Explicit Deformable Model in VTK

Release 0.00

Jérôme Velut

February 8, 2011

jerome.velut@gmail.com

Abstract

This document describes a set of classes[1] that design a generic explicit deformable model in VTK. The iterative mechanism is first introduced through an inheritance of the vtkPolyDataAlgorithm class. This vtkIterativePolyDataAlgorithm is then a based for an implementation of the deformation. Two examples of deformation is presented through an inheritage of this base class. The provided source code may be used to build a ParaView plugin that harnesses the animation feature.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3251]

Contents

[1] It is a subset of the vtkKinship library http://github.com/jeromevelut/vtkKinship

The deformable models are part of a wide family of computer graphics methods. They were initially proposed in the context of real physics simulation, in particular for solid and soft body deformations [3, 4]. One of the applications concerns the segmentation task in image processing: The snakes [1] are often cited as the seminal work that induced the huge amount of existing paper today [2]. The main idea was to minimize an energy computed under a curve evolving in an image domain. A proposed implementation was a discretization of the steepest descent algorithm, which led to an iterative displacement of each point related to a trade-off between internal and external forces. The explicit deformable models refer to this iterative deformation of the geometry, as opposed to implicit deformable models [].

In this paper, we focus on the iterative design of an explicit deformable model. First, we present the `vtkIterativePolyDataAlgorithm` class, that inherits from `vtkPolyDataAlgorithm`. It allows a VTK filter to iteratively process the input until a number of iteration is reached. A step-by-step mechanism is also implemented, bringing useful interactive capabilities -especially inside ParaView-. Second, two specialisations are presented: a brownian movement and a deformable model, designed through two classes derived from `vtkIterativePolyDataAlgorithm`. Finally, a ParaView plugin is provided and a short example is given in which the animation feature is used in order to interact with the deformable model.

1 Making PolyData algorithms iterative

1.1 basics

The design of the VTK library is based on a pipeline of data processing. The modification of an input anywhere in the pipeline triggers the execution of the whole remaining, downward processes. An obvious consequence is the impossibility to plug the output of a filter directly to an upward input 'as is': this feedback connection will create in many cases an infinite loop. However, it is possible to do almost everything in the execution method, as far as the inputs are not modified. The first element of the deformable model suit is a `vtkPolyDataAlgorithm`-inherited class, namely `vtkIterativePolyDataAlgorithm`. It implements a caching procedure and a special iteration mechanism.

1.2 Caching the input

In order to avoid a modification of the input, a call to `vtkIterativePolyDataAlgorithm::RequestData` copy the first connection of the first input port into a CachedInput member under certain condition:

- this is the first iteration (i.e. `CurrentIteration` is false)
- the filter is asked to compute all the iteration at each Update (i.e. `IterateFromZero` is true)
- Maximum number of iterations (variable `NumberOfIterations`) has been set to zero

The copy is a performed by:

```
// Copy input
this->CachedInput->DeepCopy( inputMesh );
// Reset current iteration
this->CurrentIteration = 0;
// User define initial condition
this->Reset( inputVector );
```

Here, `inputMesh` points to the first input connection of the first port. This copy implies that the `CurrentIteration` is 0. Child classes should overwrite the `Reset` function in order to implement special initial states.

1.3 Iterating

While `CurrentIteration` is less than `NumberOfIterations`, the classical iterative process is performed :

- transform the `CachedInput` and name it `IterativeOutput`
- Increment the `CurrentIteration`
- copy the `IterativeOutput` over `CachedInput`

The manipulation of `CachedInput` is implemented in the specialisations of `vtkIterativePolyDataAlgorithm::IterativeRequestData`:

```
while( this->CurrentIteration < this->NumberOfIterations )
{
   // Effective call to the iterative algorithm. Child classes
   // should override this function
   this->IterativeRequestData( inputVector );

   this->CachedInput->DeepCopy( this->IterativeOutput );
   this->CurrentIteration ++;
}
```

Two important points are illustrated in these lines of code. First, it is the responsability of the `IterativeRequestData` function to set the `IterativeOutput` data. Second, the algorithm will iterate from `CurrentIteration` to `NumberOfIterations`. It means that the whole iterations are not necessarily computed, unless it is explicitely asked by setting `IterateFromZero` to 1. It is useful for a dynamic visualisation of an iterative process.

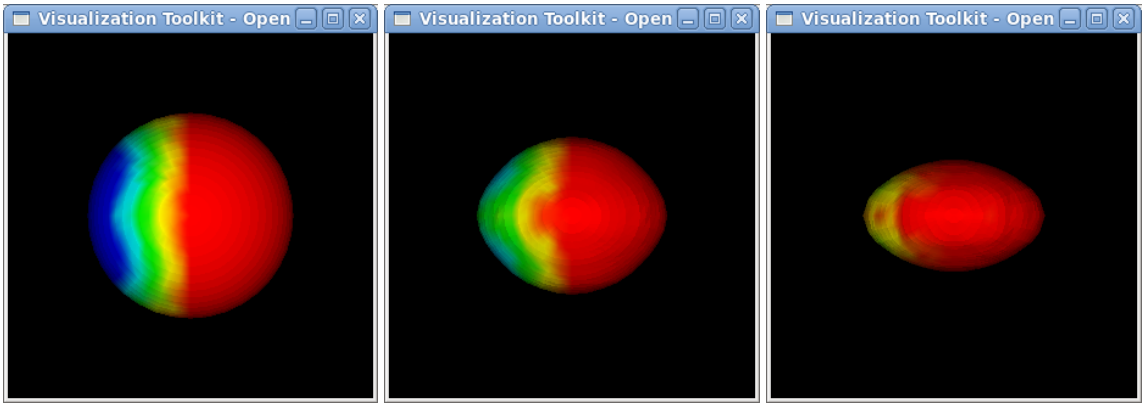
The final step, in a VTK point of view, is to copy the iterative output over the real output of the filter:

```
vtkInformation *outMeshInfo = outputVector->GetInformationObject(0);
vtkPolyData* outputMesh = vtkPolyData::SafeDownCast(
                          outMeshInfo->Get(vtkDataObject::DATA_OBJECT()));

if( this->NumberOfIterations == 0 )
   outputMesh->DeepCopy( inputMesh );
else
   outputMesh->DeepCopy( this->IterativeOutput );
```

In the case where `IterateFromZero` is 1, the iterative process could be resetted by setting `NumberOfIterations` to 0:

- the initial input cache is performed
- the output of the filter is set to inputMesh



(a) $i = 0$ (b) $i = 100$ (c) $i = 250$

Figure 1: Deformation of a spherical mesh towards an ellipsoid image at different iterations i

2 Explicit deformable model

2.1 Dynamic point warping

A simple implementation of the deformation of a point set is realized in `vtkPolyDataIterativeWarp`. An internal pipeline made of brownian vectors and point warping is built. The input of this pipeline is `CachedInput` and the `IterativeOutput` is set to be the output of the warp filter.

A basic illustration may be found in `Examples/IterativeWarp.cxx`. A point cloud `vtkPointSource` is passed through the `vtkIterativePolyDataAlgorithm` and an infinite loop increments the number of iterations before rendering the output. The example may run for a while, as it doesn't iterate from zero at each update of the filter.

2.2 Deformable mesh

This iterative point warping is easily extended to an explicit deformable mesh framework. Let the input be a polygonal mesh; let the warp vectors being computed from a second input volume: If the warp vectors point to a minimum of an energy, then each point of the mesh will convergence towards these minima.

The class `vtkDeformableMesh` implements such a behaviour. The main difference between this one and `vtkPolyDataIterativeWarp` is that the warp vectors are not randomly generated. They are expected as a vector array inside a `vtkImageData` given on input port 1.

Although the input mesh may include topological information, those are not taken into account: Each point are moved independently from each other. It means that it is not possible to introduce internal forces in the deformation process in this class. Another specialisation has to be implemented.

The example `Examples/DeformableMesh.cxx` shows this deformable model evolving in a volume of size 64^3, representing an ellipsoid of radius $(18, 14, 28)$ and centered in the volume. The external forces are computed in a classical fashion. Let I be an intensity image, $\mathcal{S}$ the 3D Sobel operator. The second input of the `vtkDeformableMesh` is then $\mathcal{S}(|\mathcal{S}(I)|)$. The input is an UV-sphere (`vtkSphereSource`). Executing the example shows the sphere deforming to an ellipsoid (figure 1).

3 Paraview plugin

The presented VTK classes are optionnally wrapped into a ParaView plugin. If the option "BUILD_PARAVIEW_PLUGINS" has been set to true during the CMake procedure, then the compilation will output a `vtkPVExplicitDeformableModel` dynamic library. This library is loadable in ParaView:

```
"Tools" -> "Manage Plugins" -> "Load New ..."
```

A new submenu `Deformable model` appears in the `Filters` menu, with both `Iterative Warp` and `Deformable Mesh`.

3.1 Basic pipeline

The ParaView filters are directly linked to the VTK classes. The parameters have then the same meaning. `Deformable Mesh` is a multiple input filter: First, the input mesh has to be selected and put as input of `Deformable Mesh`. Then, a dialog box will appear to ask for a second input containing Vectors.

When triggering the `Apply` button, the iterative polydata filter will iterate internally until `NumberOfIterations`.

3.2 Rendering of successive iterations

By harnessing the Animation View, it is possible to litteraly see the input mesh or point cloud deforming, while keeping the usual interactions on the render window available. The screenshot in figure 2 shows the settings of the animation view. The option `IterateFromZero` is important here, as it will avoid the computation of the whole iterations at each timestep.

4 Software Requirements

This software has been successfully tested with ParaView-3.8.1 and ParaView-3.9 development branch on Linux Fedora 13.

5 Conclusion

We presented a set of classes that design the iterative deformation of a point set. A generic class, inherited from `vtkPolyDataAlgorithm` is derived to specialise the deformation process. These deformable models are available in ParaView for interactive visualization of the deformation.

Future works will focus on the regularization of deformable meshes for a segmentation point of view, and the generalisation of input data structures.

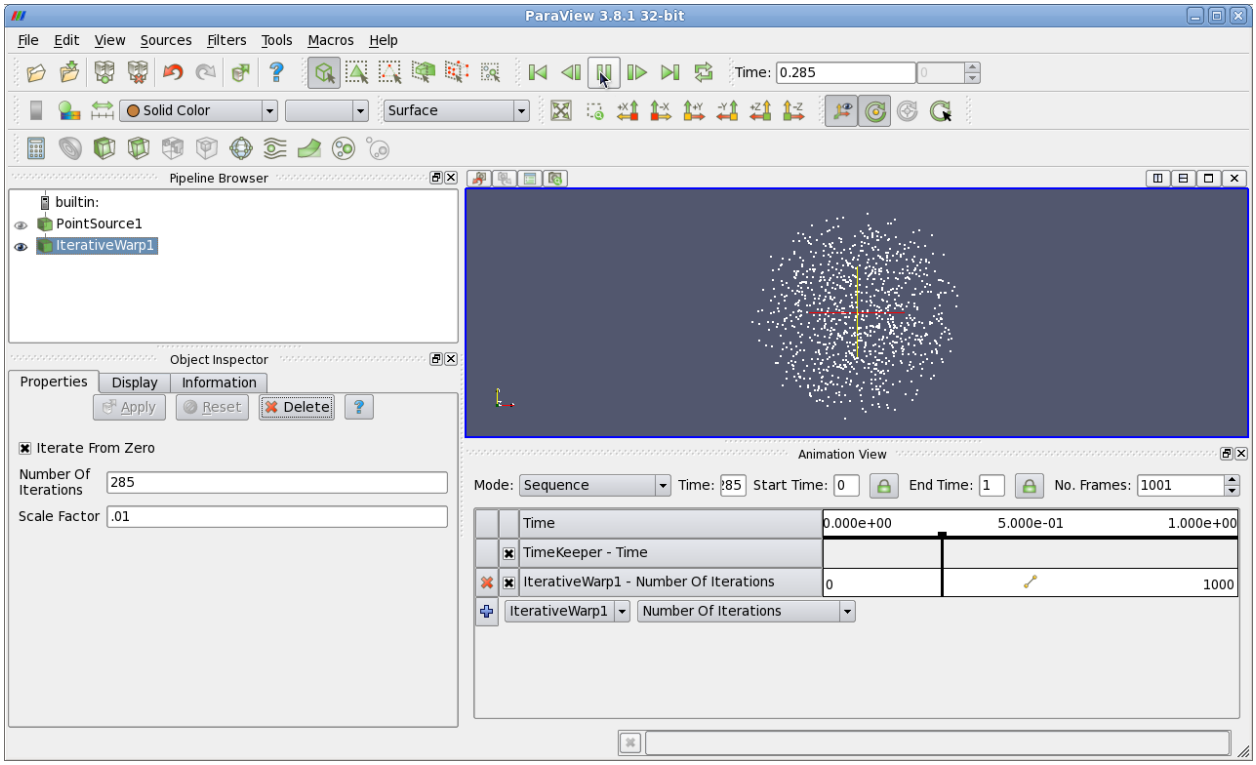


Figure 2: Setting of the animation view to see the iterative deformation of the input PolyData

References

[1] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. In *ICCV*, pages 259–268, London, Engl, 1987. IEEE, New York, NY, USA. (document)

[2] Johan Montagnat, H. Delingette, and N. Ayache. A review of deformable surfaces : Topology, geometry and deformation. *Image and Vision Computing.*, 19:1023–1040, 2001. Image and Vision Computing. (document)

[3] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH*, pages 205–214, New York, NY, USA, 1987. ACM Press. (document)

[4] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, 8(6):41 – 51, 1988. (document)