KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNA MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

GENEL KORUNUM DENKLEMİNİ SONLU HACİMLER METODU İLE MODELLEYEN VE HETEROJEN HESAPLAMA TEKNOLOJİSİNİ DESTEKLEYEN YAZILIMIN GELİŞTİRİLMESİ

BARIŞ CUMHUR

KOCAELİ 2020

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MAKİNE MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

GENEL KORUNUM DENKLEMİNİ SONLU HACİMLER METODU İLE MODELLEYEN VE HETEROJEN HESAPLAMA TEKNOLOJİSİNİ DESTEKLEYEN YAZILIMIN GELİŞTİRİLMESİ

BARIŞ CUMHUR

Prof. Dr. Hasan KARABAY
Danışman, **Kocaeli Üniversitesi** .....

Doç. Dr. Müslüm ARICI
Jüri Üyesi, **Kocaeli Üniversitesi** .....

Prof. Dr. Mehmet ŞAHİN
Jüri Üyesi, **İstanbul Teknik Üniversitesi** .....

Tezin Savunulduğu Tarih: 17.01.2020

ÖNSÖZ VE TEŞEKKÜR

Mühendislik tasarımlarında simülasyon yazılımları sıkça kullanılmakta ve öngörülerde bulunarak tasarımı şekillendirmeye yardımcı olmaktadır. Bu simülasyon yazılımlarında çözülen denklemler farklı fiziksel büyüklüklerin korunumunu ifade etsede temelde benzer yapıdaki denklemleri çözmekte ve sonuçları benzer yöntemlerle kullanıcıya sunmaktadır. Bahsedilen denklemler genel bir ifade ile literatürde "Genel Korunum Denklemi" olarak geçmektedir. Bu tez çalışması, bahsedilen genel korunum denklemini yorumlayan, modelleyen ve çözen bir yazılım kütüphanesinin geliştirilmesi adına çalışılmış ve geliştirilen yazılımsal yapılar bu dokümanda verilmiştir. Bu tez ile yaptığım çalışmalar bundan sonra yapacağım akademik çalışmaların temeli niteliğindedir.

Yüksek öğrenim sürem boyunca düşünce yapısı, bilgi birikimi ve bilgiye olan düşkünlüğü ile örnek aldığım danışman hocam Sn. Prof. Dr. Hasan KARABAY'a desteğini, tecrübesini ve bilgi birikimi esirgememesi, lisans döneminden bu yana daha iyiye doğru ilerlemem adına tavsiyelerde bulunmasından dolayı çok teşekkür ederim.

Verdiği destek ile tez çalışmama katkıda bulunan MDS Motor'a, kurucusu Sn. Doç. Dr. Metin AYDIN'a ve yardımları ile katkıda bulunan çalışma arkadaşım Murat ÖNSAL'a ayrıca teşekkürlerimi sunarım.

Her konuda olduğu gibi tez çalışmam süresince gösterdiği sabrı, ilgisi, desteği ve sevgileri sebebiyle eşim Meltem CUMHUR'a ve varlığıyla beni mutlu eden kızım Elisa CUMHUR'a minnetlerimi sunarım.

Ocak - 2020 Barış CUMHUR

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

$\mathbf{A}$	: Bir lineer sistemin katsayı matrisi
B	: Akış alanının bir özelliği
b	: Akış alanının bir intensif özelliği
$\mathbf{b}$	: Bir lineer sistemin sonuç vektörü
C	: Ağ yapısının bir elemanı
E	: Toplam enerji, (J)
e	: Toplam enerji kütlesel yoğunluğu, (J/kg)
F	: Ağ yapısının bir elemanının komşu elemanı
f	: Ağ yapısının bir elemanının komşusuyla ortak yüzü
m	: Kütle, (kg)
N	: Ağ yapısının bir elemanının komşu elemanları
$\mathbf{n}$	: Normal vektör
s	: Yüzey alanı
T	: Sıcaklık, (°C ya da K)
t	: Zaman, (s)
û	: İç enerji kütlesel yoğunluğu, (J/kg)
V	: Hacim, (m^3)
$\mathbf{v}$	: Hız vektörü, (m/s)
ρ	: Yoğunluk, (kg/m^3)
μ	: Dinamik viskozite, (Pa.s)
λ	: Yığın(bulk) viskozite, (Pa.s)
$\boldsymbol{\tau}$	: Gerilme tensörü, (Pa)
$\dot{\mathbf{q}}_s$	: Isı akısı, (W/m^2)
$\dot{q}_v$	: Isı kaynağı, (W/m^3)
Q^T	: Sıcaklığa bağlı kaynak terimi, (W/m^3)
$\mathbf{Q}^{\mathbf{v}}$	: Hıza bağlı kaynak terimi, (kg/m^2s^2)
Ω	: Problem bölgesi
Ω_i	: Problem alt bölgesi
Γ^{ϕ}	: Difüzyon katsayısı

Kısaltmalar

3B	: 3 Boyutlu
3D	: 3 Dimensional (3 Boyutlu)
CUDA	: Compute Unified Device Architecture (Hesaplama Birleşik Aygıt Mimarisi)
CPU	: Central Processing Unit (Merkezi İşleme Birimi)
CV	: Control Volume (Kontrol Hacmi)
FVM	: Finite Volume Method (Sonlu Hacimler Metodu)

GPU	: Graphics Processing Unit (Grafik İşleme Birimi)
MV	: Material Volume (Maddesel Hacim)
RAM	: Random Access Memory (Rastgele Erişimli Bellek)
RTT	: Reynolds Transport Teoremi
RMS	: Root Mean Square (kök kare ortalama)
SHM	: Sonlu Hacimler Metodu
STL	: Standard Template Library (Standart Taslak Kütüphane)
VRAM	: Video Random Access Memory (Video Rastgele Erişimli Bellek)

GENEL KORUNUM DENKLEMİNİ SONLU HACİMLER METODU İLE MODELLEYEN VE HETEROJEN HESAPLAMA TEKNOLOJİSİNİ DESTEKLEYEN YAZILIMIN GELİŞTİRİLMESİ

ÖZET

Bu çalışmada ağ yapısını dışarıdan okuyan, okunan ağ yapısı için yazılan bir korunum denklemini yorumlayan, ayrıklaştıran ve ayrıklaştırılmış haliyle çözen bir yazılım ve yazılım kütüphanesi geliştirilmiştir. Bu yazılım ve yazılım kütüphanesi, dışarıdan okunan ağ yapısı verisi üzerinde gezme ve işleme bakımından birçok kolaylık sunmaktadır. Kütüphanenin dışarıdan okuyacağı ağ yapısı, yapısal olmayan(unstructured) bir ağ yapısı formunda desteklenmektedir. Kütüphaneye, statik olarak sembolik denklem ifade edebilmeyi sağlayan modüller geliştirilmiştir ve denklemlerin yazılımın derlenmesi aşamasında yorumlanması ile SHM ayrıklaştırma şemaları arasından denkleme uygun kod yolunun seçilmesi sağlanmıştır. Ayrıklaştırma, interpolasyon, gradyen yapımı vb. gibi şemaların modüler yapıda olması sebebiyle yeni şemaların kütüphaneye eklenmesi oldukça kolaydır. Bir korunum denklemi ve sınır koşulları için derlenen yazılımın, çalışma esnasında verilen bir ağ yapısı üzerinde korunum denklemini ve sınır koşullarını ayrıklaştırması sonucu ortaya çıkan lineer denklem sisteminin katsayı matrisi, halkalı seyrek matris adı verilen bir yapı ile verimli bir şekilde bellekte depolanabilmektedir. Ayrıca kütüphane iteratör sistemine ve iteratörlerle beraber kullanılan bir algoritma sistemine sahip olduğundan, veriler ekran kartı belleğinde depolandığı veya ekran kartına aktarıldığı taktirde, veriler üzerinde yapılacak işlemleri ekran kartında paralel olarak yürütmekte mümkündür. Sonuç olarak geliştirilen bu yazılım ve kütüphane, yüksek çarpıklık ve ortogonal olmayan, yüksek ortogonal kaliteye sahip olan ve yüksek çözünürlüklü yüksek ortogonal kaliteye sahip ağ yapıları ile bir difüzyon problemi için test edilmiş ve doğrulanmıştır.

Anahtar Kelimeler: CUDA, C++, Genel Korunum Denklemi, Heterojen Hesaplama, Sonlu Hacimler Metodu.

DEVELOPMENT OF A SOFTWARE MODELING THE GENERAL TRANSPORT EQUATION WITH FINITE VOLUME METHOD AND SUPPORTING HETEROGENEOUS COMPUTING TECHNOLOGY

ABSTRACT

In this study, a software library and a software has been developed that reads the mesh from a file, interprets a conservation equation written for the mesh being read, discretize and solves it in its discrete form. The software and the software library offers many convenience in terms of navigating and processing the mesh data read from an external file. The mesh that the library will read is supported in the form of an unstructured mesh. Modules that allow to express the symbolic equations statically have been developed, and the code path suitable for the equation discretization will be selected from the FVM discretization schemes by interpreting the equations during the compilation of the software. It is very easy to add new discretization schemes, interpolation schemes, gradient constuction schemes to the library since these schemes implemented in a modular structure. The sparse coefficient matrix of the linear equation system resulting from the discretization of a conservation equation and boundary conditions on a given mesh can be stored efficiently in memory with a structure called cyclic sparse matrix. Also, since the library has an iterator system and an algorithm system used with iterators, it is possible to execute the computing code in parallel, if the data is stored in the VRAM or transferred to the VRAM. As a result, this software and library has been tested and validated for a diffusion problem with meshes which have high skewness and low orthogonal quality, high orthogonal quality and, high resolution high orthogonal quality.

Keywords: CUDA, C++, General Transport Equation, Heterogenous Computing, Finite Volume Method.

GİRİŞ

Fiziksel olayları ifade eden matematiksel denklemler, mühendislik tasarımlarının temelini oluşturur. Bu matematiksel ifadelerden çoğunun analitik çözümü yoktur ve var olan çözümler bazı özel durumlar için geçerlidir. Ancak bir mühendislik tasarımının yapılabilmesi için bu matematiksel denklemlerin özel durumlar için yapılmış analitik çözümleri yeterli değildir ve karmaşık geometriler üzerinde denklemlerin çözümleri aranır. Böyle durumlarda kullanmak adına yaklaşımlar geliştirilmiş, bu yaklaşımlar ile analitik çözüme yakınsak nümerik çözüm arayan yazılımlar geliştirilmiştir.

Bahsedilen fiziksel olayları ifade eden denklemlerden birkaçı da Reynolds Transport Denklemi'nden de türetilebilen kütle, momentum ve enerji korunumu denklemleridir ve her biri genel korunum denklemi formundadır. Mangani, Moukalled ve Darwish [1]'de de denklemlerin formundaki benzerlikten bahsetmiştir ve skaler bir değişken için genel korunum denklemini vermiştir. Bu çalışmada da denklemlerdeki form benzerliğinden esinlenilmiştir. Geliştirilen kütüphanenin bu denklemleri tanımlı oldukları bölgeler ile beraber ifade edebilecek, yorumlayacak, çözebilecek ve gerektiğinde yeni formların ve yeni yöntemlerin eklenmesine olanak sağlayacak bir kütüphane olması amaçlanmıştır. Denklemlerin yorumlanması ve ayrıklaştırılması kısmında Sonlu Hacimler Metodu tabanlı ayrıklaştırma şemaları kütüphaneye entegre edilsede Sonlu Elemanlar Metodu ve diğer metodlarında yazılıma entegre edilebilmesi amacıyla esneklik sunulmuştur.

Ayrıca günümüzde ekran kartlarının sunduğu hesaplama kapasitesi göz ardı edilemeyeceğinden geliştirilen kütüphanenin ekran kartında hesaplamaya destek vermesi de amaçlanmıştır. Programlama dilinin sunduğu olanaklar, ekran kartı kütüphanelerini kullanabilme ve derleyicisinin matematiksel işlem hızına optimize bir makine kodu üretebilmesi gibi konular göz önüne alınarak C++ programlama dili kütüphane için uygun bir dil olarak düşünülmüştür.

Çalışmanın son aşamasında zamandan bağımsız kaynak terimli difüzyon denklemi farklı sınır koşulları için test amaçlı olarak basit bir geometri için çözülmüş ve ticari bir yazılım olan Fluent yazılımından elde edilen sonuçlar ile karşılaştırılmıştır. Seçilen basit geometri yüksek çarpıklık ve düşük ortogonalliğe, yüksek ortogonalliğe ve yüksek çözünürlüklü yüksek ortogonalliğe sahip yapısal olmayan bir ağ yapıları ile hazırlanmıştır. Testler sonucunda Fluent yazılımından elde edilen sonuçlara yakın sonuçlar elde edilmiş ve aradaki farklılıkların sebebi irdelenmiştir.

1. GENEL BİLGİLER

Giriş kısmında bahsedilen genel korunum denklemi bu bölümde verilmiştir. Ancak öncesinde genel bilgilerin verilmesi faydalı olacaktır. Bölüm 1.1'de vektör hesabıyla ilgili hatırlatıcı bilgiler ve Bölüm 1.2'de geometrik hesaplarla ilgili bilgiler verilmiştir. Genel korunum denklemi ile ilgili bilgiler ise Bölüm 1.3'de verilmiştir.

1.1. Vektör Hesabı

Fiziksel problemi ifade eden denklemlerin skaler, vektörel ve tensörel ifadeler içermesi açısından vektör hesabının (vector calculus) anlaşılması gereklidir. Konuyla ilgili daha detaylı bilgiye Marsden ve Tromba'nın [2]'de verilen kitabından ulaşılabilir. Bu dokümanda skaler büyüklükler normal yazı tipiyle, vektörel büyüklükler kalın Latin harfleriyle, tensörel büyüklükler ise kalın Yunan harfleriyle gösterilmiştir. Ayrıca tüm matrislerde kalın büyük harflerle gösterilmiştir.

Vektörlerin birbirleri üzerine projeksiyonu olarak tanımlanabilecek iç çarpım operatörü,

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \|\mathbf{v}_1\|\|\mathbf{v}_2\|\cos(\mathbf{v}_1, \mathbf{v}_2) \tag{1.1}$$

olarak tanımlanır. Denklem (1.1)'de $\|\mathbf{v}_1\|$ ve $\|\mathbf{v}_2\|$ vektörlerin şiddetini göstermekle birlikte Kartezyen koordinatlarda $\mathbf{i}$, $\mathbf{j}$ ve $\mathbf{k}$ birim vektörleri ile gösterilen herhangi bir $\mathbf{v} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$ vektörünün şiddeti(uzunluğu),

$$\|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{a^2 + b^2 + c^2} \tag{1.2}$$

ile hesaplanır. Denklem (1.2)'de uzunluğu verilen vektörün normali ise,

$$\mathbf{e}_v = \frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{1.3}$$

şeklinde hesaplanan bir birim vektördür.

Vektörlerin iç çarpımları (veya projeksiyonları) skaler bir büyüklük iken çapraz çarpımları ise çarpılan iki vektöre ortogonal bir vektördür. Çapraz çarpımın sonuç vektörü yönü sağ el kuralı ile belirlenebiliyor olsa da Kartezyen koordinat eksenleri üzerindeki birim vektörler cinsinden çarpım sonucu yönleri,

$$\mathbf{i}\times\mathbf{i}=\mathbf{j}\times\mathbf{j}=\mathbf{k}\times\mathbf{k}=0 \tag{1.4}$$

$$\mathbf{i}\times\mathbf{j}=-\mathbf{j}\times\mathbf{i}=\mathbf{k} \tag{1.5}$$

$$\mathbf{j}\times\mathbf{k}=-\mathbf{k}\times\mathbf{j}=\mathbf{i} \tag{1.6}$$

$$\mathbf{k}\times\mathbf{i}=-\mathbf{i}\times\mathbf{k}=\mathbf{j} \tag{1.7}$$

şeklindedir. Denklem (1.4) - (1.7)'de verilen ilişkiler kullanılarak iki vektörün çapraz çarpımı,

$$\begin{aligned}\mathbf{v}_1\times\mathbf{v}_2 &= \left(u_1\mathbf{i}+v_1\mathbf{j}+w_1\mathbf{k}\right)\times\left(u_2\mathbf{i}+v_2\mathbf{j}+w_2\mathbf{k}\right)\\ &= u_1u_2\mathbf{i}\times\mathbf{i}+u_1v_2\mathbf{i}\times\mathbf{j}+u_1w_2\mathbf{i}\times\mathbf{k}+v_1u_2\mathbf{j}\times\mathbf{i}+v_1v_2\mathbf{j}\times\mathbf{j}\\ &\quad +v_1w_2\mathbf{j}\times\mathbf{k}+w_1u_2\mathbf{k}\times\mathbf{i}+w_1v_2\mathbf{k}\times\mathbf{j}+w_1w_2\mathbf{k}\times\mathbf{k}\\ &= \left(v_1w_2-w_1v_2\right)\mathbf{i}+\left(w_1u_2-u_1w_2\right)\mathbf{j}+\left(u_1v_2-v_1u_2\right)\mathbf{k}\end{aligned} \tag{1.8}$$

olarak hesaplanır. Denklem (1.8), aynı zamanda determinant formunda da,

$$\mathbf{v}_1\times\mathbf{v}_2=\begin{vmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k}\\ u_1 & v_1 & w_1\\ u_2 & v_2 & w_2\end{vmatrix}=\begin{bmatrix}v_1w_2-w_1v_2\\ w_1u_2-u_1w_2\\ u_1v_2-v_1u_2\end{bmatrix} \tag{1.9}$$

olarak verilebilir. Diğer bir tür çarpım ise cebirsel çarpma işlemidir. Literatürde tensör çarpımı olarak da geçmektedir ve aşağıdaki gibidir;

$$\begin{aligned}\boldsymbol{\tau}=\mathbf{v}_1\mathbf{v}_2 &= \left(u_1\mathbf{i}+v_1\mathbf{j}+w_1\mathbf{k}\right)\left(u_2\mathbf{i}+v_2\mathbf{j}+w_2\mathbf{k}\right)\\ &= u_1u_2\mathbf{ii}+u_1v_2\mathbf{ij}+u_1w_2\mathbf{ik}+v_1u_2\mathbf{ji}+v_1v_2\mathbf{jj}\\ &\quad +v_1w_2\mathbf{jk}+w_1u_2\mathbf{ki}+w_1v_2\mathbf{kj}+w_1w_2\mathbf{kk}\\ &= \begin{bmatrix}u_1u_2 & u_1v_2 & u_1w_2\\ v_1u_2 & v_1v_2 & v_1w_2\\ w_1u_2 & w_1v_2 & w_1w_2\end{bmatrix}\end{aligned} \tag{1.10}$$

Denklem (1.10), matris iç çarpımı formunda,

$$\boldsymbol{\tau} = \mathbf{v}_1\mathbf{v}_2^{\mathrm{T}} = \begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix}\begin{bmatrix} u_2 & v_2 & w_2 \end{bmatrix} = \begin{bmatrix} u_1u_2 & u_1v_2 & w_1w_2 \\ v_1u_2 & v_1v_2 & v_1w_2 \\ w_1u_2 & w_1v_2 & w_1w_2 \end{bmatrix} \tag{1.11}$$

şeklinde de gösterilebilir. Genel korunum denkleminde kullanılan bir vektörde nabla operatörüdür. Bu operatör vektörel bir türev operatörüdür ve Kartezyen koordinatlarda,

$$\nabla = \frac{\partial}{\partial \mathrm{x}}\mathbf{i} + \frac{\partial}{\partial \mathrm{y}}\mathbf{j} + \frac{\partial}{\partial \mathrm{z}}\mathbf{k} \tag{1.12}$$

şeklinde tanımlanır. Denklem (1.12) verilen operatör skaler veya vektörel bir alanın üzerine çarpım ile uygulandığında değişkenin gradyeni olarak ifade edilir;

$$\mathrm{grad}\,\mathrm{f} = \nabla \mathrm{f} = \frac{\partial \mathrm{f}}{\partial \mathrm{x}}\mathbf{i} + \frac{\partial \mathrm{f}}{\partial \mathrm{y}}\mathbf{j} + \frac{\partial \mathrm{f}}{\partial \mathrm{z}}\mathbf{k} \tag{1.13}$$

$$\mathrm{grad}\,\mathbf{f} = \nabla \mathbf{f} = \begin{bmatrix} \frac{\partial \mathrm{f_x}}{\partial \mathrm{x}} & \frac{\partial \mathrm{f_y}}{\partial \mathrm{x}} & \frac{\partial \mathrm{f_z}}{\partial \mathrm{x}} \\ \frac{\partial \mathrm{f_x}}{\partial \mathrm{y}} & \frac{\partial \mathrm{f_y}}{\partial \mathrm{y}} & \frac{\partial \mathrm{f_z}}{\partial \mathrm{y}} \\ \frac{\partial \mathrm{f_x}}{\partial \mathrm{z}} & \frac{\partial \mathrm{f_y}}{\partial \mathrm{z}} & \frac{\partial \mathrm{f_z}}{\partial \mathrm{z}} \end{bmatrix} \tag{1.14}$$

Denklem (1.13) ve Denklem (1.14)'den anlaşılacağı gibi bir skaler alanın gradyeni pozisyonla değişimi ifade etmektedir. Gradyenin bir birim vektör yönündeki projeksiyonu ise o yöndeki değişimi vermektedir. İfade etmek gerekirse,

$$\frac{\mathrm{df}}{\mathrm{dr}} = \nabla \mathrm{f} \cdot \mathbf{e}_{\mathrm{r}} = \left\| \nabla \mathrm{f} \right\| \cos\left(\nabla \mathrm{f}, \mathbf{e}_{\mathrm{r}} \right) \tag{1.15}$$

şeklindedir. Nabla operatörünün bir vektör ya da tensör alanı üzerine iç çarpım ile uygulanması ise bahsedilen vektör ya da tensör alanının divörjansı olarak isimlendirilir. Bir bölge üzerinde tanımlı vektör alanının divörjansının bölgede integrali, bölgedeki vektör alanının bölgeye net giriş – çıkış miktarını verir. Divörjans, bir vektör alanı üstünde Denklem (1.12)'nin iç çarpım ile uygulanmasıyla,

$$\operatorname{div}\mathbf{v} = \nabla \cdot \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{1.16}$$

şeklinde Kartezyen koordinatlarda yazılabilir. Divörjans operatörü bir gradyen üzerine uygulandığında ise işlem bir skaler ya da vektör alanının laplasyeni olarak adlandırılır ve bir skaler değişken üstünde,

$$\nabla \cdot \nabla f = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \tag{1.17}$$

şeklinde Kartezyen koordinatlar için yazılır. Bir vektör alanının rotasyoneli ise Denklem (1.18)'deki gibidir;

$$\operatorname{curl}\mathbf{v} = \nabla \times \mathbf{v} = \begin{bmatrix} \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} \\ \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \end{bmatrix} \tag{1.18}$$

Son olarak bu kısımda bahsedilmesi gereken ve sıkça kullanılan bir teoremde divörjans teoremidir. Bu teorem Gauss teoremi olarakta bilinmektedir. Teoremin bahsettiği dönüşüm,

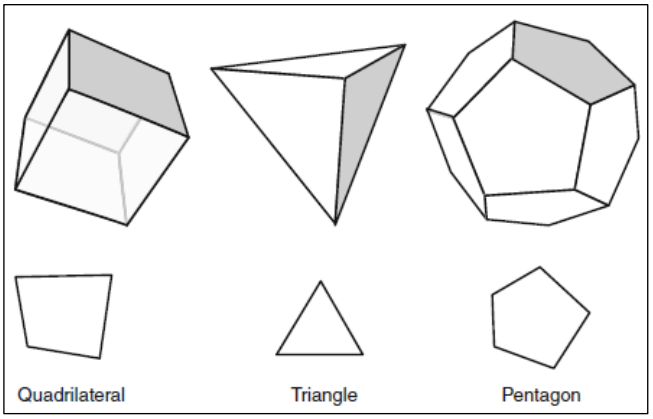
$$\int_V (\nabla \cdot \mathbf{v})\, dV = \int_{\partial V} \mathbf{v} \cdot \mathbf{n}\, ds \tag{1.19}$$

olarak verilmiştir. Burada V bir hacimsel bölgeyi, ∂V bölgeyi çevreleyen yüzeyi, $\mathbf{n}$ bu yüzeyin normal vektörlerini ve ds ise bu yüzey üzerinde diferansiyel bir alanı ifade etmektedir. Denklem (1.19) genel korunum denklemini ayrıklaştırılmasında kullanılacaktır.

1.2. Geometri

Bahsedilen genel korunum denklemi bir geometri üzerinde tanımlandığında ve sınır koşulları verildiğinde bir problemi ifade etmektedir. Problemin analitik çözümü yerine yakınsak çözümü aranırken verilen geometrik bölge alt bölgelere ayrılır ve yaklaşımlar bu bölgeler üzerinden uygulanır. Bu alt bölgeler üstünde geometrik hesaplamaların

yapılması gerekeceğinden bu kısımda hacim hesabı, alan hesabı, merkez ve alan merkezi(centroid) gibi hesaplarla ilgili ifadeler verilmiştir.



Şekil 1.1. Bir ağ yapısında karşılaşılabilecek 3B ve 2B elemanlardan bazıları[1]

3B ağ yapısında bir elemanın bir yüzündeki yüzey vektörü hesaplanırken bu yüz üçgenlere ayrılabilir ve üçgenlerin toplam yüzey vektörü, o yüzün yüzey vektörünü; herhangi bir üçgenin yüzey vektörünün normali ise yüzün normali doğrultusunu verecektir. 3B uzayda köşe vektörleri $\mathbf{r}_1$, $\mathbf{r}_2$ ve $\mathbf{r}_3$ ile verilen bir üçgenin(dolayısıyla $\mathbf{r}_1$, $\mathbf{r}_2$ ve $\mathbf{r}_3$ birbirinden farklı vektörlerdir) yüzey vektörü,

$$\mathbf{s} = \frac{1}{2}\left(\mathbf{r}_2 - \mathbf{r}_1\right) \times \left(\mathbf{r}_3 - \mathbf{r}_1\right) \tag{1.20}$$

ile hesaplanabilir. Konveks bir dörtgenin yüzey vektörü köşegen vektörleri olan $\mathbf{p}$ ve $\mathbf{q}$ cinsinden,

$$\mathbf{s} = \frac{1}{2}\mathbf{p} \times \mathbf{q} \tag{1.21}$$

ile hesaplanabilse de üçgenden daha fazla kenara sahip herhangi bir konveks poligonun yüzey vektörü genel olarak aşağıdaki gibi,

$$\mathbf{s}_{\text{poligon,n}} = \sum_{i=1}^{n} \mathbf{s}_i \tag{1.22}$$

$$\mathbf{s}_i = \frac{1}{2}\left(\mathbf{r}_i - \mathbf{r}_c\right) \times \left(\mathbf{r}_i' - \mathbf{r}_c\right) \tag{1.23}$$

ile hesaplanabilir. Denklem (1.23)'de $\mathbf{r}_c$ poligon içerisinde keyfi bir noktanın (tercihen merkezin), $\mathbf{r}_i$ poligona ait bir köşenin ve $\mathbf{r}_i^{'}$ $\mathbf{r}_i$ köşesine hep aynı yöndeki komşu köşenin konum vektörleridir.

Bilindiği üzere bir poligonun merkezi ile alan merkezi bazı özel poligonlar için çakışıktır ve bu özel poligonlardan biri de üçgendir. Ancak herhangi bir poligonun alan merkezi yine poligonun alt üçgenlere ayrılması yoluyla hesaplanabilir. Üçgenin merkezi ve alan merkezi,

$$\mathbf{x}_C = \mathbf{x}_G = \frac{1}{3}\left(\mathbf{r}_1 + \mathbf{r}_2 + \mathbf{r}_3\right) \tag{1.24}$$

şeklinde yazılabilir. Denklem (1.24)'de $\mathbf{x}_C$ alan merkezini, $\mathbf{x}_G$ merkezi göstermektedir. Bir poligonun alan merkezini hesaplamak için ise poligon içindeki keyfi bir nokta etrafında alt üçgenlere ayrılır. Bu keyfi nokta poligon merkezi olarak,

$$\mathbf{x}_{G,poligon,n} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{r}_i \tag{1.25}$$

şeklinde seçilebilir ve $\mathbf{x}_{G,poligon,n}$ etrafındaki alt üçgenlerin alan merkezleri $\mathbf{x}_{C,i}$, alanları A_i olmak üzere poligonun alan merkezi,

$$\mathbf{x}_{C,poligon,n} = \frac{\sum_{i=1}^{n}\mathbf{x}_{C,i}A_i}{\sum_{i=1}^{n}A_i} \tag{1.26}$$

ile hesaplanabilir.

Hacimsel hesaplamalarda polihedron, piramitlere ayrılarak hesaplanması istenen büyüklük hesaplanabilir. Bu noktada verilmesi gereken piramitin hacim merkezidir ve

$$\mathbf{x}_{C,piramit} = \frac{3}{4}\mathbf{x}_{C,taban} + \frac{1}{4}\mathbf{r} \tag{1.27}$$

olarak verilebilir. Denklem (1.27)'de $\mathbf{r}$ piramitin tepe köşesini gösteren konum vektörüdür.

1.3. Genel Korunum Denklemi

Bu kısımda kütle, momentum ve enerji korunum denklemleri kısaca verilecek olup son olarak tüm denklemler genel korunum denklemi ile özetlenecektir. Bu kısımda bahsedilen Reynold Transport Teoremi (RTT) [1] ve [3] bunlardan birkaçı olmak üzere birçok kaynaktan detaylı incelenebilir.

B akış alanının herhangi bir özelliği (kütle, momentum, enerji, vb.), b ise buna bağlı $b = dB/dm$ ile verilen intensif bir özellik olmak üzere RTT,

$$\left(\frac{dB}{dt}\right)_{MV} = \frac{d}{dt}\left(\int_{V(t)} b\rho dV\right) + \int_{S(t)} b\rho \mathbf{v}_r \cdot \mathbf{n} dS \tag{1.28}$$

olarak [1] ve [3]'de verilmiştir. Denklem (1.28)'de ρ özkütleyi, dV MV(Material Volume – Maddesel Hacim)'de bir diferansiyel hacmi, $V(t)$ hacmin zamana bağlı değişimini, $S(t)$ MV yüzeyinin zamana bağlı değişimini, dS MV yüzeyinden bir diferansiyel alanı, $\mathbf{n}$ MV yüzeyinin normalini ve $\mathbf{v}_r$'de MV'nin bağıl hızını ifade etmektedir. Denklem (1.28)'in sol tarafı Lagrange yaklaşımı ile ifade edilen özellikleri gösterirken sağ taraf ise Euler yaklaşımındaki karşılığını vermektedir. Bu bakımdan RTT Lagrange ve Euler yaklaşımları arasında dönüşüm niteliğindedir. Denklem (1.28) 'de Maddesel hacmin (ya da kontrol hacminin) zamana göre değişmediği ve hareket etmediği kabul edilirse ve divörjans teoremi uygulanırsa,

$$\left(\frac{dB}{dt}\right)_{MV} = \int_V \left(\frac{\partial}{\partial t}(b\rho) + \nabla \cdot (b\rho \mathbf{v})\right) dV \tag{1.29}$$

olur. Burada hız vektörü $\mathbf{v} = u\mathbf{i} + v\mathbf{j} + w\mathbf{k}$ olarak tanımlanmıştır. $B = m$ seçilmesi durumunda $b = 1$ olur ve kütle korunum denklemi akı formunda,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1.30}$$

şeklindedir. Burada denklemin sağ tarafı Lagrange yaklaşımında sistemin kütlesi zamana göre değişmeyeceğinden dolayı sıfırdır.

$\mathbf{B} = m\mathbf{v}$ seçilmesi durumunda ise $\mathbf{b} = \mathbf{v}$ olur ve momentumun korunumu,

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = \mathbf{f} \tag{1.31}$$

denklemi ile ifade edilir. Denklem (1.31)'de $\mathbf{f}$ diferansiyel kontrol hacmi üzerine etkiyen kuvvetleri göstermektedir. Kontrol hacmi üzerine etkiyen normal ve teğetsel gerilmeler göz önüne alındığında, Denklem (1.31)'in,

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \mathbf{f}_{diger} \tag{1.32}$$

şeklinde olduğu [1]'de de verilmiştir. Denklem (1.32)'de p basıncı $\boldsymbol{\tau}$ ise gerilme tensörünü ifade etmektedir. Ayrıca $\mathbf{f}_{diger}$ terimi manyetik kuvvetler, yer çekimi vb. diğer kuvvetleri göstermektedir. Newtonian bir akışkan için gerilme tensörü,

$$\boldsymbol{\tau} = \mu\left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right) + \lambda(\nabla \cdot \mathbf{v})\mathbf{I} \tag{1.33}$$

olarak [1]'de gösterilmiştir. Denklem (1.33) da μ moleküler dinamik viskoziteyi, λ yığın(bulk) viskoziteyi ifade eder ve genellikle $\lambda = -(2/3)\mu$ şeklinde verilir. Denklem (1.33), Denklem (1.32)'de yerine yazıldığında ise,

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla p + \nabla \cdot (\mu \nabla \mathbf{v}) + \underbrace{\nabla \cdot \left(\mu (\nabla \mathbf{v})^T\right) + \nabla \cdot (\lambda \nabla \cdot \mathbf{v}) + \mathbf{f}_b}_{\mathbf{Q}^v} \tag{1.34}$$

$$\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = \nabla \cdot (\mu \nabla \mathbf{v}) - \nabla p + \mathbf{Q}^v \tag{1.35}$$

olur. $\mathbf{Q}^v$ terimi kaynak terimi olarak yazılmıştır. Enerji korunumu için ise E toplam enerji, e toplam enerji kütlesel yoğunluğu(specific total energy) olmak üzere $B = E = me$ olarak seçilmelidir. Termodinamiğin birinci kanununun uygulanması ile E eşitliğinin elde edilmesi aşamaları atlanarak enerji denkleminin iç enerji kütlesel yoğunluğu (specific internal energy) cinsinden ifadesi,

$$\frac{\partial}{\partial t}(\rho \hat{u}) + \nabla \cdot (\rho \mathbf{v} \hat{u}) = -\nabla \cdot \dot{\mathbf{q}}_s - p\nabla \cdot \mathbf{v} + \boldsymbol{\tau} : \nabla \mathbf{v} + \dot{q}_v \tag{1.36}$$

şeklinde Mangani, Moukalled ve Darwish'in [1] ile verilen kitabında ifade edilmiştir. İç enerji kütlesel yoğunluğu $\hat{u}$'nun sıcaklık (T) ve basınç (p) cinsinden ifadesi ile Newtonian bir akışkan için,

$$\boldsymbol{\tau} : \nabla \mathbf{v} = \lambda \underbrace{\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right)^2}_{\Psi} + \mu \underbrace{\left(\begin{array}{l} 2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2 + 2\left(\frac{\partial w}{\partial z}\right)^2 \\ + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)^2 \end{array} \right)}_{\Phi} \tag{1.37}$$

olmak üzere enerji denklemi,

$$\frac{\partial}{\partial t}\left(\rho c_p T\right) + \nabla \cdot \left(\rho c_p \mathbf{v} T\right) = \nabla \cdot \left(k \nabla T\right) + \underbrace{\rho T \frac{Dc_p}{Dt} - \left(\frac{\partial \left(\mathrm{Ln}\,\rho\right)}{\partial \left(\mathrm{Ln}\,T\right)} \right)_p \frac{Dp}{Dt} + \lambda \Psi + \mu \Phi + \dot{q}_v}_{Q^T} \tag{1.38}$$

şeklinde [1]'de verilmiştir. Denklem (1.38)'de D/Dt olarak görülen türev operatörü, maddesel türev operatörüdür ve zamana herhangi bir φ alanı için,

$$\frac{D\varphi}{Dt} = \frac{\partial \varphi}{\partial t} + \left(\mathbf{v} \cdot \nabla\right)\varphi \tag{1.39}$$

şeklinde tanımlı bir operatördür. Denklem (1.38), kaynak terimiyle ifade edilmesi halinde,

$$\frac{\partial}{\partial t}\left(\rho c_p T\right) + \nabla \cdot \left(\rho c_p \mathbf{v} T\right) = \nabla \cdot \left(k \nabla T\right) + Q^T \tag{1.40}$$

şeklinde yazılabilir. Kütle korunumu veren Denklem (1.30), Newtonian bir akışkan için momentum korunumu veren Denklem (1.35) ve Newtonian bir akışkan için enerji korunumu veren Denklem (1.40)'nın benzer formlarda oldukları açıkça görülmektedir. Burdan yola çıkarak genel korunum denklemi,

$$\frac{\partial}{\partial t}\left(\rho \phi\right) + \nabla \cdot \left(\rho \mathbf{v} \phi\right) = \nabla \cdot \left(\Gamma^{\phi} \nabla \phi\right) + Q^{\phi} \tag{1.41}$$

ile ifade edilir ve [1]'de de bu şekilde verilmiştir. Denklem (1.41)'de denklemin sol tarafındaki ilk terim zamana bağlı terim, ikinci terim konveksiyon terimi; denklemin sağ tarafındaki ilk terim difüzyon terimi, ikinci terimi de kaynak terimidir.

2. SONLU HACİMLER METODU

Denklem (1.41)'nin verilen sınır koşulları ve geometri ile bilgisayar ortamında nümerik modellemesini sağlayabilmek amacıyla kullanılabilecek yöntemlerden biri Sonlu Hacimler Metodu (SHM)'dir. Bu bölümde, ayrıklaştırılmış ifade elde edilecek olup yazılımda bu ifadeler parçalar halinde kullanılacaktır.

Literatürde, denklem (1.41)'de transport edilen değişken olarak ifade edilen ϕ değişkeni için ağ yapısı elemanları üzerinde farklı konumlar seçilmiştir. ϕ skaler değişkeninin tanımlandığı yere göre hücre merkezli SHM (cell – centered FVM) ve köşe tabanlı SHM (vertex-based FVM) bu konumlar ile geliştirilen SHM şemalarına örnektir. [4], [5]'de köşe tabanlı SHM ile ilgili daha detaylı bilgiler bulunabilir. İki yönteminde SHM ayrıklaştırma şemasının geliştirilmesinde avantajlarının ve dezavajlarının olduğundan Mangani, Moukalled ve Darwish [1]'de bahsetmiştir.

Bu tezin içerdiği çalışmalar kapsamında geliştirilen kütüphanede taslak sınıflar köşe tabanlı SHM'nin implementasyonunu desteklese de öncelikle hücre merkezli SHM tercih edilmiştir.

2.1. Problem Alanı

Ayrıklaştırma şemalarının ayrıntılarını vermeden önce problem geometrisiyle ilgili detayların verilmesi gerekir. Yazılımın geliştirilmesi aşamasında ağ yapısını dışardan alması ve kendine özgü bir veri formatında bellekte işlenmeye hazır bir şekilde depolaması sağlanmıştır. Fluent yazılımından alınacak sonuçların geliştirilen yazılım ile kıyaslanabilmesi adına dışardan okunan ağ yapısı dosyası formatı Fluent'in ağ yapısı dosyası formatı olarak seçilmiştir. Bununla ilgili detaylar Bölüm 3.1'de verilmiştir.

Problemin tanımlandığı geometri $\Omega \in \mathbb{R}^3$ ile verilen öklit uzayında tanımlı bir bölge olsun. Ayrıklaştırma adımlarının uygulanabilmesi adına Ω bölgesi, M adet Ω_i alt bölgelerine ayrılsın. Yani,

$$\Omega = \left\{ \Omega_i \middle| i = \overline{1:M}, \forall \Omega_i \in \Omega \right\} \tag{2.1}$$

dır. Denklem (2.1)'deki herhangi bir Ω_i elemanı kolaylık açısından C ile gösterilecektir. C elemanının, Ω_i elemanları arasından komşuları N ile gösterilmek üzere,

$$\begin{aligned} C &= \Omega_i \qquad 1 \le i \le M,\ \Omega_i \in \Omega \\ N &= \left\{ \Omega_i \middle| \partial\Omega_i \cap C \neq \varnothing, i = \overline{1:M}, \Omega_i \neq C, \Omega_i \in \Omega \right\} \end{aligned} \tag{2.2}$$

şeklinde tanımlanır. C elemanının N ile gösterilen komşularından herhangi biri olan N_i ile ortak yüzü,

$$f_i = \partial C \cap \partial N_i \qquad 1 \le i \le \text{count}(N) \tag{2.3}$$

ile gösterilmek üzere bu yüzler herhangi biri f, f ortak yüzüne sahip komşu işe F ile gösterilebilir. C elemanının hiçbir elemanla ortak olmayan herhangi bir b_i ile gösterilen yüzü ise sınır yüzüdür. C elemanının tüm sınır yüzleri $\partial C \cap \partial\Omega$ ile ifade edilebilecekken bu sınır yüzlerden herhangi biri kolaylık açısından b ile gösterilsin. Ve böylelikle ayrıklaştırma aşamasında gerekli bölgeler tanımlanmış olur.

2.2. Yarı Ayrıklaştırılmış Korunum Denklemi

[1]'de de verildiği üzere SHM'de ayrıklaştırma işleminin ilk adımı Denklem (1.41) ile verilen denklemin herhangi bir C elemanında integralini almaktadır. Bu aşama,

$$\int_C \frac{\partial}{\partial t}(\rho\phi)dV + \int_C \nabla\cdot(\rho\mathbf{v}\phi)dV = \int_C \nabla\cdot\left(\Gamma^\phi\nabla\phi\right)dV + \int_C Q^\phi dV \tag{2.4}$$

ile gösterildiği gibidir. Denklem (2.4)'e Denklem (1.19) ile gösterilen Gauss teoremi uygulanırsa,

$$\int_C \frac{\partial}{\partial t}(\rho\phi)dV + \int_{\partial C} \rho\mathbf{v}\phi\cdot\mathbf{n}ds = \int_{\partial C} \Gamma^\phi\nabla\phi\cdot\mathbf{n}ds + \int_C Q^\phi dV \tag{2.5}$$

şekline dönüşecektir. Denklem (2.5)'in zamandan bağımsız hali,

$$\int_{\partial C} \rho \mathbf{v}\phi \cdot \mathbf{n} ds - \int_{\partial C} \Gamma^{\phi} \nabla \phi \cdot \mathbf{n} ds = \int_{C} Q^{\phi} dV \tag{2.6}$$

şeklindedir. Denklem (2.6)'da ilk terim konveksiyon, ikinci terim difüzyon, üçüncü terim ise kaynak terimidir. Denklem (2.6)'nın ayrıklaştırılmış formunun, her bir terimin ayrı ayrı ayrıklaştırılmasından ve bu ayrıklaştırılmaların tekrar tek denklem ile ifadesinden elde edilebileceği görülebilmektedir. Ayrıklaştırılmış terimlere ait denklemler birleştirildiğinde Denklem (2.6),

$$\mathbf{A}\boldsymbol{\varphi} = \mathbf{b} \tag{2.7}$$

formunda olacaktır. Burada $\boldsymbol{\varphi}$ vektörü ϕ alanının ifade edildiği ayrık noktalardaki değerlerini ifade eden vektördür. Denklem (2.6)'deki konveksiyon teriminin ayrıklaştırma şeması geliştirilen kütüphanede kolayca implemente edilebilecekken bu tez kapsamına konveksiyon terimi dahil edilmemiştir. Bölüm 2.3 ve Bölüm 2.4'de difüzyon teriminin ve kaynak teriminin ayrıklaştırılması anlatılmıştır.

2.3. Difüzyon Teriminin Ayrıklaştırılması

Denklem (2.6) ile verilen yarı ayrıklaştırılmış zamandan bağımsız genel korunum denkleminde difüzyon teriminin ayrıklaştırılması için C elemanının her F elemanı ile komşuluğundaki ortak yüzü olan f 'in normali yönündeki gradyenin ve b sınır yüzleri var ise bu yüzlerin normali yönündeki gradyenin bilinmesi gerektiği denklemden görülebilmektedir. Bu noktada C elemanı bir iç eleman olarak düşünülsün. C elemanının sınır eleman olma durumunda uygulanan ayrıklaştırma adımları Bölüm 2.5'de sınır koşulları adı altında açıklanacaktır.

Ağ yapısının ortogonal olması durumunda normal yöndeki gradyen komşu elemanların merkezlerindeki(centroid) ϕ_C ve ϕ_F değerleri cinsinden ifade edilebilecekken, ortogonal olmayan bir ağ yapısı buna ek olarak yüzdeki gradyenide bilmek gereklidir ve Mangani, Moukalled ve Darwish [1]'de bu durumdan bahsetmiştir. Ayrıca ikinci mertebeden bir doğruluk için integrallenecek değerlerin yüz merkezinde (yüz üzerinde tek noktada) ifade edilmesi yeterlidir. Bu durumdan Gauss integrasyonu (Gauss integration) adı altında [1]'de bahsedilmiştir. Dolayısıyla ikinci mertebeden doğrulukla C elemanının herhangi bir f iç yüzü için,

$$\int_f -\Gamma^{\phi}\nabla\phi\cdot\mathbf{n}ds \cong -\Gamma_f^{\phi}\left(\nabla\phi\right)_f\cdot\mathbf{n}_f s_f \tag{2.8}$$

olur. C elemanı bir sınır eleman ise bu elemanın bir b sınır yüzü için,

$$\int_b -\Gamma^{\phi}\nabla\phi\cdot\mathbf{n}ds \cong -\Gamma_b^{\phi}\left(\nabla\phi\right)_b\cdot\mathbf{n}_b s_b \tag{2.9}$$

olacaktır. Denklem (2.8) ve Denklem (2.9)'da ϕ için yapılacak bir değişim profili kabulü ile $\left(\nabla\phi\right)_f\cdot\mathbf{n}_f$ ifadesi ortak yüzde veya $\left(\nabla\phi\right)_b\cdot\mathbf{n}_b$ ifadesi sınır yüzde yazılabilir. [1]'de bahsedildiği üzere değişim profilinin ϕ 'nin fiziksel olmayan değişimlerine izin vermemesi ve örneğin sıcaklık için iki nokta arasında, bu noktalardaki değerlerinden yüksek bir değere sahip olmaması gerekir. Ayrıca lineer bir profil değişimi kabul edilmesi durumunda ağ yapısının çözünürlüğü arttıkça kabul edilen değişim profilinin bir etkisi kalmayacaktır. Bu noktada yapılacak bir lineer değişim kabulünün ortogonal bir ağ yapısı için ikinci mertebeden doğru olduğu Taylor serisi açılımı ile ifade edilecek bir merkezi fark ifadesinde kolayca görülebilir. Ayrıca Denklem (2.8)'da uygulanan yüzün alan merkezinde alınan integrasyon yaklaşımı da ikinci mertebeden doğru olduğundan lineer profil değişimi kabulü bu noktada uygun olacaktır. Ve şuda belirtilmelidir ki; literatürde yüksek mertebeden yaklaşımlar ile ilgili birçok çalışma olmasına karşılık bu yaklaşımlar beraberinde hesaplama maliyeti de getirmektedir.

Lineer profil değişimi kabulü ile ortogonal bir ağ yapısı için Denklem (2.8) şu şekilde ifade edilebilir;

$$\int_f -\Gamma^{\phi}\nabla\phi\cdot\mathbf{n}ds \cong -\Gamma_f^{\phi}\frac{\phi_F-\phi_C}{\delta_f}s_f \tag{2.10}$$

Denklem (2.10)'da δ_f, C ve F komşu elemanlarının 3B durumda hacim merkezleri (centroid) arasındaki mesafedir ve,

$$\delta_f = \left\|\mathbf{r}_F-\mathbf{r}_C\right\| \tag{2.11}$$

şeklinde ifade edilebilir. Bahsedildiği üzere Denklem (2.10) ortogonal ağ yapıları için ikinci mertebeden doğruluğa sahiptir. Ortogonal olmayan ağ yapılarında ise (bkz.

Şekil 2.1) ortak yüzdeki akı yüzü paylaşan elemanların merkezlerindeki ϕ değerleri cinsinden ortak yüzün normali yönünde değil de eleman merkezlerini birleştiren doğru parçasının doğrultusunda yazılabilir. Denklem (2.8) için gereken normal yöndeki akı bilinen gradyen alanından bir düzeltme ile elde edilebilir [1]. Böylelikle $\mathbf{r}_F$ F elemanının merkezi(centroid) ve $\mathbf{r}_C$ C elemanının merkezi olmak üzere F elemanının merkezinden C elemanının merkezini gösteren normal vektörün,

$$\mathbf{e}_f = \frac{\mathbf{r}_F - \mathbf{r}_C}{\left\|\mathbf{r}_F - \mathbf{r}_C\right\|} \tag{2.12}$$

olarak yazılmasıyla ve ortak yüz f'in bu yöndeki yüzey vektörü bileşeni,

$$\mathbf{E}_f = \mathrm{E}_f \mathbf{e}_f \tag{2.13}$$

ile gösterilmesiyle ortogonal olmayan bir ağ yapısı için Denklem (2.8)'da verilen $(\nabla\phi)_f \cdot \mathbf{n}_f \mathrm{s}_f$ ifadesi,

$$\begin{aligned}(\nabla\phi)_f \cdot \mathbf{n}_f \mathrm{s}_f &= (\nabla\phi)_f \cdot \mathbf{s}_f \\ &= (\nabla\phi)_f \cdot \mathbf{E}_f + (\nabla\phi)_f \cdot (\underbrace{\mathbf{s}_f - \mathbf{E}_f}_{\mathbf{T}_f}) \\ &= (\nabla\phi)_f \cdot \mathbf{E}_f + (\nabla\phi)_f \cdot \mathbf{T}_f\end{aligned} \tag{2.14}$$

şeklinde gösterilebilir. Bu tanımlamalardan da anlaşılabileceği üzere ortogonal kalite, komşu elemanların merkezlerini birleştiren doğru ile ortak yüzün normali arasındaki açı ile ilişkilidir.

Denklem (2.14)'de f yüzünden geçen akı ifadesi ϕ 'nin lineer profil değişimi kabulü ile,

$$\int_f -\Gamma^\phi \nabla\phi \cdot \mathbf{n} \mathrm{ds} \cong -\Gamma_f^\phi \frac{\phi_F - \phi_C}{\delta_f} \mathrm{E}_f - \Gamma_f^\phi (\nabla\phi)_f \cdot \mathbf{T}_f \tag{2.15}$$

şeklinde yazılır. Denklem (2.15)'de son terim f yüzünden geçen akıya ortogonal olmayan katkıyı göstermektedir. Bu terim lineerleştirilemeyen bir kısmı ifade etmektedir ve bilinen gradyen alanından hesaplanmasu gerektiğinden [1]'de de

bahsedilmiştir. Gradyen alanının hesabı ise Bölüm 2.6'de açıklanacaktır. Denklem (2.15)'in tanımlı olduğu bölgenin içindeki herhangi bir C iç elemanının tüm yüzlerindeki integrali,

$$\sum_{i=1}^{\text{count(f)}} \int_{f_i} -\Gamma^{\phi} \nabla\phi \cdot \mathbf{n} \mathrm{d}s = \sum_{i=1}^{\text{count(f)}} \left[-\Gamma_{f_i}^{\phi} \frac{\phi_{F_i} - \phi_C}{\delta_i} E_{f_i} - \Gamma_{f_i}^{\phi} \left(\nabla\phi\right)_{f_i} \cdot \mathbf{T}_{f_i} \right] \tag{2.16}$$

şeklinde yazılarak bir C iç elemanı için ayrıklaştırılmış difüzyon terimi elde edilir. Denklem(2.16)'nın Denklem (2.7)'deki formda yazılacağını düşünerek,

$$\sum_{i=1}^{\text{count(f)}} \int_{f_i} -\Gamma^{\phi} \nabla\phi \cdot \mathbf{n} \mathrm{d}s = \alpha\phi_C + \sum_{i=1}^{\text{count(f)}} \alpha_i \phi_{F_i} + \alpha_v \tag{2.17}$$

$$\alpha_i = -\Gamma_{f_i}^{\phi} \frac{1}{\delta_i} E_{f_i}, \tag{2.18}$$

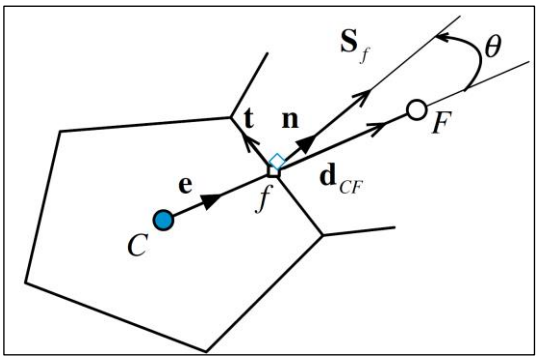
$$\alpha = \sum_{i=1}^{\text{count(f)}} -\alpha_i \tag{2.19}$$

$$\alpha_v = -\sum_{i=1}^{\text{count(f)}} \Gamma_{f_i}^{\phi} \left(\nabla\phi\right)_{f_i} \cdot \mathbf{T}_{f_i} \tag{2.20}$$

ifadelerini elde etmek mümkündür. Şekil 2.1'de ortogonal olmama durumuyla ilgili bir görsel verilmiştir. Denklem (2.16)'da gösterilen E_f ve $\mathbf{T}_f$ için farklı yöntemler literatürde mevcuttur. Bu yöntemlerden birkaçı geliştirilen yazılımda seçenek olarak sunulmuş olsa da burada bu yaklaşımlardan biri olan ve [1]'de "over – relaxed approach" ismiyle,

$$\mathbf{E}_f = \left(\frac{s_f}{\cos\theta} \right) \mathbf{e}_f = \left(\frac{s_f^2}{s_f \cos\theta} \right) \mathbf{e}_f = \frac{\mathbf{s}_f \cdot \mathbf{s}_f}{\mathbf{e} \cdot \mathbf{s}_f} \mathbf{e}_f \tag{2.21}$$

şeklinde verilen yaklaşımdan bahsetmek yeterli olacaktır. Zira bu yaklaşım için [1]'de de ortogonalliğin azalmasıyla komşu hücre merkezlerinde tanımlı ϕ değerleri cinsinden yazılan ayrıklaştırmanın hücre merkezlerindeki ϕ'e bağlılığı artacağından ve bu sebeple doğruluk ve stabilitenin artacağından bahsedilmiştir.



Şekil 2.1. Ağ elemanlarında ortogonal olmama durumu[1]

Denklem (2.16)'de ortak yüzde ifade edilmesi gereken diğer bir terim de $\Gamma^{\phi}_{f_i}$ ile gösterilen difüzyon katsayısıdır. [1]'de bahsedildiği üzere bu değerin yerel doğruluğu yerine yüzden geçen akının doğruluğu daha önemlidir. Bu sebeple bir boyutlu kaynak terimsiz ısı difüzyonu düşünülürse arayüzdeki difüzyon katsayısının,

$$\frac{(\mathbf{r}_F-\mathbf{r}_C)\cdot\mathbf{n}_f}{\Gamma^{\phi}_f}=\frac{(\mathbf{r}_f-\mathbf{r}_C)\cdot\mathbf{n}_f}{\Gamma^{\phi}_C}+\frac{(\mathbf{r}_f-\mathbf{r}_F)\cdot\mathbf{n}_f}{\Gamma^{\phi}_F} \tag{2.22}$$

olması gerektiği görülecektir. Burada $\mathbf{n}_f$ ortak yüzün normalini göstermektedir.

2.4. Kaynak Teriminin Ayrıklaştırılması

Denklem (2.6)'daki genel korunum denkleminde son terim olan kaynak teriminin ϕ 'e bağlı bir ifade olduğu Bölüm 1.3'de genel korunum denkleminin elde ediliş aşamalarından görülebilir. Kaynak teriminin bir sabit olması durumunda Denklem (2.6),

$$\int_C Q^{\phi}dV = Q^{\phi}_C V_C \tag{2.23}$$

şeklinde yazılabilecekken ϕ 'nin bir fonksiyonu olan terimler için bilinen ϕ alanından direkt olarak hesaplanması, iteratif çözümde instabiliteye sebep olacaktır. Buna [1]'de kaynak terimin sayısal büyüklüğündeki büyük değişimler sebep olarak gösterilmiştir. Bu durum ile başa çıkabilmek adına Q^{ϕ}_C 'nin Taylor serisi açılımı benzeri bir yaklaşım ile lineerleştirilebileceği Mangani, Moukalled ve Darwish tarafından [1]'de söylenmiştir. Bu yöntem ile C elemanı için hesaplanacak kaynak terimi,

$$Q_C^{\phi}\left(\phi_C\right) = Q_C^{\phi}(\phi_C^*) + \left(\frac{\partial Q_C^{\phi}}{\partial \phi_C}\right)^* \left(\phi_C - \phi_C^*\right) \tag{2.24}$$

olarak yazılır. Denklem (2.24)'de * üst indisli ifadeler bilinen değerlerden hesaplanan ifadelerdir. Bu şekilde elde edilen kaynak terimi Denklem (2.7)'deki denklem sistemindeki katsayı matrisinin köşegenini negatif olarak etkilemesi Scarborough (matris köşegeninin dominantlığı) kriterinin sağlanmasını garanti altına alacaktır. Ancak pozitif etkisi negatif olan köşegen değerlerini sıfıra yaklaştıracak ve bu da Scarborough kriterinin sağlanmamasına sebep olabilecektir. Ancak diyagonal dominantlık yakınsamayı garanti ederken bir lineer sistemin diyagonal dominant olmayan katsayı matrisi de iteratif bir çözüm yöntemiyle çözüme yakınsayabilir.

2.5. Sınır Koşulları

ϕ değerinin sınırda verilmesi anlamına gelen Dirichlet, ϕ akısının sınırda verilmesi manasına gelen Neumann ve ϕ akısının sınırda ϕ 'e bağlı olarak ifadesi anlamına gelen Robin sınır koşulları genel olarak herhangi bir C sınır elemanının herhangi bir b sınır yüzü için,

$$p\left(\nabla\phi\right)_b \cdot \mathbf{n}_b + q\phi_b + m = 0 \tag{2.25}$$

formunda yazılabilir. Denklem (2.25), $p = 0$ olması durumunda Dirichlet, $q = 0$ olması durumunda Neumann, tüm katsayılarının sıfırdan farklı olması durumunda ise Robin sınır koşuluna karşılık gelmektedir. Bu sınır koşulları problemin bir konveksiyon problemi ya da bir difüzyon problemi olması durumunda kullanılabilmektedir.

Sınır koşulu denklemlerinin ayrıklaştırması da difüzyon teriminin ayrıklaştırma aşamalarına benzerdir ve ortogonal olmayan ağ yapısı için düzeltmenin dahil edildiği durumda,

$$p\left(\nabla\phi\right)_b \cdot \mathbf{n}_b + q\phi_b + m = p\frac{1}{s_b}\left(\left(\nabla\phi\right)_b \cdot \mathbf{E}_b + \left(\nabla\phi\right)_b \cdot \mathbf{T}_b\right) + q\phi_b + m = 0 \tag{2.26}$$

şeklinde yazılabilir. Burada $\mathbf{E}_b$ (difüzyon teriminin ayrıklaştırma aşamasında kullanılan yüzey vektörü ve onun bileşenleri notasyonuna benzer şekilde) b sınır yüzün yüzey vektörünün sınır eleman ve sınır yüz merkezlerini (centroid) birleştiren doğrunun doğrultusunda bileşenidir. $\mathbf{T}_b$ ise onun $\mathbf{s}_b$ yüzey vektörüne tamamlayanı şeklindedir. Denklem (2.21) ile verilen denklem bir sınır yüz için,

$$\mathbf{E}_b = \left(\frac{s_b}{\cos\theta}\right)\mathbf{e}_b = \left(\frac{s_b^2}{s_f\cos\theta}\right)\mathbf{e}_b = \frac{\mathbf{s}_b\cdot\mathbf{s}_b}{\mathbf{e}\cdot\mathbf{s}_b}\mathbf{e}_b \tag{2.27}$$

şeklinde ifade edilebilir. Burada,

$$\mathbf{e}_b = \frac{\mathbf{r}_b - \mathbf{r}_C}{\left\|\mathbf{r}_b - \mathbf{r}_C\right\|} \tag{2.28}$$

şeklindedir.

Denklem (2.26)'in ϕ_b için ayrıklaştırılmış hali,

$$p\left(\nabla\phi\right)_b\cdot\mathbf{n}_b + q\phi_b + m = \frac{p}{s_b}\left[\frac{\phi_b-\phi_C}{\delta_b}E_b + \left(\nabla\phi\right)_b\cdot\mathbf{T}_b\right] + q\phi_b + m = 0 \tag{2.29}$$

şeklinde yazıldığında δ_b ifadesi yine Denklem (2.11)'dekine benzer şekilde sınır yüz ile sınır hücre merkezlerinin arasındaki mesafedir. Denklem (2.29) düzenlenmiş formda,

$$\phi_b = \frac{\frac{p}{s_b}\frac{E_b}{\delta_b}}{\frac{p}{s_b}\frac{E_b}{\delta_b}+q}\phi_C - \frac{m+\frac{p}{s_b}\left(\nabla\phi\right)_b\cdot\mathbf{T}_b}{\frac{p}{s_b}\frac{E_b}{\delta_b}+q} \tag{2.30}$$

şeklindedir. Bu form sınır yüzdeki ϕ_b değerine bağlı herhangi bir ifadede kullanılabilir. Örneğin sınır yüzden geçen difüzyon akısı yazılmak istendiğinde,

$$\begin{aligned} -\Gamma_b^\phi\left(\nabla\phi\right)_b\cdot\mathbf{s}_b &= -\Gamma_b^\phi\left(\nabla\phi\right)_b\cdot\mathbf{E}_b - \Gamma_b^\phi\left(\nabla\phi\right)_b\cdot\mathbf{T}_b \\ &= -\Gamma_b^\phi\frac{\phi_b-\phi_C}{\delta_b}E_b - \Gamma_b^\phi\left(\nabla\phi\right)_b\cdot\mathbf{T}_b \end{aligned} \tag{2.31}$$

olur ve Denklem (2.30)'dan ϕ_b yerine yazılırsa,

$$-\Gamma_b^\phi \left(\nabla\phi\right)_b \cdot \mathbf{s}_b = \frac{\Gamma_b^\phi q}{\frac{p}{s_b} + q\delta_b / E_b}\phi_C + \frac{\Gamma_b^\phi \left(m - \frac{q\delta_b}{E_b}\left(\nabla\phi\right)_b \cdot \mathbf{T}_b \right)}{\frac{p}{s_b} + q\delta_b / E_b} \tag{2.32}$$

şeklinde sınırdaki akı ifadesi elde edilir. Bu ifade de örneğin Neumann sınır koşulu için ($q = 0$ ve $p = -\Gamma_b^\phi$),

$$-\Gamma_b^\phi \left(\nabla\phi\right)_b \cdot \mathbf{s}_b = -ms_b \tag{2.33}$$

şeklinde sadeleşir.

2.6. Gradyen Yapımı ve İnterpolasyonu

Difüzyon teriminin ayrıklaştırılması aşamasında komşu hücrelerin merkezlerindeki ϕ değerleri cinsinden ifade edilemeyen ayrıklaştırma parçalarının bilinen gradyen alanından hesaplanması gerektiği gösterilmiştir. Bu aşamada kullanılabilecek birkaç yöntem varken en yaygın kullanılanları Green – Gauss gradyeni ile en küçük kareler (least squares) gradyenidir.

Gradyen yapımı şemaları genel olarak bilinen ϕ değişkeni alanını kullanarak yine ϕ değişkenlerinin tanımlı olduğu yerlerde $\nabla\phi$'nin hesaplanması şeklinde bir yol izler. En küçük kareler gradyeni ise [1]'de de bahsedildiği üzere komşu elemanların herhangi birindeki ϕ_F değerinin ϕ_C ve $\nabla\phi_C$ cinsinden minimum hata ile ifade edilmesi temeline dayalıdır. G_C hata fonksiyonu olmak üzere bahsedilen durum [1]'de de,

$$G_C = \sum_{i=1}^{\text{count(F)}} \left\{ w_i \left[\phi_{F_i} - \left(\phi_C + \nabla\phi_C \cdot \left(\mathbf{r}_{F_i} - \mathbf{r}_C \right) \right) \right]^2 \right\} \tag{2.34}$$

ifadesinin gradyen terimine göre minimizasyonu şeklinde açıklanmıştır. Bu ifadeden elde edilecek olan gradyenin en az birinci mertebeden doğru olduğu yine [1]'de belirtilmiştir. Burada w_i terimi bir ağırlıklandırma faktörünü ifade etmektedir. Bir C elemanında hesaplamak istenen gradyenin, C elemanının merkezine merkezi yakın

olan komşusundaki ϕ değerinin etkisinin daha fazla olması gerektiği açıktır. Bu sebeple komşuluklar yönündeki hatalar Denklem (2.34)'deki toplam hataya, merkez uzaklıklarıyla ters orantılı olacak şekilde ağırlıklandırılıp dahil edilecek olursa,

$$w_i = \frac{1}{\left\| \mathbf{r}_{F_i} - \mathbf{r}_C \right\|} \tag{2.35}$$

şeklinde alınmalıdır. Denklem (2.34) ile ifade edilen toplam hata fonksiyonunun minimizasyonu ise,

$$\frac{\partial G_C}{\partial\left(\frac{\partial \phi}{\partial x}\right)} = \frac{\partial G_C}{\partial\left(\frac{\partial \phi}{\partial y}\right)} = \frac{\partial G_C}{\partial\left(\frac{\partial \phi}{\partial z}\right)} = 0 \tag{2.36}$$

şartlarıyla sağlanabilir. Denklemlerin yazılıp düzenlenmesiyle C elemanına ait gradyen ifadesi,

$$\nabla\phi_C = \left[\sum_{i=1}^{\text{count(F)}} w_i \left(\mathbf{r}_{F_i} - \mathbf{r}_C\right)\left(\mathbf{r}_{F_i} - \mathbf{r}_C\right) \right]^{-1} \left[\sum_{i=1}^{\text{count(F)}} w_i \left(\mathbf{r}_{F_i} - \mathbf{r}_C\right)\left(\phi_{F_i} - \phi_C\right) \right] \tag{2.37}$$

şeklinde elde edilebilir. Denklem (2.37)'den C elemenlarının merkezlerinde elde edilen gradyenin Denklem (2.16)'de kullanılabilmesi için ortak yüzlere taşınması (yada interpole edilmesi) gerekir. Bu aşamada Mangani, Moukalled ve Darwish [1]'de lineer interpolasyon ve interpole edilmiş gradyenin komşu elemanları birleştiren doğru parçasının doğrultusunda düzeltilmesini önermiştir. Lineer interpolasyon adımı,

$$\nabla\phi_f^{'} = \frac{\left\| \mathbf{r}_F - \mathbf{r}_f \right\|}{\left\| \mathbf{r}_F - \mathbf{r}_C \right\|} \nabla\phi_C + \frac{\left\| \mathbf{r}_C - \mathbf{r}_f \right\|}{\left\| \mathbf{r}_F - \mathbf{r}_C \right\|} \nabla\phi_F \tag{2.38}$$

ile ifade edilir. Denklem (2.38), $\mathbf{e}_f = \left(\mathbf{r}_F - \mathbf{r}_C\right) / \left\| \mathbf{r}_F - \mathbf{r}_C \right\|$ ve $\delta_f = \left\| \mathbf{r}_F - \mathbf{r}_C \right\|$ olmak üzere düzeltilmiş hali,

$$\nabla\phi_f = \nabla\phi_f^{'} + \left(\frac{\phi_F - \phi_C}{\delta_f} - \nabla\phi_f^{'} \cdot \mathbf{e}_f \right) \mathbf{e}_f \tag{2.39}$$

şeklindedir.

Bu noktaya kadar açıklanan yöntemler ve şemalar, yazılımda implemente edilmiş olanların bir kısmı olmakla beraber ikinci mertebeden bir doğruluk için kullanılabilecek şemalardır. Bu yöntemlerin doğrulukları ve stabiliteleri ile ilgili detaylı bilgiye [1] ve [5]'den ulaşılabilir.

2.7. Problem Matrisi

Denklem (2.17), Denklem (2.18), Denklem (2.19) ve Denklem (2.20)'de verilen ayrık denklemler, sınır koşulları ile beraber Ω bölgesindeki tüm Ω_i elemanları için yazıldığında Denklem (2.7) formunda bir denklem sistemi vercektir. Denklem (2.7) 'deki **A** katsayı matrisi, Ω bölgesi M adet Ω_i adet elemanına ayrıldığı için M x M boyutunda bir seyrek(sparse) matris, $\boldsymbol{\varphi}$ vektörü M adet skalerden oluşan bilinmeyen vektörü ve **b** ise M adet sabitten oluşan sonuç vektörüdür. Denklem (2.7) $\boldsymbol{\varphi}$ için çözüldüğünde yakınsak problem çözümü bulunmuş olacaktır. Bölüm 3.1'de ve Bölüm 3.12'de bahsedildiği gibi **A** matrisinin deseni ağ elemanların birbirine bağlılıkları (komşulukları) ile ilgilidir. Bu durum kısaca şöyle açıklanabilir; ağ yapısının iç yüzlerinde (ortak yüzlerde) yazılan ayrıklaştırmalar, hem C elemanı hemde onun herhangi bir komşusu olan F elemanı için yazılmalıdır. Dolayısıyla C elemanının merkezinde tanımlı ϕ_C hem kendi için yazılan ayrıklaştırılmış denklemde hemde komşusu için yazılan ayrıklaştırmış denklemde bilinmeyen olarak yazılacaktır. Sonuçta oluşacak olan matris deseni Şekil 2.2'dekine benzer bir desen olacaktır ve bir difüzyon problemi için bu desenin gösterdiği sıfırdan farklı elemanların değerleri simetriktir (yani M x M boyutlu $\mathbf{A} = \left(a_{ij}\right)$ matrisi için $i \neq j$ olmak üzere $\forall a_{ij} = a_{ji}$).

Denklem (2.7)'in bilinen bir $\boldsymbol{\varphi} = \boldsymbol{\varphi}^*$ alanı için,

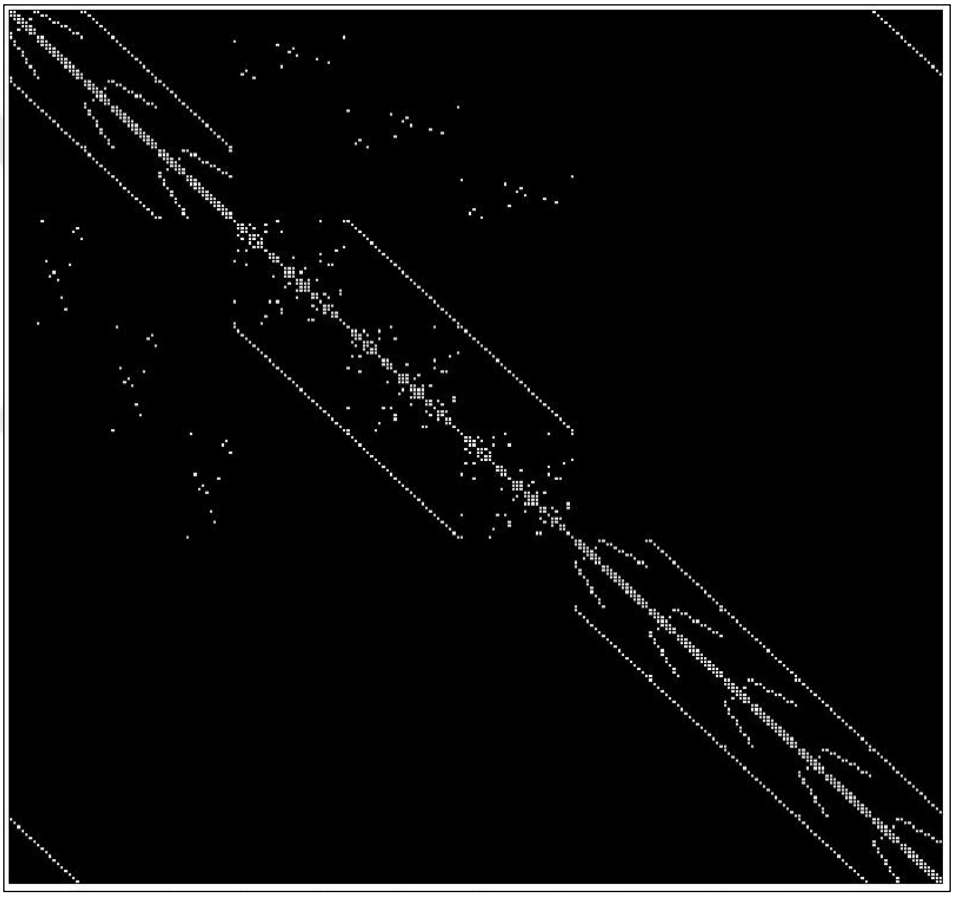
$$\mathbf{r} = \mathbf{b} - \mathbf{A}\boldsymbol{\varphi}^* \tag{2.40}$$

formunda yazılışı literatürde kalıntı form olarak geçmektedir. Denklem (2.7) veya Denklem (2.40)'ın çözümüyle ilgili yöntemlerden bazıları [1] ve [6]'da verilmiştir. Denklem (2.40)'da $\boldsymbol{\varphi}^*$ eğer yakınsak bir çözüm ise $\mathbf{r} \cong 0$'dır. Problem sisteminin çözümü sırasında da bu kriter aranır ve çözüm alanının yakınsaması **r** ile kontrol edilir. Bununla ilgili olarak literatürde farklı yöntemler vardır ve en çok kullanılanı

normalize edilmiş en büyük kalıntıdır (scaled residual). Normalize edilmiş en büyük kalıntı (max scaled residual),

$$\mathbf{r}_{\text{scaled}} = \frac{\max\limits_{i=1:M}\left[b_i - \sum\limits_{j=1}^{\text{count}(\Omega)} a_{ij}\phi_j \right]}{\max\limits_{i=1:M}\left[a_{ii}\phi_i \right]} \tag{2.41}$$

şeklinde [1]'de verilmiştir. Burdaki a_{ij} elemanları sistemin katsayı matrisin olan **A**'nın elemanlarıdır.



Şekil 2.2. 239 elemanlı bir ağ yapısında tanımlanan difüzyon problemine ait katsayı matrisinin deseni

3. YAZILIM VE KÜTÜPHANE

Denklem (1.41)'de verilen genel korunum denklemin yakınsak çözümünü istenen herhangi bir geometri için elde edebilmek amacıyla bir yazılım ve kütüphane geliştirildiğinden bu tez dokümanının önceki kısımlarında bahsedilmişti. İlerleyen kısımlarda geometrinin alınması, problemin tanımlanması, modellenmesi ve çözümü adımları için yazılım kütüphanesine yazılmış yapılardan bazıları açıklanmıştır. Bu bölümde "std" ad uzayı ile verilen türler C++ STL altında bulunan türlerdir. C++ standart kütüphanesi hakkında daha fazla bilgi için [7] ve [8] incelenebilir. Bu bölümde verilen bilgilerin detaylarına ise kaynak koddan ve dokümantasyonundan ulaşılabilir.

3.1. Ağ Yapısı

Kütüphane taslaklardan (C++ programlama dilindeki template sınıf ve yapılar) oluşmaktadır ve ağ yapısını ilgili bellek bölgesinde (RAM ya da GPU RAM) tutma ve ilgili bilgileri sunma gibi görevleri üstlenen "mesh" sınıfı taslak parametresi olarak "Allocator" adında bir tür parametresi ilk taslak parametresi olarak almaktadır. C++ STL'nin sunduğu "std::allocator" taslağından türetilen yada benzer şekilde davranan bir sınıf ile mesh(ağ yapısı) sınıf taslağından bir mesh sınıfı türü tanımlanabilir. Mesh sınıfının aldığı "Allocator" parametresi sınıfa ait konteynırların hangi bellek bölgesinde (CPU ya da GPU RAM – alternatif bir uzak sistem için yazılmış "allocator" ise o uzak sistemde) veri tutacağını belirtmektedir. Ayrıca bu konteynırlardan std::begin ve std::end ile alınan iteratörler "foreach", "transform", "reduce" vb. algoritmalara parametre olarak verildiğinde algoritma fonksiyonu, algoritmaya parametre olarak verilen fonksiyonun (ya da "functor"un) hangi sistemde yürütüleceğine karar verme amacıyla "Allocator" sınıfının ayırdığı bellek bölgesinin hangi sistemde ayrıldığına bakacaktır. Ve böylece ağ yapısı üzerinde yapılan işlemler CPU ya da GPU üstünde yürütülebilecektir.

Bu sınıf polihedral yapısal olmayan 3B (polyhedral unstructured 3D mesh) bir ağ yapısını tutabilmekte ve üzerinde işlem yapabilmektedir. Bir mesh dosyasında nokta

alanının koordinatları, bu nokta alanının ne şekilde bağlanarak (face-node connectivity) ağ yapısının yüzlerini oluşturduğu, bir yüzün hangi hücreye ait olduğu ya da hangi hücre çiftinin ortak yüzü olduğu (face-cell connectivity) gibi bilgiler yer almaktadır. Böyle bir mesh dosyasını okuyabilecek sınıfın altında bulunması gereken fonksiyonlar yazılımda "mesh_loader" adında bir soyut taban sınıf (abstract base class) ile belirtilmiştir. mesh_loader'dan türetilen sınıflar(derived class) mesh sınıfının yapıcı fonksiyonuna parametre olarak verilebilir. Mesh sınıfı mesh_loader ile verilen dosyadan ağ yapısını okuduktan sonra noktalar, yüzler ve hücreler arasındaki tüm bağlantı kombinasyonlarını gerektiğinde hesaplayabilir. Ağ yapısı yükleyicisi (mesh_loader sınıfından türetilen sınıf) olarak şu anda yalnızca Fluent ticari yazılımının kullandığı ağ yapısını okuyabilen bir sınıf mevcuttur. Fluent ticari yazılımının kullandığı ağ yapısı dosya formatı ile ilgili bilgiye [9]'den ulaşılmıştır.

Mesh sınıfına bağlı birçok sınıf mevcuttur. Bu sınıflardan bazıları nokta, yüz ve hücrelere tanımlayıcı (identifier – kısaca id) olma görevini, bazıları indis olma görevini üstlenmiştir. Bunlar dışında nokta, yüz ve hücreler için iteratör olma görevini üstlenen sınıflarda mevcuttur. Bu sınıflardan bazıları Tablo 3.1'de verilmiştir.

Tablo 3.1. Ağ yapısı elementlerinin erişimi için yazılmış sınıf ve sınıf taslaklarından bazıları

Grup	**Sınıf ismi**	**Açıklama**
Tanımlayıcılar	mesh_node_id	Nokta tanımlayıcısı
	mesh_face_id	Yüz tanımlayıcısı
	mesh_cell_id	Hücre tanımlayıcısı
	mesh_node_zone_id	Nokta bölgesi tanımlayıcısı
	mesh_face_zone_id	Yüz bölgesi tanımlayıcısı
	mesh_cell_zone_id	Hücre bölgesi tanımlayıcısı
İndisler	mesh_node_index_in_zone	Nokta bölgesi içinde indis
	mesh_face_index_in_zone	Yüz bölgesi içinde indis
	mesh_cell_index_in_zone	Hücre bölgesi içinde indis
	mesh_node_neighbor_node_index	Nokta komşuluğunda nokta indisi
	mesh_node_neighbor_face_index	Nokta komşuluğunda yüz indisi
	mesh_node_neighbor_cell_index	Nokta komşuluğunda hücre indisi
	mesh_face_neighbor_node_index	Yüz komşuluğunda nokta indisi
	mesh_ face_neighbor_face_index	Yüz komşuluğunda yüz indisi
	mesh_ face_neighbor_cell_index	Yüz komşuluğunda hücre indisi
	mesh_cell_neighbor_node_index	Hücre komşuluğunda nokta indisi
	mesh_ cell_neighbor_face_index	Hücre komşuluğunda yüz indisi
	mesh_cell_neighbor_cell_index	Hücre komşuluğunda hücre indisi
Elementler	mesh_nodes	Noktalar
	mesh_faces	Yüzler
	mesh_cells	Hücreler

Tablo 3.1. (Devam) Ağ yapısı elementlerinin erişimi için yazılmış sınıf ve sınıf taslaklarından bazıları

Grup	Sınıf ismi	Açıklama
Bölgeler	mesh_node_zone	Nokta bölgesi
	mesh_face_zone	Yüz bölgesi
	mesh_face_zone	Hücre bölgesi
	mesh_node_zones	Nokta bölgeleri
	mesh_face_zones	Yüz bölgeleri
	mesh_face_zones	Hücre bölgeleri
Elementlere ve komşuluklara bakışlar	mesh_node_view	Nokta bakışı
	mesh_face_view	Yüz bakışı
	mesh_cell_view	Hücre bakışı
	mesh_node_node_neighbors_view	Noktanın nokta komşuları
	mesh_node_face_neighbors_view	Noktanın yüz komşuları
	mesh_node_cell_neighbors_view	Noktanın hücre komşuları
	mesh_face_node_neighbors_view	Yüzün nokta komşuları
	mesh_face_face_neighbors_view	Yüzün yüz komşuları
	mesh_face_cell_neighbors_view	Yüzün hücre komşuları
	mesh_cell_node_neighbors_view	Hücrenin nokta komşuları
	mesh_ cell_face_neighbors_view	Hücrenin yüz komşuları
	mesh_ cell_cell_neighbors_view	Hücrenin hücre komşuları

Tablo 3.1'de verilen bazı sınıf ve sınıf taslaklarına ek olarak element iteratörleri de mevctuttur. İteratörler, her element barındıran sınıfın altında mevcuttur. Bunların herbiri bu tez dokümanında açıklanmayacaktır. İstendiği taktirde kaynak kod ve dokümantasyonundan incelenebilir ve sonuç olarak ağ yapısının verileri üzerinde istenildiği gibi gezmek mümkündür.

3.2. Sonlu Hacimler Alanı

Bu sınıf iki adet taslak parametresi (template parameter) alır. Bunlardan ilki fvm_domain_discretization_policy adında bir yapıyı parametre olarak kabul eden "DiscretizationPolicy" taslak parametresidir. fvm_domain_discretization_policy yapısı ile kütüphanede hücre merkezli ve köşe merkezli SHM birbirinden ayrılır. Zira köşe merkezli SHM (vertex – centered FVM) olarak literatürde geçen SHM, ağ yapısının üzerine ikincil bir ağ yapısı oturtarak literatürde "dual mesh" (ikili ağ) adı verilen ağ yapısını oluşturmaktadır.

fvm_domain sınıfıının aldığı ikinci taslak parametresi ise mesh sınıfınında da aldığı Allocator parametresidir. "DiscretizationPolicy" adındaki ilk taslak parametresinin hücre merkezli SHM'yi belirtmesi durumunda fvm_domain sınıfı mesh sınıfından kalıtım alır ve böylelikle onun nokta, yüz ve hücre veri yapısına sahip olur. mesh

sınıfından kalıtım alınan sınıflardaki yüz alanları, eleman hacimleri vb. depolama yerlerinin varlığını kontrol ettiğinden böyle bir depolama alanı var ise bunları hesaplar ve gerekli yerde depolar.

fvm_domain sınıf taslağı mesh sınıf taslağına benzer sınıf ve taslaklarla beraber kullanılabilecekken, bunların yanında fvm_interior, fvm_boundary, fvm_interface ve fvm_region gibi sınıf taslakları da fvm_domain'in veri yapısında tanımlayıcılar üzerinden işlem yapmaktadır.

3.3. Sonlu Hacimler Bölgesi

Bu sınıf taslağı Bölüm 3.4'de açıklanacak olan fvm_variable sınıf taslağında ve fvm_variable'ın kullanıldığı diğer sınıf taslaklarında (fvm_equation, fvm_problem vb.) türetilen objenin üstünde işlem yapacağı bölgeyi belirtmek üzere tasarlanmıştır. "fvm_domain" ve dolayısıyla hücre merkezli SHM için mesh sınıfının altındaki bölge(zone) tanımlayıcıları ile işlem yapmaktadır. Bir fvm_region, fvm_subdomain'lerin birleşimini veya fvm_boundary'lerin birleşimini ifade edebilir. Sınıfın kesişim, birleşim gibi diğer bir fvm_region ile işlemlerini belirten fonksiyonları vardır.

3.4. SHM Değişkeni

Sıcaklık, hız, ısı transfer katsayısı gibi genel korunum denkleminde ve sınır koşulu denklemlerinde geçebilecek ifadelerin yerini tutma görevi vardır. Şekil 3.1'de fvm_variable'ın tanımlanma satırları verilmiştir. Burdaki "DiscretizationPolicy" parametresi, fvm_domain taslak sınıfının taslak parametresi olan "DiscretizationPolicy" isimli parametresinden farklıdır ve fvm_discretization_policy adındaki bir yapıyı kabul etmektedir. Bu yapıyla ilgili bir görsel Şekil 3.2'de verilmiştir. Görsellerden anlaşılabileceği üzere fvm_variable, değişkenin tanımlı olduğu fvm_region'daki verilerini tutmak dışında nasıl ayrıklaştıracağı, interpole edileceği ve değişkenin kontrol hacminin neresinde tanımlı olduğu bilgilerini de tutmaktadır.

```
template<typename FvmRegion, typename DiscretizationPolicy, typename VariableType, algex_variable_symbol Symbol, typename ...Dependency>
class fvm_variable :
    public fvm_variable_data<fvm_variable<FvmRegion, DiscretizationPolicy, VariableType, Symbol, Dependency...>, VariableType>,
    public algex_variable<
        fvm_variable<FvmRegion, DiscretizationPolicy, VariableType, Symbol, Dependency...>,
        Symbol,
        VariableType,
        fvm_symbolic_coordinate_t,
        Dependency...
    > {
```

Şekil 3.1. fvm_variable sınıf taslağının tanımlanma satırları

```
template<
    FVM_VARIABLE_LOCATION VariableLocation,
    FVM_VARIABLE_DISCRETIZATION_SCHEME VariableDiscretizationScheme = FVM_VARIABLE_DISCRETIZATION_SCHEME_UNSPECIFIED,
    FVM_VARIABLE_INTERPOLATION_SCHEME VariableInterpolationScheme = FVM_VARIABLE_INTERPOLATION_SCHEME_UNSPECIFIED,
    FVM_GRADIENT_DISCRETIZATION_SCHEME GradientDiscretizationScheme = FVM_GRADIENT_DISCRETIZATION_SCHEME_UNSPECIFIED,
    FVM_GRADIENT_CONSTRUCTION_SCHEME GradientConstructionScheme = FVM_GRADIENT_CONSTRUCTION_SCHEME_UNSPECIFIED,
    FVM_GRADIENT_INTERPOLATION_SCHEME GradientInterpolationScheme = FVM_GRADIENT_INTERPOLATION_SCHEME_UNSPECIFIED,
    FVM_DIVERGENCE_DISCRETIZATION_SCHEME DivergenceDiscretizationScheme = FVM_DIVERGENCE_DISCRETIZATION_SCHEME_UNSPECIFIED,
    FVM_DIVERGENCE_CONSTRUCTION_SCHEME DivergenceConstructionScheme = FVM_DIVERGENCE_CONSTRUCTION_SCHEME_UNSPECIFIED,
    FVM_DIVERGENCE_INTERPOLATION_SCHEME DivergenceInterpolationScheme = FVM_DIVERGENCE_INTERPOLATION_SCHEME_UNSPECIFIED,
    FVM_CURL_DISCRETIZATION_SCHEME CurlDiscretizationScheme = FVM_CURL_DISCRETIZATION_SCHEME_UNSPECIFIED,
    FVM_CURL_CONSTRUCTION_SCHEME CurlConstructionScheme = FVM_CURL_CONSTRUCTION_SCHEME_UNSPECIFIED,
    FVM_CURL_INTERPOLATION_SCHEME CurlInterpolationScheme = FVM_CURL_INTERPOLATION_SCHEME_UNSPECIFIED,
    FVM_INTEGRATION_SCHEME IntegrationScheme = FVM_INTEGRATION_SCHEME_UNSPECIFIED
>
struct fvm_discretization_policy {
    static constexpr auto variable_location = VariableLocation;
    static constexpr auto variable_discretization_scheme = VariableDiscretizationScheme;
    static constexpr auto variable_interpolation_scheme = VariableInterpolationScheme;
    static constexpr auto gradient_discretization_scheme = GradientDiscretizationScheme;
    static constexpr auto gradient_construction_scheme = GradientConstructionScheme;
    static constexpr auto gradient_interpolation_scheme = GradientInterpolationScheme;
    static constexpr auto divergence_discretization_scheme = DivergenceDiscretizationScheme;
    static constexpr auto divergence_construction_scheme = DivergenceConstructionScheme;
    static constexpr auto divergence_interpolation_scheme = DivergenceConstructionScheme;
    static constexpr auto curl_discretization_scheme = CurlDiscretizationScheme;
    static constexpr auto curl_construction_scheme = CurlConstructionScheme;
    static constexpr auto curl_interpolation_scheme = CurlInterpolationScheme;
    static constexpr auto integration_scheme = IntegrationScheme;
};
```

Şekil 3.2. fvm_discretization_policy taslak yapısının tanımlanma satırları

Bu sınıf taslağı ile tanımlanan sınıflar ayrıklaştırma şemasını denklemin yapısıyla beraber belirleyen sınıflardır. Ayrıca değişken lokasyonun seçilmesine de izin veren taslak yapısı, sınır yüzlerde ısı transfer katsayısı gibi değişkenler tanımlanmasına da izin vermektedir. Ancak ayrıklaştırılacak olan değişken (şuanda) kontrol hacminin merkezinde tanımlı olmak zorundadır. Şekil 3.3'de sıcaklık için tanımlama verilmiştir.

```
using temperature_t =
    fvm_variable<
        fvm_subdomain<fvm_domain<>>,
        fvm_discretization_policy<
            FVM_VARIABLE_LOCATION_CV,
            FVM_VARIABLE_DISCRETIZATION_SCHEME_UNSPECIFIED,
            FVM_VARIABLE_INTERPOLATION_SCHEME_LINEAR_WITH_GRADIENT_CORRECTION,
            FVM_GRADIENT_DISCRETIZATION_SCHEME_SURFACE_NORMAL_GRADIENT_FORMULATION_WITH_CORRECTION,
            FVM_GRADIENT_CONSTRUCTION_SCHEME_LEAST_SQUARES_GRADIENT,
            FVM_GRADIENT_INTERPOLATION_SCHEME_LINEAR_WITH_CORRECTION,
            FVM_DIVERGENCE_DISCRETIZATION_SCHEME_GREEN_GAUSS_THEOREM,
            FVM_DIVERGENCE_CONSTRUCTION_SCHEME_UNSPECIFIED,
            FVM_DIVERGENCE_INTERPOLATION_SCHEME_UNSPECIFIED,
            FVM_CURL_DISCRETIZATION_SCHEME_UNSPECIFIED,
            FVM_CURL_CONSTRUCTION_SCHEME_UNSPECIFIED,
            FVM_CURL_INTERPOLATION_SCHEME_UNSPECIFIED,
            FVM_INTEGRATION_SCHEME_SECOND_ORDER_GAUSS_INTEGRATION
        >,
        scalar,
        temprature_sym
    >;
```

Şekil 3.3. Sıcaklık için tanımlanmış bir fvm_variable türü

3.5. SHM Değişken İnterpolatörü

Hücre merkezli SHM'de Green-Gauss gradyeni hesaplanırken değişkenlerin hücre merkezlerinde hücre yüzlerine interpole edilmesi gerekebilir. Bu sınıf taslağı temelde bu amaç için yazılmıştır. Ancak kütüphane kullanılırken bu taslak sınıf başka amaçlar içinde kullanılabilir. Örneğin bir ifadenin yüzde hesaplanabilmesi için öncelikle ifadedeki değişkenlerin yüze interpole edilmesi gerekebilir. fvm_variable_interpolator taslak parametresi olarak fvm_variable taslağından oluşturulmuş bir tür kabul etmekte ve fvm_discretization_policy altında değişken interpolasyon şeması için seçilmiş seçeneğe göre işlem yapmaktadır. Eğer taslak parametresi olarak geçilen değişken türünde interpolasyon şeması belirtilmemişse fvm_variable_interpolator::interpolate fonksiyonun çağrılması çalışma aşamasında "not_implemented_exception" hatasına sebep olacaktır.

3.6. SHM Gradyen Yapıcısı

SHM ayrıklaştırmasında yazılan düzeltme faktörlerinin hesaplanması ve çözüm alanından bilgi alınması gibi durumlarda değişkenin gradyeninin hesaplanması gerekebilir. Böyle bi durumda fvm_gradient_constructor sınıf taslağı bilinen değişken alanından gradyen hesaplama amacıyla kullanabilir. Bu sınıf (şuanda) en küçük kareler gradyeni ve hücre tabanlı Green – Gauss gradyeni hesaplayabilmektedir. Hesaplama sonuçları ilgili değişkenin gradyen verileri kısmında saklanmaktadır.

3.7. SHM Gradyen İnterpolatörü

Hücre merkezli SHM'de ayrıklaştırmadaki düzeltme faktörlerinin hesaplanabilmesi için gradyenin hücre merkezlerinde hesaplanması yeterli değildir ve hesaplanan gradyenlerin hücre yüzlerine interpole edilmesi gerekir. fvm_gradient_interpolator seçilen interpolasyon şemasına göre gradyen interpolasyonunu gerçekleştirir ve ilgili değişkenin gradyen verileri kısmında sonuçları saklar.

3.8. SHM Denklemi ve Cebirsel Denklem

Kütüphanede var olan ayrıklaştırma şemalarını kullanarak bir korunum denkleminin bir geometride çözümünü elde etmek için öncelikle denklem ifade edilmelidir. algex ve fvm_equation taslak sınıfları bu amaca yönelik yazılmış taslaklardır. algex sınıfı

Şekil 3.4'de mavi işaretlenmiş kısımla gösterildiği gibi olan bir denklemi yazma aşamasında kullanılmakta ve eşitlik kontrol operatörü "==" ile fvm_equation sınıf taslağından bir tür tanımlanmakta, bu türden de bir obje oluşturulmaktadır.

```
fvm_equation diffusion =
    divergence<fvm_symbolic_coordinate_t>(operand: -(conductivity * gradient<fvm_symbolic_coordinate_t>(temperature))) + source
        == zero_algex_v;
```

Şekil 3.4. Geliştirilen kütüphane ile yazılmış kaynak terimli zamandan bağımsız difüzyon denklemi

Burada algex sınıfı üstünde tanımlı operatörler gerekli kontrolleri yaparak bir işlem ağacı oluşturmaktadır. Gerekli kontrollerden kastedilen ise örneğin skaler bir ifadeyle vektörel bir ifadenin toplanamayacağı ve dolayısıyla toplama operatörünün tanımsız olacağı şeklindedir. Şekil 3.5'de fvm_equation sınıfının ilk birkaç satırı verilmiştir.

```
template<typename Algex>
class fvm_equation  {

    static_assert(is_algex_v<Algex>, "fvm equation requires an algex type");
    static_assert(count_fvm_variables_v<Algex> > 0, "equation must be linked with a fvm domain");
    static_assert(count_fvm_variables_v<Algex> == algex_variable_count_v<Algex>, "there are non-fvm variables in algex.");
```

Şekil 3.5. fvm_equation taslak sınıfının ilk birkaç satırı

Denklemin tanımlı olduğu bölge ise denklemi oluşturan fvm_variable türlerinin tanımlı olduğu bölgelerin kesişimi şeklinde hesaplanmaktadır. Bu bölgenin bir fvm_equation objesinden istenmesi durumunda fonksiyon kesişim bölgesini gösteren fvm_region objesi döndürecektir.

3.9. SHM Problemi

Bir problemi ifade etmek için problem için yazılan denklemlerin tanımlı olduğu bölgelere, Şekil 3.6'daki gibi bir denkleme ve sınır denklemlerine ihtiyaç vardır. Ayrıca denklemi oluşturan değişkenlerin başka bir geometride tanımlı olması denklemi tanımsız yapacaktır. Bu gibi zorunluluklar fvm_problem sınıf taslağı altında statik ve dinamik olarak kontrol edilir.

Statik kontroller programın derlenmesi aşamasında yapılırken dinamik kontroller program çalışırken yapılır. Kontroller sonrasında fvm_problem verilen denklemlerden tanımlanır. Buna bir örnek Şekil 3.6'de verilmiştir.

```
fvm_equation diffusion =
    divergence<fvm_symbolic_coordinate_t>( operand: -(conductivity * gradient<fvm_symbolic_coordinate_t>(temperature))) + source
        == zero_algex_v;

fvm_equation robin = (conductivity * inner_product( lhs: gradient<fvm_symbolic_coordinate_t>(temperature), rhs: differential( expr: fvm_symbolic_normal)))
                  == (heat_transfer_coeff * (temperature - temperature_inf));

fvm_equation neumann = (-conductivity * inner_product( lhs: gradient<fvm_symbolic_coordinate_t>(temperature), rhs: differential( expr: fvm_symbolic_normal)))
    == boundary_flux;

fvm_equation dirichlet = temperature == boundary_temperature;


fvm_problem problem = define_fvm_problem( [&]temperature, diffusion, robin, dirichlet, neumann);
```

Şekil 3.6. fvm_problem objesinin oluşturulması

Problem tanımlanırken çağırılan fvm_problem yapıcı fonksiyonu, argüman olarak geçilen denklemleri tanımlı oldukları bölgelere göre fvm_problem_equation ya da fvm_problem_boundary_equation olarak sınıflandırır. fvm_problem_equation ve fvm_problem_boundary_equation taslak sınıfları sonraki başlıklarda açıklanacaktır. Değişkenlerin tanımlı oldukları bölge türleri statik olarak belirlendiğinden bir denklem dinamik olarak türünü değiştiremez ve bölgelere özgü tanımlayıcılar ile sadece sınır yüzlerde ya da iç bölgelerde tanımlanabilir.

Ayrıklaştırma adımlarının yürütülmesini ve düzeltmelerin uygulanmasını, problem çözme aşamasının en tepesinde bu sınıf üstlenmiştir. Şekil 3.7'de divörjansın hacim integrali şeklindeki bir ifadeyi ayrıklaştırma adımlarına yönlendiren bir fonksiyon verilmiştir.

```
template<
    typename Algex,
    typename FvmVariable,
    typename... BoundaryEquations,
    std::enable_if_t<
        std::conjunction_v<
            is_fvm_discretization_policy_configured_with<
                typename fvm_variable_traits<FvmVariable>::discretization_policy,
                FVM_DIVERGENCE_DISCRETIZATION_SCHEME_GREEN_GAUSS_THEOREM
            >
    >, int> = 0,
    typename Allocator = typename fvm_variable_traits<FvmVariable>::allocator,
    typename VariableType = typename fvm_variable_traits<FvmVariable>::field_variable_type,
    typename CoeffMatrixType = csparse_matrix<VariableType, rebind_alloc<Allocator, VariableType>>,
    typename SourceMatrixType = matrix<VariableType, rebind_alloc<Allocator, VariableType>>
>
pair<CoeffMatrixType, SourceMatrixType> _apply_discretization(
    const _fvm_volume_integral_type<_fvm_divergence_algex<Algex>>& expr,
    const FvmVariable& variable,
    const fvm_problem_boundary_equations<FvmVariable, BoundaryEquations...>& boundary_eqns) {

    return _apply_discretization(
        _integrate_fvm_expr_inner_product_surface(
            get_divergence_operand(
                _get_fvm_integral_operand(expr))
            ),
        variable,
        boundary_eqns);
}
```

Şekil 3.7. Divörjansın hacim integralini ayrıklaştıran bir fonksiyon

3.10. SHM Problem Denklemi

Şekil 3.4'de verilen denklem gibi fvm_subdomain'den oluşan bölgeler üstünde tanımlı denklemleri tutmak ve üzerinde işlem yapmak gibi görevleri vardır. Altında ifade ettiği denklemi bilinmeyen vektörü ile beraber tutar. Denklemin ayrıklaştırılma sonuçlarını katsayı matrisi, sonuç vektörü ve düzeltme vektörünü de altında tutarak istendiği taktirde kesiştiği başka bölge için ayrıklaştırılan matrise tuttuğu ayrıklaştırma bilgilerini yazabilir. Şekil 3.8'de taslak sınıfın ilk birkaç satırı verilmiştir.

```
template<typename FvmVariable, typename FvmEquation>
class fvm_problem_equation : public FvmEquation  {

    static_assert(is_fvm_variable_v<FvmVariable>, "type isn't a fvm variable");
    static_assert(is_fvm_equation_v<FvmEquation>, "type isn't a fvm equation");
```

Şekil 3.8. fvm_problem_equation taslak sınıfının ilk satırları

3.11. SHM Problem Sınır Denklemi

Sınıf taslağı isminden de anlaşılacağı üzere sınır koşulu denklemlerini tutar. Bir sınır koşulu denklemi Şekil 3.9'de gösterilen şekilde fvm_variable taslağından oluşturulmuş türler barındırır ve örnek birkaç denklem Şekil 3.10'de verilmiştir.

```
using boundary_temperature_t =
    fvm_variable<
        fvm_boundary<fvm_domain<>>,
        fvm_discretization_policy<
            FVM_VARIABLE_LOCATION_CV_FACE
        >,
        scalar,
        temprature_b_sym
    >;
```

Şekil 3.9. Sınırda verilecek sıcaklık koşulu için değişken türü tanımlaması

```
fvm_equation robin = (conductivity * inner_product(lhs: gradient<fvm_symbolic_coordinate_t>(temperature), rhs: differential(expr: fvm_symbolic_normal)))
                  == (heat_transfer_coeff * (temperature - temperature_inf));

fvm_equation neumann = (-conductivity * inner_product(lhs: gradient<fvm_symbolic_coordinate_t>(temperature), rhs: differential(expr: fvm_symbolic_normal)))
    == boundary_flux;

fvm_equation dirichlet = temperature == boundary_temperature;
```

Şekil 3.10. Sınır koşulu denklemleri

Bir sınır koşulu denkleminin ayrıklaştırılmış hali de bu sınıf altında, Bölüm 2'de kullanılan notasyonla $P\phi_b + Q\phi_C + N = 0$ formunda tutulmaktadır. Burada P, Q ve N

birer skaler alandır. Bu ayrıklaştırma formunun $A\phi = b$ formunda yazılan global denklem sistemine entegre edilebilmesi için öncelikle kontrol hacminde korunan formda yazılmalıdır. Bu form örneğin difüzif akı formudur. Ayrıca $P\phi_b + Q\phi_C + N = 0$ şeklinde tutulan ayrıklaştırma sayesinde sınır hücre merkezindeki değişken cinsinden sınır yüzdeki ϕ_b değeri alınabilmektedir.

Sınır koşulu denklemleri fvm_problem altında toplu olarak fvm_problem_boundary_equations taslak sınıfı ile tutulmaktadır. Bu taslak ise "std::tuple" taslak sınıfına benzer tuple adlı bir sınıftan kalıtım alarak statik olarak sınır koşulu denklemlerini sıralamaktadır. Bu taslak sınıfların ilk birkaç satırı Şekil 3.11 ve Şekil 3.12'da verilmiştir.

```
template<typename FvmVariable, typename FvmEquation>
class fvm_problem_boundary_equation : public FvmEquation  {

    static_assert(is_fvm_variable_v<FvmVariable>, "type isn't a fvm variable");
    static_assert(is_fvm_equation_v<FvmEquation>, "type isn't a fvm equation");
```

Şekil 3.11. fvm_problem_boundary_equation taslak sınıfının ilk birkaç satırı

```
template<typename FvmVariable, typename... FvmEquations>
class fvm_problem_boundary_equations
    : public tuple<fvm_problem_boundary_equation<FvmVariable, FvmEquations>...> {
```

Şekil 3.12. fvm_problem_boundary_equations taslak sınıfının ilk birkaç satırı

3.12. Halkalı Seyrek Matris

SHM ile ayrıklaştırılan bir problemin katsayı matrisi seyrek matris yapısında olacaktır. Bu durumdan önceki başlıklar altında da bahsedilmişti ve Şekil 2.2 ile bu seyrek matris yapısına bir örnek verilmişti. Seyrek matris elemanlarının dizilme şekli, ayrıklaştırma aşamasından görülebileceği gibi ağ yapısındaki bir iç yüzün sahibi veya komşuluklarının indislerine bağlıdır. Bir yüzden bir yönde geçen akı yüzün komşuluğundaki bir kontrol hacmi için yazıldığında benzer bir akı ifadesi diğer kontrol hacmi içinde yazılmaktadır. Bu durumda hesaplanan $\mathbf{A} = \left[a_{ij}\right]$ katsayı matrisinde herhangi bir i indisli kontrol hacminin bir j indisli kontrol hacmi ile komşuluğu var ise matrisin a_{ii}, a_{jj}, a_{ij} ve a_{ji} elemanları sıfırdan farklıdır. Bu elemanların sıralı şekilde birbirlerine eşit uzaklıkta olması ve bu elemanlar üstünden bir halka geçmesi

sebebiyle “halkalı seyrek matris” ifadesi kullanılmış olup bahsi geçen halkalar ağ yapısının herhangi bir iç yüzü için oluşmaktadır. Benzer bir matris yapısından “lduMatrix” adı altında [1]’de de bahsedilmiştir. Ancak yapıların bir noktada ayrılması ve bağımsız düşünülmüş olmaları sebebiyle bu matris yapısı için “cyclic sparse matrix” ismi seçilmiştir. Bu taslak sınıfın amacı ise sıfırdan farklı elemanların verimli bir şekilde depolanması ve gerekli matris işlemlerin verimli bir şekilde yapılmasıdır. Ayrıklaştırma aşamasında matrisin kullanımına Şekil 3.13’de bir örnek verilmiştir.

```
using region_type = typename fvm_variable_traits<FvmVariable>::region_type;

const region_type& region = variable.region();

auto diff_algex = _get_fvm_integral_operand(expr).lhs_operand();

prepare_evaluation_at_cvs(diff_algex);

pair<CoeffMatrixType, SourceMatrixType> result = { {region.cv_count(), region.interiors().face_count()}, {} };
CoeffMatrixType& A = result.first;

using other_region_type = algex_region_type<Algex>;

const other_region_type& other_region = algex_region(diff_algex);

for (const fvm_interior<FvmDomain>& interior : other_region.interiors()) {
    for (const fvm_cv_face_const_view<FvmDomain>& face : interior) {
        uint32 find = region.index_of(face.id());
        uint32 lcvind = region.index_of(face.lower_cv().id());
        uint32 ucvind = region.index_of(face.upper_cv().id());
        A.make_cycle(find, lcvind, ucvind);
        scalar eval = interior_harmonic_evaluate(diff_algex, face);
        scalar diff = _grad_coeff(face)* eval;
        A.cycle_upper_value(find) = diff;
        A.cycle_lower_value(find) = diff;
    }
}

A.sum_negative_cycles_to_diag();

boundary_eqns.add_uncorrected_system_to(result, expr, region);

return result;
```

Şekil 3.13. csparse_matrix’in eleman atama işlemine bir örnek

Sıkıştırılmış bir veri depolama yapısına sahip bu sınıf yine yazılımda implemente edilmiş square_matrix isimli bir sınıfa dönüşerek kapladığı bellek bölgesinde (yerinde), [6]’de verilen LU ayrıklaştırma yöntemiyle ayrıklaştırılabilmektedir. Zira matris seyrek bir yapıda olmasına rağmen LU ayrıklaştırması ile ayrıklaştırılmış hali yine bir seyrek matris olmak zorunda değildir. LU ayrıklaştırması sonrası $\mathbf{A}\boldsymbol{\varphi} = \mathbf{b}$ formundaki sistemin herhangi bir $\mathbf{b}$ vektörü için $\boldsymbol{\varphi}$ ’nin çözümü elde edilebilir. Ancak ayrıklaştırılmış bir korunum denklemi o anda bilinen bir ϕ skaler alanından düzeltme hesaplanması ile $\mathbf{b}$ sonuç vektörünün değiştirilmesini gerektirdiğinden ayrıklaştırılmış korunum denkleminin çözümünün iteratif bir çözüm olması daha doğrudur. Bu sebeple

csparse_matrix taslak sınıfının sunduğu veri yapısı ile iteratif çözüm yapabilecek eşlenik gradyen ve önkoşullu eşlenik gradyen yöntemleride yazılmıştır. Önkoşullandırma matrisi ise "Diagonal Incomplete LU" (DILU) ayrıklaştırma yöntemi ile hazırlanabilmektedir. DILU ayrıklaştırma yönteminin yazılmasında [1]'den yararlanılmış ve depolama gereksinimlerinden yine [1]'de bahsedilmiştir. Farklı önkoşullama yöntemleri de iteratif çözücüye kolayca eklenebilir. Ayrıca belirtilmelidir ki "allocator" ve "iterator" sistemi sayesinde direkt çözücüde, iteratif çözücüde ve önkoşullayıcıda kullanılan algoritmalar ekran kartında da yürütülebilir.

3.13. CUDA için Yazılmış Kısımlar

Kütüphane paralel hesaplama yöntemi olarak CUDA kütüphanesine odaklı şekilde yazılmıştır. CUDA ile ilgili dokümantasyona [10]'dan ulaşılabilir. İteratör sisteminde varolan taslak yapılar örneğin ekran kartında yürütülmeyi de destelemektedir. İteratör sistemine ait taslak yapıların bir listesi Tablo 3.2'de verilmiştir.

Tablo 3.2. İteratör sistemine ait sınıf taslaklarının bir listesi

Sınıf	**Açıklama**
constant_iterator	Sabit iteratör
counting_iterator	Permütasyon iteratörü
generator_iterator	Sayan iteratör
permutation_iterator	Üreten iteratör
conversion_iterator	Dönüşüm iteratörü
reverse_iterator	Ters iteratör
value_iterator	Değer iteratörü
packed_iterator	Paket iteratör

İteratör sistemi dışında CUDA bellek bölgesinde varolan bir bölgeyi manipüle edebilmek amacıyla uzak işaretçi(system_ptr) ve uzak referans (sytem_ref) taslak sınıflarıda kütüphanede mevcuttur. Örneğin CUDA bellek bölgesinde tanımlı bir dizi işaretçisi CPU'da dereferans edildiğinde system_ref taslak sınıfı türünde bir değer döndürecektir ve zaten işaretçisinin kendisi de CPU'da yürütülen kod için bir system_ptr'dir.

Ayrıca CUDA ile yürütülecek "Kernel" fonksiyonunun, yürütme konfigürasyonunu yapan "kernel_launch_policy" adında bir taslak sınıf mevcuttur. Bu taslak sınıf bir "Kernel" için eğer özel olarak konfigüre edilmişse ona göre, edilmemiş ise CUDA kütüphanesinin sunduğu hesaplama fonksiyonlarının verdiği şekilde grid ve blok

sayısını konfigüre eder ve gerekli dinamik paylaşımlı bellek boyutunu belirler. CUDA'a iletilmesi gereken bu konfigüsrasyonun ve "Kernel"in iletilmesi işlemide kernel_traits<Kernel>::launch fonksiyonuna aittir.

Yukarıda bahsedilen "Kernel" yapıları "transform", "reduce" vb. algoritmalar için kütüphanede bulunmaktadır. Bunun dışında ekran kartında yürütme desteği sunulan birçok kısım bulunmaktadır. Detaylı bilgi için kaynak kod ve dokümantasyon incelenebilir.

4. ÇÖZÜM VE DOĞRULAMA

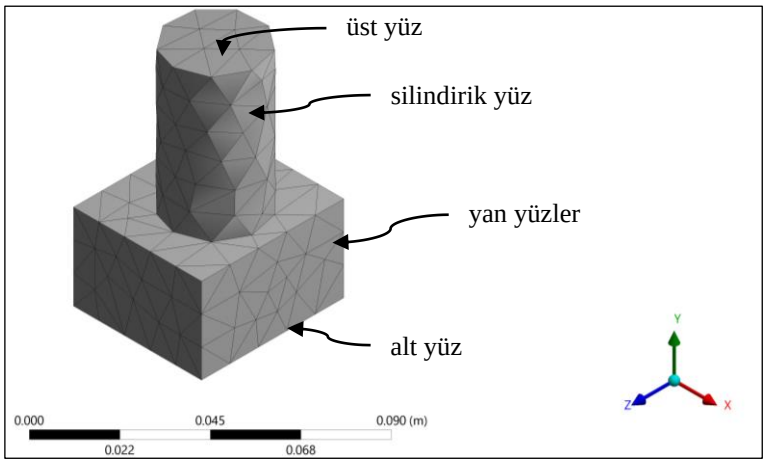
Geliştirilen kütüphanede SHM ile ilgili kısmın doğrulanması amacıyla basit bir geometri üstünde çözüm yapılmış ve sonuçlar, ticari bir yazılım olan Fluent yazılımının sonuçlarıyla karşılaştırılmıştır. Karşılaştırmalar hücre merkezlerindeki sıcaklık değerleri arasındaki farklar, sınırlardan geçen toplam akı ve net akı dikkate alınarak yapılmıştır.

Bahsedilen karşılaştırmalar üç farklı durum için yapılmış olup her durumda sınır koşulları ve geometri aynı; ancak ağ yapısı farklıdır. İlgili geometri Şekil 4.1'de verilmiştir. Sınır bölgeleri Şekil 4.1'de gösterilen "üst yüz", "alt yüz", "silindirik yüz" ve "yan yüzler" ile gösterilen bölgeler ile tanımlanmıştır. Her durum için geçerli olan sınır koşulları ve ısı üretimi Tablo 4.1'de verilmiştir. Ayrıca tüm testlerde, geliştirilen yazılımda çözücü olarak önkoşullu eşlenik gradyen çözücü kullanılmıştır.

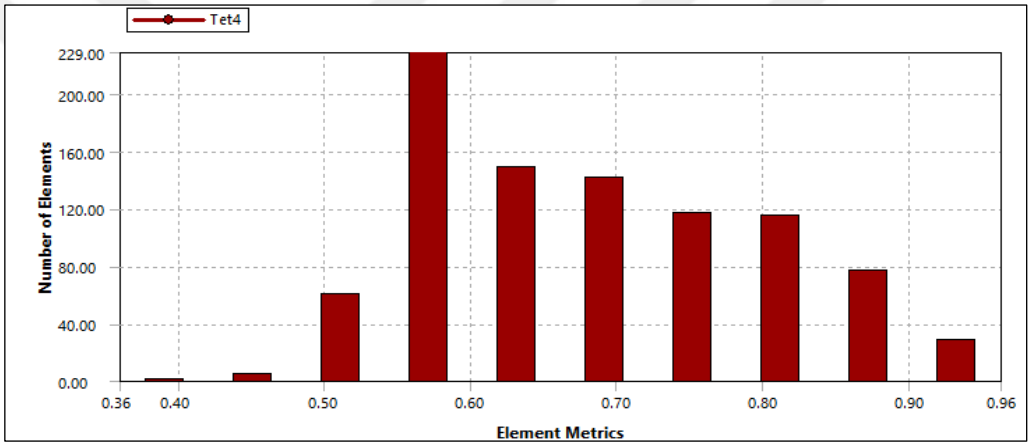
Tablo 4.1. Sınır koşulları

Bölge Adı	**Bölge Tanımlayıcısı**	**Koşul Tipi**	**Koşul Değeri**
Üst Yüz	5	Sıcaklık Sınır Koşulu	500K
Alt Yüz	7	Sıcaklık Sınır Koşulu	100K
Silindirik Yüz	6	Konveksiyon Sınır Koşulu	500 W/m^2K katsayı ile 300K'e
Yan Yüzler	8	Akı Sınır Koşulu	100000 W/m^2
Diğer Yüzler	-	Adyabatik	-
Tüm Hacim	2	Isı Üretimi	100000 W/m^3

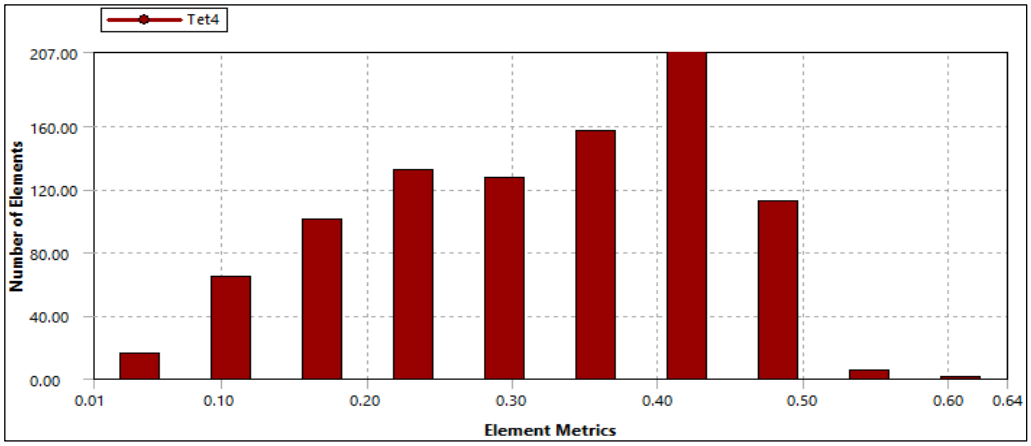
İlk durumda düşük çözünürlüğe, yüksek çarpıklığa ve düşük orotogonalliğe sahip Şekil 4.1'deki ağ yapısı alınmıştır. Ansys ortamından ağ yapısı için alınan çarpıklık grafiği Şekil 4.3'de, ortogonallik grafiği ise Şekil 4.2'de verilmiştir. Bu durum için sonuçlar Tablo 4.2'de özetlenmiştir. Görüldüğü üzere sıcaklıklar ve yüzlerden geçen akılarda bi fark mevcuttur. Bu farklılık ikinci durumda daha detaylı irdelenmiştir.



Şekil 4.1. Düşük çözünürlüklü, yüksek çarpıklığa ve düşük ortogonalliğe sahip ağ yapısı



Şekil 4.2. Düşük çözünürlüklü, yüksek çarpıklığa ve düşük ortogonalliğe sahip ağ yapısının ortogonal kalitesi



Şekil 4.3. Düşük çözünürlüklü, yüksek çarpıklığa ve düşük ortogonalliğe sahip ağ yapısının çarpıklığı

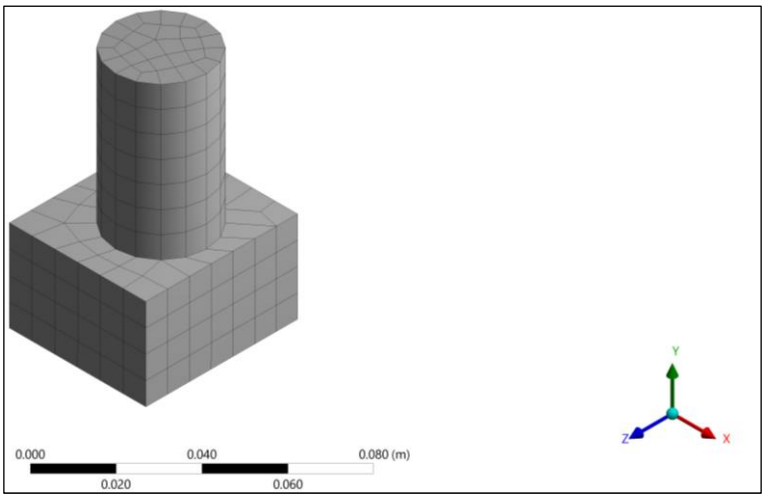
Tablo 4.2. Düşük çözünürlüklü, yüksek çarpıklığa ve düşük ortogonalliğe sahip ağ yapısı için karşılaştırma

Bölge Adı	Toplam ısı üretimi	Ortalama Sıcaklık Farkı	Toplam Akı (Fluent)	Toplam Akı (Yazılım)
Üst Yüz		0,3079K	913,03351 W	920,494751 W
Alt Yüz			-1384,9688 W	-1391,567749 W
Silindirik Yüz			-108,1617 W	-109,017014 W
Yan Yüzler			570 W	
Tüm Hacim	10,0932 W			
		Net Akı	-10,09699 W	-10,090252 W
		Net	-3,80E-03 W	2,95E-03 W

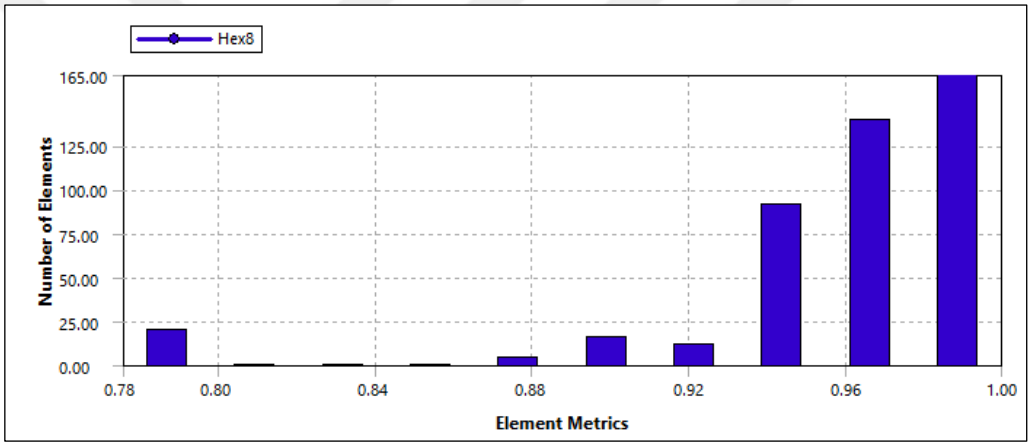
İkinci durumda düşük çözünürlüklü ve birinci durumdakine kıyasla yüksek ortogonal kaliteye sahip bir ağ yapısı kullanılmıştır. Ağ yapısı ile ilgili bir görsel Şekil 4.4'de, ortogonal kalitesi Şekil 4.5'de çarpıklığı ise Şekil 4.6'da verilmiştir. Bu durumda elde edilen sonuçlar ilk durumda elde edilen sonuçlara kıyasla çok daha yakındır. Buradan anlaşılacağı üzere Fluent'in ortogonalliğe bağlı olarak kullandığı kontrol hacmi yüzeyine normal yönde gradyen düzeltme yaklaşımı geliştirilen yazılımda kullanılan yaklaşımdan farklıdır. Sonuçlar Tablo 4.3'de özetlenmiştir.

Tablo 4.3. Düşük çözünürlüklü, yüksek çarpıklığa ve yüksek ortogonal kaliteye sahip ağ yapısı için karşılaştırma

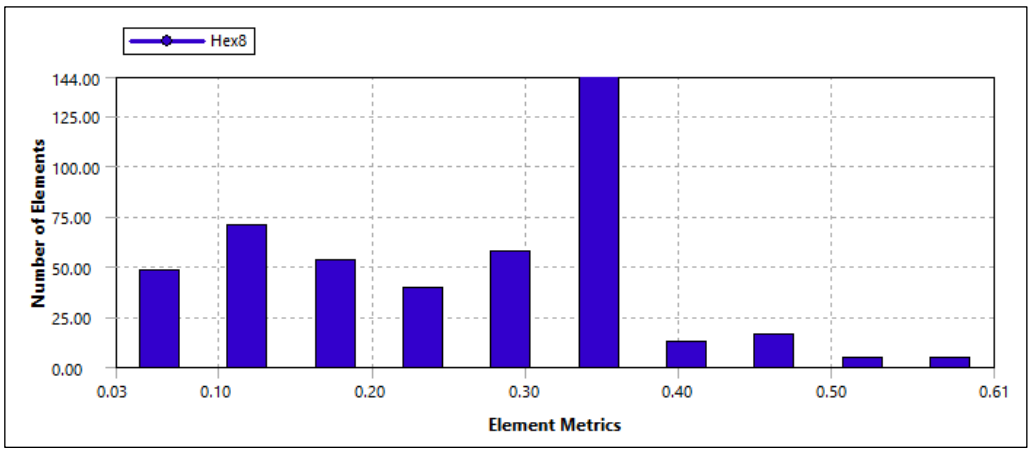
Bölge Adı	Toplam ısı üretimi	Ortalama Sıcaklık Farkı	Toplam Akı (Fluent)	Toplam Akı (Yazılım)
Üst Yüz		8,04E-02K	937,35449 W	937,096802 W
Alt Yüz			-1405,9288 W	-1405,559082 W
Silindirik Yüz			-111,62403 W	-111,736008 W
Yan Yüzler			570 W	
Tüm Hacim	10,194022 W			
		Net Akı	-10,19834 W	-10,197895 W
		Net	-4,47E-03 W	-3,87E-03 W



Şekil 4.4. Düşük çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısı



Şekil 4.5. Düşük çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısının ortogonal kalitesi

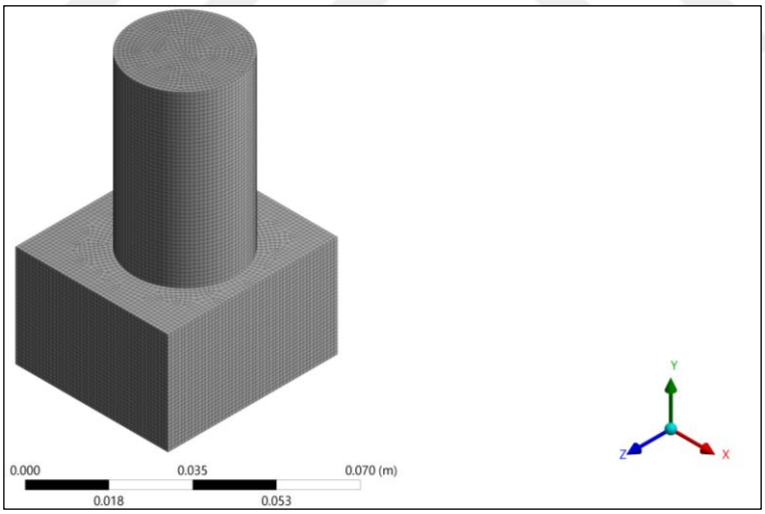


Şekil 4.6. Düşük çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısının çarpıklığı

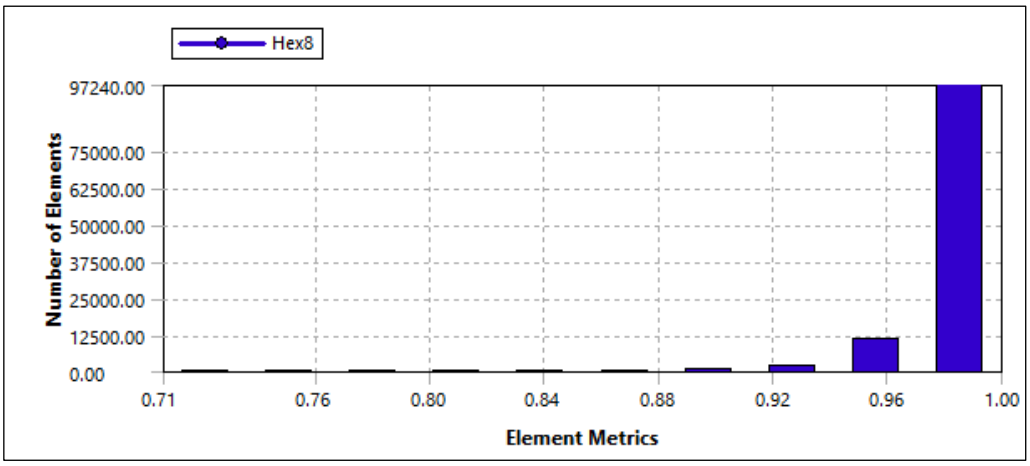
Son olarak yüksek çözünürlüklü ve yüksek ortogonal kaliteye sahip bir ağ yapısı için yapılan test tekrarlanmıştır. Kullanılan ağ yapısı Şekil 4.7'de, ortogonal kalite Şekil 4.8'de ve çarpıklıkta Şekil 4.9'de verilmiştir. Bu durumda elde edilen sonuçlarda ise Fluent'in çözümü ve geliştirilen yazılımın verdiği çözüm alanı arasındaki fark rahatlıkla yuvarlama hatası denilebilecek kadar küçüktür. İlk durumda düşük çözünürlükve düşük ortogonal kaliteye sahip ağ yapısı için alınan çözümlerden farklıdır ve bu da beklenen bir durumdur. Karşılaştırma Tablo 4.4'de özetlenmiştir.

Tablo 4.4. Yüksek çözünürlük ve yüksek ortogonal kaliteye sahip ağ yapısı için karşılaştırma

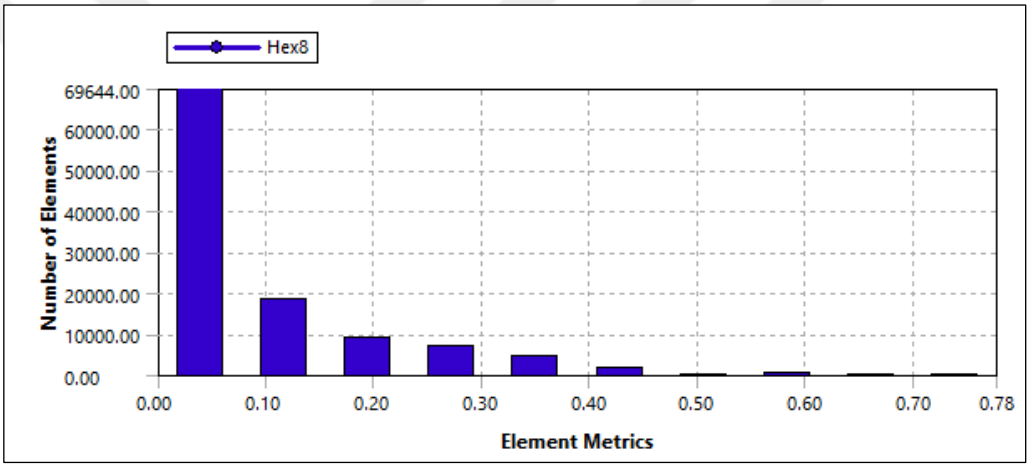
Bölge Adı	**Toplam ısı üretimi**	**Ortalama Sıcaklık Farkı**	**Toplam Akı (Fluent)**	**Toplam Akı (Yazılım)**
Üst Yüz		5,277E-08 K	973,94043	973,94397
Alt Yüz			-1447,6664	-1447,665161
Silindirik Yüz			-106,55096	-106,551903
Yan Yüzler			569,999 W	
Tüm Hacim	10,281748 W			
		Net Akı	-10,27693	-10,276181
		Net	-5,18E-03	5,58E-03



Şekil 4.7. Test amaçlı kullanılan yüksek çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısı



Şekil 4.8. Yüksek çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısının ortogonal kalitesi



Şekil 4.9. Yüksek çözünürlüğe ve yüksek ortogonalliğe sahip ağ yapısının çarpıklığı

5. SONUÇLAR VE ÖNERİLER

Bu çalışmada, ilk olarak gerekli temel bilgiler, genel korunum denklemi ile ilgili açıklamalar ve sonlu hacimler metodu hakkında bilgiler verilmiştir. Tez kapsamında geliştirilen yazılım ve kütüphane ile ilgili olarak yapısal açıklamalar yapılmıştır. Geliştirilen yazılım ve kütüphane genel korunum denklemi formunda bir denklemi verilen bir geometri üzerinde yorumlayabilmekte, modelleyebilmekte ve çözebilmektedir. Yazılım yapısal olmayan 3B bir ağ yapısına destek vermektedir. Ayrıca yazılan kodun ekran kartında da yürütülebileceği belirtilmiştir. Bu çalışma bu konuda yapılacak olan çalışmalara bir temel niteliğinde olup ortaya çıkan kütüphane olabildiğince yeni eklemelere açık bir şekilde geliştirilmiştir. Yeni SHM şemaları yanısıra sonlu elemanlar metodu gibi metodlar eklenerek kütüphane geliştirilebilir. Bunun yanında çözücü ile eklemeler, matris veri yapıları ile ilgili eklemeler, farklı ağ yapıları için sınıflar vs. birçok ekleme yapılabilir. Son olarak kod bir difüzyon denklemi için aynı geometri üstünde farklı durumlarda test edilmiştir. Testler sonucunda beklenen sonuçlar elde edilmiş, kodun verdiği çözüm doğrulanmış ve sonuçlar bu tez dokümanında açıklanmıştır.

KAYNAKLAR

[1] Mangani L., Moukalled F. and Darwish M., *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab*, 1st ed., Springer, Switzerland, 2016.

[2] Marsden J. E. and Tromba A., *Vector Calculus*, 6rd Ed., W. H. Freeman and Company, New York, 2012.

[3] Çengel Y. A. and Cimbala J. M., *Fluid Mechanics: Fundamentals and Applications*, 1st ed., McGraw-Hill, New York, 2006.

[4] Patankar S. V., *Numerical Heat Transfer and Fluid Flow*, 1st ed., Hemisphere Publishing Corporation, New York, 1980.

[5] Versteeg H. K. and Malalasekera M., *An Introduction to Computational Fluid Dynamics: Finite Volume Method*, 2nd ed., Bell & Bain Limited, Glasgow, 2007.

[6] Press W. H., Teukolsky S. A., Vetterling W. T. and Flannery B. P., *Numerical Recipes*, 3rd ed., Cambiridge University Press, New York, 2007.

[7] https://en.cppreference.com/, (Ziyaret tarihi : 31/12/2019).

[8] https://isocpp.org/, (Ziyaret tarihi : 31/12/2019).

[9] ANSYS Fluent User's Guide, ANSYS, Inc., 2008.

[10] https://docs.nvidia.com/cuda/, (Ziyaret tarihi : 01/01/2020).

KİŞİSEL YAYIN VE ESERLER

[1] Onsal M., **Cumhur B.**, Demir Y., Yolacan E., Aydin M., Rotor Design Optimization of A New Flux Assisted Consequent Pole Spoke Type Permanent Magnet Torque Motor for Low Speed Applications, *IEEE Transactions on Magnetics*, DOI: 10.1109/TMAG.2018.2832076.

ÖZGEÇMİŞ

1993 İzmir doğumlu. İlköğretimi Merkez Efendi İlköğretim Okulu'nda, Lise öğrenimini Manisa Lisesi'nde tamamladıktan sonra 2012 yılında Kocaeli Üniversitesi Makina Mühendisliği bölümünde üniversite eğitimine başladı. Üniversite eğitimi devam ederken 2014 yılında çift anadal programı ile aynı üniversitenin Mekatronik Mühendisliği bölümüne kaydoldu. 2016 yılında da Makina Mühendisi olarak mezun oldu. 2017 yılında yüksek lisans eğitimine Kocaeli Üniversitesi Fen Bilimleri Enstitüsü Makina Mühendisliği Anabilim Dalı'nda başladı. Aynı zamanda 2016 yılından beri MDS Motor Ltd. bünyesinde Ar-Ge Mühendisi olarak çalışmaktadır.