

新EasyISTRの紹介

(EasyISTR ver 3.52.250701)

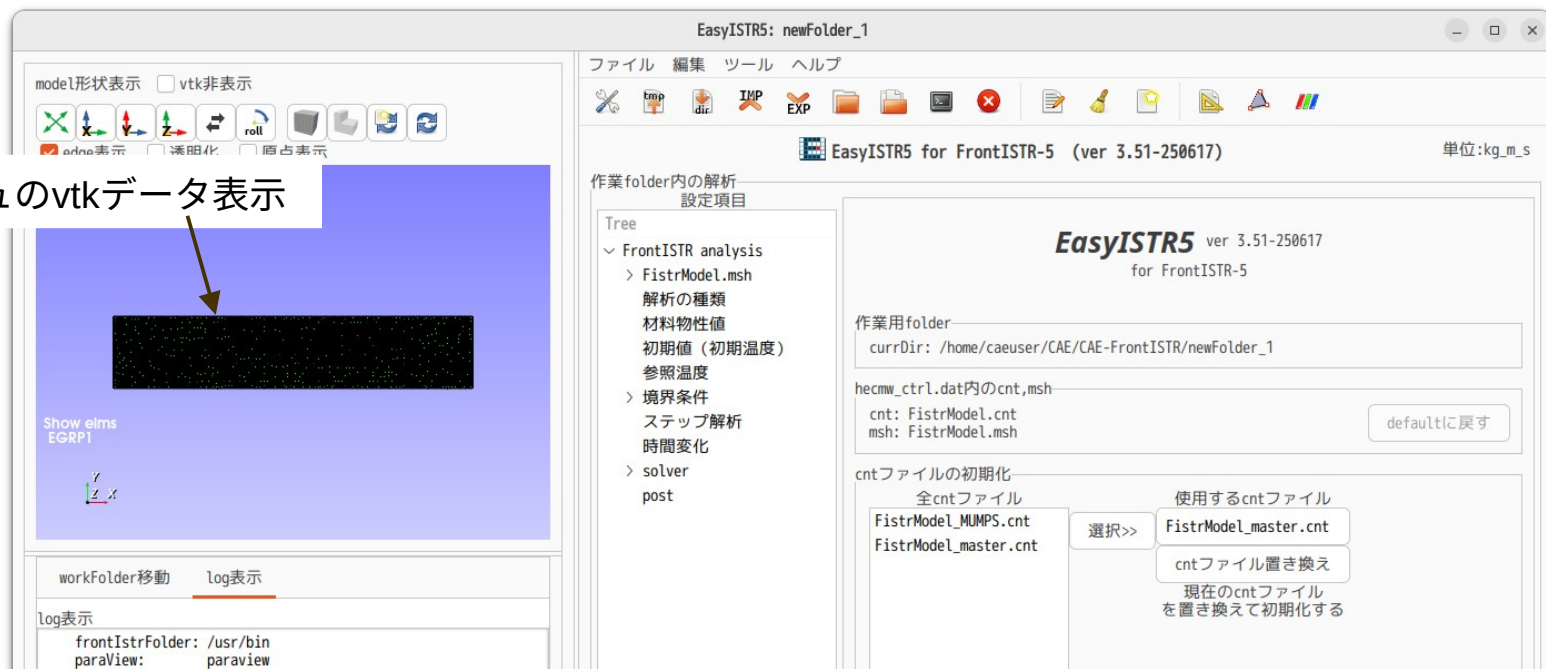
1. vtk関連処理の高速化
 - 1-1. vtuファイルのbinary形式を追加
 - 1-1-1. binary形式の内容
 - 1-1-2. Base64とは
 - 1-1-3. Base64によるデータセット作成方法
 - 1-1-4. binary、asciiへの切り替え方法
 - 1-1-5. binary形式による効果
 - 1-2. vtuファイル作成方法修正
2. mesh作成の為のmeshSize設定方法修正
 - defaultのmeshSizeの設定でメッシュ作成できない事がある為

1. vtk関連処理の高速化

1-1. vtuファイルのbinary形式を追加

EasyISTRでは、操作性向上の為、メッシュ形状をviewer上に表示させている。
メッシュの規模が増えるに従って、vtkデータの読み込み、表示までに時間が掛かってしまう。
この為、起動時には、制限時間の5秒までに表示できなかった場合、
メッシュを非表示でEasyISTRを起動するようにしている。
vtkデータの読み込み表示時間を早めるために、vtkデータのbinary化を検討する。

メッシュのvtkデータ表示



1-1-1. binary形式の内容

vtkのxmlフォーマット（vtuファイル）のbinary化は、
VTK document「VTK File Formats」に以下の記述がある。

format="ascii" — The data are listed in ASCII directly inside the DataArray element. Whitespace is used for separation.

format="binary" — The data are encoded in **base64** and listed contiguously inside the DataArray element. Data may also be compressed before encoding in base64. The byte-order of the data matches that specified by the byte_order attribute of the VTKFile element.

format="appended" — The data are stored in the appended data section. Since many DataArray elements may store their data in this section, the offset attribute is used to specify where each DataArray's data begins. This format is the default used by VTK's writers.

format="binary"：binary化は、「Base64」でencodeされたdataを使う。

1-1-2. Base64とは

「Base64」とは、生のbinaryデータを6bit毎に分け、 $2^6=64$ 種類の文字を割り当てて変換する。64種類の文字は、「A-Z, a-z, 0-9, +, /」の64種類と空文字の「=」の文字を使って表現する。生のbinaryデータが6bitで割り切れない場合、「=」を使って調整する。

code変換表 (wikipediaより抜粋)

ビット列	Base64文字	ビット列	Base64文字	ビット列	Base64文字	ビット列	Base64文字
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/

vtkのDataArrayにセットするデータセットは、
「8バイトのheader（binaryデータのbyte数）とbinaryデータ」
をBase64で変換する。

以下が変換例になる。

ascii形式

```
<DataArray type="Int32" Name="offsets" format="ascii">  
  4 8 12 16 20 24  
</DataArray>
```

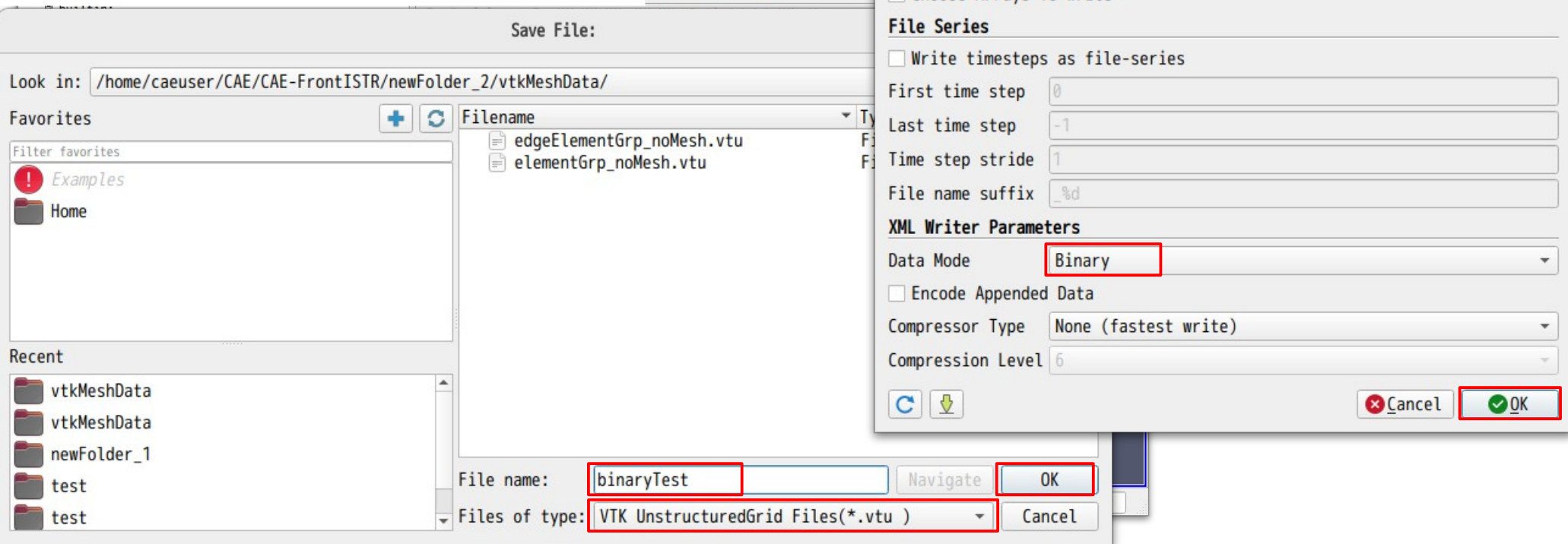
binary形式（Base64でencode）

```
<DataArray type="Int32" Name="offsets" format="binary">  
  GAAAAAAAAAEAAAAQAAAAwAAAAQAAAAFAAAABgAAAA=  
</DataArray>
```

↑
headerを含むbinaryを変換した結果

paraviewは、表示している内容をVTUのbinary形式（base64によるencode）で出力する事ができる。

メニュー「File」>「Save Data...」を選択



binary形式が出力されるので、参考になる。

ascii形式とbinary形式の比較

ascii形式

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="1.0" byte_order="LittleEndian"
header_type="UInt64">
<UnstructuredGrid>
<Piece NumberOfPoints="8" NumberOfCells="6">
<Points>
<DataArray type="Float32" NumberOfComponents="3" format="ascii" >
0.000000e+00 0.000000e+00 1.000000e+00
1.000000e+00 0.000000e+00 1.000000e+00
2.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 1.000000e+00 1.000000e+00
1.000000e+00 1.000000e+00 1.000000e+00
2.000000e+00 1.000000e+00 0.000000e+00
0.000000e+00 1.000000e+00 0.000000e+00
</DataArray>
</Points>
<Cells>
<DataArray type="Int32" Name="connectivity" format="ascii" >
0 3 2 1 4 5 6 7
0 1 5 4 1 2 6 5
2 3 7 6 3 0 4 7
</DataArray>
<DataArray type="Int32" Name="offsets" format="ascii">
4 8 12 16 20 24
</DataArray>
<DataArray type="UInt8" Name="types" format="ascii" >
9 9 9 9 9 9
</DataArray>
</Cells>
</Piece>
</UnstructuredGrid>
</VTKFile>
```

binary形式(Base64でencode)

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="1.0" byte_order="LittleEndian"
header_type="UInt64">
<UnstructuredGrid>
<Piece NumberOfPoints="8" NumberOfCells="6">
<Points>
<DataArray type="Float32" NumberOfComponents="3" format="binary" >
YAAAAAAAAAAAAAAAAAAgD8AAIA/
AAAAAAAAAgD8AAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAPwAAgD8AAIA/
AACAPwAAgD8AAABAAACAPwAAAAAAAAAACAPwAAAA=
</DataArray>
</Points>
<Cells>
<DataArray type="Int32" Name="connectivity" format="binary" >
YAAAAAAAAAAAAAAAAAwAAAAIAAABAAAAABAAAAUAAAGAAAAABwAAAAAAAAABAAAAAQAA
AABAAAAAgAAAAYAAAAFAAAAAGAAAAMAAAAHAAABgAAAAMAAAAAAAAAABAAAAcAAAA=
</DataArray>
<DataArray type="Int32" Name="offsets" format="binary">
GAAAAAAAAAAEAAAACAAAAAwAAAAQAAAAFAAAABgAAAA=
</DataArray>
<DataArray type="UInt8" Name="types" format="binary" >
BgAAAAAAAAAJCQkJCQk=
</DataArray>
</Cells>
</Piece>
</UnstructuredGrid>
</VTKFile>
```

太字がbinary特有の項目

vtuのbinary形式は、全て文字データの為、テキストエディタで編集・保存が可能。

1-1-3. Base64によるデータセット作成方法

pythonには、標準でbase64モジュールがインストールされているので、これを使う。
base64、numpyモジュールを使って、encode（binary変換）、decode（復元）ができる。

binary形式に変換（encode）

```
packVals = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0] #変換するlist
nBytes   = [len(packVals) * 4] #byte数
buffer   = np.array(nBytes, dtype="uint64").tobytes() #byte数をセット
buffer += np.array(packVals, dtype="float32").tobytes() #変換するlistをセット
binData  = base64.b64encode(buffer) #bufferを変換
```

binary形式を復元（decode）

```
buffer = base64.b64decode(binData) #binDataをdecode
nBytes = np.frombuffer(buffer[:8]), dtype="uint64" #byte数を取得
packVals = np.frombuffer(buffer[8:]), dtype="float32" #float値を取得
```

以下のwebやparaViewが出力したbinary形式を参考にしながら、作成した。

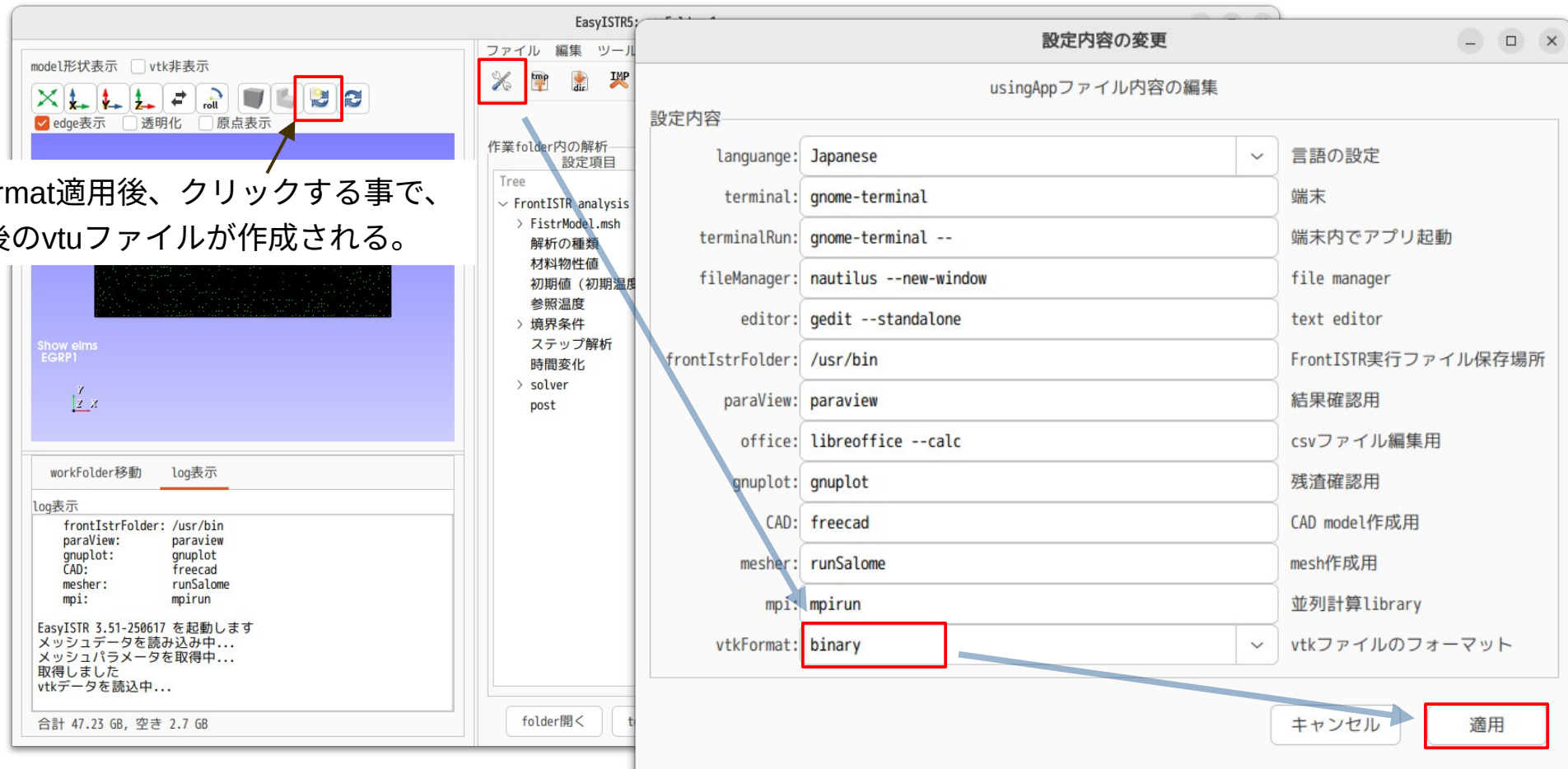
<https://discourse.vtk.org/t/binary-dataarray-in-xml-using-python-numpy-what-are-the-leading-int32-values/8013/15>

<https://qiita.com/ak-yoshi/items/e955be686ac69a01f215>

1-1-4. binary、asciiへの切り替え方法

EasyISTR上で、vtuファイルの形式をascii、binaryで選択できる。（defaultは、binary形式）

vtkFormat適用後、クリックする事で、適用後のvtuファイルが作成される。



1-1-5. binary形式による効果

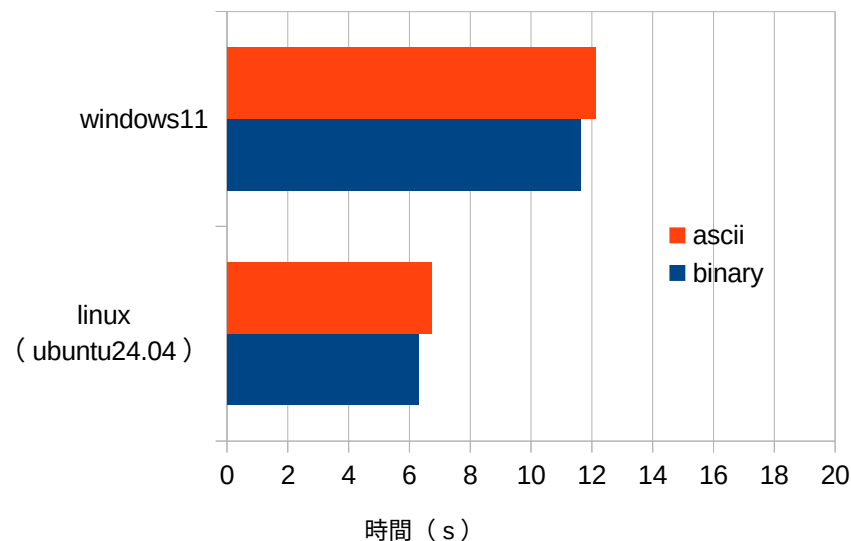
binary形式であれば、ascii形式に比べて、vtuファイル読み込み、モデル表示の時間が早まる事を期待して、binary形式が扱える様に修正した。

どのくらいの効果があるか確認した結果が以下になる。

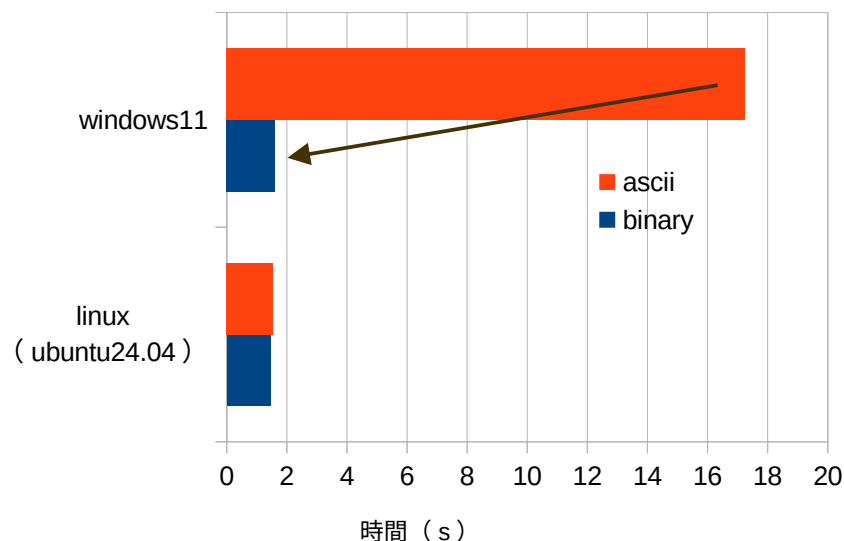
モデル：四面体2次要素、要素数:32.7万要素

PC：cpu:Ryzen7、memory:32GB (VMwear 16GB)

vtu ファイル作成時間



vtu ファイル読込 - 表示までの時間



windowsにおいて、vtuファイル読込-表示までの時間がbinaryの方が格段に早くなる。

この時間は、EasyISTRの起動時間に影響を与えるので、windowsの起動時間が早くなる事を意味する。

1-2. vtuファイル作成方法修正

vtuファイルを作成する上でネックになっている処理は、

- 1) Fistr形式のメッシュを読み込み、str文字をintやfloatに変換する処理
- 2) vtuファイル作成の為、solidの表面要素を取得する処理

になる。

要素数を増やして、処理時間を確認した結果が下表になる。

四面体2次要素、要素数：123万要素の大規模メッシュで確認

	linux(ubuntu24.04)	windows11
vtuファイル(ascii)作成-表示までの時間（s）	50	85
1) メッシュ読み込み、str文字を数値に変換（s）	9.40	11.57
2) solidの表面要素を取得（s）	8.74	13.12

前記した処理時間は、全体の約1／3に相当する。

<Fistr形式メッシュ読み込み、str文字を数値に変換 対応>

時間が掛かっているのが、str文字を数値に変換する処理になる。

vtuファイルを作成する時は、メッシュ作成、メッシュ変換（meshio含む）のときになる。

これら処理中には、メッシュデータを数値で保持している為、この数値データを

pickleモジュールを使って、binaryでファイルに保存しておき、

vtuファイル作成時は、保存したbinaryファイルを読み込み、vtuを作成。

binaryファイルを読み込む事で、str文字を数値に変換する事はなくなる。

<solidの表面要素の取得 対応>

メッシュ作成とunv2fistr変換については、処理途中で、solidの外表面要素を取得して、未定義のsurfaceGroupがあれば「otherS」でsurfaceGroupを定義している。

この為、vtuファイル作成時、このsurfaceGroupを使えば、solidの表面要素が取得できる。

現状は、全ての要素の各faceについて、

faceに隣接する要素が存在するかしないかを調べている。（これに時間が掛かる）

対応後の効果確認

対応後、同様に処理時間を確認した結果が以下になる。

(差が確認できるように、100万要素モデルで確認)

四面体2次要素、要素数:123万要素	対応前(ascii)		対応後(ascii)	
	linux	windows	linux	windows
vtuファイル(ascii)作成-表示までの時間 (s)	50	85	28	65
1) メッシュ読み込み, str文字を数値に変換 (s)	9.40	11.57	2.02	2.28
2) solidの表面要素を取得 (s)	8.74	13.12	1.42	1.67

四面体2次要素、要素数:123万要素	対応後(binary)	
	linux	windows
vtuファイル(binary)作成-表示までの時間 (s)	16	25
1) メッシュ読み込み, str文字を数値に変換 (s)	1.88	2.43
2) solidの表面要素を取得 (s)	1.40	1.60

今回の対応で、vtuファイル作成時間を 2 / 3 に低減できる。

binary形式にすることで、さらに 1 / 2 に低減できる。

(1.項のascii、binaryの確認は、この対応を反映した状態で確認している。)

vtuファイルのフォーマットをbinaryにすることにより、
windowsでは、vtuファイルの読込-表示までの時間が10倍（17s⇒1.6s）早まっている。
（123万要素でも数秒で表示できる。）

これにより、EasyISTRの起動時間が、大幅に改善されている為、

- ・従来は、vtk表示に時間が掛かり過ぎる場合、
時間制限を設定して、vtk非表示でEasyISTRを起動させていた。
- ・今回から、この時間制限を廃止する。

さらに、vtuファイル作成時間を短縮している為、100万要素の大規模モデルでも、
EasyISTR上で、違和感なく操作設定できる。

2. mesh作成の為のmeshSize設定方法修正

現状、defaultで設定しているmeshSizeの設定では、メッシュが作成できない事がある。

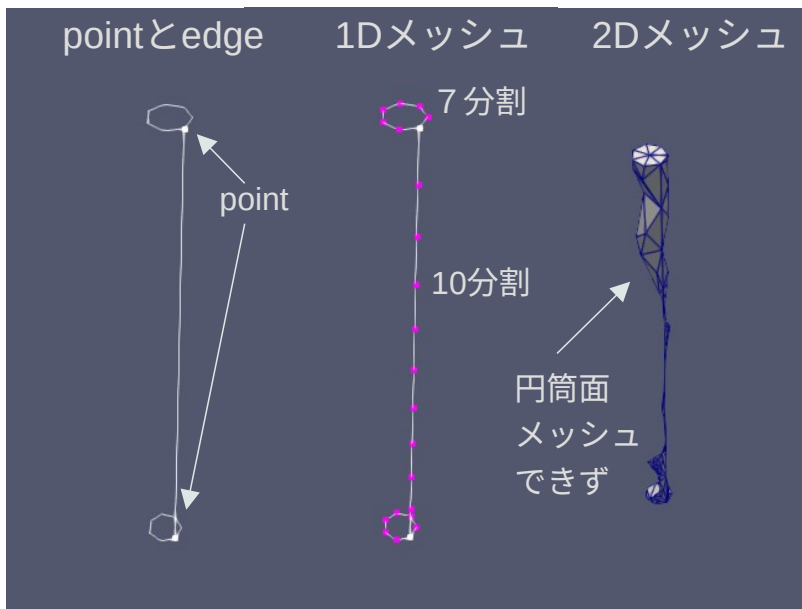
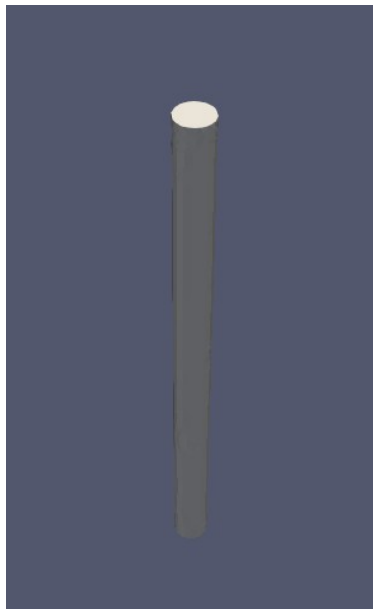
- 1) 細長い円柱形状の場合、メッシュ作成できず
- 2) 細かい部品を多数含む場合、メッシュ作成できず

以下の様な円柱形状は、defaultの設定でメッシュが作成できない。

メッシュは、1Dメッシュ(edge)→2Dメッシュ(surface)→3Dメッシュ(solid)を作成する

Φ20 x 300 mmの円柱

1Dメッシュと2Dメッシュ



1Dメッシュでは、

円形edge：7分割

直線edge：10分割

される。(defaultの設定)

2Dメッシュでは、

上下面はメッシュができるが、
円筒面は、直線edgeの分割が粗い
為、メッシュが作成できない。

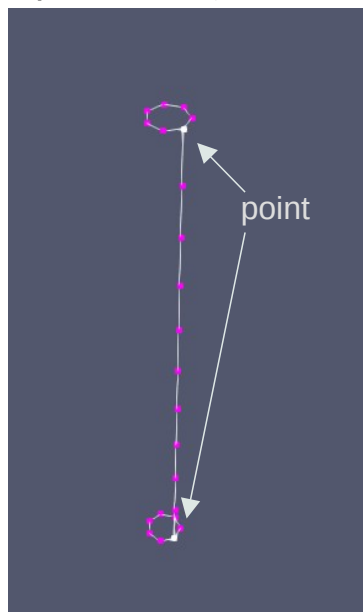
<対応方法>

各pointについて、pointに接続されるedgeの要素sizeを確認し、
そのpointに、edgeのmin要素Sizeを設定後、1Dメッシュを作成する。
→ 円形edge、直線edgeとも同じ要素sizeが設定される。

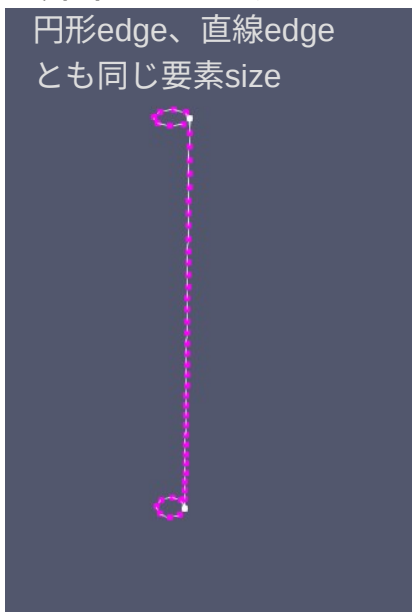
対応後のメッシュ作成方法

仮に1Dメッシュ作成 → 全pointについて、そのpointのmin要素sizeを設定
→ 再度1Dメッシュ → 2Dメッシュ → 3Dメッシュを作成すると、うまくメッシュが切れる。

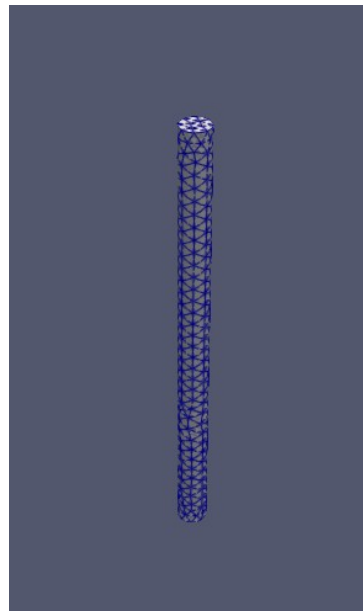
従来の1Dメッシュ



今回の1Dメッシュ



今回の2Dメッシュ



2) 細かい部品を多数含む場合、メッシュ作成できず

下図の様なモデルの場合、defaultの設定では、メッシュが切れない。

(以下のモデルは、PrePoMax1.1.0のModelsフォルダ内の「Valve_assm.STEP」ファイルを引用)

Gmshのthread並列を外して、maxMeshSizeの設定を小さくすると、メッシュが切れる。

maxMeshSizeは、CADモデルのXYZ各方向のサイズ（長さ）から、算出している。

今回のモデルの様に細かい部品で構成されたモデルに対応する為には、

部品サイズに応じたmaxMeshSizeに設定する必要がある。

→ 1Dメッシュ（edge）の最大要素サイズをmaxMeshSizeに設定する様に修正。



Gmshのthread並列に関しては、今回のモデルの様な場合、エラー停止する。

singleCoreでGmshを実行するとエラー発生しない。

(エラー発生原因不明)

「Valve_asem.STEP」のメッシュ作成結果



部品点数が25個ある複雑なモデル

defaultの設定でメッシュが作成できる。

PrePoMax1.1.0のModelsフォルダ内には、他に以下のCADファイルがある。

Assembly.STEP	Generator_bracket.step	Sheet-Metal-Bracket.STEP
Assembly_T.STEP	Hertz.STEP	Support_plate.STEP
Bearing_bracket.STEP	Nosilec.STEP	Support_plate_no_submodel.STEP
Contact.STEP	PBA-648-000.STEP	Support_plate_submodel.STEP
Crane.STEP	Plates_Bolt.STEP	Tensile_sample.STEP
Engine-piston.STEP	Profile.STEP	Valve_assm.STEP

これらの全18ファイルを使って、defaultの設定でメッシュが切れるか確認した結果、
全てのファイルがdefaultの設定でメッシュが切れた。

この為、現在の設定で、ほぼメッシュが切れる状態になっている。

(もし、エラー発生した場合、maxMeshSizeを小さくして、再度トライする。)

