H. Dwyer

Aug 2019

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

**EndoVantage Challenge Notes**

Received zip file Friday 8/02/19. Result due Monday morning, 8/04/19.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

8/02/19

Unpacked zip file.

Printed the instructions and problem statement.

I understand the 2D triangulation, I think

* the grid is square in the examples, 200x200
* the grid has the same number of subdivisions vertically and horizontally. The number is controlled by a global constant DX\_GLOBAL, which is the dimension of a triangle in X.
* there is no DY\_GLOBAL ... we use DX\_GLOBAL in both directions
* the 2D projection of the elements will all be axis-aligned right triangles of the same size and with

What should we do if DX\_GLOBAL does not evenly divide the dimension of the grid?

Looking at the code, I see that there are implementations for most (all?) required procedures.

We will use a timer to compare performance of the original implementation and any changes I make.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

There does not seem to be any procedure to set Z values for the nodes.

If the Z values are all 0.0 then all the elements are the same shape and area, all the normals are vertical, and the area matrix is filled with identical values.

The problem description contains “upon arrival” ... but I have not heard that I am invited to visit (?)

I assume that I will send in a solution and they will evaluate it before making a decision on how to proceed.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

We are computing a matrix which contains the areas of the elements.

We then compute . I know that the matrix will be square and symmetric.

I note that the routine which computes this matrix product does not use the symmetry.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

It does not appear that the supplied procedures make any use of parallel processing.

I will explore this as time allows.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

8/03/19

Installed VS 2019. Successfully created a “Hello world” console application.

VS 2019 runs very slowly on my machine

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

There does not seem to be a VS project included in the supplied files. The main routine seems to be a simple console application. The emphasis is clearly on run time, so I will not take the time to do even a simple user interface.

**The next step would seem to be to get the supplied code into a VS 2019 console app and get it to execute.**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Added KPoint, Element, Node, and Vector to console app.

It builds, runs, says “Hello”.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Moved body of supplied main.cpp into VS solution

Builds, executes. Returns 441, 800

using

DX\_GLOBAL = 10

double dx = DX\_GLOBAL;

double width = 200;

double height = 200;

With 20 intervals there are 21 endpoints: 21\*21=441

With 20 intervals there are 20\*20 rectangles and 20\*20\*2=800 triangles.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

Placed code and header files under version control in branch **EndoTestBranch**.

Added this file to version control.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

Added progress messages.

* With DX\_GLOBAL = 10 we go into the MatrixMatrix procedure and do not come out for a while (I did not wait).
* With DX\_GLOBAL = 100 we get through all the code with a time of about 0.003 sec
* Array Y contains ridiculous values and the norm is ‘inf’

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

procedure **SetupModel**

Decreased the operation count slightly.

Found that two local dynamic arrays had not been deleted, so added two delete[] statements.

With DX\_Global = 10, time for **SetupModel** is approx. 0.045 sec.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

procedure **AllocateArrays**

Decreased the operation count slightly.

With DX\_Global = 10, time for **SetupModel** and **AllocateArrays** is approx. 0.071 sec.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

procedure **PopulateArrays**

Decreased the operation count slightly.

Moved call to get element area into the outer loop.

With DX\_Global = 10, time for **SetupModel**, **AllocateArrays** and **PopulateArrays** is approx. 0.078 sec.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

class **Element**

Made the center, area and face normal into protected data fields and added booleans to indicate if they have been computed.

Modified **GetCenter**, **GetArea**, and **GetFaceNormal** so that the values are computed only as required.

Combined the computation of the face normal and the area into one routine.

With DX\_Global = 10, time for **SetupModel**, **AllocateArrays** and **PopulateArrays** is approx. 0.071 sec.

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

8/04/19

procedure **MatrixMatrix**

Created a debug procedure **Debug\_MockAreaMatrix** to allow me to create a matrix **area** of any size, for testing of **MatrixMatrix**.

Using size 3x3 it gives

The result from **MatrixMatrix** is

Calculation using a calculator confirms that the values in ATA are correct.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

We know that a matrix product will always be symmetric.

Procedure **MatrixMatrix** does not take advantage of this. By doing so, we can cut execution time in half.

Time test

|  |  |  |  |
| --- | --- | --- | --- |
| n | time for  **Debug\_MockAreaMatrix** | time for  **MatrixMatrix** (orig) | time for  **MatrixMatrix** (rewrite) |
| 300 | 0.020 | 0.355 | 0.087 |
| 600 | 0.077 | 6.174 | 2.812 |
| 1200 | 0.281 | 82.800 | 41.298 |

As expected, **Debug\_MockAreaMatrix** is very clearly .

The behavior of **MatrixMatrix** is not as expected ... the rewrite did cut the time in half, but:

* doubling n from 300 to 600 increased time by a factor of about 17.4 ( )
* doubling n from 600 to 1200 increased time by a factor of about 13.4 ( )