Subject: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 09:43:35 GMT
View Forum Message <> Reply to Message

Hi,

Some queries regarding the appropriateness of Arrys or Vectors. Please consider a generic scenario:

1.- In the manual it is mentioned that Arrays are significantly slower than Vectors. In a real

2.- Programming wise, are there important differences? For example, are Arrays more complex to program and debug?.

3.- Debugging: If we store a class in Array, how to visualize the data of the classes stored in the Array with the debugger? Apparently this is not possible, which penalizes Array in comparison to Vectors (please correct me if I am wrong).

4.- Memory consumption wise: manual says that for small size of Array, this is more memory consuming than its Vector counterpart and improvements appears as Arrys sizes grows. Any hints about this? Any real figures from real applications?

5.- Pointers: apparently if you need pointers to classes, then Arrays is the only option available. What other limitations Arrys have compared with Vectors and vice versa?.

6.- In general (perhaps this is a non sense question) what is your preferred container?.
Thank you.

Best wishes,
Javier

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 10:20:44 GMT
View Forum Message <> Reply to Message

1) in general, no. if OTOH you use the containers in a 'high performance scenario, loops, creation, destruction, adding, it might be of interest to use Vector, especially for intrinsic types. but for GUI applications as storage for some user data structure, it has no perceptible difference.

2) the most important difference is, that Vector holds the mem space for the objects itself in a sequence and creates the objects (calls constructor) there, while Array creates the objects somewhere else on the heap, and stores only the pointers to them. thats why Adding to Vector can render any previous referenece to an object inside it as invalid, thats also why the objects need to be Moveable<>. see manual. Arrays dont need those precautions. in Vector, you cant Detach an element, and Attach() it in another Vector, in Array, you can, whithout even touching

the element itself (handling over pointers in Array).
programming with both is quite the same, infact, Upp has MACROS which apply to a variety of Containers, because they have a huge subset of common Methods, like GetCount(), Add(), etc.
debugging is not really easy, since Upp doesnt yet (AFAIK) support expressions, so things like looking at vec[2] are not possible. but you can look at v[0] indirectly. inspectiong them in terms of count, allocated mem etc is no problem.

3) explained above, they both lack the same. maybe try to hassle with the Vector<void*> vector inside the Array, '*(MyClass*)arr.vector[12]' or sth.

4) depends on use case and whether it really matters, best is to check the implementations, they are not too hard to understand, knowing the basics from manual.

5) correctly speaking, if you need references in which ever way to classes, beeing it pointers, or refs, that should not get invalid once you manipulate the container, (adding is dangerous, needs growing and 'moving' objects in Vector), Array is the only choice. it also grows, but only moves the pointers to the objects, which remain in same place.

6) Vector<> for intrinsic types, Buffers, 'oldschool' arrays (MyType ar[12] replacement, Arrays, if type is quite complex, or need to 'move' them around, by detaching/ataching somewhere else.

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 10:43:57 GMT
View Forum Message <> Reply to Message

Thanks kohait, well explained.

Regarding debugging, I really need to how to inspect an Array. How to do it in an effective manner?
I am open to any suggestion.

On the other hand, my application requires extensive access to containers in a loop, so Array option does not look very attractive.

Cheers,
Javier

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 10:51:31 GMT
View Forum Message <> Reply to Message

for inspecting you could use a debugger function, that simply 'reads' the references

```
void ContInspector(const Vector<MyType>& v)
{
   for(int i = 0; i<v.GetCount(); i++)
   {
      const MyType& t = v[i];
      int d = 0; //dummy, set breakpoint here and inspect t
   }
}
```

what kind of data is the content of the containers?

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 11:08:55 GMT
View Forum Message <> Reply to Message

Data are geometric objects (that require to be painted with OpenGL in a loop). For example:

```
template<class T>
class Point3D:MoveableAndDeepCopyOption<Point3D<T>>{
private:
 T x,y,z,w;
 unsigned int id,layer;
 bool picked;
 void Copy(const Point3D<T>& src){
  x=src.x;
  y=src.y;
  z=src.z;
  w=src.w;
  id=src.id;
  picked=src.picked;
  layer=src.layer;
 }
public:
 Point3D (){
  x=T();
  y=T();
  z=T();
  w=static_cast<T>(1.0);
  id=0;picked=0;
 }
 Point3D (T a, T b, T c,unsigned int e, bool f, unsigned int g, T d=static_cast<T>(1.0)){
  x=a;y=b;z=c;w=d;
  id=e;
  picked=f;
  layer=g;
 }
```

```
 Point3D (Point3D<T> &point){
  x=point.get_x();y=point.get_y();z=point.get_z();w=point.get_w();
  id=point.get_id();
  picked=point.get_picked();
  layer=point.get_layer();
 }
 Point3D (const Point3D<T> &src,int){Copy(src);}
 void set_x(T a){x=a;}
 void set_y(T a){y=a;}
 void set_z(T a){z=a;}
 void set_w(T a){w=a;}
 void set_id(unsigned int a){id=a;}
 void set_picked(bool a){picked=a;}
 void set_layer(unsigned int a){layer=a;}
 T get_x()const{return x;}
 T get_y()const{return y;}
 T get_z()const{return z;}
 T get_w()const{return w;}
 unsigned int get_id(){return id;}
 bool get_picked(){return picked;}
 unsigned int get_layer(){return layer;}
 void operator*(const T& obj){
  x=x*obj;
  y=y*obj;
  z=z*obj;
  w=w*obj;
 }
 Point3D<T> operator+(const Point3D<T> &obj){
  return Point3D<T>(x+obj.get_x(),y+obj.get_y(),z+obj.get_z(),w+obj.get_w());
 }
 void Serialize(Stream& s){
  s%x%y%z%w%id%picked%layer;
 }
 String ToString()const{
  String s;
  s<<"x:"<<x<<","<<"y:"<<y<<","<<"z:"<<z<<","<<"w:"<<w<<","<<"id:"<<id<<","<<"picked:"<<picke
d<<", "<<"layer:"<<layer;
  return s;
 }
};
```

I am using other geometrical structures, for example line3D, nurbs lines and surfaces.

Let us take a line. Here is the question: the two points that form part of a line, is it better to reference them by using pointers (hence Array has to be used to store the point3D) or by emulating their content or by storing a simple identifier (an unsigned int which looks the easiest way to do it)?.

Remark: I have tried to debug an Arry<int> array and added an integer array.Add(1): when debugging U++ crashes when clicking on the .vector, what is it going on wrong? It is weird, is it not?

Thanks,
Javier

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 11:36:22 GMT
View Forum Message <> Reply to Message

i'd recommend to use Vector here, see Core/Gtypes.h for Size and Point implementations, they also are templates, and are only Moveable<>, no need for you to MoveableAndDeepCopyOption<> them. it is a lightweight pointer class as i can see. it is best copied. i suppose T is double or float..

picked is misleading... Upp has own usage for pick stuff, think of a different name for it.

also, use the initializer list in constructors, and the implicit logic of classes (aka implicit copy constructor).

think of what you really need to have getters/setters to. better make the members public in this case.

```
template<class T>
class Point3D : Moveable<Point3D<T> >
{
public:
 T x,y,z,w;
 unsigned int id,layer;
 bool taken;
public:
 Point3D ()
  : x(T())
  , y(T())
  , z(T())
  , w((T)1.0)
  , id(0)
  , layer(-1)
  , taken(0)
 {}

 Point3D (const T& a, const T& b, const T& c, unsigned int e, bool f, unsigned int g, const T& d=(T)1.0)
  : x(a)
  , y(b)
```

```
  , z(c)
  , w(d)
  , id(e)
  , layer(g)
  , taken(f)
 {}

 //use implicit copy constructor

 //this one should probaly be Point3D<T> operator*(const T& o) const, since yours is modifying
object
 void operator*(const T& o){
  x*=o;
  y*=o;
  z*=o;
  w*=o;
 }

 Point3D<T> operator+(const Point3D<T> &o){
  return Point3D<T>(x+o.x(),y+o.y(),z+o.z(),w+o.w());
 }

 void Serialize(Stream& s){
  s%x%y%z%w%id%taken%layer;
 }

 String ToString()const{
  String s;
  s<<"x:"<<x<<","<<"y:"<<y<<","<<"z:"<<z<<","<<"w:"<<w<<","<<"id:"<<id<<","<<"taken:"<<taken<
<", "<<"layer:"<<layer;
  return s;
 }
};


GUI_APP_MAIN
{
 Vector<Point3D<double> > v;
 v.Add();

 Vector<Point3D<double> > vw;
 vw <<= v;

 Point3D<double> p;
 p = v[0];
}
```

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 16:23:21 GMT
View Forum Message <> Reply to Message

Roger. DeepCopyOption is not needed. Your hints are also correct, thanks.

But, what about my question regarding Array<int> and the debugger? U++ is crashing when clicking in the container during debug. It is frustrating to see that the only way to inspect an Array is through a function.

Thank you.

Javier

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 16:54:21 GMT
View Forum Message <> Reply to Message

i try to avoid inspecting containers where possible and try to check things beforehand.. i'd also prefer to be able to inspect containers. array is IMHO not too hard, one'd need to make the internal Vector<void*> vector a typed vector, Vector<T*>, and have the Vector itself be inspectable, which is the hard part. dont know how and who could realize that.

in case of the crash i cant help much, i had it once, but it disappeared somewhere..and i didnt care much again. sorry.

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 17:20:33 GMT
View Forum Message <> Reply to Message

Thanks.

(for example, a Line2D needs to point to its two constituent points Point2D).

An additional question, please. Have a look to these two classes:

```
template<class T>
class Point2D:Moveable<Point2D<T>>{
private:
 T x,y;
public:
```

```
 Point2D():x(T()),y(T()){}
 Point2D(const T &a, const T &b): x(a),y(b){}
 T get_x()const{return x;}
 T get_y()const{return y;}
 void set_x(T a){x=a;}
 void set_y(T a){y=a;}
 Point2D<T> operator+(const Point2D<T> &obj){
  return Point3D<T>(x+obj.x,y+obj.get_y());
 }
};
```

and

```
template<class T>
class Line2D:Moveable<Line2D<T>>{
private:
 Point2D<T> *p1,*p2;
public:
 Line2D():p1(),p2(){}
 Line2D(const Point2D<T> &point1, const Point2D<T> &point2):
  p1(&point1),
  p2(&point2)
 {}
 Point2D<T> get_p1() const{return *p1;}
 Point2D<T> get_p2() const{return *p2;}
 void set_p1(const Point2D<T> &p){p1=&p;}
 void set_p2(const Point2D<T> &p){p2=&p;}
};
```

The compiler is claiming about the constructor in Point2D; it says that it cannot assign const pointers to *p1 and *p2. Why is this?. I would appreciate your advice.

Thanks.

Cheers,

Javier

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 18:04:56 GMT
View Forum Message <> Reply to Message

ok. *p1 and *p2 have to be const.

Subject: Re: Arrys vs Vectors
Posted by Didier on Tue, 21 Dec 2010 18:45:17 GMT
View Forum Message <> Reply to Message

Hi,

The UPP debugger, from my point of view, and at least under linux, is not very usable but this point is not very important since DDD exists.

So if you are working in linux, just use DDD: you can do all the inspections you want and it works perfectly.

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 18:48:43 GMT
View Forum Message <> Reply to Message

why cant Line2D have Point2D copies in it?

BTW: considered using Painter?

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 20:54:47 GMT
View Forum Message <> Reply to Message

Quote:why cant Line2D have Point2D copies in it?

Actually this is the current design (+Vectors) to store the geometrical classes. I am dubious about the appropriateness of using Array + pointers.

In general the construction of geometrical entities is done from bottom up: first points, then lines/curves that contain end/intermediate points, then surfaces than contain lines or curves, then solid objects that contain surfaces. So, every class needs to reference somehow to its constitutive lower classes (a line to its two points, etc). So my question is:

1.- either to copy the classes of the lower elements inside the upper element: I think this would imply more space to store de data; this solution can be solved by using Vector as container. I think this option is faster since Vector is faster than Array.

2.-or to use pointers to achieve the same, then the model will be smaller. This needs the usage of Array. This option looks slower than 1.

Thanks.

Best wishes,
Javier

Subject: Re: Arrys vs Vectors
Posted by 281264 on Tue, 21 Dec 2010 21:04:47 GMT
View Forum Message <> Reply to Message

Answering to Didier I have to say that Windows is my choice; so, any other suggestion?.

Perhaps for others is not but, IMHO, to me the debugger is crucial, since the inspection of classes/variables is very important (yes, typically one tests a class before it is implemented; what I mean is the run time behavior, mainly when some mathematical calculations are involved). If U++ has wonderful containers, why not a wonderful debugger capable to make most of it? I wish U++ developers hear me and make some improvements.

Best wishes,
Javier

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Tue, 21 Dec 2010 21:12:00 GMT
View Forum Message <> Reply to Message

looks like it will be a heavy duty 3d Model thing
the basic question is in fact: do the upper layers really really need to have the exactly same point references, because the points maybe are crucial for the model parametrisation, and you modify the points and expect the whole model to be able to recalculate on top of this or are the higher primitives like lines independant from the point instances after creation?

i would do a mix of both:

storing Points not as refs or pointers but as-is copies in Lines etc.. but then, from there, use Pointers to Lines and the like in the upper layers. i think here you will be able to achieve both, easy use and good overall performance.

it's not quite easy to determine the right approach, but again, like a friend of mine used to say: 90% percent of programming is choosing the right approach, which is 90% based on choosing the right data structures

Subject: Re: Arrys vs Vectors
Posted by mirek on Sat, 25 Dec 2010 09:57:17 GMT

281264 wrote on Tue, 21 December 2010 11:23Roger. DeepCopyOption is not needed. Your hints are also correct, thanks.

But, what about my question regarding Array<int> and the debugger? U++ is crashing when clicking in the container during debug.

Sorry about that. I would like to fix it, can you describe the exact way how to crash it?

Quote:
 It is frustrating to see that the only way to inspect an Array is through a function.

Well, over years, I have rather got used to debug things trough .logs. Especially if larger data (-> containers) or GUI are in play, logging IMO is much more productive approach.

I believe it worth to try for you, just place DDUMPC(container) here and there and see how it goes...

Another option is to use msdev.exe (Visual C++ environment) for debugging. I know Tom does this all the time. You can configure theide to use msdev as 'external' debugger. And today, it is free. But I doubt it is any helpfil with containers either.

Certainly, improving debugger would be nice to

Hm, now thinking about it, maybe something like:

```
template <class T>
class Array : public MoveableAndDeepCopyOption< Array<T> > {
protected:
#ifdef _DEBUG
 Vector<T *> vector;
#else
 Vector<void *> vector;
#endif
```

might be helpful...

Mirek

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Sun, 26 Dec 2010 09:07:50 GMT

View Forum Message <> Reply to Message

Quote:
Vector<T *> vector;

why not generally do it like that? what was the reason for the void* ?

Subject: Re: Arrys vs Vectors
Posted by 281264 on Mon, 27 Dec 2010 10:51:27 GMT
View Forum Message <> Reply to Message

Thank you for your answers and advises.

Regarding DUMPC et al. I am getting used to them and find DUMPC and Logfile very useful. It works very well, thanks. Remark: By the way,

**Query: what are DUMPCC, DUMPCCC and DDUMPC for?. So far I only know DUMP and DUMPC.

Another question more related to C++: the usage of pointers and Arrays of pointers as class

**Query: if I have an Array<ptr*> as a variable private member of a class, does it work? Is it any limitation in the NTL containers for being used as containers of pointers to classes as member variables of a class?

Regarding DeepCopyOptionness I am using it in all my classes for I find it very useful.

**Query: Does it penalize or increases the complexity of the code in any way whatsoever?

Debugging: I have tried Microsoft Windbg as external debugger but it does not inspect the NTL containers. Nevertheless, as Mirek suggests, the usage of DUMPS et al. is ok. The crash in the debugger faded away after a U++ re-installation. An improvement that I recommend is the increase of the number lines displayed to visualize a variable (so far is 20 lines). I think this is useful and easy for you guys to do.

**Query: Is there any way to preserve the U++ personal configuration between installations?

Best wishes and Merry Christmas,

Javier

Subject: Re: Arrys vs Vectors
Posted by mirek on Mon, 27 Dec 2010 12:23:49 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Sun, 26 December 2010 04:07Quote:
 Vector<T *> vector;

why not generally do it like that? what was the reason for the void* ?

With void *, the name signatures of most non-inline functions generated for template are the same, so there is basically only one code in .exe for all Arrays, regardless of T. Results in smaller .exe.

For debug mode we do not care about .exe size too much...

Subject: Re: Arrys vs Vectors
Posted by mirek on Mon, 27 Dec 2010 12:29:27 GMT
View Forum Message <> Reply to Message

281264 wrote on Mon, 27 December 2010 05:51Thank you for your answers and advises.

Regarding DUMPC et al. I am getting used to them and find DUMPC and Logfile very useful. It works very well, thanks. Remark: By the way,

**Query: what are DUMPCC, DUMPCCC and DDUMPC for?. So far I only know DUMP and DUMPC.

DUMPCC is for dumpning container of containers...

Quote:
Another question more related to C++: the usage of pointers and Arrays of pointers as class

Well, the U++ way is to try to avoid pointers as much as possible...

Why not use proper objects instead?

Quote:
**Query: if I have an Array<ptr*> as a variable private member of a class, does it work? Is it any limitation in the NTL containers for being used as containers of pointers to classes as member variables of a class?

No. Just you have to keep in mind that the object then in container is pointer...

Also, if you need to store pointers, Vector<ptr*> is most likely the better option.

Quote:
Regarding DeepCopyOptionness I am using it in all my classes for I find it very useful.

**Query: Does it penalize or increases the complexity of the code in any way whatsoever?

DeepCopy _Option_ is definitely fine as it is only used explicitly.

Still, I surprised you are using that so much. I found using myself the deep-copy-option quite rarely.

Quote:
**Query: Is there any way to preserve the U++ personal configuration between installations?

I guess once you get accustomed, you will start recompiling theide based on svn - then you just replace theide.exe and keep configuration.

I guess the "U++ installation" should perhaps rather be considered to be "U++ starter pack"

Mirek

Subject: Re: Arrys vs Vectors
Posted by kohait00 on Mon, 27 Dec 2010 13:15:44 GMT
View Forum Message <> Reply to Message

Quote:
so there is basically only one code in .exe for all Arrays,

NIIICE thumbs up.

Subject: Re: Arrys vs Vectors
Posted by gprentice on Mon, 27 Dec 2010 20:13:42 GMT
View Forum Message <> Reply to Message

281264 wrote**Query: Is there any way to preserve the U++ personal configuration between installations?
Javier

http://www.ultimatepp.org/appideFiles$en-us.html

Personal configuration is in theide.cfg if the U++ executable is called theide.cfg. You can preserve your configuration by preserving the content of the root installation folder other than theide.exe and .map

Graeme