf{dl} \\ &= 0.5 \cdot \sum_{i=1}^{n-2} (v_1, v_{i+1}) \times (v_1, v_{i+2}) \end{aligned} \quad (5)$$

Utilizando v_1 como um centro de projeção, o polígono plano foi dividido seqüencialmente em vários triângulos formados por v_1 e cada uma das diferentes arestas do polígono em seqüência. A área do polígono é a soma vetorial das áreas de todos os triângulos como definido na equação (3). A equação (5) vale tanto para polígonos convexos e para polígonos concâvos.

A propriedade integral de um poliedro Q pode ser descrita por

$$I = \int_Q f(x, y, z) \cdot dv \quad (6)$$

Onde a função f é um polinômio. Com uma transformação linear definida como

$$\begin{aligned} x &= g_x(u, v, w) \\ y &= g_y(u, v, w) \\ z &= g_z(u, v, w) \end{aligned} \quad (7)$$

A equação (6) ficará

$$I = \iiint_Q f(g_x, g_y, g_z) \cdot |J| \cdot du \cdot dv \cdot dw \quad (8)$$

Onde o Jacobiano J vale

$$J = \begin{vmatrix} \frac{\partial g_x}{\partial u} & \frac{\partial g_x}{\partial v} & \frac{\partial g_x}{\partial w} \\ \frac{\partial g_y}{\partial u} & \frac{\partial g_y}{\partial v} & \frac{\partial g_y}{\partial w} \\ \frac{\partial g_z}{\partial u} & \frac{\partial g_z}{\partial v} & \frac{\partial g_z}{\partial w} \end{vmatrix} \quad (9)$$

O integrando f na equação (6) é um polinômio que pode ser genericamente representado por

$$f(x, y, z) = \sum_{n_2, n_2, n_3} x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \quad (10)$$

onde n_1, n_2 e n_3 são inteiros. Para calcular a integral descrita na equação (6), podemos focar a nossa atenção apenas em um termo

$$I = \iiint_Q x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \cdot dx \cdot dy \cdot dz \qquad (11)$$

Vamos apenas analisar o caso em que Q é um 3D simplex, i.e., um tetraedro, com quatro vértices (v_0, v_1, v_2, v_3) com o vértice v_0 localizado na origem. As coordenadas dos vértices são

$$\begin{aligned} v_0 &= (0,0,0) \\ v_1 &= (x_1, y_1, z_1) \\ v_2 &= (x_2, y_2, z_2) \\ v_3 &= (x_3, y_3, z_3) \end{aligned} \qquad (12)$$

Define-se a transformação linear $\mathbf{T}$ como

$$\mathbf{T} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \qquad (13)$$

que relaciona o antigo sistema de coordenadas (x, y, z) com o novo sistema de coordenadas (X, Y, Z) por

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad (14)$$

Com esta transformação, o tetraedro $Q = (v_0, v_1, v_2, v_3)$ da equação (12) é transformado em um tetraedro unitário ortogonal $W = (v'_0, v'_1, v'_2, v'_3)$ com coordenadas

$$\begin{aligned} v'_0 &= (0,0,0) \\ v'_1 &= (1,0,0) \\ v'_2 &= (0,1,0) \\ v'_3 &= (0,0,1) \end{aligned} \qquad (15)$$

Baseando-se na transformação exibida na equação (14), a integral na equação (8) ficará

$$
\begin{aligned}
I &= \iiint_{Q} x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \cdot dx \cdot dy \cdot dz \\
&= \|T\| \cdot \iiint_{W} \left(x_1 X + x_2 Y + x_3 Z\right)^{n_1} \left(y_1 X + y_2 Y + y_3 Z\right)^{n_2} \left(z_1 X + z_2 Y + z_3 Z\right)^{n_3}
\end{aligned}
\qquad (16)
$$

Onde o Jacobiano $\|T\|$ é igual ao valor absoluto do determinante da matriz **T**. A seguir descrevemos a fórmula para calcular a integral do polinômio $x^{n_1} \cdot y^{n_2} \cdot z^{n_3}$ sobre um tetraedro W ortogonal unitário como descrito na equação (15).

$$
\begin{aligned}
\int_{W} x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \cdot dv &= \int_{0}^{1} \int_{0}^{1-z} \int_{0}^{1-z-y} x^{n_1} \cdot y^{n_2} \cdot z^{n_3} \cdot dx \cdot dy \cdot dz \\
&= \frac{n_1! \cdot n_2! \cdot n_3!}{(n_1 + n_2 + n_3 + 3)!}
\end{aligned}
\qquad (17)
$$

É possível observar que o tetraedro arbitrário pode ser sempre transformado em um tetraedro unitário ortogonal por meio da matriz de transformação **T** como na equação (14). Portanto, a integral de um polinômio sobre um tetraedro pode ser calculada simbolicamente pelas equações (16) e (17). Para calcular a integral envolvida na equação (16), primeiro devemos decompor o integrando

$$
\begin{aligned}
I &= \|T\| \cdot \iiint_{W} \left(x_1 X + x_2 Y + x_3 Z\right)^{n_1} \left(y_1 X + y_2 Y + y_3 Z\right)^{n_2} \left(z_1 X + z_2 Y + z_3 Z\right)^{n_3} \\
&= \|T\| \sum_i \sum_j \sum_k c(i, j, k) \int_{W} X^i Y^j Z^k \\
&= \|T\| \sum_i \sum_j \sum_k c(i, j, k) \frac{i! \cdot j! \cdot k!}{(i + j + k + 3)!}
\end{aligned}
\qquad (18)
$$

Onde a função $c(i, j, k)$ representa o coeficiente do termo $X^i Y^j Z^k$ na expação do integrando, que pode ser descrito como

$$
\left(x_1 X + x_2 Y + x_3 Z\right)^{n_1} \left(y_1 X + y_2 Y + y_3 Z\right)^{n_2} \left(z_1 X + z_2 Y + z_3 Z\right)^{n_3} = \sum_{i+j+k=n_1+n_2+n_3} c(i, j, k) x^i y^j z^k \qquad (19)
$$

Os seguintes exemplos exibem como devemos calcular o volume e o centro de massa de um 3D-simplex. O volume do tetraedro é dado por:

$$
V = \int_{Q} dv = \|T\| \int_{w} dV = \|T\| \frac{0!}{3!} = \frac{\|T\|}{6} \qquad (20)
$$

O baricentro (x_0, y_0, z_0) de um tetraedro é dado por

$$x_0 = \frac{1}{V}\int_Q x \cdot dv = \frac{\|T\|}{V}\iiint_W \left(x_1 \cdot X + x_2 \cdot Y + x_3 \cdot Z\right)\cdot dX \cdot dY \cdot dZ \qquad (21)$$

$$\begin{aligned} x_0 &= 1/4\left(x_1 + x_2 + x_3\right) \\ y_0 &= 1/4\left(y_1 + y_2 + y_3\right) \\ z_0 &= 1/4\left(z_1 + z_2 + z_3\right) \end{aligned} \qquad (22)$$

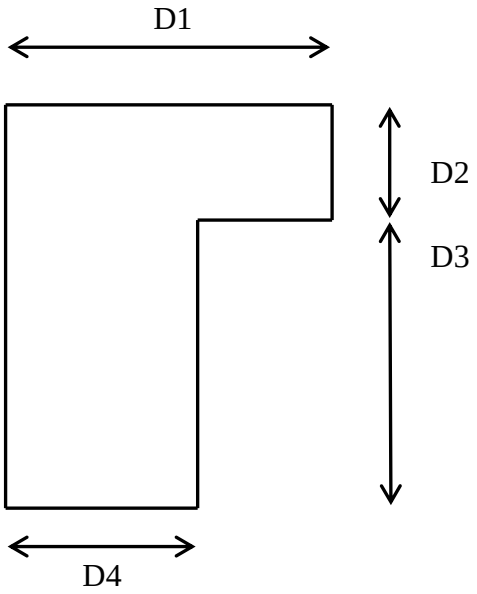
ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Mello Moraes, nº2231 CEP05508-900 São Paulo SP

Departamento Engenharia Mecatrônica e de Sistemas Mecânicos

Anexo 2 – Criando o sólido primitivo L

O sólido primitivo L é definido pelo desenho abaixo:



A Figura acima define um polígono, e para criarmos o sólido com volume temos a dimensão D5 que define de quanto este polígono será extrudado.

Você deverá utilizar os seguintes operadores de Euler:

`MVSF` (*Make Vertex Solid Face*): este operador cria um sólido inicial com apenas uma face e um vértice; o acionamento do operador é feito pela rotina:

```
SPTYPE MSD_lowMVSF(Id sn, Id fn, Id vn, Id dn, float xx, float yy, float zz)
```

`sn`: identificador do sólido a ser criado
`fn`: identificador da face a ser criada
`dn`: identificador do shell a ser criado
`vn`: identificador do vértice a ser criado
`xx, yy, zz`: coordenadas do vértice criado

MEV (*Make Edge Vertex*): este operador adiciona a um sólido uma aresta e um vértice. A aresta é criada conectando-se um vértice já existente ao novo vértice criado; o acionamento deste operador é feito pela rotina:

```
int MSD_highSVME3(Id sn, Id v1, Id v2, Id v3, Id v4, Id f1, Id f2, Id f3, Id f4,
                                                          float x, float y, float z)
```

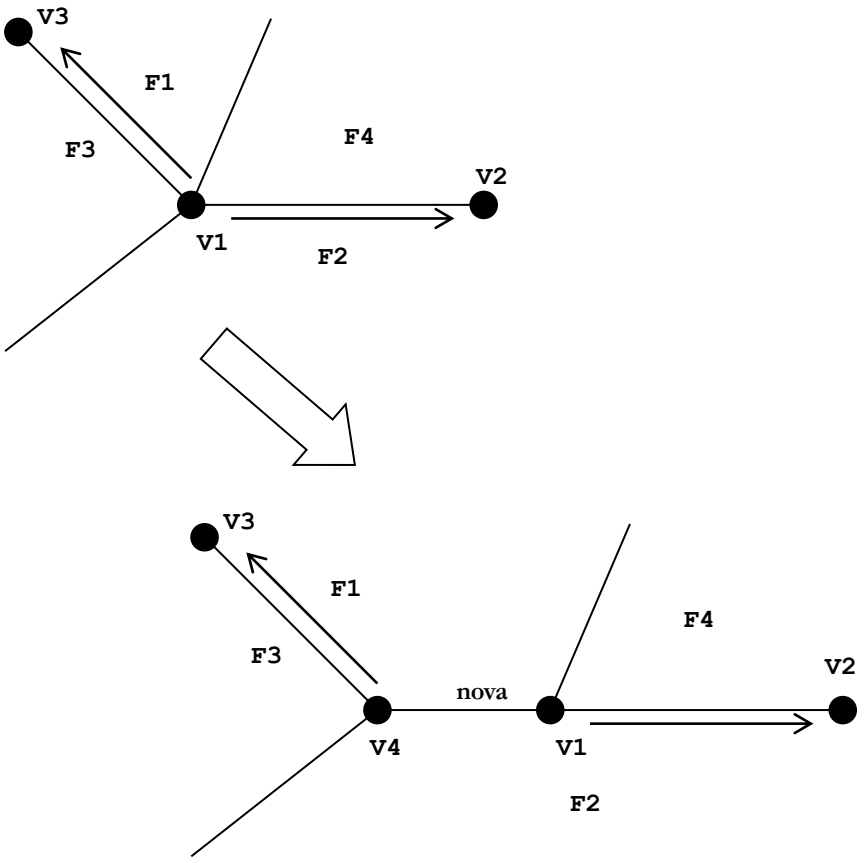
sn: identificador do sólido a ser modificado

a aresta e1 é definida por **f1**, **f3**, **v1** e **v2** e a aresta e2 é definida por **f2**, **f4**, **v1** e **v3**

x, **y**, **z**: coordenadas do vértice a ser criado

v4: identificador do novo vértice a ser criado.

O operador **MEV** possui a seguinte semântica:



MEF (*Make Edge Face*): este operador adiciona ao sólido uma aresta e uma face. A face é criada pela divisão de uma face já existente acrescentando-se a nova aresta; o acionamento deste operador é feito pela rotina:

```
int MSD_highMEF2(Id sn, Id v1, Id v2, Id v3, Id v4, Id f1, Id f2, Id f3, Id f4)
```

`sn`: identificador do sólido a ser modificado

a aresta e1 é definida por `f1`, `f2`, `v1` e `v3` e a aresta e2 é definida por `f1`, `f3`, `v2` e `v4`

`f4`: identificador da nova face a ser criada.

O operador **MEF** possui a seguinte semântica: