汎用境界要素解析インターフェイスプログラムの開発

関西大学 環境都市工学部 建築学科
建築環境工学第Ⅰ研究室
建 21-0071　野村 悠帆
指導教員　豊田 政弘

目次

第 1 章　序論 …… 1
1.1　研究背景 …… 1
1.2　研究目的 …… 2

第 2 章　境界要素法の概要 …… 3
2.1　境界要素法の特徴 …… 3
2.2　Gauss の発散定理 …… 4
2.3　Green の定理 …… 4
2.4　Kirchhoff-Helmholtz の境界積分方程式 …… 5
2.5　境界積分方程式 …… 7

第 3 章　研究手法 …… 10
3.1　メッシュの作成方法 …… 10
3.2　開発したインターフェイスプログラム …… 14

第 4 章　研究条件 …… 17
4.1　対象空間 …… 17
4.2　計算条件 …… 18

第 5 章　研究結果 …… 19
5.1　他手法との比較による精度の検証 …… 19
5.2　積分方法の違いによる影響の検討 …… 20

第 6 章　結論 …… 22

参考文献 …… 23

付 録 A　プログラムのソースコード …… 24
A.1　Control.cs …… 24
A.2　Viewer.cs …… 27
A.3　Surf.cs …… 32
A.4　Vector3D.cs …… 34
A.5　Point3D.cs …… 37

第 1 章　序論

1.1　研究背景

建築音響分野における音場解析の手法には、波動音響理論、すなわち、音波の波動性を考慮した理論に基づいたものがある。この理論に基づいた手法で音場解析を行うには音波の挙動を記述する波動方程式を解く必要があるが、複雑な形状に適用する場合は計算量が膨大になる。

しかし近年では計算機性能の発展に伴い、波動音響理論に基づいた数値解析手法が広く利用されつつある。数値解析シミュレーションは音場の予測とその最適化に大きな役割を果たしており、ますます重要性が増している。設計初期の段階で問題点を予測し、改善策を講じることができるうえ、複雑な形状や材料特性にも対応できるため、実際に実験を行うよりも効率的かつ低コストであると言える。

波動音響理論に基づいた数値解析手法のひとつとして、境界要素法（Boundary Element Method、BEM）が挙げられる。境界要素法とは解析対象である空間を囲む境界を離散化することで解を得る手法である。境界要素法は時間領域有限差分法（Finite-Difference Time-Domain 法、FDTD 法）や有限差分法（Finite Element Method、FEM）などの他の数値解析手法と比較すると、計算の効率や精度の面で有利になる場合があり、高度なシミュレーション技術が求められる近年では、境界要素法の適用はより重要なものとなっている。境界要素法を用いた市販の数値解析ソフトウェアは既にいくつか存在しているが、その価格は比較的高価であり、気軽に使用できるものは少ないのが現状である。

1.2　研究目的

本研究の目的は、境界要素法を用いた数値解析ソフトウェアのユーザーインターフェース（UI）の開発、及び他の解析手法との比較、積分計算の違いによる精度の検証を行うことで、その利便性と実用性を低コストで向上させることである。

今回、境界要素法の計算エンジンについては本研究の指導教員である豊田によって提供される。本研究で開発する UI が、解析を行う形状モデルの読み込み、計算に必要な条件の設定、ならびに計算エンジンが読み込み可能な形式での書き出しといった役割を果たすことで、境界要素法による数値解析シミュレーションの一連の流れを可能にする。

第 2 章　境界要素法の概要

2.1　　境界要素法の特徴

境界要素法の最大の特徴として、計算が空間の境界のみに限定されるため、計算時のメモリを節約できること、およびメッシュ作成が簡単であることが挙げられる。また、開領域空間を誤差なく計算できることも特徴の一つである。

空間をある単位で区切ることを離散化と呼ぶが、境界要素法においては空間を取り囲む境界のみを離散化する。そのイメージを図 2.1.1 に示す。境界上の離散化された点を「節点」と呼び、節点を結んで構成される境界の小区間を「要素」と呼ぶ。計算負荷や精度は節点数や要素の粗さに大きく依存するため、解析の目的に応じた適切な離散化が重要となる。

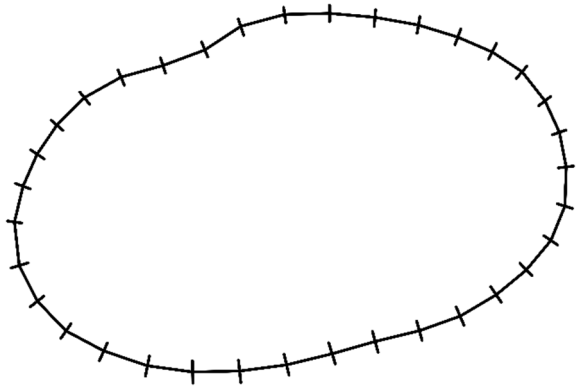

図 2.1.1 境界要素法 離散化イメージ

2.2 Gauss の発散定理

Gauss の発散定理とはベクトル場の発散の体積積分を、その境界面での面積分に変換する定理である。閉曲面 S で囲まれた領域 V において、ベクトル関数$\mathbf{A} = [A_x A_y A_z]^{\mathrm{T}}$について、次のように表される。

$$\int_{\mathrm{V}} \nabla^{\mathrm{T}} \mathbf{A} dV = \int_{\mathrm{S}} \mathbf{A}^{\mathrm{T}} \mathbf{n} dS \tag{2.1}$$

ここで、$\mathbf{n}= \left[n_x n_y n_z\right]^{\mathrm{T}}$は境界面の外向き単位法線ベクトルである。この定理を用いることで、領域内部の積分を境界面上の積分に書き換えることができる。

2.3 Green の定理

式(2.1)に、$\mathbf{A}= u\nabla v$を代入する。左辺は、

$$\begin{aligned}
\nabla^{\mathrm{T}}\mathbf{A} &= \frac{\partial}{\partial x}\left(u\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(u\frac{\partial v}{\partial y}\right) + \frac{\partial}{\partial z}\left(u\frac{\partial v}{\partial z}\right) \\
&= \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + u\frac{\partial^2 v}{\partial x^2} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} + u\frac{\partial^2 v}{\partial y^2} + \frac{\partial u}{\partial z}\frac{\partial v}{\partial z} + u\frac{\partial^2 v}{\partial z^2} \\
&= u\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) + \left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial u}{\partial z}\frac{\partial v}{\partial z}\right) \\
&= u\nabla^2 v + (\nabla u)^{\mathrm{T}}\nabla v
\end{aligned} \tag{2.2}$$

となり、右辺は、

$$\mathbf{A}^{\mathrm{T}}\mathbf{n} = u(\nabla \mathrm{v})^{\mathrm{T}}\mathbf{n} = u\left(\frac{\partial v}{\partial x}n_x + \frac{\partial v}{\partial y}n_y + \frac{\partial v}{\partial z}n_z\right) = u\frac{\partial v}{\partial \mathbf{n}} \tag{2.3}$$

となるため、

$$\int_V \{u\nabla^2 v + (\nabla u)^{\mathrm{T}}\nabla v\}\, dV = \int_S u\frac{\partial v}{\partial \mathbf{n}} dS \tag{2.4}$$

が導かれる。また、同様に$\mathbf{A}= v\nabla u$の場合を考えると、式(2.1)は、

$$\int_V \{v\nabla^2 u + (\nabla v)^{\mathrm{T}}\nabla u\}\, dV = \int_S u\frac{\partial u}{\partial \mathbf{n}} dS \tag{2.5}$$

となり、式(2.4)、式(2.5)の差をとると、

$$\int_V \{u\nabla^2 v - v\nabla^2 u\}\, dV = \int_S \left(u\frac{\partial v}{\partial \mathbf{n}} - u\frac{\partial u}{\partial \mathbf{n}}\right) dS \tag{2.6}$$

となる。式(2.4)、式(2.6)を Green の定理と呼び、式(2.4)は Green の第一恒等式、式(2.6)は Green の第二恒等式である。

2.4 Kirchhoff-Helmholtz の積分方程式

音波の伝播を記述する基本的な微分方程式として、Helmholtz 方程式があり、次のように表される。

$$\nabla^2 p + k^2 p = 0 \tag{2.7}$$

p：空間的な波の形状を決める関数　　$k[m^{-1}]=\omega/c$：波数

ω[rad/s]：角周波数　　c[m/s]：音速

Helmholtz 方程式の右辺に負符号のデルタ関数を与えた場合の解を Green 関数と呼ぶ。さらに Green 関数のうち、自由空間を対象としたものを基本解と呼ぶ。基本解$\mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})$を導入すると、式(2.7)より、

$$\nabla^2 \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime}) + k^2 \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime}) = -\delta(\mathbf{r}_{\mathrm{xx}\prime}) \tag{2.8}$$

と表され、$\mathbf{r}$ は位置ベクトル、$\mathbf{r}_{\mathrm{xx}\prime}$は xx'間の距離である。デルタ関数[1/s]は単位体積当たりの体積速度である。3 次元音場の場合、この解は

$$\mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime}) = \frac{e^{ikr_{\mathrm{xx}'}}}{4\pi \mathbf{r}_{\mathrm{xx}\prime}} \tag{2.9}$$

である。

式(2.8)を Green の第二恒等式（式(2.6)）に代入すると、

$$\int_\mathrm{V} \{p\nabla^2 \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime}) - \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})\nabla^2 p\}\,\mathrm{dV} = \int_\mathrm{S} \left(p\frac{\partial \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})}{\partial \mathbf{n}} - \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})\frac{\partial \mathrm{p}}{\partial \mathbf{n}}\right)\mathrm{dS} \tag{2.10}$$

となる。ここでヘルムホルツ方程式（式(2.7)）を使うと、

$$C(\mathbf{r}_\mathrm{x})p(\mathbf{r}_\mathrm{x}) + \int_S p(\mathbf{r}_{\mathrm{x}\prime})\frac{\partial \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})}{\partial \mathbf{n}}\,dS = \int_V \mathrm{G}(\mathbf{r}_\mathrm{x},\mathbf{r}_{\mathrm{x}\prime})\frac{\partial p(\mathbf{r}_{\mathrm{x}\prime})}{\partial \mathbf{n}}\,dV \tag{2.11}$$

となる。このように領域積分が消去され、境界積分のみが残る。これを Kirchhoff-Helmholtz の積分方程式と呼ぶ。

均質な媒質で満たされた空間中に点音源、受音点 p、境界面上の点 q があり、q における速度ポテンシャルを$\varphi(\mathbf{r}_\mathrm{q})$[m²/s]、領域の境界を S、領域を V、S の内部を V'とする。φは式(2.7)を満たすので、

$$\nabla^2\varphi(\mathbf{r}_\mathrm{q}) + k^2\varphi(\mathbf{r}_\mathrm{q}) = 0 \qquad (\mathbf{2.12})$$

基本解を G($\mathbf{r}_\mathrm{p}$,$\mathbf{r}_\mathrm{q}$)とすると、式(2.11)、式(2.12)より、

$$C(\mathbf{r}_\mathrm{p})\varphi(\mathbf{r}_\mathrm{p}) + \int_S \varphi(\mathbf{r}_\mathrm{q})\frac{\partial \mathrm{G}(\mathbf{r}_\mathrm{p},\mathbf{r}_\mathrm{q})}{\partial \mathbf{n}_\mathrm{q}}dS = \int_V \mathrm{G}(\mathbf{r}_\mathrm{p},\mathbf{r}_\mathrm{q})\frac{\partial \varphi(\mathbf{r}_\mathrm{q})}{\partial \mathbf{n}_\mathrm{q}}dV \qquad (\mathbf{2.13})$$

となる。ここで、$C(\mathbf{r}_\mathrm{p})$は係数であり、受音点 p の位置によって異なる。p が領域 V 内に存在する場合は、速度ポテンシャルは完全に再現されるため、$C(\mathbf{r}_\mathrm{p}) = 1$、p が境界 S 上にある場合は、領域内外からの影響が半々となり、積分が半分だけ寄与するため、$C(\mathbf{r}_\mathrm{p}) = 1/2$、p が領域外部 V'にある場合は、内部の影響を受けないため、$C(\mathbf{r}_\mathrm{p}) = 0$となる。これを踏まえ、式(2.13)は次のように表すことができる。

$$\varphi_\mathrm{D}(\mathbf{r}_\mathrm{p}) + \int_S \left\{\varphi(\mathbf{r}_\mathrm{q})\frac{\partial \mathrm{G}(\mathbf{r}_\mathrm{p},\mathbf{r}_\mathrm{q})}{\partial \mathbf{n}_\mathrm{q}} - \mathrm{G}(\mathbf{r}_\mathrm{p},\mathbf{r}_\mathrm{q})\frac{\partial \varphi(\mathbf{r}_\mathrm{q})}{\partial \mathbf{n}_\mathrm{q}}\right\}dS = \begin{cases} \varphi(\mathbf{r}_\mathrm{p}) & (\mathrm{p} \in \mathrm{V}) \\ \dfrac{\varphi(\mathbf{r}_\mathrm{p})}{2} & (\mathrm{p} \in \mathrm{S}) \\ 0 & (\mathrm{p} \in \mathrm{V}') \end{cases} \qquad (\mathbf{2.14})$$

特にp ∈ Sの場合を Kirchhoff-Helmholtz の境界積分方程式と呼ぶ。

2.5　境界積分方程式

S 上の速度ポテンシャル$\boldsymbol{\varphi}$とその法線微分を求める方法を述べる。境界積分方程式を解くにあたって、S 上の境界条件を設定する必要がある。本研究では境界 S の垂直入射表面インピーダンス比が$Z_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})$[-]である場合を考える。

垂直入射表面インピーダンス比は垂直入射の音波に対する表面インピーダンスを、媒質の音響インピーダンスで基準化したものであり、次のように表される。

$$Z_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}}) = \frac{Z_S}{\rho_0 c_0} \tag{2.15}$$

Z_S [Pa · s/m] $= p/v$：表面インピーダンス　p [Pa]：表面での音圧
v [m/s]：表面の法線方向の粒子速度　ρ_0 [kg/m^3]：媒質密度　c_0 [m/s]：媒質中の音速

また、$p = -i\omega\rho_0\varphi$であり、$v(\mathbf{r}_{\mathrm{q}}) = \frac{\partial\varphi(\mathbf{r}_{\mathrm{q}})}{\partial\mathbf{n}_{\mathrm{q}}}$ であることから、

$$Z_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}}) = \frac{1}{\rho_0 c_0}\frac{p(\mathbf{r}_{\mathrm{q}})}{-v(\mathbf{r}_{\mathrm{q}})} = \frac{1}{\rho_0 c_0}\frac{-i\omega\rho_0\varphi(\mathbf{r}_{\mathrm{q}})}{\dfrac{\partial\varphi(\mathbf{r}_{\mathrm{q}})}{\partial\mathbf{n}_{\mathrm{q}}}}$$
$$\leftrightarrow \frac{\partial\varphi(\mathbf{r}_{\mathrm{q}})}{\partial\mathbf{n}_{\mathrm{q}}} = -\frac{ik\varphi(\mathbf{r}_{\mathrm{q}})}{Z_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})} = -ikA_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})\varphi(\mathbf{r}_{\mathrm{q}}) \tag{2.16}$$

となる。ここで$A_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})$[-]は垂直入射表面アドミッタンス比である。式(2.14)より、境界積分方程式は

$$\varphi(\mathrm{r_p}) - 2\int_S \left\{\varphi(\mathbf{r}_{\mathrm{q}})\frac{\partial \mathrm{G}(\mathbf{r}_{\mathrm{p}},\mathbf{r}_{\mathrm{q}})}{\partial\mathbf{n}_{\mathrm{q}}} + ikA_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})\varphi(\mathbf{r}_{\mathrm{q}})\mathrm{G}(\mathbf{r}_{\mathrm{p}},\mathbf{r}_{\mathrm{q}})\right\}dS = 2\varphi_{\mathrm{D}}(\mathbf{r}_{\mathrm{p}}) \tag{2.17}$$

となる。これを解くために S を N 個の微小な平面要素に離散化する。p の位置に対応する要素が i 番目（i=1, …, N）であるとし、その要素の中心を$\mathbf{r}_i$とする。q の位置に対応する要素が j 番目（j=1, …, N）であるとし、その要素の中心を$\mathbf{r}_j$、要素の面積を$\Delta S_j[m^2]$とする。また、ΔS_j上では速度ポテンシャル$\boldsymbol{\varphi}(\mathbf{r}_j)$は一定であるものと仮定する。この時、式(2.17)は

$$\varphi(\mathbf{r}_i) - 2\sum_{j=1}^{N}\varphi(\mathbf{r}_{\mathrm{q}})\left\{\int_{\Delta S_j}\frac{\partial \mathrm{G}(\mathbf{r}_i,\mathbf{r}_{\mathrm{q}})}{\partial\mathbf{n}_{\mathrm{q}}}dS + ikA_{\mathrm{n}}(\mathbf{r}_{\mathrm{q}})\int_{\Delta S_j}\mathrm{G}(\mathbf{r}_i,\mathbf{r}_{\mathrm{q}})dS\right\} = 2\,\varphi_{\mathrm{D}}(\mathbf{r}_i) \tag{2.18}$$

と近似される。ここで、

$$\int_{\Delta S_j}\mathrm{G}(\mathbf{r}_i,\mathbf{r}_{\mathrm{q}})dS = \int_{\Delta S_j}\frac{e^{ikr_{i\mathrm{q}}}}{4\pi\mathrm{r}_{i\mathrm{q}}}dS \tag{2.19}$$

である。$i=j$の場合は$\mathrm{r}_{iq}=0$となり、発散するため計算上の工夫をする必要がある。同一平面内に *i, j* 番目の要素がない場合には、関数の積分をより高精度に計算することができる、Gauss-Legendre 積分などを用いて式(2.19)を計算すればよい。同一平面内に *i , j* 番目の要素がある場合で$i \neq j$のときは、図 2.5.1 のようにr_{iq}, θ をとり、

$$\begin{aligned}\int_{\Delta S_j} \frac{e^{ikr_{iq}}}{4\pi \mathrm{r}_{iq}} dS &= \frac{1}{4\pi}\int_{\theta_1}^{\theta_2}\int_{r_1(\theta)}^{r_2(\theta)} \frac{e^{ikr_{iq}}}{\mathrm{r}_{iq}} \mathrm{r}_{iq} d\mathrm{r}_{iq} d\theta \\ &= \frac{1}{4\pi}\int_{\theta_1}^{\theta_2} \left[\frac{e^{ikr_{iq}}}{ik}\right]_{r_1(\theta)}^{r_2(\theta)} d\theta \\ &= \frac{1}{4\pi ik}\int_{\theta_1}^{\theta_2} \left(e^{ikr_2(\theta)} - e^{ikr_1(\theta)}\right) d\theta \\ &= \frac{1}{4\pi ik}\oint e^{ik\mathrm{r}(\theta)}\, d\theta\end{aligned} \qquad (\mathbf{2.20})$$

と積分変換することで求めることができる。一方、$i=j$のときは、図 2.5.2 のように、

$$\begin{aligned}\int_{\Delta S_j} \frac{e^{ikr_{iq}}}{4\pi \mathrm{r}_{iq}} dS &= \frac{1}{4\pi}\int_{0}^{2\pi}\int_{0}^{\mathrm{r}(\theta)} \frac{e^{ikr_{iq}}}{\mathrm{r}_{iq}} \mathrm{r}_{iq} d\mathrm{r}_{iq} d\theta \\ &= \frac{1}{4\pi}\int_{0}^{2\pi} \left[\frac{e^{ikr_{iq}}}{ik}\right]_{0}^{\mathrm{r}(\theta)} d\theta \\ &= \frac{1}{4\pi ik}\int_{0}^{2\pi} \left(e^{ik\mathrm{r}(\theta)} - 1\right) d\theta \\ &= \frac{1}{4\pi ik}\oint e^{ik\mathrm{r}(\theta)}\, d\theta - \frac{1}{2ik}\end{aligned} \qquad (\mathbf{2.21})$$

で求めることができる。

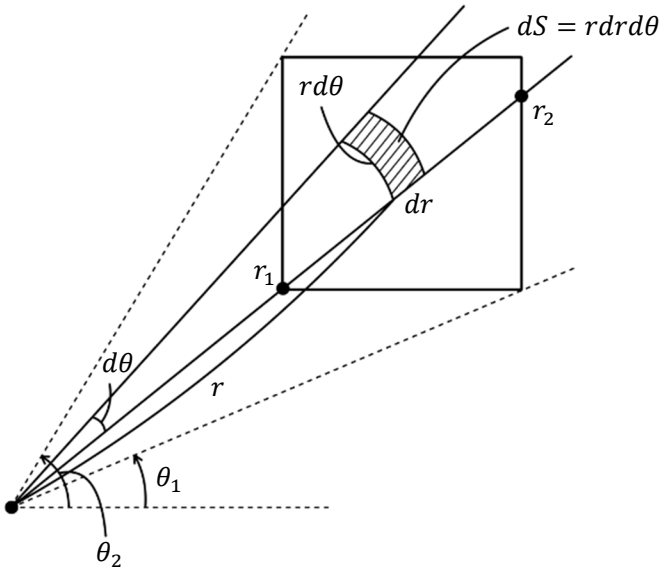

図 2.5.1 同一平面内での積分

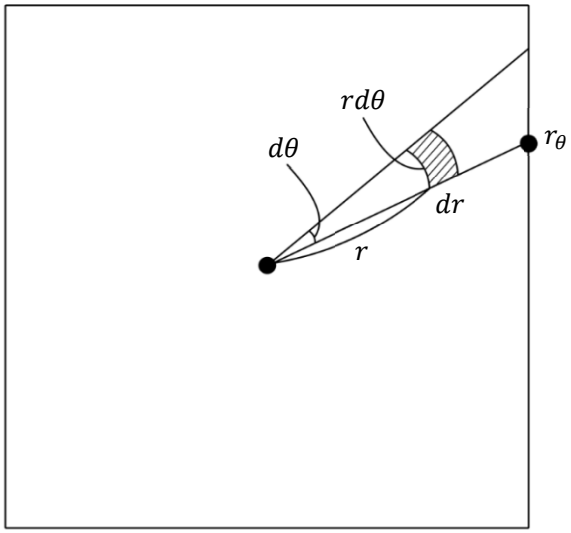

図 2.5.2 同一要素内での積分

第 3 章　研究手法

3.1　　メッシュの作成方法

境界要素法の計算には、解析対象物の表面にメッシュを作成する必要がある。ここではその手順について示す。本研究ではメッシュ作成のためのツールとして、フリーソフトウェアである Gmsh [1]を用いた。Gmsh は 3D モデルを読み込み、任意のサイズにメッシュを再分割することができる。

まず、解析対象物の 3D モデルを形状モデラ（CAD ソフトウェア）で作成する。本研究では Shade3D [2]を用いて、(x, y, z)＝(7700, 4800, 2800) mm の直方体音場を作成した。作成したモデルを stl ファイルとして書き出す。今回は room.stl と名付けて書き出した。Gmsh では geo ファイルとして読み込む必要があるため、図 3.1.1 のようなコードで stl ファイルを読み込む geo ファイルを作成した。1 行目の赤い枠で囲ったファイル名を変更することで任意の stl ファイルを読み込むことが可能である。

```
1  Merge "room.stl";↓
2  ↓
3  angle = 1e1;↓
4  includeBoundary = 1;↓
5  forceParametrizablePatches = 0;↓
6  ↓
7  ClassifySurfaces{angle * Pi / 180, includeBoundary, forceParametrizablePatches};↓
8  ↓
9  CreateGeometry;↓
10 ↓
11 Surface Loop(1) = Surface{:};↓
12 ↓
13 //Transfinite Surface "*";↓
14 //Recombine Surface "*";↓
15 ↓
16 Mesh.MeshSizeExtendFromBoundary = 0;↓
17 Mesh.MeshSizeFromPoints = 0;↓
18 Mesh.MeshSizeFromCurvature = 0;↓
19 ↓
20 DefineConstant[↓
21   MeshSizeTmp = {1000, Min 0, Max 1e100, Step 1, Name "MeshSize"}↓
22 ];↓
23 ↓
24 Mesh.MeshSizeMin = MeshSizeTmp;↓
25 Mesh.MeshSizeMax = MeshSizeTmp;↓
26 ↓
27 Mesh.SurfaceFaces = 1;↓
28 //Mesh.RecombineAll = 1;↓
29 //Mesh.SubdivisionAlgorithm = 1;↓
30 [EOF]
```

図 3.1.1 作成した geo ファイル

作成した geo ファイルを Gmsh に読み込む。その様子を図 3.1.2 に示す。

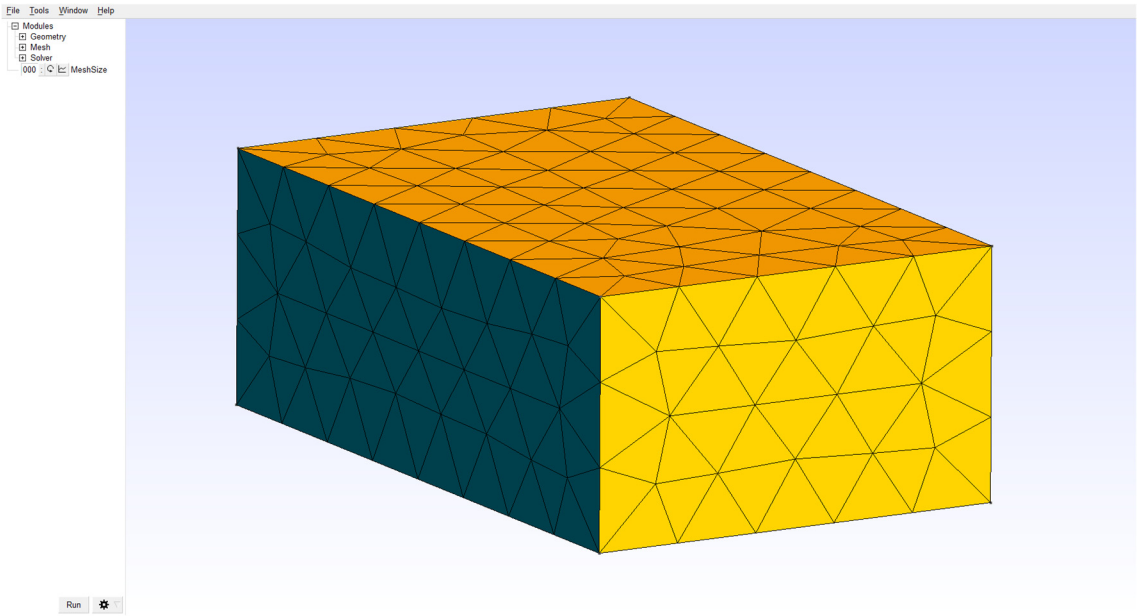

図 3.1.2 Gmsh の GUI

［Mesh］の下にある［2D］（図 3.1.3）をクリックし、メッシュの再分割を行う。

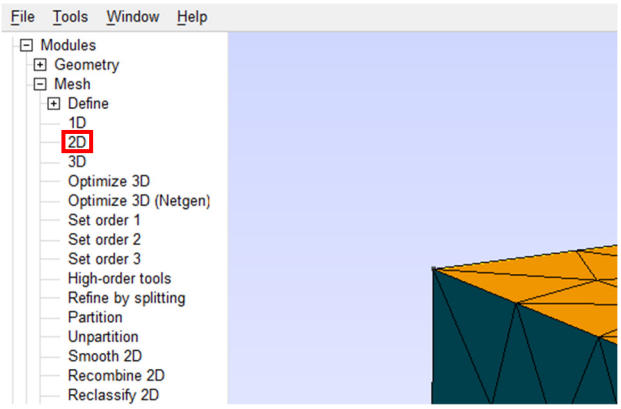

図 3.1.3 分割を行う[2D]のボタン

[Gmsh]の MeshSize（図 3.1.4）の値を調節することで分割後のメッシュの粗さを変更することが可能である。値が小さいほどメッシュは小さくなり、値が大きいほどメッシュは大きくなる。メッシュが小さいほど解析の精度は上がるが、計算の処理時間は長くなる。また、[Parameters]の Angle for surface detection（図 3.1.4）から、エッジを認識する角度を変更することができる。角度が小さいほど、隣接する面を独立した面として認識しやすい。小さくしすぎると、別の面として分離されてしまう面が多く、メッシュに穴

が開くなど不具合が生じる場合がある。また逆に大きすぎると、独立した面と認識しにくいため、再分割を行った際に曖昧な形状になる場合がある。他にも様々なパラメーターが存在するが、特にこれら二つの数値を適切に設定することで望み通りの再分割が可能となる。

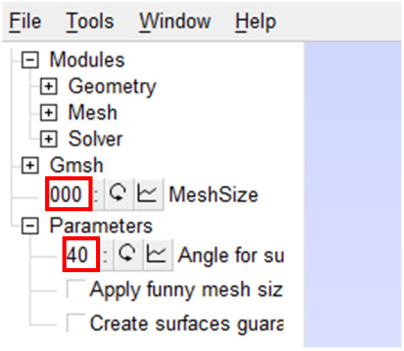

図 3.1.4 MeshSize , Angle for surface detection の調節欄

図 3.1.5 はメッシュサイズを小さくした際と大きくした際の再分割を比較した図である。

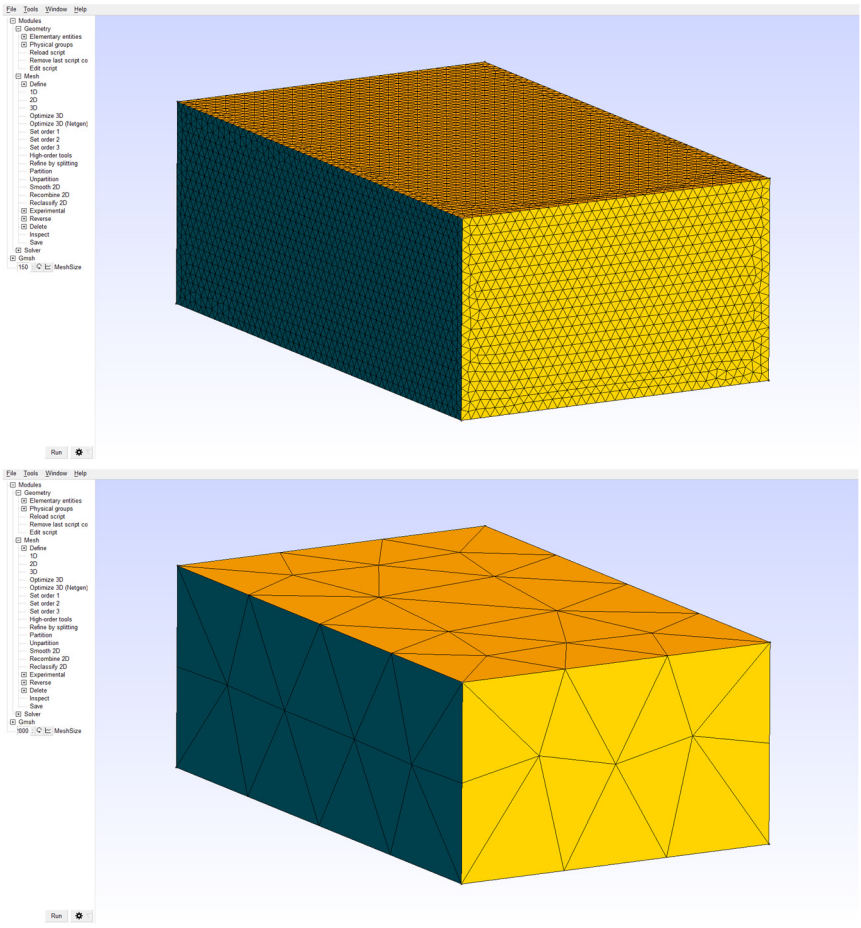

図 3.1.5 メッシュサイズの大小の比較

また、メッシュサイズについては田中らの研究[3]にあるように、要素の最大の長さを波長の 1/8 以下に保てば実用上問題ないとされている。したがって、これを目安にメッシュサイズを調節するのがよい。後述するが、開発したインターフェイスプログラムの中では読み込んだモデルの最大の要素の長さを確認することができる機能を設けた。最後に[File]→[Export]より Gmsh から再び stl ファイルとして書き出す。ここでは、再び room.stl ファイルと名付けて書き出した。

3.2　開発したインターフェイスプログラム

ここでは開発したインターフェイスプログラムの概要とその役割について述べる。プログラミング言語は C#、開発環境は Microsoft Visual Studio2022 [4]、3D 形状を表示するための API（Application Programming Interface）として OpenGL を用いた。Gmsh から書き出した stl ファイルを読み込み、形状の確認や計算に必要な条件の設定を行うことができる。2 つのウインドウからなり、数値の設定を行うのが Control ウインドウ（図 3.2.1）、形状の確認をするのが Viewer ウインドウ（図 3.2.2）である。

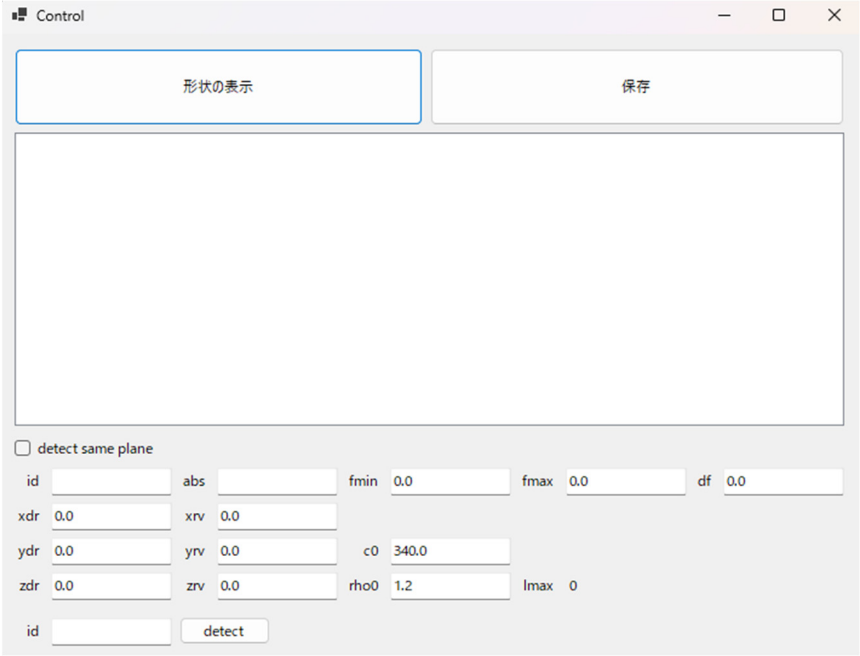

図 3.2.1 Control ウインドウ

プログラムに読み込むファイルを room.stl に設定した後、Control ウインドウの「形状の表示」のボタンを押すと、Viewer ウインドウが展開される。

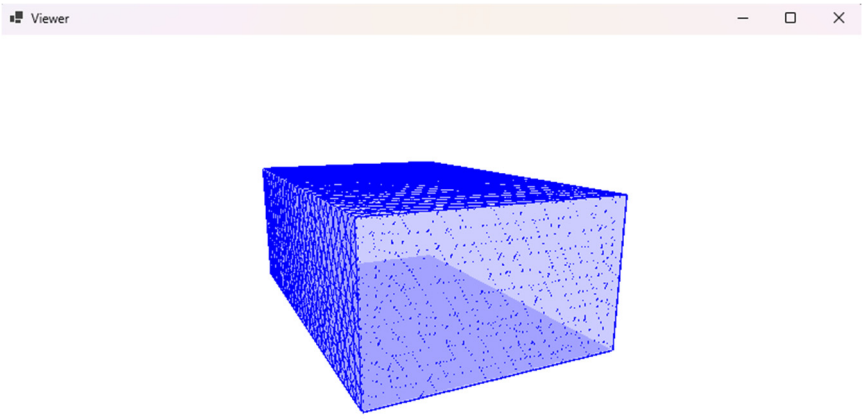

図 3.2.2 Viewer ウインドウ

この時、Control ウインドウにはそれぞれの面番号と、その面（三角形）を構成する 3 点の座標（x, y, z）が表示される（図 3.2.3）。Control ウインドウのリストから任意の面を選択することができ、Viewer ウインドウでは選択した面は赤く表示される（図 3.2.4）。

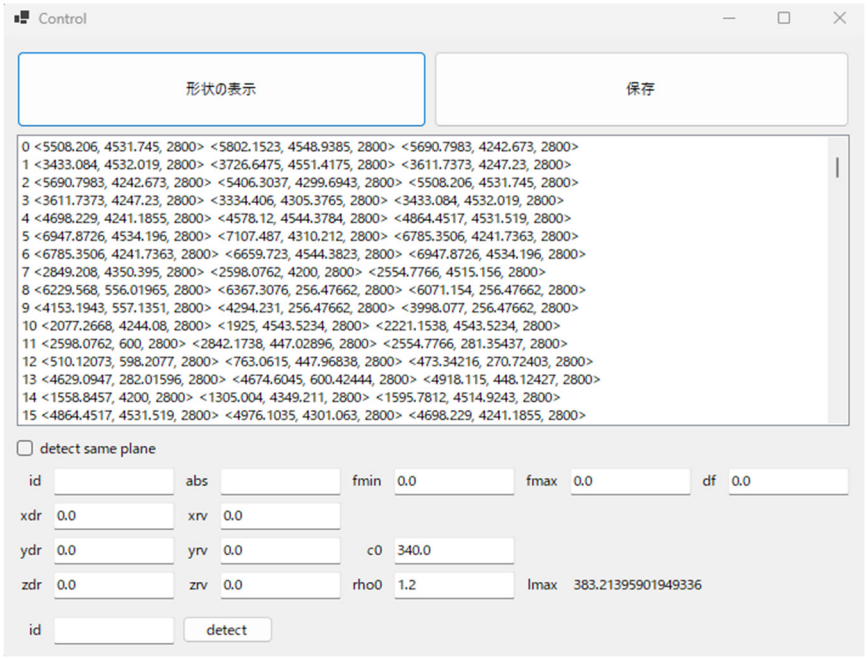

図 3.2.3 読み込み後の Control ウインドウ

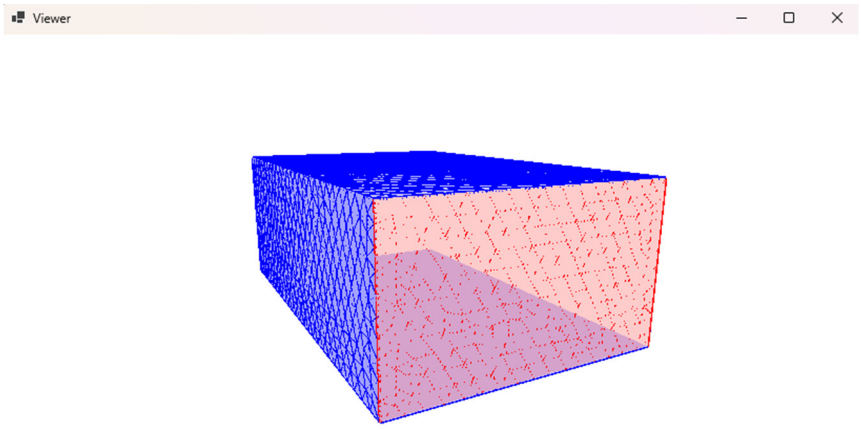

図 3.2.4 面選択した際の Viewer ウインドウ

Control ウインドウにおける入力（テキストボックス）について図 3.2.5 に示す。id、lmax は stl ファイルを読み込んだ際に自動で決定される。その他は自身で入力する必要がある。吸音率（abs）に関しては面を選択してから数値を入力することで設定することができる。面ごとに個別の吸音率を設定することも可能である。

id	要素番号
abs	吸音率
fmin	解析周波数の最小値[Hz]
fmax	解析周波数の最大値[Hz]
df	解析周波数の離散化幅[Hz]
xdr	音源位置の *x* 座標
ydr	音源位置の *y* 座標
zdr	音源位置の *z* 座標
xrv	受音点位置の *x* 座標
yrv	受音点位置の *y* 座標
zrv	受音点位置の *z* 座標
c0	空気の音速[m/s]
rho0	空気の密度[kg/m^3]
lmax	要素の最大長さ[mm]

図 3.2.5 Control ウインドウのテキストボックス

Control ウインドウの detect same plane のチェックボックスは、要素が同一平面上にあるかどうかを検知する機能である。チェックを入れて形状の表示のボタンを押すと、同一平面上に存在する全ての要素に同じ id が与えられる。最後に保存のボタンを押すと、入力した情報がバイナリファイルとして書き出され、それを計算エンジンで読み込むことで結果を得ることができる。

第 4 章　研究条件

4.1　　対象空間

解析を行う対象空間を図 4.1.1 に示す。(x, y, z)＝(7700, 4800, 2800) mm の直方体音場であり、音源位置は(x, y, z)＝(50, 50, 50) mm、受音点位置は(x, y, z)＝(7650, 4750, 2750) mm とする。

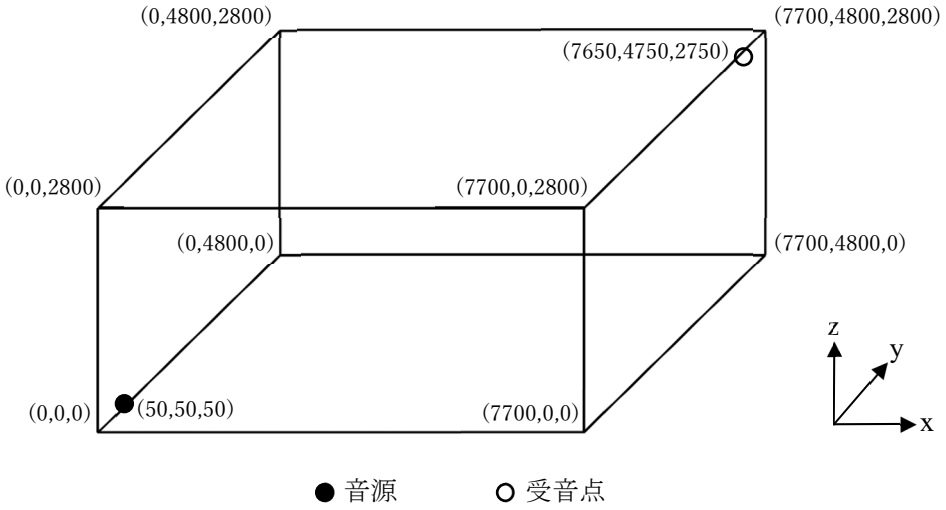

図 4.1.1 解析対象空間

音源と受音点を隅部に配置しているのは、これらの点が音圧のモードの節になりにくいからである。モードの節に音源を置くと、そのモードの固有振動が励振されない。一方、モードの節に受音点を置くと、音圧が常に 0 となり、観測されない。したがって、音場の全てのモードを観測するために、音源と受音点を隅部に配置している。

4.2　計算条件

計算条件について図 4.1.2 に示す。これらをそれぞれ Control ウインドウで入力し、計算を行う。

吸音率 (abs)	0.01
解析周波数の最小値［Hz］(fmin)	10
解析周波数の最大値［Hz］(fmax)	100
解析周波数の離散化幅［Hz］(df)	0.5
音源位置の座標 (xdr, ydr, zdr)	(50, 50, 50)
受音点位置の座標 (xrv, yrv, zrv)	(7650, 4750, 2750)
空気の音速［m/s］(c0)	340
空気の密度［$\mathrm{kg/m^3}$］(rho0)	1.2

図 4.2.2 入力する計算条件

また、この場合のメッシュサイズについては、解析周波数の最大値が 100 Hz であることから、

$$\lambda = \frac{c}{f} = \frac{340}{100} = 3.4\ [\mathrm{m}]$$

であり、これに 1/8 をかけると 425 mm となる。したがって、lmax が 425 mm 以下となるようにメッシュサイズを調節すればよい。

第 5 章　研究結果

5.1　　他手法との比較による精度の検証

書き出したバイナリファイルを計算エンジンに読み込み、計算を行った。その結果を他の解析手法による結果と比較し、開発したプログラムの精度の検証を行った。同条件下において、FDTD 法を用いて解析を行ったものと比較した結果を図 5.1.1 に示す。

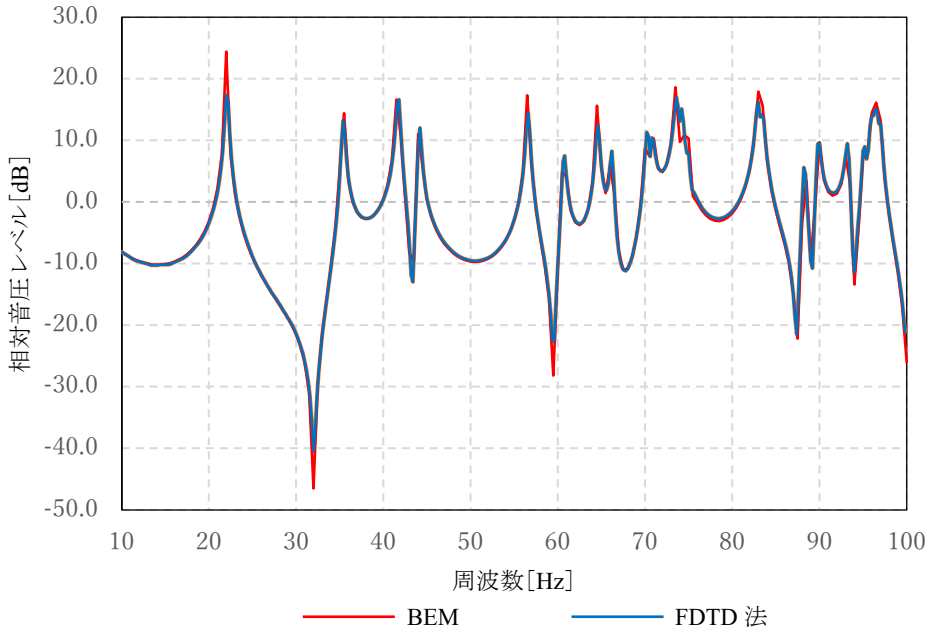

図 5.1.1　BEM　FDTD 法　比較結果

ピークになっている部分で多少の誤差が見られるが、おおよそ一致していることがグラフより読み取れる。このことから、現段階で境界要素法による数値解析プログラムとしての機能及び精度は十分に保つことができており、解析に使用できる状態にあると言える。

5.2　積分方法の違いによる影響の検討

ここでは、積分の方法を変えて計算を行うことにより、結果にどれほど影響するかを検証する。

まず、Control ウインドウの detect same plane のチェックボックスにチェックを入れた場合と入れない場合の結果の違いを比較する。ここではチェックを入れた場合を detect、入れない場合を nodetect と呼ぶことにする。積分方法の違いについては、同一平面上の要素間積分について、detect の場合は式(2.20)、nodetect の場合は式(2.19)を Gauss-Legendre 積分で計算した。比較した結果を図 5.2.1 に示す。

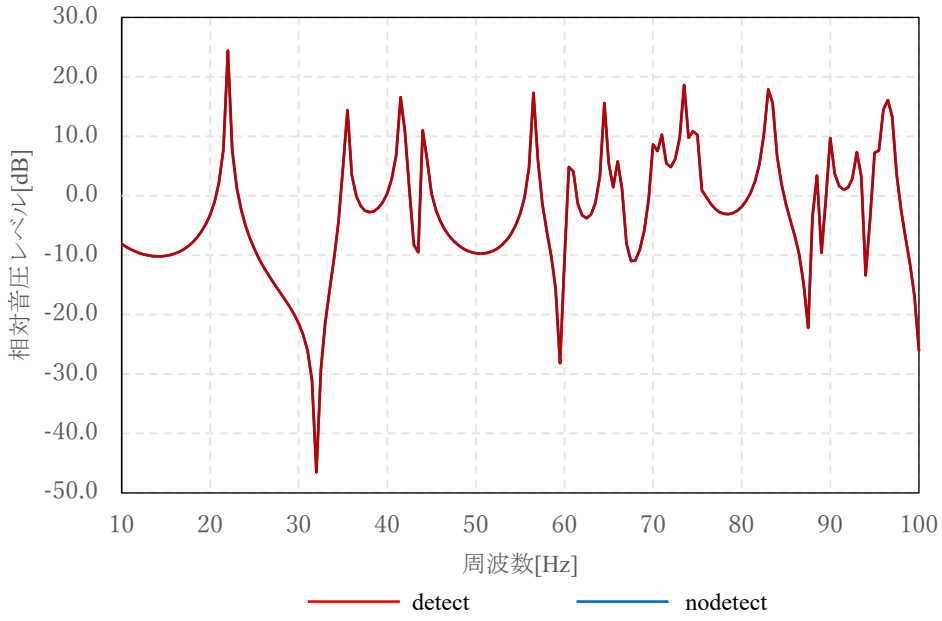

図 5.2.1　detect　nodetect　比較結果

グラフから見て分かるように、完全に一致しており、要素が同一平面上にある場合とない場合の積分方法の違いは計算の結果に影響しないことが分かった。

次に、同一要素の場合に、式(2.21)の計算を行わずに、式(2.19)を Gauss-Legendre 積分で計算した場合の結果への影響について検証する。計算結果を図 5.2.2 に示す。

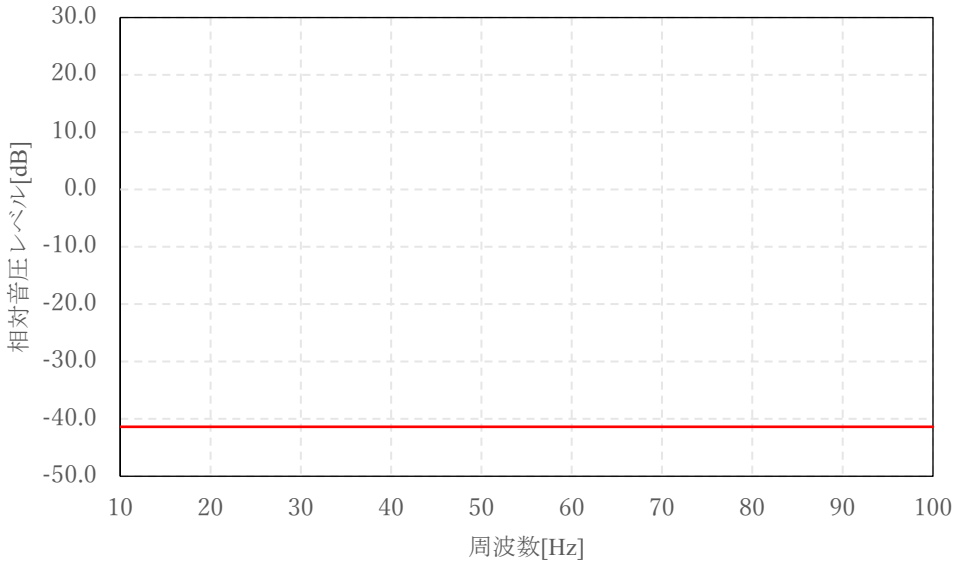

図 5.2.2 同一要素の Gauss-Legendre 積分による計算結果

このような結果となり、Gauss-Legendre 積分では同一要素間積分について適切な結果が得られなかった。したがって、同一要素の場合には、式(2.21)の計算を行う必要があることが分かった。

第 6 章　結論

本研究では境界要素法を用いた数値解析プログラムの UI 開発および精度の検証を通して、実用可能な解析ツールの構築を目指した。開発した UI は基本的な音響解析に対応可能な機能を備え、メッシュ作成と計算の間の役割を果たしたことで数値解析の一連の流れを実現することができた。また、ユーザーが直感的に操作できる環境を提供したことで、解析の効率を高めることができた。精度の検証においても、既存の手法による解析結果や異なる計算方法による結果との比較を行うことで、十分な解析精度を確保していることを示した。今後の課題としては、UI の利便性、計算精度のさらなる向上、計算速度の最適化を実現することで、実務レベルでの運用の可能性を高めることが挙げられる。

参考文献

[1] Gmsh, https://gmsh.info/

[2] Shade3D, https://shade3d.jp/

[3] 田中俊光, 藤川猛, 阿部亨, “宇津野秀夫,境界要素法による二次的音場の解析（第 2 報, 伝達マトリックス形解放の提案と消火器モデルへの適用）”, 日本機械学会論文集（C 編）50 巻 460 号 p.2356-2363,1984

[4] Microsoft Visual Studio, https://visualstudio.microsoft.com/ja/

付 録A　プログラムのソースコード

ここでは開発したインターフェイスプログラムのソースコードを紹介する。

A.1　Control.cs

```
using OpenCvSharp;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SurfaceAnalyzer;

namespace _3DView
{
    public partial class Control : Form
    {
        PolygonModel polygon;
        bool[] listnum;
        List<Surf> surflist;
        bool changeflag = false;
        double err = 1e-10f;
        double allmax = 0.0f;

        public Control()
        {
            InitializeComponent();
        }

        Viewer viewer;
        private void button1_Click_1(object sender, EventArgs e)
        {
            viewer = new Viewer();
            viewer.Show();
        }
        private void button2_Click_1(object sender, EventArgs e)
        {
            viewer = new Viewer();
            viewer.Show();

            polygon = SurfaceAnalyzer.LoadData.LoadSTL(@"room.stl", true);
            surflist = new List<Surf>();
            int gpcount = 0;
            for (int i = 0; i < polygon.Faces.Count; i++)
            {
                listBox1.Items.Add(i.ToString() + " "
                    + polygon.Faces[i].Vertices[0].P.ToString() + " "
                    + polygon.Faces[i].Vertices[1].P.ToString() + " "
                    + polygon.Faces[i].Vertices[2].P.ToString());

                Point3D pt0 = new Point3D(polygon.Faces[i].Vertices[0].P.X,
                    polygon.Faces[i].Vertices[0].P.Y,
                    polygon.Faces[i].Vertices[0].P.Z);
                Point3D pt1 = new Point3D(polygon.Faces[i].Vertices[1].P.X,
                    polygon.Faces[i].Vertices[1].P.Y,
                    polygon.Faces[i].Vertices[1].P.Z);
                Point3D pt2 = new Point3D(polygon.Faces[i].Vertices[2].P.X,
                    polygon.Faces[i].Vertices[2].P.Y,
                    polygon.Faces[i].Vertices[2].P.Z);
                Surf surf = new Surf(pt0, pt1, pt2, 0);
                surflist.Add(surf);

                if(allmax < surf.lmax)
                {
                    allmax = surf.lmax;
                }

                if (checkBox1.Checked)
                {
                    bool sameplaneflag = false;
                    for (int j = 0; j < surflist.Count - 1; j++)
                    {
```

```
                        if ((Math.Abs(surflist[surflist.Count - 1].Norm.Unit.x -
surflist[j].Norm.Unit.x) < err) &&
                            (Math.Abs(surflist[surflist.Count - 1].Norm.Unit.y -
surflist[j].Norm.Unit.y) < err) &&
                            (Math.Abs(surflist[surflist.Count - 1].Norm.Unit.z -
surflist[j].Norm.Unit.z) < err) &&
                            (Math.Abs(surflist[surflist.Count - 1].Planed - surflist[j].Planed)
< err))
                        {
                            surflist[surflist.Count - 1].group = surflist[j].group;
                            sameplaneflag = true;
                            break;
                        }
                    }
                    if (!sameplaneflag)
                    {
                        gpcount++;
                        surflist[surflist.Count - 1].group = gpcount;
                    }
                }
                else
                {
                    gpcount++;
                    surflist[surflist.Count - 1].group = gpcount;
                }
            }

            listnum = new bool[polygon.Faces.Count];
            for (int i = 0; i < polygon.Faces.Count; i++)
            {
                listnum[i] = false;
            }
            foreach (int i in listBox1.SelectedIndices)
            {
                listnum[i] = true;
            }
            viewer.Render(polygon, listnum);

            allmaxLabel.Text = allmax.ToString();
        }

        private void button3_Click_1(object sender, EventArgs e)
        {
            using (BinaryWriter writer = new BinaryWriter(File.OpenWrite("BinaryFile2.bin")))
            {
                // 書き込み
                writer.Write(double.Parse(fminTextBox.Text));
                writer.Write(double.Parse(fmaxTextBox.Text));
                writer.Write(0.0d);
                writer.Write(double.Parse(dfTextBox.Text));
                writer.Write(16.0d);
                writer.Write(double.Parse(c0TextBox.Text));
                writer.Write(double.Parse(rho0TextBox.Text));
                writer.Write(double.Parse(xdrTextBox.Text) / 1000.0d);
                writer.Write(double.Parse(ydrTextBox.Text) / 1000.0d);
                writer.Write(double.Parse(zdrTextBox.Text) / 1000.0d);
                writer.Write(1.0d);
                writer.Write(double.Parse(xrvTextBox.Text) / 1000.0d);
                writer.Write(double.Parse(yrvTextBox.Text) / 1000.0d);
                writer.Write(double.Parse(zrvTextBox.Text) / 1000.0d);
                writer.Write((double)polygon.Faces.Count);
                for (int i = 0; i < polygon.Faces.Count; i++)
                {
                    writer.Write((double)polygon.Faces[i].Vertices[0].P.X / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[0].P.Y / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[0].P.Z / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[1].P.X / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[1].P.Y / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[1].P.Z / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[2].P.X / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[2].P.Y / 1000.0d);
                    writer.Write((double)polygon.Faces[i].Vertices[2].P.Z / 1000.0d);
                    writer.Write((double)surflist[i].Norm.Unit.x / 1000.0d);
                    writer.Write((double)surflist[i].Norm.Unit.y / 1000.0d);
                    writer.Write((double)surflist[i].Norm.Unit.z / 1000.0d);
                    writer.Write((double)((1.0d - Math.Sqrt(1.0d - surflist[i].absc)) / (1.0d +
Math.Sqrt(1.0d - surflist[i].absc))));
                    writer.Write(0.0d);
                    writer.Write((double)surflist[i].group);
                }
            }
        }
```

```
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            for (int i = 0; i < polygon.Faces.Count; i++)
            {
                listnum[i] = false;
            }
            changeflag = true;

            foreach (int i in listBox1.SelectedIndices)
            {
                listnum[i] = true;
                absTextBox.Text = surflist[i].absc.ToString();
                idTextBox.Text = surflist[i].group.ToString();
            }
            changeflag = false;

            viewer.Render(polygon, listnum);
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            if (!changeflag)
            {
                double tmp = 0.0f;
                if (double.TryParse(absTextBox.Text, out tmp))
                {
                    foreach (int i in listBox1.SelectedIndices)
                    {
                        surflist[i].absc = tmp;
                    }
                }
            }

        }
        private void iddetectButton_Click(object sender, EventArgs e)
        {
            int iddetect = 0;
            if (int.TryParse(iddetectTextBox.Text, out iddetect))
            {
                for(int i = 0;i < listBox1.Items.Count;i++)
                {
                    listBox1.SetSelected(i, false);
                    if (surflist[i].group == iddetect)
                    {
                        listBox1.SetSelected(i, true);
                    }
                }
            }
        }
    }
}
```

A.2 Viewer.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Reflection.Emit;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using OpenTK;
using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;
//using OpenTK.Mathematics;
using SurfaceAnalyzer;


namespace _3DView
{
    public partial class Viewer : Form
    {
        #region Camera__Field

        bool isCameraRotating;      //カメラが回転状態かどうか
        Vector2 current, previous;  //現在の点、前の点
        float zoom = 0.0f;                  //拡大度
        double rotateX = 1, rotateY = 0, rotateZ = 0;//カメラの回転による移動
        float theta = 0;
        float phi = 0;
        Vector3 eye = new Vector3(1.3f, -2.25f, 1.5f);

        float _lensPosX = 2000.0f;
        float _lensPosY = 2000.0f;
        float _lensPosZ = 2000.0f;
        Point3D pt0 = new Point3D(2000.0f, 2000.0f, 2000.0f); // カメラ位置
        float _lensFocusX = 0.0f;
        float _lensFocusY = 0.0f;
        float _lensFocusZ = 0.0f;
        Point3D pt1 = new Point3D(0.0f, 0.0f, 0.0f); // 注視点
        Vector3D vec1 = new Vector3D();
        Vector3D vec2 = new Vector3D(0.0f, 1.0f, 0.0f);
        Vector3D vecX = new Vector3D();
        Vector3D vecY = new Vector3D();
        bool[] _keys = new bool[256];
        System.Drawing.Point _oldMousePoint = System.Drawing.Point.Empty;

        #endregion

        public Viewer()
        {
            InitializeComponent();

            AddglControl();
        }

        // glControl の追加
        GLControl glControl;
        private void AddglControl()
        {
            SuspendLayout();

            int width = this.Width;
            int height = this.Height;

            //GLControl の初期化
            glControl = new GLControl();

            glControl.Name = "SHAPE";
            glControl.Size = new System.Drawing.Size(width, height - 80);
            glControl.Location = new System.Drawing.Point(0, 0);
            glControl.SendToBack();

            //イベントハンドラ
            glControl.Load += new EventHandler(glControl_Load);
            glControl.Resize += new EventHandler(glControl_Resize);
```

```
            //glControl.MouseDown += new
System.Windows.Forms.MouseEventHandler(this._3DView_MouseDown);
            glControl.MouseMove += new
System.Windows.Forms.MouseEventHandler(this._3DView_MouseMove);
            //glControl.MouseUp += new
System.Windows.Forms.MouseEventHandler(this._3DView_MouseUp);
            glControl.MouseWheel += new
System.Windows.Forms.MouseEventHandler(this._3DView_MouseWheel);
            //glControl.KeyDown += new
System.Windows.Forms.KeyEventHandler(this._3DView_KeyDown);
            glControl.PreviewKeyDown += new
System.Windows.Forms.PreviewKeyDownEventHandler(this._3DView_PreviewKeyDown);
            glControl.KeyUp += new System.Windows.Forms.KeyEventHandler(this._3DView_KeyUp);

            Controls.Add(glControl);

            ResumeLayout(false);
        }

        private void glControl_Load(object sender, EventArgs e)
        {
            GLControl s = (GLControl)sender;
            s.MakeCurrent();

            GL.ClearColor(Color4.White);
            GL.Enable(EnableCap.DepthTest);

            // アルファブレンドを有効化
            GL.Enable(EnableCap.Blend);
            GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

            Update();
        }

        private void glControl_Resize(object sender, EventArgs e)
        {
            GL.Viewport(0, 0, glControl.Size.Width, glControl.Size.Height);
            GL.MatrixMode(MatrixMode.Projection);
            //Matrix4 projection = Matrix4.CreatePerspectiveFieldOfView((float)Math.PI / 4,
            //    (float)glControl.Size.Width / (float)glControl.Size.Height, 1.0f, 256.0f);
            Matrix4 projection = Matrix4.CreatePerspectiveFieldOfView((float)Math.PI / 4,
                (float)glControl.Size.Width / (float)glControl.Size.Height, 0.1f, 100000.0f);
            GL.LoadMatrix(ref projection);

            Update();
        }

        private void _3DView_MouseMove(object sender, MouseEventArgs e)
        {
            // スペース(keys[32])を押しながらだとカメラ・注視点ともに移動
            if (this._keys[32])
            {
                if (e.Button == MouseButtons.Left)
                {
                    // カメラ・注視点移動
                    vecX = (vec1 ^ vec2);
                    vecY = (vecX ^ vec1);

                    this._lensPosX += (float)(vecX.Unit.x * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.x * (e.Location.Y - this._oldMousePoint.Y));
                    this._lensFocusX += (float)(vecX.Unit.x * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.x * (e.Location.Y - this._oldMousePoint.Y));
                    this._lensPosY += (float)(vecX.Unit.y * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.y * (e.Location.Y - this._oldMousePoint.Y));
                    this._lensFocusY += (float)(vecX.Unit.y * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.y * (e.Location.Y - this._oldMousePoint.Y));
                    this._lensPosZ += (float)(vecX.Unit.z * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.z * (e.Location.Y - this._oldMousePoint.Y));
                    this._lensFocusZ += (float)(vecX.Unit.z * (e.Location.X -
this._oldMousePoint.X)
                        - vecY.Unit.z * (e.Location.Y - this._oldMousePoint.Y));

                    pt0.x = (double)this._lensPosX;
                    pt0.y = (double)this._lensPosY;
                    pt0.z = (double)this._lensPosZ;
                    pt1.x = (double)this._lensFocusX;
                    pt1.y = (double)this._lensFocusY;
```

```
                pt1.z = (double)this._lensFocusZ;

                vec1 = pt1 - pt0;
            }
        }
        else
        {
            if (e.Button == MouseButtons.Left)
            {
                // カメラ移動
                double L1 = vec1.Length;
                double theta = -(double)(e.Location.X - this._oldMousePoint.X) / 300.0d;
                double phi = (double)(e.Location.Y - this._oldMousePoint.Y) / 300.0d;

                vecX = (vec1 ^ vec2);
                vecY = (vecX ^ vec1);

                Vector3D XX = new Vector3D();
                Vector3D XXT = new Vector3D();
                Vector3D YY = new Vector3D();
                Vector3D YYT = new Vector3D();

                XXT = (L1 * Math.Tan(theta) * vecX.Unit - vec1);
                XX = L1 * (XXT.Unit) + vec1;
                YYT = (L1 * Math.Tan(phi) * vecY.Unit - vec1);
                YY = L1 * (YYT.Unit) + vec1;

                this._lensPosX += (float)(XX.x + YY.x);
                this._lensPosY += (float)(XX.y + YY.y);
                this._lensPosZ += (float)(XX.z + YY.z);

                pt0.x = (double)this._lensPosX;
                pt0.y = (double)this._lensPosY;
                pt0.z = (double)this._lensPosZ;

                vec1 = pt1 - pt0;
            }
        }
        // マウスの位置を記憶
        this._oldMousePoint = e.Location;

        Update();
    }

    private void _3DView_MouseWheel(object sender, MouseEventArgs e)
    {
        if (e.Delta != 0)
        {
            // xyz 座標を更新
            this._lensPosX += (float)(vec1.x * e.Delta / 10000.0f);
            this._lensPosY += (float)(vec1.y * e.Delta / 10000.0f);
            this._lensPosZ += (float)(vec1.z * e.Delta / 10000.0f);

            pt0.x = (double)this._lensPosX;
            pt0.y = (double)this._lensPosY;
            pt0.z = (double)this._lensPosZ;

            vec1 = pt1 - pt0;

            Update();
        }
    }

    private void _3DView_PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
    {
        // 押されたキーコードのフラグを立てる
        if ((int)e.KeyCode < this._keys.Length)
        {
            this._keys[(int)e.KeyCode] = true;
        }

        // 上下キーでマウスホイール操作イベントの代わり
        if (this._keys[38])
        {
            // xyz 座標を更新
            this._lensPosX += (float)(vec1.x / 100.0f);
            this._lensPosY += (float)(vec1.y / 100.0f);
            this._lensPosZ += (float)(vec1.z / 100.0f);

            pt0.x = (double)this._lensPosX;
            pt0.y = (double)this._lensPosY;
            pt0.z = (double)this._lensPosZ;
```

```
                vec1 = pt1 - pt0;
            }

            if (this._keys[40])
            {
                // xyz 座標を更新
                this._lensPosX += (float)(-vec1.x / 100.0f);
                this._lensPosY += (float)(-vec1.y / 100.0f);
                this._lensPosZ += (float)(-vec1.z / 100.0f);

                pt0.x = (double)this._lensPosX;
                pt0.y = (double)this._lensPosY;
                pt0.z = (double)this._lensPosZ;

                vec1 = pt1 - pt0;
            }

            Update();
        }

        private void _3DView_KeyUp(object sender, KeyEventArgs e)
        {
            // 放したキーコードのフラグを下ろす
            if ((int)e.KeyCode < this._keys.Length)
            {
                this._keys[(int)e.KeyCode] = false;
            }

            Update();
        }

        PolygonModel Polygon;
        bool[] listnumlocal;
        public void Update()
        {
            if (Polygon == null) return;
            Render(Polygon, listnumlocal);
        }

        public void Render(PolygonModel polygon, bool[] listnum)
        {
            Polygon = polygon;
            listnumlocal = listnum;

            // バッファのクリア
            GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

            // カメラ設定
            Vector3 lensPosition = new Vector3(this._lensPosX, this._lensPosY,
this._lensPosZ);
            Vector3 lensFocus = new Vector3(this._lensFocusX, this._lensFocusY,
this._lensFocusZ);
            Vector3 worldTop = new Vector3(0.0f, 1.0f, 0.0f);
            Matrix4 modelView = Matrix4.LookAt(lensPosition, lensFocus, worldTop);

            // 表示設定
            GL.MatrixMode(MatrixMode.Modelview);
            GL.LoadMatrix(ref modelView);

            // 3D 形状の表示
            DrawPolygons(polygon, listnum);

            // バッファの入れ替え
            glControl.SwapBuffers();
        }

        private void DrawPolygons(PolygonModel polygon, bool[] listnum)
        {
            if (polygon == null) return;

            //描画
            GL.Begin(PrimitiveType.Triangles);

            //三角形を描画
            for (int l = 0; l < polygon.Faces.Count; l++)
            {
                //! [Something went wrong]()

                var normal = polygon.Faces[l].Normal();
                //GL.Color4(Math.Abs(normal.X), Math.Abs(normal.Y), Math.Abs(normal.Z), 0.5f);
                if (listnum[l] == true)
```

```
                {
                    GL.Color4(1.0f, 0.0f, 0.0f, 0.2f);
                }
                else
                {
                    GL.Color4(0.0f, 0.0f, 1.0f, 0.2f);
                }
                GL.Normal3(N2TK(normal));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[0].P));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[1].P));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[2].P));
            }

            GL.End();

            GL.LineWidth(3.0f);
            //三角形を描画
            for (int l = 0; l < polygon.Faces.Count; l++)
            {
                //! [Something went wrong]()

                GL.Begin(PrimitiveType.LineStrip);
                var normal = polygon.Faces[l].Normal();
                //GL.Color4(Math.Abs(normal.X), Math.Abs(normal.Y), Math.Abs(normal.Z),
256.0f);
                if (listnum[l] == true)
                {
                    GL.Color4(1.0f, 0.0f, 0.0f, 1.0f);
                }
                else
                {
                    GL.Color4(0.0f, 0.0f, 1.0f, 1.0f);
                }
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[0].P));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[1].P));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[2].P));
                GL.Vertex3(N2TK(polygon.Faces[l].Vertices[0].P));
                GL.End();
            }
        }

        // Numerics.Vector3 を OpenTK.Vector3 に変換します。
        private static OpenTK.Vector3 N2TK(System.Numerics.Vector3 vec3) => new
Vector3(vec3.X, vec3.Z, vec3.Y);

        // 画像の保存
        public OpenCvSharp.Mat GetMat()
        {
            int width = glControl.Width;
            int height = glControl.Height;

            float[] floatArr = new float[width * height * 3];
            OpenCvSharp.Mat ret = new OpenCvSharp.Mat(height, width,
OpenCvSharp.MatType.CV_32FC3);

            // dataBuffer への画像の読み込み
            IntPtr dataBuffer = Marshal.AllocHGlobal(width * height * 12);
            GL.ReadBuffer(ReadBufferMode.Front);
            GL.ReadPixels(0, 0, width, height, PixelFormat.Bgr, PixelType.Float, dataBuffer);

            // img への読み込み
            Marshal.Copy(dataBuffer, floatArr, 0, floatArr.Length);

            // opencvsharp.Mat への変換
            Marshal.Copy(floatArr, 0, ret.Data, floatArr.Length);

            // 破棄
            Marshal.FreeHGlobal(dataBuffer);

            return ret;
        }
    }
}
```

A.3 Surf.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _3DView
{
    public class Surf
    {
        private Point3D p0, p1, p2;
        public double absc;
        public int group;
        public Surf()
        {
            this.p0 = new Point3D();
            this.p1 = new Point3D();
            this.p2 = new Point3D();
            this.absc = 0.0f;
            this.group = 0;
        }

        public Surf(Point3D p0, Point3D p1, Point3D p2)
        {
            this.p0 = p0;
            this.p1 = p1;
            this.p2 = p2;
            this.absc = 0.0f;
            this.group = 0;
        }
        public Surf(Point3D p0, Point3D p1, Point3D p2, double absc)
        {
            this.p0 = p0;
            this.p1 = p1;
            this.p2 = p2;
            this.absc = absc;
            this.group = 0;
        }
        public Surf(Point3D p0, Point3D p1, Point3D p2, int group)
        {
            this.p0 = p0;
            this.p1 = p1;
            this.p2 = p2;
            this.absc = 0.0f;
            this.group = group;
        }
        public Surf(Point3D p0, Point3D p1, Point3D p2, double absc, int group)
        {
            this.p0 = p0;
            this.p1 = p1;
            this.p2 = p2;
            this.absc = absc;
            this.group = group;
        }
        public Vector3D Norm
        {
            get
            {
                Vector3D result = new Vector3D();
                Vector3D vec1 = new Vector3D(p0, p1);
                Vector3D vec2 = new Vector3D(p0, p2);
                result = vec1 ^ vec2;
                return result;
            }
        }

        public double Planed
        {
            get
            {
                double a = this.Norm.Unit.x;
                double b = this.Norm.Unit.y;
                double c = this.Norm.Unit.z;
                double x = this.p0.x;
                double y = this.p0.y;
                double z = this.p0.z;
                double result = -a*x-b*y-c*z;
                return result;
```

```
            }
        }

        public double lmax
        {
            get
            {
                Vector3D vec01 = new Vector3D(p0, p1);
                Vector3D vec12 = new Vector3D(p1, p2);
                Vector3D vec20 = new Vector3D(p2, p0);
                return Math.Max(vec20.Length,Math.Max(vec01.Length, vec12.Length));
            }
        }
    }
}
```

A.4 Vector3D.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _3DView
{
    public class Vector3D
    {
        #region 定数宣言

        //private double err = 1e-10;
        private double err = 1e-13;

        private double px, py, pz;

        #endregion

        #region コンストラクタ

        public Vector3D()
        {
            px = 0.0d;
            py = 0.0d;
            pz = 0.0d;
        }

        public Vector3D(Point3D pt0, Point3D pt1)
        {
            this.px = pt1.x - pt0.x;
            this.py = pt1.y - pt0.y;
            this.pz = pt1.z - pt0.z;
        }

        public Vector3D(float px, float py, float pz)
        {
            this.px = (double)px;
            this.py = (double)py;
            this.pz = (double)pz;
        }

        public Vector3D(double px, double py, double pz)
        {
            this.px = px;
            this.py = py;
            this.pz = pz;
        }

        #endregion

        #region 演算子オーバーライド

        // 加算
        public static Vector3D operator +(Vector3D vec1, Vector3D vec2)
        {
            Vector3D result = new Vector3D();

            result.px = vec1.px + vec2.px;
            result.py = vec1.py + vec2.py;
            result.pz = vec1.pz + vec2.pz;

            return result;
        }

        // 減算
        public static Vector3D operator -(Vector3D vec1, Vector3D vec2)
        {
            Vector3D result = new Vector3D();

            result.px = vec1.px - vec2.px;
            result.py = vec1.py - vec2.py;
            result.pz = vec1.pz - vec2.pz;

            return result;
        }

        // 負
```

```
        public static Vector3D operator -(Vector3D vec1)
        {
            Vector3D result = new Vector3D();

            result.px = -vec1.px;
            result.py = -vec1.py;
            result.pz = -vec1.pz;

            return result;
        }

        // 内積
        public static double operator *(Vector3D vec1, Vector3D vec2)
        {
            double result;

            result = vec1.px * vec2.px + vec1.py * vec2.py + vec1.pz * vec2.pz;

            return result;
        }

        // 定数倍
        public static Vector3D operator *(double a, Vector3D vec1)
        {
            Vector3D result = new Vector3D();

            result.px = a * vec1.px;
            result.py = a * vec1.py;
            result.pz = a * vec1.pz;

            return result;
        }

        // 外積
        public static Vector3D operator ^(Vector3D vec1, Vector3D vec2)
        {
            Vector3D result = new Vector3D();

            result.px = vec1.py * vec2.pz - vec1.pz * vec2.py;
            result.py = vec1.pz * vec2.px - vec1.px * vec2.pz;
            result.pz = vec1.px * vec2.py - vec1.py * vec2.px;

            return result;
        }

        #endregion

        #region プロパティ

        internal double x
        {
            get
            {
                return px;
            }
            set
            {
                px = value;
            }
        }

        internal double y
        {
            get
            {
                return py;
            }
            set
            {
                py = value;
            }
        }

        internal double z
        {
            get
            {
                return pz;
            }
            set
            {
                pz = value;
```

```
            }
        }

        public double Length
        {
            get
            {
                double result = Math.Sqrt(px * px + py * py + pz * pz);
                return result;
            }
        }

        // 立体角・距離取得用の単位ベクトル
        public Vector3D Unit
        {
            get
            {
                Vector3D result = new Vector3D();

                if (Math.Abs(Length) > err)
                {
                    result.px = px / Length;
                    result.py = py / Length;
                    result.pz = pz / Length;
                }
                else
                {
                    result.px = 0.0d;
                    result.py = 0.0d;
                    result.pz = 0.0d;
                }

                return result;
            }
        }

        // 面法線方向比較用の単位ベクトル
        public Vector3D UnitPML
        {
            get
            {
                Vector3D result = new Vector3D();

                if (Math.Abs(Length) > err)
                {
                    result.px = px / Length;
                    result.py = py / Length;
                    result.pz = pz / Length;
                }
                else
                {
                    result.px = 0.0d;
                    result.py = 0.0d;
                    result.pz = 0.0d;
                }

                return result;
            }
        }

        public string DataOut
        {
            get
            {
                return "x=" + this.px.ToString() + " y=" + this.py.ToString() + " z=" +
this.pz.ToString();
            }
        }

        #endregion
    }
}
```

A.5 Point3D.cs

```
using OpenTK;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _3DView
{
    public class Point3D
    {
        #region 定数宣言

        private double err = 1e-10;

        private double px, py, pz;

        #endregion

        #region コンストラクタ

        public Point3D()
        {
            px = 0.0d;
            py = 0.0d;
            pz = 0.0d;
        }

        public Point3D(float px, float py, float pz)
        {
            this.px = (double)px;
            this.py = (double)py;
            this.pz = (double)pz;
        }

        public Point3D(double px, double py, double pz)
        {
            this.px = px;
            this.py = py;
            this.pz = pz;
        }

        #endregion

        #region 演算子オーバーライド

        // 2 点からベクトル作成
        public static Vector3D operator -(Point3D pt1, Point3D pt2)
        {
            Vector3D result = new Vector3D(pt1.px - pt2.px, pt1.py - pt2.py, pt1.pz - pt2.pz);
            return result;
        }

        // 負
        public static Vector3D operator -(Point3D pt1)
        {
            Vector3D result = new Vector3D(-pt1.px, -pt1.py, -pt1.pz);
            return result;
        }

        // ベクトル足し算
        public static Point3D operator +(Point3D pt1, Vector3D vec)
        {
            Point3D result = new Point3D(pt1.x + vec.x, pt1.y + vec.y, pt1.z + vec.z);
            return result;
        }

        #endregion

        #region 検索

        public bool FindSamePoint(Point3D pt1)
        {
            if (Math.Abs(this.px - pt1.px) < err)
            {
                if (Math.Abs(this.py - pt1.py) < err)
                {
                    if (Math.Abs(this.pz - pt1.pz) < err)
```

```
                    {
                        return true;
                    }
                }
            }
            return false;
        }

        #endregion

        #region プロパティ

        internal double x
        {
            get
            {
                return px;
            }
            set
            {
                px = value;
            }
        }

        internal double y
        {
            get
            {
                return py;
            }
            set
            {
                py = value;
            }
        }

        internal double z
        {
            get
            {
                return pz;
            }
            set
            {
                pz = value;
            }
        }

        public string DataOut
        {
            get
            {
                return string.Format("x={0:E}, y={1:E}, z={2:E}", this.px, this.py, this.pz);
            }
        }

        #endregion
    }
}
```