Zpráva s popisem metodologie, implementace a specifikací rozhraní modulu pro přenos dat mezi sítěmi

MDT (modul pro Mesh Data Transfer), je knihovna sloužící pro mapování dat mezi plošnými sítěmi konečných prvků. Knihovna je napsána v jazyce C a vydána pod otevřenou licencí GNU LGP. Zdrojové kódy, volně dostupné na domovské stránce http://mech.fsv.cvut.cz/mdt včetně dokumentace, využívají pouze standardní knihovny jazyka C a mohou být přeloženy na platformách Linux, Windows i Mac OS. Knihovna zatím funguje jako samostatná aplikace, jejímž vstupem je popis sítí (zdrojové a cílové) a výstupem je mapování mezi sítěmi.

1 Použitá metodologie mapování

Pro mapování veličin z uzlů jedné (zdrojové) sítě do uzlů sítě druhé (cílové) je použita interpolace pomocí bázových funkcí. To vyžaduje lokalizovat pro každý uzel cílové sítě nejbližší prvek sítě zdrojové a stanovit jeho parametrické souřadnice (v případě trojúhelníkové sítě plošné souřadnice) v rámci tohoto prvku. V těch se vyčíslí hodnoty bázových funkcí odpovídající jednotlivým uzlům lokalizovaného prvku, jež se následně použijí jako koeficienty interpolace uzlových hodnot cílové veličiny na tomto prvku.

Kritickou fází algoritmu, zejména jde-li o přenos dat mezi velkými sítěmi (čítajícími stovky tisíc uzlů či prvků), je lokalizace prvků zdrojové sítě pro jednotlivé uzly cílové sítě. Efektivní metodou pro lokalizaci je prostorová dekompozice problému (konečněprvkové sítě) s rychlým přístupem do jednotlivých částí sítě. Protože rovnoměrná dekompozice na buňky (kvádry) o stejné velikosti není vhodná pro lokálně zahuštěné sítě, jež jsou typicky výstupem adaptivní analýzy, byla pro dekompozici použita metoda oktalového stromu. Ta je založena na hierarchické dekompozici krychle, obepínající zdrojovou síť, postupným rekurzivním dělením vždy na 8 menších krychlí (oktantů) o poloviční hraně. Do jednotlivých terminálních (dále nedělených) oktantů se postupně zapisují uzly zdrojové sítě a to dle jejich prostorové příslušnosti k danému oktantu. Pokud počet uzlů zapsaných v daném oktantu dosáhne určitého limitu, který je rozhodující pro rychlost vyhledávání v rámci oktantu, je oktant rozdělen a uzly v něm zapsané jsou přerozděleny do jeho potomků dle jejich prostorové příslušnosti. Protože uzly sítě jsou obecně v prostoru rozloženy nerovnoměrně, je hloubka oktalového stromu proměnná a zohledňuje tak lokální zjemnění sítě. Po té, co jsou do oktalového stromu zaregistrovány všechny uzly zdrojové sítě, může začít vlastní lokalizace.

Vyhledání zdrojového prvku (pro daný cílový uzel) začne nejdříve lokalizací terminálního oktantu, ke kterému je cílový uzel prostorově příslušný. Uzly zapsané v tomto oktantu jsou potenciálně vrcholem hledaného zdrojového prvku. Proto se postupně vyšetřují prvky (zdrojové sítě) sdílející uzly zapsané v lokalizovaném oktantu a to v pořadí dle vzrůstající vzdálenosti od cílového uzlu. Pokud cílový uzel (resp. jeho průmět do roviny prvku, jedná-li se o zakřivenou síť)padne do některého z vyšetřovaných prvků, je zdrojový prvek lokalizován. Pokud se tak nestane (např. proto, že v lokalizovaném terminálním oktantu není zapsán žádný uzel nebo proto, že žádný z uzlů zdrojového prvku není v oktantu registrován), začne se prohledávat postupně zvětšující se okolí (ve tvaru koule) cílového uzlu. Vertikálním průchodem oktalové struktury se postupně detekují

terminální oktanty, které mají neprázdný průnik s krychlí opsanou kulovému okolí. V nich se pak vyšetřují zdrojové prvky sdílející ty uzly, které padnou do kulového okolí a současně nepadnou do kulového okolí (o menší velikosti) neúspěšně vyšetřovaného v předchozím kroku.

Protože síť konečných prvků je pouze aproximací skutečné geometrie (obecně se zakřivenou hranicí), není zaručeno, že vždy existuje zdrojový prvek, do nějž cílový uzel (na zakřivené hranici) padne. V takových případech je zapotřebí k uzlu cílové sítě lokalizovat nejbližší prvek zdrojové sítě. Aby se s touto eventualitou nemuselo počítat ve všech případech, jsou uzly cílové sítě na základě souvislosti jejich okolí (příslušejícího témuž regionu, jsou-li zpracovávané sítě víceregionální) klasifikovány do dvou skupin - vnitřní a hraniční. Pro vnitřní uzly se kulové okolí (viz výše) zvětšuje tak dlouho, dokud není zdrojový prvek, do něhož cílový uzel padne, nalezen. V případě hraničních uzlů se kulové okolí postupně zvětšuje pouze několikrát a z vyšetřovaných prvků se vybere ten, který je cílovému uzlu nejblíže.

2 Popis implementace a datového rozhraní

Knihovna MDT slouží k mapování dat mezi trojúhelníkovými sítěmi MKP. Pro každý uzel cílové sítě je nalezen odpovídající prvek zdrojové sítě, do kterého uzel padne nebo kterému je nejblíže, a stanoví plošné souřadnice uzlu v tomto prvku. Ty se následně využijí (jako bázové funkce) pro mapování veličin z uzlů lokalizovaného prvku zdrojové sítě do vyšetřovaného uzlu sítě cílové. Obě sítě musí být diskretizací téhož geometrického modelu.

Na vstupu se očekává popis zdrojové i cílové sítě, tj. jejich geometrie (souřadnice uzlů), konektivita (čísla uzlů jednotlivých prvků) a příslušnost prvků k jednotlivým regionům. Číslování se předpokládá spojité začínající od 1. Na výstupu jsou pak pro uzly jednotlivých regionů cílové sítě uvedeny nejbližší prvek (příslušný regionu) zdrojové sítě, čísla jeho uzlů a plošné souřadnice cílového uzlu promítnutého do roviny tohoto prvku. V současné verzi se vstup i výstup realizují přes datové soubory.

Formát vstupního souboru:

počet_uzlů počet_prvků
číslo_uzlu souřadnice_x souřadnice_y souřadnice_z
číslo_prvku číslo_regionu počet_uzlů_prvku čísla_uzlů_prvku

Formát výstupního souboru:

číslo_regionu počet_uzlů
čislo_uzlu_cílové_sítě číslo_prvku_zdrojové_sítě čísla_uzlů_prvku plošné_souřadnice

Datové struktury:

Struktura nodes_list (alias LIST)

Slouží pro seznam uzlů, které jsou v OCTANTu. Tato struktura umožnuje přidávání uzlů do seznamu aniž bychom dříve znali počet uzlů v seznamu.

Struktura LIST obsahuje 2 parametry:

- *long* ***nodenum*** *- číslo uzlu*

- *struct **nodes_list *next** - ukazatel na další část seznamu*

Struktura octant (alias OCTANT)

OCTANT je pomyslná krychle, na které je rozdělen prostor, kde se síť nachází.

Struktura OCTANT obsahuje 5 parametrů:

- *int t**erminal** - určuje jestli je OCTANT terminální či nikoliv (1- je Terminal; 0-není Terminal);*
- *double **coordinates[4]** - určuje souřadnice (0. pozice - délka hrany, 1. pozice - x středu, 2. pozice - y středu, 3. pozice - z středu)*
- *int **num_of_nodes** - počet uzlů v OCTANTu*
- *LIST ***nodes_list** - určuje čísla uzlů nacházející se v OCTANTu*
- *struct **octant *childs[8]** - pole ukazatelů na potomky OCTANTu*

Variabilní parametry:

Parametry pro debug a ladění výkonu knihovny:

- ***MAX_NODE_NUM** - počet uzlů nacházejících se v oktantu (doporučené hodnoty 10-100)*
- ***NUM_OF_VALUES** - počet hodnot různých veličin uložených v uzlech*
- ***OCT_RESIZE** - zvětšení octree (1.1 odpovídá zvětšení na 110% tedy zvětšení o 10%)*
- ***INVESTIGATED_AREA_RESIZE** – zvětšení oblasti ve které jsou hledány nejbližší uzly, pokud dosud nebyly žádné nalezeny (1.5 odpovídá zvětšení o 0.5 velikosti oblasti na každou stranu)*
- ***MAX_NODE_VISIT** – počet uzlů pro které je u všech prvků jim příslušících porovnávána nejmenší záporná plošná souřadnice*

Globální proměnné:

- *double ***NODE_field_maped, *NODE_field_orig** - obsahuje data o uzlech zdrojové(nebo cílové) sítě v tomto pořadí: 0-číslo uzlu, 1-souřadnice x, 2-souřadnice y, 3-souřadnice z*
- *double **answer[4]** - na první pozici je číslo prvku s největší zápornou plošnou souřadnicí, na druhé až čtvrté pozici jsou jeho plošné souřadnice x,y a z*
- *double **search_centre[3]** - souřadnice středu vyhledávání ve formátu x, y, z*
- *double **x_maped, y_maped, z_maped** - jsou souřadnice uzlu cílové sítě*
- *double **search_radius, previous_radius** - poloměr hledání a předchozí poloměr hledání*
- *long **number_of_visited_nodes** - počet navštívených prvků*
- *long **maped_node_num** - číslo aktuálního prvku cílové sítě*
- *long **nodes_orig, nodes_maped** - počet prvků obou sítí*
- *long ***ELEMENT_field_orig, *ELEMENT_field_maped** – pole prvků zdrojové (nebo cílové) sítě*
- *long***NODE_field_conected_to_elements_positions(_MAPED)** – obsahuje číslo určující pořadí příslušného uzlu v následujícím poli (NODE_field_conected_to_elements_list)*
- *long ***NODE_field_conected_to_elements_list(_MAPED)** - obsahuje čísla prvků v pořadí jednotlivých uzlů*
- *long ***elements_searched_list** - obsahuje informaci, zda byl prvek zdrojové sítě v souvislosti s aktuálním uzlem cílové sítě už vyšetřován*

Funkce

int main() – HLAVNÍ FUNKCE

Stručný popis: Alokuje proměnné. Ověří si soubor zdrojové sítě funkcí **verify_FILE** a otevře ho. Načte z něj všechna data pomocí funkcí **create_1D_FIELD** a zavře ho. Ověří si soubor cílové sítě funkcí **verify_FILE** a otevře ho. Načte z něj všechna data pomocí funkcí **create_1D_FIELD** a zavře ho. Poté zavolá funkce **establish_conectivity_1** a **establish_conectivity_2** s parametry zdrojové sítě a uloží jejich návratová pole. Dále zavolá funkce **establish_conectivity_1** a **establish_conectivity_2** s parametry cílové sítě a uloží jejich návratová pole. Pomocí funkce zjistí meze prostoru zdrojové sítě **find_OCTREE_BOUNDRY** a může dále vytvořit první OCTANT funkcí **create_first_OCTANT.** Funkcí **grow_octree_of_OCTANTS** pak načítá do OCTANTu uzly a nechává ho růst**.** Nakonec zavolá funkci **locate_nodes_in_octant** aby lokalizovala cílové prvky v uzlech zdrojové sítě. Na závěr otevře soubor new_mesh.txt a uloží pro něj data pro mapování.

int verify_FILE (FILE *g) - FUNKCE PRO KONTROLU SOUBORU

Požaduje 1 parametr: ukazatel na soubor již otevřený funkci main

V*rací:* 0 - pokud soubor sítě neexistuje; 1 - pokud soubor sítě existuje;

double* create_1D_FIELD(FILE *g, long n) - FUNKCE PRO ALOKACI 1D POLE TYPU DOUBLE A NAČTENÍ HODNOT DO POLE

Požaduje 2 parametry: velikost pole a ukazatel na soubor již otevřený funkci main

V*rací:* ukazatel na pole

long* create_1D_FIELD (FILE *g, long n) - FUNKCE PRO ALOKACI 1D POLE TYPU LONG A NAČTENÍ HODNOT DO POLE

Požaduje 2 parametry: velikost pole a ukazatel na soubor již otevřený funkci main

Vrací: ukazatel na pole

double* find_OCTREE_BOUNDRY (double field[], long nodes) - FUNKCE PRO ZJIŠTENÍ MIN. A MAX. SOUŘADNICE VŠECH UZLŮ

Požaduje 2 parametry: 1d_pole uzlů a počet uzlů (obojí zdrojové sítě)

Vrací: ukazatel na pole maximálních a minimálních hodnot

Stručný popis: Projde souřadnice všech uzlů a do pole o velikosti 6 si uloží maximální a minimální hodnoty jednotlivých souřadnic x,y a z.

void create_first_OCTANT (OCTANT *octant, double boundry[]) - FUNKCE PRO VYTVOŘENÍ PRVNÍHO OCTANTU

Požaduje 2 parametry: OCTANT, a pole max. a min. souřadnic

Stručný popiss: Najde největší rozdíl max. a min. souřadnic a zvětší ho hodnotou OCT_RESIZE (aby uzle nepadaly mimo octree) a uloží ho jako hranu OCTANTu. Poté vypočte souřadnice jeho středu a nastaví OCTANT jako terminal a všechny další parametry OCTANTu na nulové hodnoty.

void grow_octree_of_OCTANTS (OCTANT *octant, double field[], long nodes) - FUNKCE PRO TVORBU OCTREE

Požaduje 3 parametry: OCTANT, a pole a počet uzlů (zdrojové sítě)

Stručný popis: Prochází postupně všechny uzly, a když najde koncový OCTANT, do kterého uzel spadá, uloží ho do seznamu LIST (pomocí funkce ***add_to_LIST***). Pokud ale při ukládání nového uzlu do seznamu v OCTANTu překročí seznam velikost MAX_NODE_NUM zavolá funkci **divide_OCTANT.**

void divide_OCTANT (OCTANT *parent, double field[]) - FUNKCE PRO ROZDĚLENÍ OCTANTU MEZI POTOMKY

požaduje 2 parametry: OCTANT, který chceme dělit a 1d_pole uzlů

Stručný popis: Vytvoří 8 OCTANTů potomků. Všem potomkům se uloží délka poloviční hrany rodiče. Souřadnice středů OCTANTů se vypočtou tak, aby nevznikly 2 stejné OCTANTy. Potomci se nastaví jako terminal (tedy koncové OCTANTy ve stromu) a odkazy na jejich potomky se nastaví na hodnotu NULL. Jednotlivé uzly rodiče se roztřídí podle souřadnic mezi potomky a pomocí funkce ***add_to_LIST*** se přidají do jejich seznamů uzlů. Seznam uzlů rodiče se poté vymaže funkcí ***delete_LIST***

long *establish_conectivity_1 (long nodes_maped, long ELEMENT_field [], long triangEL) - FUNKCE PRO ALOKOVÁNÍ A VYPLNĚNÍ POLE POŘADÍ V POLI KONEKTIVITY

Požaduje 3 parametry: počet uzlů, pole prvků a počet prvků

Vrací: pole pořadí uzlů v poli konektivity (podrobněji v komentáři ve zdrojovém kódu)

Stručný popis: Nejdříve projde všechny pole prvků a zjistí, kolikrát jsou jednotlivé uzly využity. Poté podle využití jednotlivých uzlů vyplní postupně pole pořadí.

long *establish_conectivity_2 (long list_of_positions[], long ELEMENT_field [], long nodes, long triangEL) - FUNKCE PRO ALOKOVÁNÍ A VYPLNĚNÍ POLE KONEKTIVITY

Požaduje 4 parametry: Pole pořadí v poli konektivity, počet uzlů, pole prvků a počet prvků

Vrací: pole konektivity

Stručný popis: Alokuje si pole aktuálního pořadí v poli konektivity. Poté prochází jednotlivé prvky a ukládá je do úseků patřící příslušným uzlům. Zároveň si v poli aktuálního pořadí posouvá odkazy na úseky v poli konektivity tak, aby se nepřepisovaly již dříve zapsané prvky.

long* locate_nodes_in_octant(OCTANT *octant, long nodes_maped, long triangEL_maped, long triangEL_orig) - FUNKCE PRO LOKALIZACI UZLŮ CÍLOVÉ SÍTĚ V PRVCÍCH ZDROVÉ SÍTĚ

Požaduje 4 parametry: OCTANT, počet uzlů cílové sítě, počet prvků cílové a zdrojové sítě

Vrací: Pole, kde pozice označuje mapovaný uzel a hodnota prvek, ve kterém se uzel nachází, (nebo je mu nejblíže).

Stručný popis: Každý uzel cílové sítě lokalizuje v příslušném OCTANTu. Pokud jsou v octantu uloženy nějaké uzly zdrojové sítě, hledá přes ně prvky, do kterého by mohl cílový uzel spadat. Pokud nenalezne prvek, do kterého uzel spadá, a nesplní dosud podmínku, že počet prohledaných prvků musí být větší než *MAX_NODE_VISIT* spustí funkci ***search.*** Pokud funkce ***search*** vrátí hodnotu 1(tzn. že prvek byl lokalizován) ukončí se hledání. Pokud ne, a funkce ***investigate_surrounding_area*** vrátí nesouvislé okolí (hodnota 0), zkontroluje se, zda nebyl překročen *MAX_NODE_VISIT.* Když limit překročen nebyl, poloměr koule se zvětší o *INVESTIGATED_AREA_RESIZE* a pokračuje se v hledání. Hledání se ukončí, pokud byl uzel lokalizován přesně v prvku, nebo byl překročen *MAX_NODE_VISIT.*

double *get_point_from_node_field (long point_num, double node_field[]) - FUNKCE PRO EXTRAKCI BODU Z POLE UZLŮ

Požaduje 2 parametry: pole uzlů a číslo bodu

Vrací: bod (pole se souřadnicemi x, y a z)

double *create_vector_from_point1_to_point2 (double point1[], double point2[]) - FUNKCE PRO VYTVOŘENÍ VECTORU ZE DVOU BODŮ

Požaduje 2 parametry: 2 body (ve formátu předchozí funkce)

Vrací: vektor (pole se směrnicemi x, y a z)

double get_skalar_of_vectors(double vector1[], double vector2[]) - FUNKCE PRO SKALÁRNÍ SOUČIN VEKTORŮ

Požaduje 2 parametry: 2 vektory (ve formátu předchozí funkce)

Vrací: skalární součin

double *create_vector_NORMAL_to_line_across_point (double point_on_line[], double point_off_line[], double line_vector[]) - FUNKCE PRO VYTVOŘENÍ NORMÁLOVÉHO VECTORU PŘÍMKY PROCHÁZEJÍCÍHO DANÝM BODEM MIMO PŘÍMKU

Požaduje 3 parametry: bod na přímce, bod, kterým normála povede a směrový vektor přímky (ve formátu předchozích funkcí)

Vrací: normálový vektor

double *get_area(double point_on_line[], double point_on_line2[], double point_off_line[], double line_vector[])-FUNKCE PRO VYPOČTENÍ PLOCHY TROJÚHELNÍKA

Požaduje 4 parametry: 2 body trojúhelníka, třetí bod trojúhelníka, a směrový vektor úsečky mezi prvními 2ma body (ve formátu předchozích funkcí)

Vrací: plochu trojúhelníku

void add_to_LIST (LIST **ppl, long node) - FUNKCE PRO PŘIDÁNÍ HODNOTY DO SEZNAMU TYPU LIST

Požaduje 2 parametry: adresu seznamu a hodnotu, kterou přidáváme

Stručný popis: Vytvoří novou strukturu typu LIST. Uloží do ní přidávanou hodnotu a ukazatel na původní část seznamu LIST. Do příslušného OCTANTu pak uloží ukazatel na novou strukturu LIST (původní část seznamu tedy zůstává neměnná, další hodnoty se přidávají "na začátek").

void delete_LIST (LIST **ppl)- FUNKCE PRO SMAZÁNÍ SEZNAMU

Požaduje 1 parametr: adresu seznamu

Stručný popis: Postupně projde všechny články seznamu a smaže je, aniž by nějakou vynechal.

int search(OCTANT *octant)- FUNKCE HLEDÁNÍ DALŠÍCH BLÍZKÝCH BODŮ

Požaduje 1 parametr: OCTANT (vše ostatní jsou z důvodu rekurzivity funkce globální proměnné)

Vrací: 1 - pokud přímo funkce ***locate_node_in_element*** najde prvek, do kterého uzel spadá, nebo 0 - pokud se prvek, do kterého by uzel spadal, přímo nenajde

Stručný popis: Prochází octree a hledá koncový OCTANT, který se překrývá s původním OCTANTem zvětšeným o *INVESTIGATED_AREA_RESIZE.* Toto prohledávání probíhá rekurzivně. Až najde koncový (terminální) OCTANT zkusí, jestli jeho uzly patří do pomyslné koule zvětšeného okolí. Ty, které do ní spadají, otestuje funkcí ***locate_node_in_element.*** Uzly, které spadaly do předchozích (menších) koulí netestuje znovu, ale přeskakuje.

int locate_node_in_element(long maped_node_num)- FUNKCE NA LOKALIZACI UZLU V PRVCÍCH

Požaduje 1 parametr: uzel zdrojové sítě (vše ostatní jsou z důvodu rekurzivity nadřazené funkce globální proměnné)

Vrací: 1 - pokud přímo najde prvek, do kterého uzel spadá, nebo 0 - pokud prvek nepatří do žádného prvku v okolí uzlu

Stručný popis: Nejdříve načte všechny okolní prvky. Poté je postupně prochází a hledá, do kterého mapovaný uzel spadá. U každého prvku postupně vypočte pomocí funkcí pro práci s vektory: 1) normálový vektor každé strany a 2) vektor z bodu, nacházejícího se na dané straně, do cílového bodu. Pokud jsou skalární součin těchto dvou vektorů pro všechny strany trojúhelníka kladné, nachází se bod uvnitř trojúhelníku (tedy prvku). V tomto případě vrátí hodnotu 1 a uloží si příslušný prvek. Dále také počítá plošné souřadnice na prvcích a ukládá si nejmenší nalezenou zápornou souřadnici (největší a zároveň menší než nula.) Pokud nenajde prvek, v kterém by se uzel přímo nacházel, vrátí hodnotu 0.

long* search_list_for_closest(LIST *pl, int nodes) - FUNKCE PRO VYTVOŘENÍ TŘÍDĚNÉHO SEZNAMU NEJBLIŽŠÍCH UZLŮ ORIGINÁLNÍ SÍTĚ V LISTU UZLŮ

Požaduje 2 parametry: LIST uzlů, počet uzlů v LISTu uzlů

vrací: pole seznam uzlů seřazený od nejbližšího po nejvzdálenější.

Stručný popis: Prochází seznam typu LIST a načítá si souřadnice uzlů v seznamu. Z nich vypočte vzdálenost od cílového bodu. Do jednoho pole si postupně uloží čísla uzlů a do druhého na stejné pozice jejich vzdálenosti od cílového bodu. Poté se obě pole paralelně seřadí od nejmenší vzdálenosti k největší. Řazení probíhá systémem: Prohoď větší hodnotu na aktuální pozici za menší hodnotu na další pozici, nebo pokračuj beze změny.

void investigate_surrounding_area (void) - FUNKCE NA VYŠETŘENÍ SOUVISLOSTI OKOLÍ (TEDY ZDA JE PRVEK OBKLOPEN ZE VŠECH STRAN UZLY)

Nepožaduje parametry: (všechny užívané parametry jsou globální)

Stručný popis: Zkoumá všechny okolní prvky, které uzel využívají, a spočítá všechny uzly těchto prvků. Pokud se každý uzel opakuje alespoň 2krát, pak je okolí souvislé. Nakonec se uloží do pole souvislosti, zda je okolí kolem bodu souvislé, nebo není.