Is is possible to convert C# double[,,] array to double[] without making a copy

Asked 16 years, 3 months ago Modified 16 years, 3 months ago Viewed 8k times

6

I have huge 3D arrays of numbers in my .NET application. I need to convert them to a 1D array to pass it to a COM library. Is there a way to convert the array without making a copy of all the data?

I can do the conversion like this, but then I use twice the ammount of memory which is an issue in my application:

```
  double[] result = new double[input.GetLength(0) * input.GetLength(1) *
input.GetLength(2)];
  for (i = 0; i < input.GetLength(0); i++)
    for (j = 0; j < input.GetLength(1); j++)
      for (k = 0; k < input.GetLength(2); k++)
        result[i * input.GetLength(1) * input.GetLength(2) + j *
input.GetLength(2) + k)] = input[i,j,l];
  return result;
```

c# .net arrays

Share Improve this question Follow

asked Sep 18, 2008 at 19:29

Hallgrim

15.5k ● 11 ● 48 ● 54

5 Answers

Sorted by: Highest score (default)

8

I don't believe the way C# stores that data in memory would make it feasible the same way a simple cast in C would. Why not use a 1d array to begin with and perhaps make a class for the type so you can access it in your program as if it were a 3d array?

Share Improve this answer Follow

answered Sep 18, 2008 at 19:33

Loren Segal

3,272 ● 1 ● 30 ● 29

6

Unfortunately, C# arrays aren't guaranteed to be in contiguous memory like they are in closer-to-the-metal languages like C. So, no. There's no way to convert double[,,] to double[] without an element-by-element copy.

Share Improve this answer Follow

answered Sep 18, 2008 at 19:32

Yes - that Jake.

17.1k ● 15 ● 72 ● 99

3

Consider abstracting access to the data with a Proxy (similar to iterators/smart-pointers in C++). Unfortunately, syntax isn't as clean as C++ as operator() not available to overload and operator[] is single-arg, but still close.

Of course, this extra level of abstraction adds complexity and work of its own, but it would allow you to make minimal changes to existing code that uses double[,,] objects, while allowing you to use a single double[] array for both interop and your in-C# computation.

```
class Matrix3
{
    // referece-to-element object
    public struct Matrix3Elem{
        private Matrix3Impl impl;
        private uint dim0, dim1, dim2;
        // other constructors
        Matrix3Elem(Matrix3Impl impl_, uint dim0_, uint dim1_, uint dim2_) {
            impl = impl_; dim0 = dim0_; dim1 = dim1_; dim2 = dim2_;
        }
        public double Value{
            get { return impl.GetAt(dim0,dim1,dim2); }
            set { impl.SetAt(dim0, dim1, dim2, value); }
        }
    }

    // implementation object
    internal class Matrix3Impl
    {
        private double[] data;
        uint dsize0, dsize1, dsize2; // dimension sizes
        // .. Resize()
        public double GetAt(uint dim0, uint dim1, uint dim2) {
            // .. check bounds
            return data[ (dim2 * dsize1 + dim1) * dsize0 + dim0 ];
        }
        public void SetAt(uint dim0, uint dim1, uint dim2, double value) {
            // .. check bounds
            data[ (dim2 * dsize1 + dim1) * dsize0 + dim0 ] = value;
        }
    }

    private Matrix3Impl impl;
```

```
    public Matrix3Elem Elem(uint dim0, uint dim1, uint dim2){
        return new Matrix2Elem(dim0, dim1, dim2);
    }
    // .. Resize
    // .. GetLength0(), GetLength1(), GetLength1()
}
```

And then using this type to both read and write -- 'foo[1,2,3]' is now written as 'foo.Elem(1,2,3).Value', in both reading values and writing values, on left side of assignment and value expressions.

```
void normalize(Matrix3 m){

    double s = 0;
    for (i = 0; i < input.GetLength0; i++)
        for (j = 0; j < input.GetLength(1); j++)
            for (k = 0; k < input.GetLength(2); k++)
    {
        s += m.Elem(i,j,k).Value;
    }
    for (i = 0; i < input.GetLength0; i++)
        for (j = 0; j < input.GetLength(1); j++)
            for (k = 0; k < input.GetLength(2); k++)
    {
        m.Elem(i,j,k).Value /= s;
    }
}
```

Again, added development costs, but shares data, removing copying overhead and copying related developtment costs. It's a tradeoff.

Share Improve this answer Follow

answered Sep 18, 2008 at 20:32

Aaron

3,474 ● 25 ● 26

1

As a workaround you could make a class which maintains the array in one dimensional form (maybe even in closer to bare metal form so you can pass it easily to the COM library?) and then overload operator[] on this class to make it usable as a multidimensional array in your C# code.

Share Improve this answer Follow

answered Sep 18, 2008 at 19:34

Tobi

81.4k ● 6 ● 34 ● 37

Without knowing details of your COM library, I'd look into creating a facade class in .Net and exposing it to COM, if necessary.

1

Your facade would take a double[,,] and have an indexer that will map from [] to [,,].

Edit: I agree about the points made in the comments, Lorens suggestion is better.

Share

Improve this answer

Follow

edited Sep 18, 2008 at 19:49

answered Sep 18, 2008 at 19:33

Robert Jeppesen

7,857 ● 3 ● 37 ● 50

This won't work. COM will **need** to have the literal double[] array. If anything, you'd want to create a .NET class that made the double[] act like a double[,,]. – Jonathan Rupp Sep 18, 2008 at 19:36

I am not able to change the COM library. – Hallgrim Sep 18, 2008 at 19:42