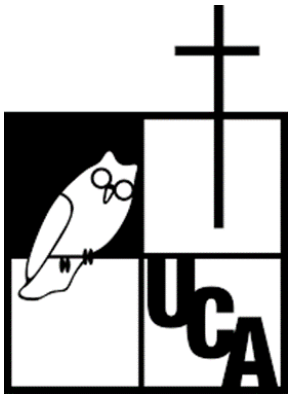
UNIVERSIDAD CENTROAMERICANA "JOSÉ SIMEÓN CAÑAS"

FACULTAD DE INGENIERÍA Y ARQUITECTURA



Asignatura:

Simulación de computadoras

Sección 01

Equipo de Proyecto: Equipo 04

Catedrático:

Jorge Alfredo López Sorto

Integrantes:

Melvin Alexander Díaz Ayala 00008821

Luis Alexander Hernandez Martinez 00129020

Marcos Antonio Hernandez Grande 00007518

Mario Antonio Martínez Villatoro 00072520

Fecha de entrega: 28 de junio de 2024.

Introducción a la Implementación del Método de Elementos Finitos

Implementar el Métdodo de Elementos Finitos (MEF) para resolver el problema de transferencia de calor plantea un desafío significativo. La dificultad fundamental yace en aproximar un modelo de transferencia continuo con un sistema discreto, como una computadora. La naturaleza continua de un fenomeno físico, como distribución de calor, requiere una ejecución precisa de la matemática lo cuál significa cierta dificultad y complejidad para resolver este problema de manera analítica.

Al momento de traducir estos modelos a una aproximación discreta adecuada para el análisis computacional nos encontramos con varios problemas dentro de su resolución y su implementación, entre ellas están: la discretización del modelo físico, la debida contextualización de la aproximación del modelo, la creación del mallado y la aproximación de la sumas continuas.

Además al momento de realizar la implementación del MEF, se requiere diseñar una estructura de código adecuada para facilitar su desarrollo y depuración. La presencia de programación modular y la utilización de funciones se convierte en una necesidad para conseguir la mejor implementación de no solo el MEF sino cualquier código.

Diferencias entre MEF 2D y 3D

Después de la resolución del MEF en dos dimensiones, se necesita regresar al modelo físico inicial para lograr la interpretación del método en tres dimensiones.

Las principales diferencias entre el módelo bidimensional y tridimensional son las distintas dimensiones de los componentes que están presentes en ambas soluciones. Desde la implicación de un tercer eje dentro del sistema de coordenadas hasta el aumento de dimensiones de la matriz jacobiana que nos permite realizar la transiciones entre sistemas de magnitudes escalares y vectoriales.

Modificaciones y Adiciones a la Implementación del MEF

A continuación, se destacarán los cambios ***principales*** dentro del código en C++ del Método de Elementos Finitos proporcionado por la UCA SV en Github.

´

Geometría

Dentro de la geometría del proceso del MEF, debido a la adición de una nueva dimensiones al problema de transferencias de calor, se necesita adicionar la nueva coordenada z para la clase Nodo con sus respectivas funciones de acceso y colocar valores.

```
class Node {
    //TODO: add z coordinate
    private:
        int ID;
        float x_coordinate;
        float y_coordinate;
        float z_coordinate;
    public:

        /*
        *
        Other class functions
        *
        */

        //TODO: add z getter and setter
        void set_z_coordinate(float z_value){
            z_coordinate = z_value;
        }
        float get_z_coordinate(){
            return z_coordinate;
        }
};
```

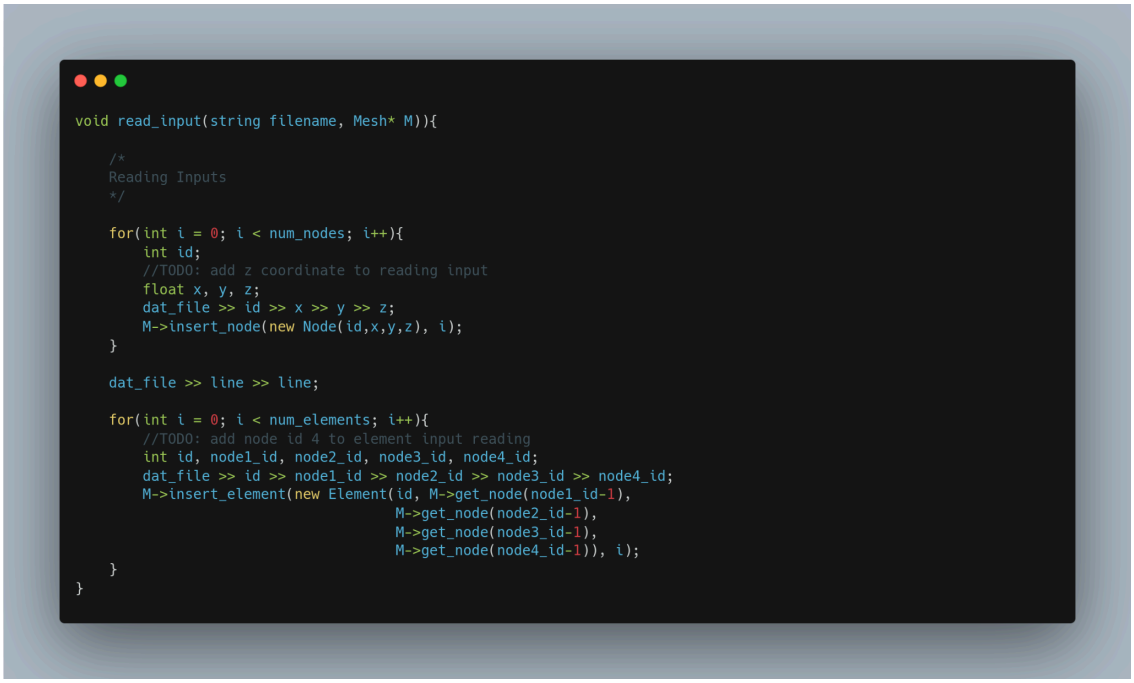
A su vez, al momento de transicionar a un modelo tridimensional, los elementos fundamentales del MEF se convierten en tetraedros los cuales están descritos por 4 nodos, esto deberá actualizarse en el archivo correspondiente de la clase Elemento.

```
class Element {
    //TODO: Add new 4th node to Element
    private:
        int ID;
        Node* node1;
        Node* node2;
        Node* node3;
        Node* node4;
    public:
        /*
        Other class functions
        */
        //TODO: getter and setter for 4th row
        void set_node4(Node* node){
            node4 = node;
        }
        Node* get_node4(){
            return node4;
        }
};
```

Figura 1. Clase de C++ que describe un Elemento en MEF

Entrada y Salida

Después de agregarle el nuevo campo a la clase Nodo, se debe proceder a actualizar la función de lectura de datos del archivo .dat en las partes donde los nodos intervienen. Debido a que estos se les ha otorgado un nuevo campo, este debe verse atribuido al momento de la instanciación de las variables nodo y elemento.



```
void read_input(string filename, Mesh* M)){

    /*
    Reading Inputs
    */

    for(int i = 0; i < num_nodes; i++){
        int id;
        //TODO: add z coordinate to reading input
        float x, y, z;
        dat_file >> id >> x >> y >> z;
        M->insert_node(new Node(id,x,y,z), i);
    }

    dat_file >> line >> line;

    for(int i = 0; i < num_elements; i++){
        //TODO: add node id 4 to element input reading
        int id, node1_id, node2_id, node3_id, node4_id;
        dat_file >> id >> node1_id >> node2_id >> node3_id >> node4_id;
        M->insert_element(new Element(id, M->get_node(node1_id-1),
                                      M->get_node(node2_id-1),
                                      M->get_node(node3_id-1),
                                      M->get_node(node4_id-1)), i);
    }
}
```

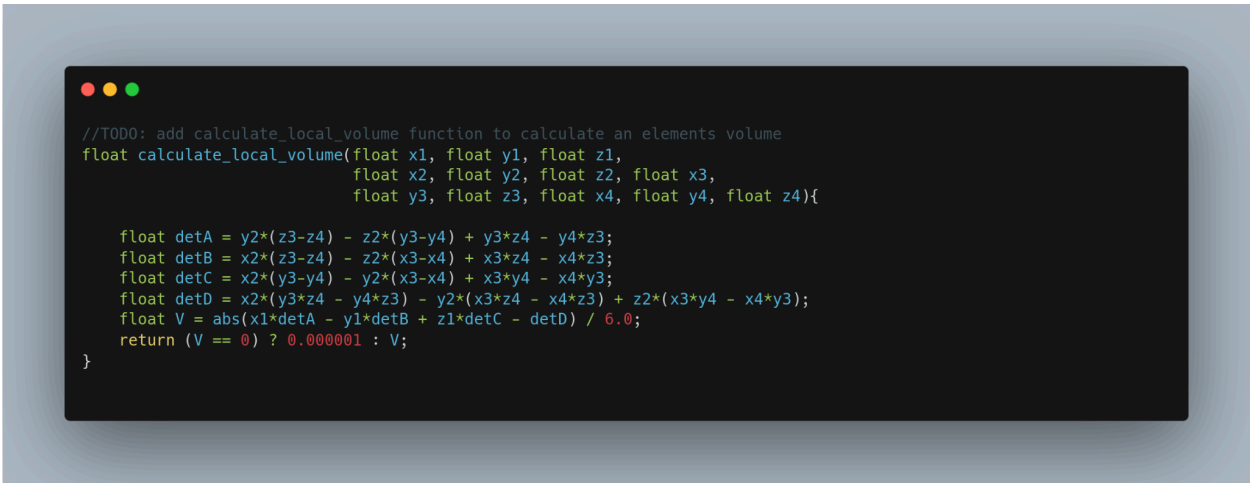
Figura 2. Función de lectura de archivo .dat

Proceso del MEF

Debido a la extensión espacial del MEF, este se ve afectado en casi todos los aspectos que requieren algún tipo de cálculo que requiera la estructuras modificadas debido a las transformaciones dimensionales que han sufrido estos componentes.

Volumen

El cálculo del volumen es procede a verse afectado en la cantidad de parámetros que recibe; la función del cálculo de volumen ahora recibe coordenadas del eje z, y un cuarto punto que proviene del nuevo nodo que se le agrega al componente base (pasar de triangulos a tetraedros).



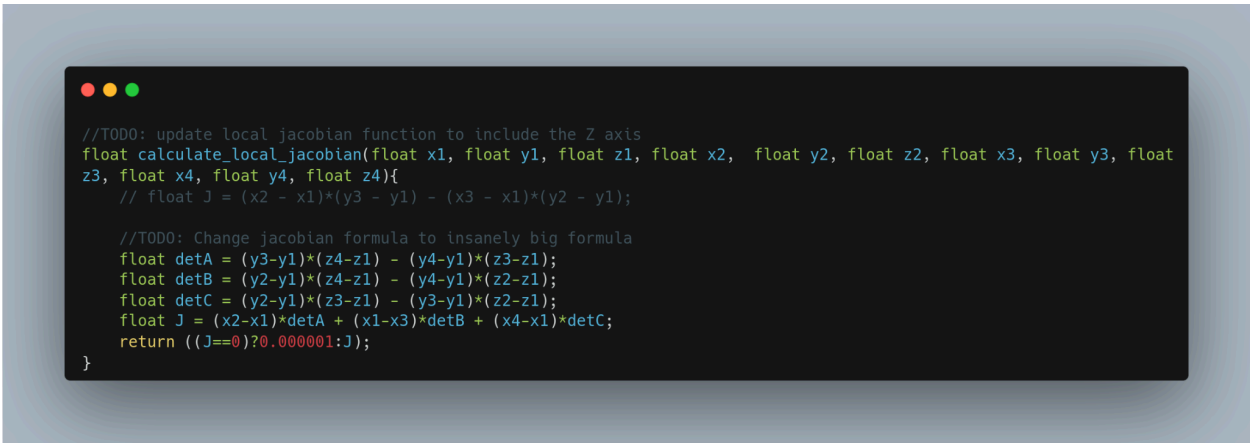
```
//TODO: add calculate_local_volume function to calculate an elements volume
float calculate_local_volume(float x1, float y1, float z1,
                             float x2, float y2, float z2, float x3,
                             float y3, float z3, float x4, float y4, float z4){

    float detA = y2*(z3-z4) - z2*(y3-y4) + y3*z4 - y4*z3;
    float detB = x2*(z3-z4) - z2*(x3-x4) + x3*z4 - x4*z3;
    float detC = x2*(y3-y4) - y2*(x3-x4) + x3*y4 - x4*y3;
    float detD = x2*(y3*z4 - y4*z3) - y2*(x3*z4 - x4*z3) + z2*(x3*y4 - x4*y3);
    float V = abs(x1*detA - y1*detB + z1*detC - detD) / 6.0;
    return (V == 0) ? 0.000001 : V;
}
```

Figura 3. Función para calcular el volumen de un tetraedro.

Jacobiano

Gracias a la adición de una nueva dimensión al método, esto causó repercusión en pasos del MEF como la interpolación y la definición de funciones de forma. Lo cuál hasta cierto punto, al momento de la resolución de la integral de la izquierda, esto requiere luego de la debida transformación espacial lo cuál produce la presencia del Jacobiano de dimensiones 3x3.



```
//TODO: update local jacobian function to include the Z axis
float calculate_local_jacobian(float x1, float y1, float z1, float x2,  float y2, float z2, float x3, float y3, float
z3, float x4, float y4, float z4){
    // float J = (x2 - x1)*(y3 - y1) - (x3 - x1)*(y2 - y1);

    //TODO: Change jacobian formula to insanely big formula
    float detA = (y3-y1)*(z4-z1) - (y4-y1)*(z3-z1);
    float detB = (y2-y1)*(z4-z1) - (y4-y1)*(z2-z1);
    float detC = (y2-y1)*(z3-z1) - (y3-y1)*(z2-z1);
    float J = (x2-x1)*detA + (x1-x3)*detB + (x4-x1)*detC;
    return ((J==0)?0.000001:J);
}
```

Figura 4. Calculo de la determinante del Jacobiano

Matrices Locales y Ensamblaje

Al presentar la solución la integral de la izquierda, nos presentamos con la declaración o definición de nuevas matrices que se utilizan al momento de crear la matriz A y B.

```
//TODO: change function to meet the dimensions of new A matrix in 3d
void calculate_local_A(Matrix* A, float x1, float y1, float z1,
                       float x2, float y2, float z2,
                       float x3, float y3, float z3,
                       float x4, float y4, float z4){

    //Calculating Matrix A cells
    float a = (y3-y1)*(z4-z1) - (y4-y1)*(z3-z1);
    float b = (x4-x1)*(z3-z1) - (x3-x1)*(z4-z1);
    float c = (x3-x1)*(y4-y1) - (x4-x1)*(y3-y1);
    float d = (y4-y1)*(z2-z1) - (y2-y1)*(z4-z1);
    float e = (x2-x1)*(z4-z1) - (x4-x1)*(z2-z1);
    float f = (x4-x1)*(y2-y1) - (x2-x1)*(y4-y1);
    float g = (y2-y1)*(z3-z1) - (y3-y1)*(z2-z1);
    float h = (x3-x1)*(z2-z1) - (x2-x1)*(z3-z1);
    float i = (x2-x1)*(y3-y1) - (x3-x1)*(y2-y1);
    //Transposing
    A->set(a,0,0);A->set(b,0,1);A->set(c,0,2);
    A->set(d,1,0);A->set(e,1,1);A->set(f,1,2);
    A->set(g,2,0);A->set(h,2,1);A->set(i,2,2);
}
```

```
//TODO: refactor calculate B to meet the following:
void calculate_B(Matrix* B){
    B->set(-1,0,0);  B->set(1,0,1);  B->set(0,0,2);  B->set(0,0,3);
    B->set(-1,1,0);  B->set(0,1,1);  B->set(1,1,2);  B->set(0,1,3);
    B->set(-1,2,0);  B->set(0,2,1);  B->set(0,2,2);  B->set(1,2,3);
}
```

Figura 5 y 6. Creación y asignación de valores para matrices A y B

Adicionalmente, se tiene que modificar el proceso de creación de matriz local K y b al necesitar la demás coordenadas para poder seguir el plantamiento del MEF 3D.

```
//TODO: change function to implement all changes to new K matrices
void create_local_K(Matrix* K, int element_id, Mesh* M){
    K->set_size(4,4);
    float k = M->get_problem_data(THERMAL_CONDUCTIVITY);
    float x1 = M->get_element(element_id)->get_node1()->get_x_coordinate(), y1 = M-
>get_element(element_id)->get_node1()->get_y_coordinate(), z1 = M->get_element(element_id)->get_node1()-
>get_z_coordinate(),
          x2 = M->get_element(element_id)->get_node2()->get_x_coordinate(), y2 = M-
>get_element(element_id)->get_node2()->get_y_coordinate(), z2 = M->get_element(element_id)->get_node2()-
>get_z_coordinate(),
          x3 = M->get_element(element_id)->get_node3()->get_x_coordinate(), y3 = M-
>get_element(element_id)->get_node3()->get_y_coordinate(), z3 = M->get_element(element_id)->get_node3()-
>get_z_coordinate(),
          x4 = M->get_element(element_id)->get_node4()->get_x_coordinate(), y4 = M-
>get_element(element_id)->get_node4()->get_y_coordinate(), z4 = M->get_element(element_id)->get_node4()-
>get_z_coordinate();
    float Volume = calculate_local_volume(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4);
    float J = calculate_local_jacobian(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4);
    Matrix B(3,4), A(3,3);
    calculate_B(&B);
    calculate_local_A(&A, x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4);
    Matrix Bt(4,3), At(3,3);
    transpose(&B,3,4,&Bt);
    transpose(&A,3,3,&At);

    Matrix res1, res2, res3;
    product_matrix_by_matrix(&A,&B,&res1);
    product_matrix_by_matrix(&At,&res1,&res2);
    product_matrix_by_matrix(&Bt,&res2,&res3);

    product_scalar_by_matrix(k*Volume/(J*J),&res3,4,4,K);
}
```

```
void create_local_b(Vector* b, int element_id, Mesh* M){
    b->set_size(4);

    float Q = M->get_problem_data(HEAT_SOURCE);
    float x1 = M->get_element(element_id)->get_node1()->get_x_coordinate(), y1 = M-
>get_element(element_id)->get_node1()->get_y_coordinate(), z1 = M->get_element(element_id)->get_node1()-
>get_z_coordinate(),
          x2 = M->get_element(element_id)->get_node2()->get_x_coordinate(), y2 = M-
>get_element(element_id)->get_node2()->get_y_coordinate(), z2 = M->get_element(element_id)->get_node2()-
>get_z_coordinate(),
          x3 = M->get_element(element_id)->get_node3()->get_x_coordinate(), y3 = M-
>get_element(element_id)->get_node3()->get_y_coordinate(), z3 = M->get_element(element_id)->get_node3()-
>get_z_coordinate(),
          x4 = M->get_element(element_id)->get_node4()->get_x_coordinate(), y4 = M-
>get_element(element_id)->get_node4()->get_y_coordinate(), z4 = M->get_element(element_id)->get_node4()-
>get_z_coordinate();
    float J = calculate_local_jacobian(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4);

    b->set(Q*J/24.0f, 0);
    b->set(Q*J/24.0f, 1);
    b->set(Q*J/24.0f, 2);
    b->set(Q*J/24.0f, 3);
}
```

Figura 7 y 8. Funciones de creación de matrices local K y b (**KT** = **b**)