Field Data Changes

Kitware Inc.

October 10, 2001

1 Changes to vtkFieldData and vtkDataArray

vtkFieldData's interface for adding and naming is unnecessarily complicated. There are many different methods for adding arrays. Some of these take a name, some do not. Furthermore, since it is possible to store array without associated names, it is difficult to differentiate arrays. The only reliable and unambiguous way of accessing arrays is through array indices (which can be change with the available Add/Set commands). To reduce this complexity, I introduced several changes to vtkFieldData and vtkDataArray.

1.1 vtkDataArray

In order to make vtkDataArray a more independent class, I moved some of the functionality from a few other classes into vtkDataArray. The most important of these two are names and lookup tables. Each instance of vtkDataArray will have it's own name and lookup table. At the creation of vtkDataArray, a (unique) name is assigned. This name can be changed with

```
void SetName(const char* name);
```

Note that this method will not accept a null pointer. Although it is possible to assign an empty name (`""`), I do not recommend it.

The lookup table which was originally in vtkScalars moved to vtkDataArray. Setting the lookup table of vtkScalars is still possible and this will in turn set the underlying vtkDataArray's lookup table. However, if the data array is changed with `void vtkAttributeData::SetData(vtkDataArray *)`, the lookup table will be lost. Furthermore, trying to set the lookup table of a scalar object with a null data array will not work (it is a no-op).

Another thing to note (this has nothing to do with my changes) is that `ext` in `virtual int Allocate(const int sz, const int ext=1000)` is no longer used. I found this out while going over `vtkFloatArray.cxx`. Instead of using `ext`, all data arrays increase their internal storage by 100% when the user tries to access insert the last id.

1.2 vtkFieldData

Adding and Removing Arrays

Since each array will now have a name, it is easier (and safer) to access the data arrays in the field data by name. Therefore in order to simplify vtkFieldData's interface and to make array access safer, I removed the following methods

```
void SetArray(int i, vtkDataArray *);
int AddArray(vtkDataArray *array, const char *name);
int AddReplaceArray(vtkDataArray *array, const char *name);
int AddNoReplaceArray(vtkDataArray *array, const char *name);
```

Instead of these methods, you can use

```
void vtkDataArray::SetName(const char* name);
```

followed by

```
int AddArray(vtkDataArray *array);
```

If there already is an array with same name in the field data, it will be replaced. The recommended way of adding multiple arrays is shown in the following example:

```
vtkFloatArray *arrays[3];
// ... Create arrays here

vtkFieldData* fd = vtkFieldData::New();
fd->SetNumberOfArrays(3);
arrays[0]->SetName("array0");
fd->AddArray(arrays[0];
arrays[1]->SetName("array1");
fd->AddArray(arrays[1];
arrays[2]->SetName("array2");
fd->AddArray(arrays[2];
```

It is still possible to add more arrays after this point because the field data object can dynamically reallocate it's internal pointer list and increase it's capacity. I also added a method for removing an array:

```
void RemoveArray(const char *name);
```

vtkFieldData::Iterator

C++ users can use vtkFieldData::Iterator to access some or all the data arrays in a field data. Here are two examples:

```
// ... Create a field data called fd and add some arrays
vtkFieldData::Iterator it(fd);
vtkDataArray* da;
for(da=it.Begin(); !it.End(); da=it.Next())
  {
  // Do something with da
  }

// ... Create a field data called fd and add some arrays
int indices[2] = {0, 2}
vtkFieldData::Iterator it(fd, indices, 2);
vtkDataArray* da;
for(da=it.Begin(); !it.End(); da=it.Next())
  {
  // Do something with da (where da will be array 0 and 2)
  }
```

This iterator is used in vtkDataSetAttributes (more on this later).

2 vtkDataSetAttributes

2.1 Inheritance

Instead of containing a vtkFieldData, vtkDataSetAttributes (therefore, vtkPointData and vtkCellData which are subclasses of vtkDataSetAttributes) now inherits from vtkFieldData. The old and new class hierarchies are show in figures 1 and 2. Since vtkPointData and vtkCellData are now subclasses of vtkFieldData, there is no more need to interact with a FieldData member to add and remove arrays (note that vtkDataObject still has a FieldData member which is manipulated as usual). GetFieldData() is still available and is implemented as follows:

```
vtkFieldData* GetFieldData() { return this; }
```

It is no longer possible to SetFieldData(). Furthermore, it is no longer necessary to check if a FieldData exists before adding/getting arrays. Therefore, something like this

```
if (!pd->GetFieldData())
  {
  vtkFieldData* fd = vtkFieldData::New();
```

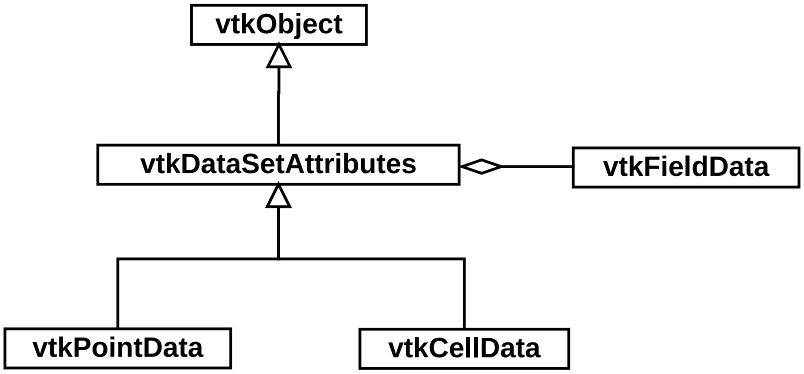

Figure 1: Class hierarchy before the changes.

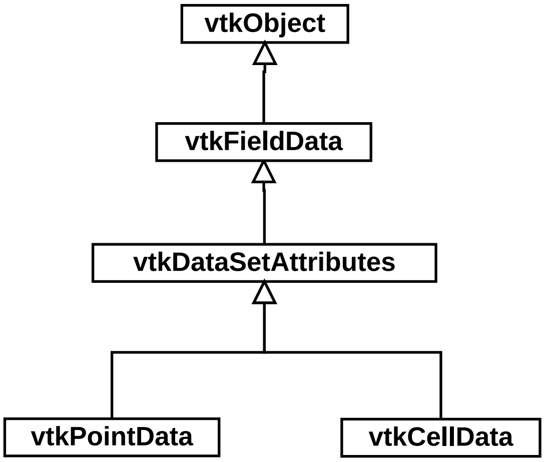

Figure 2: Class hierarchy after the changes.

```
    pd->SetFieldData(fd);
    fd->Delete();
    }
  pd->GetFieldData()->AddReplaceArray(someArray, someName);
```

can now be replaced with

```
  pd->AddArray(someArray);
```

2.2 Attributes

In the old vtkDataSetAttributes, all attributes were contained as member variables:

```
// Old vtkDataSetAttributes
class VTK_EXPORT vtkDataSetAttributes : public vtkObject
{
public:
...
protected:
...
  // support manipulation and access of attribute data
  vtkScalars *Scalars;
  vtkVectors *Vectors;
  vtkNormals *Normals;
  vtkTCoords *TCoords;
  vtkTensors *Tensors;
...
}
```

This meant that data arrays were stored separately in two different places: in the FieldData member and inside each attribute member. Now, all the data arrays are stored as fields in vtkDataSetAttributes. An array of indices determines which arrays correspond to which attributes:

```
// New vtkDataSetAttributes
class VTK_EXPORT vtkDataSetAttributes : public vtkObject
{
public:
...
  // Always keep NUM_ATTRIBUTES as the last entry
  enum AttributeTypes {
    SCALARS=0,
    VECTORS=1,
    NORMALS=2,
    TCOORDS=3,
    TENSORS=4,
    NUM_ATTRIBUTES
  };
...
protected:
...
  int AttributeIndices[NUM_ATTRIBUTES]; //index to attribute array in
                                        //field data
...
}
```

There are now new methods to set/get attributes by passing vtkDataArray (instead of vtkAttributeData):

```
  void SetScalars(vtkDataArray* da);
  vtkDataArray* GetActiveScalars();
  void SetVectors(vtkDataArray* da);
  vtkDataArray* GetActiveVectors();
  void SetNormals(vtkDataArray* da);
  vtkDataArray* GetActiveNormals();
  void SetTCoords(vtkDataArray* da);
  vtkDataArray* GetActiveTCoords();
  void SetTensors(vtkDataArray* da);
  vtkDataArray* GetActiveTensors();
```

Since vtkAttributeData is actually a (relatively useless) shell around vtkDataArray, storing only the underlying data array is enough — specially after moving the lookup table into the data array, see section 1.1 — (see figures 3 and 4). However, to avoid breaking compatibility with the filter written in the old style (as of now, all of them, really), it is still possible to set/get attributes by passing in a vtkAttributeData. For this, an array of vtkAttributeData pointers is stored. If an attribute is requested with, for example GetScalars(), and there is no corresponding vtkAttributeData (because that attribute was set by passing a data array), one is created:

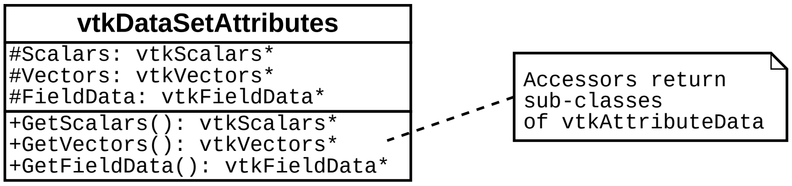
vtkDataSetAttributes
#Scalars: vtkScalars*
#Vectors: vtkVectors*
#FieldData: vtkFieldData*
+GetScalars(): vtkScalars*
+GetVectors(): vtkVectors*
+GetFieldData(): vtkFieldData*
Accessors return sub-classes of vtkAttributeData

Figure 3: before

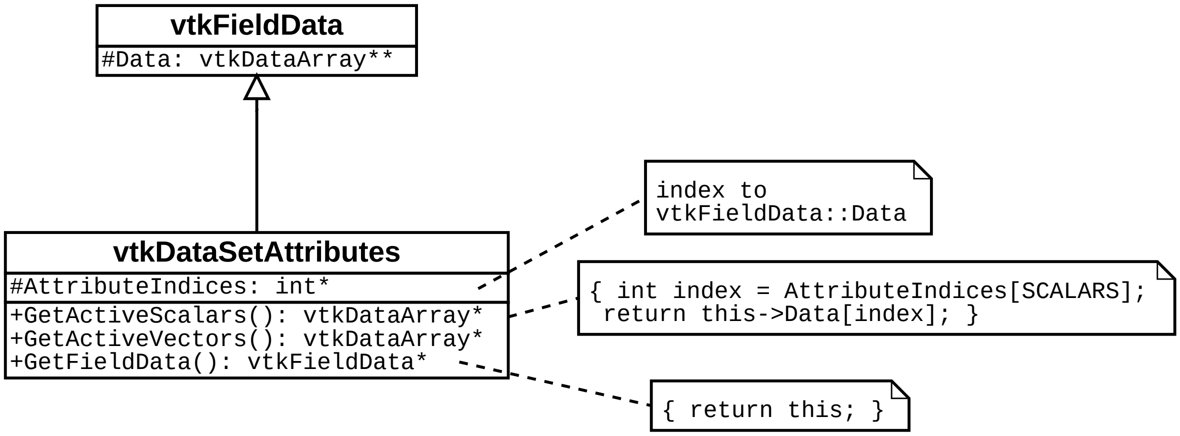
vtkFieldData
#Data: vtkDataArray**
vtkDataSetAttributes
#AttributeIndices: int*
+GetActiveScalars(): vtkDataArray*
+GetActiveVectors(): vtkDataArray*
+GetFieldData(): vtkFieldData*
index to vtkFieldData::Data
{ int index = AttributeIndices[SCALARS];
return this->Data[index]; }
{ return this; }

Figure 4: after

```
vtkScalars* vtkDataSetAttributes::GetScalars()
{
  vtkScalars* retVal=0;

  if (this->Attributes[SCALARS] || (this->AttributeIndices[SCALARS] == -1))
    {
    retVal = (vtkScalars*)this->GetAttributeData(SCALARS);
    }
  else
    {
    this->Attributes[SCALARS] = vtkScalars::New();
    this->Attributes[SCALARS]->SetData(this->GetActiveScalars());
    retVal = (vtkScalars*)this->Attributes[SCALARS];
    }

  return retVal;
}
```

If an attribute is set by passing a vtkAttributeData, that pointer is also stored. Hopefully, in time, all filter will use data arrays (or at least not use the SetAttribute(vtkAttributeData*) interface) and this additional complexity will be removed. I will start changing all vtk filters so that they do not use subclasses of vtkAttributeData unless they absolutely need to. Furthermore, all the functionality in vtkAttributeData subclasses which is needed will be moved to helper classes (for example ComputeMaxNorm() in vtkVectors).

2.3 Copying/Passing Data

vtkDataSetAttributes has helper functions to facilitate writing filter which have to pass (i.e. copy the pointer), copy (i.e. copy the data point by point (cell by cell) specifying source and destination point/cell ids) or interpolate data. It is possible to individually select which attributes to copy/pass. However, it was possible to individually select arrays (it was either all fields or none). Since all attributes are now stored as arrays, this is no longer enough. Therefore, I added functionality which allows users to individually select which arrays will be copied as well as attributes. Note that the decision about an attribute always overrides the decision about an array (i.e. if an attribute is not to be copied, the corresponding array is not copied whether it is flagged or not and, similary, if an attribute is to be copied, the corresponding array is copied whether it is flagged or not). The new methods are

```
  void CopyFieldOn(const char* name);
  void CopyFieldOff(const char* name);
```

(by default, all fields are passed/copied, as before).

3 Changes to filter

vtkFieldDataToAttributeDataFilter and vtkAttributeDataToFieldDataFilter will be deprecated in VTK 4.0. You should use one or more of vtkMergeFields, vtkSplitField, vtkAssignAttribute, vtkRearrangeFields instead.