TECHNICAL REPORT
VTKMODELING, AN EXAMPLE OF EXTENDING VTK

Shaoting Zhang, Xiaoxu Wang and Dimitris Metaxas

The Center for Computational Biomedicine Imaging and Modeling (CBIM)
Rutgers University

ABSTRACT

This report introduces vtkModeling, a new toolkit of deformable models. It is created by extending subclasses of vtkAlgorithm in Visualization Toolkit 5.0 and can be wrapped with Python. Four deformation algorithms are integrated into vtkModeling: Laplacian Surface Editing and Optimization, As-Rigid-As Possible deformation, Mass Spring System and Meshless method. The whole extending procedure as well as several algorithms are presented in this report. All codes and documents can be downloaded from Sourceforge.net.

Index Terms— Visualization Toolkit, vtkModeling, CMake, Python, Laplacian Surface Editing, Optimization, Meshless

1. INTRODUCTION

Deformable models are significant in many fields like computer graphics, biomedical imaging and computer vision. Many open source codes are provided for popular deformation algorithms, such as Finite Element Method and Free Form Deformation. But they may not be easy to use for cross-platform and cross-language purpose. On the other hand, many novel deformation algorithms have been advanced in the past decade in computer graphics community. Some of them may not be well-known in biomedical and vision fields. Thus we decide to develop a toolkit which consists of novel deformation algorithms and is easy to use by different programming languages.

In this report a new modeling toolkit, vtkModeling, is introduced. Four deformation algorithms are integrated so far: Laplacian Surface Editing [1] and Optimization [2], Deformation with Moving Least Square [3], Meshless method [4] and Mass Spring System. The toolkit is created by extending classes in Visualization Toolkit 5.0. It is cross-platform, cross-language. A matrix solver is also provided.

Section 2 introduces related toolkits. Section 3 shows the whole procedure to create new algorithms by extending Visualization Toolkit. Section 4 briefly presents several deformation algorithms and numerical methods in vtkModeling. Section 5 discusses future work.

2. RELATED TOOLKITS

2.1. Visualization Toolkit (VTK)

The Visualization Toolkit (VTK) is an open source graphics toolkit [5]. It is a platform independent graphics engine with parallel rendering support. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. Table 1 shows basic class types in VTK. Every VTK program is based on a pipeline: a data source at the beginning, a renderer at the end, in between can be multiple algorithms. These algorithms can be extended and thus new algorithms can be created and integrated into the VTK pipeline. Section 3 will discuss procedures to extend VTK algorithms.

Table 1. Class types a VTK programmer will encounter

Class Types	Examples
Containers	vtkDataObject
Algorithms	vtkAlgorithm
Actors	vtkProp
Interactors	vtkRenderWindowInteractor
Renderers	vtkRenderer
Arrays	vtkDataArray
Commands	vtkCommand

2.2. CMake

CMake is a family of tools designed to build, test and package software [6]. It is used to control the software compilation process using simple platform and compiler independent configuration files. It generates native makefiles and workspaces that can be used in any compiler environment. Combining CMake and VTK, we can generate wrappers for Python, Java and TCL/TK.

To use CMake, we need to create CMakeLists files to describe the project. One file for each directory. A typical CMakeLists file contains list of all source files, compile targets, include directories and used libraries. A simple CMakeLists example can be found at [7].

2.3. vtkModeling

vtkModeling [8] is an extension of Visualization ToolKit 5.0. Basically it is a collection of deformation algorithms, like Laplacian Surface Editing and Optimization, Moving Least Square deformation, Mass Spring System and Meshless Method. Section 3 will analyze this project and discuss how to create new VTK algorithms. Section 4 will briefly introduce several algorithms in vtkModeling.

3. EXTENDING VTK

Four approaches can be employed to extend VTK, which is showed in table 2. VtkModeling is created by extending subclasses of vtkAlgorithm, which is the first method in table 2 and will be discussed in section 3.1.

Table 2. Four approaches to extend VTK

Problem	Extension Point
New algorithm	Subclass a subclass of vtkAlgorithm
New data type	Subclass vtkDataObject
New interaction	vtkInteractorStyleUser
New callback	Subclass vtkCommand

3.1. Create New Algorithms

In this section we will introduce how to create new VTK algorithms and we will use vtkLaplacianSurface class in vtkModeling as an example.

Firstly we should figure out the proper input and output for the new algorithm. In vtkLaplacianSurface, vtkPolyData is preferable since the connectivity information is required. Secondly we need to extend proper subclass of vtkAlgorithm according to the input and output, like vtkPolyDataAlgorithm for laplacian surface editing. We can always change the data type of input and output by extra codes, but sticking to correct subclass can save time. After collecting these information, we can write codes based on the skeleton one. Here are the skeleton codes of vtkLaplacianSurface.

Table 3. vtkLaplacianSurface.h

```
#include <vtkPolyDataAlgorithm.h>
class vtkPolyData;

class VTK_EXPORT vtkLaplacianSurface : public vtkPolyDataAlgorithm {
public:
    static vtkLaplacianSurface* New();
    vtkTypeMacro(vtkLaplacianSurface,vtkPolyDataAlgorithm);
    void PrintSelf(ostream& os, vtkIndent indent);
    //Get parameters for lse. This declaration is enough, no definition needed.
    vtkSetObjectMacro(ControlIds, vtkIdList);      // Input, control points' id
    vtkSetObjectMacro(ControlPoints, vtkPoints); // Input, control points' coordinates
protected:
    vtkLaplacianSurface();
    ~vtkLaplacianSurface();
    int RequestData(vtkInformation *, vtkInformationVector **, vtkInformationVector *);
    int FillInputPortInformation(int, vtkInformation*);
    //data
    vtkIdList* ControlIds;    // control points' id
    vtkPoints* ControlPoints;// control points' coordinates
private:
    vtkLaplacianSurface(const vtkLaplacianSurface&);// Not implemented.
    void operator=(const vtkLaplacianSurface&);// Not implemented.
};
```

Table 4. RequestData method in vtkLaplacianSurface.cpp

```
int vtkLaplacianSurface::RequestData(
    vtkInformation *vtkNotUsed(request),
    vtkInformationVector **inputVector,
    vtkInformationVector *outputVector) {
    // get the info objects
    vtkInformation *inInfo = inputVector[0]->GetInformationObject(0);
    vtkInformation *outInfo = outputVector->GetInformationObject(0);
    // get the input and ouptut
    vtkPolyData *input = vtkPolyData::SafeDownCast(
                                inInfo->Get(vtkDataObject::DATA_OBJECT()));
    vtkPolyData *output = vtkPolyData::SafeDownCast(
                                 outInfo->Get(vtkDataObject::DATA_OBJECT()));
    doLSE(input, output);
    return 1;
}
```

$vtkSetObjectMacro$ is employed to get object-type parameters. To get primitive-type parameters we can use $vtkSetMacro$. $RequestData$ is the most important method in this class. It handles input and output data. $input$ holds a pointer to the input dataset. It cannot be modified directly, but we can always make a $deepcopy$ and deal with that copy one. $output$ holds a pointer to the output dataset of this filter, which may be used by other filter later. It is empty in the beginning. Dataset can be filled in at anytime. Laplacian Surface algorithms can be implemented in $doLSE$ method with parameters $input$ and $output$.

3.2. Write CMakeLists

In section 3.1 we create a new algorithm by extending subclass of vtkAlgorithm. Now we need to write a CMakeLists to maintain project conveniently. Table 5 is an example for vtkLaplacianSurface class.

Two things are worth to mention. Firstly $WRAP\_EXCLUDE$ 1 means that these files will not be wrapped for script languages like python. These files may not be satisfy with warping criterion, besides, they just like private methods. They can be invoked by other C++ files, like $vtkLaplacianSurface.cpp$ in this case. Secondly $ADD\_EXECUTABLE$ will generate a new C++ project. Here a test project will be generated. It invokes method in vtkLaplacianSurface for debugging purpose.

3.3. Python Bindings

Using CMake and VTK Wrap mechanisms, VTK and vtkModeling can be invoked by script languages like Python [9]. Python is a dynamic object-oriented programming language that can be used for many kinds of software development. We can use C++ to write complicated algorithms, and use Python to invoke them and develop projects. Thus bindings can combine fast execution with rapid development. The trade-off is that not all codes can be recognized by the VTK wrapper, so we need

Table 5. CMakeLists for vtkLaplacianSurface

```
SET (LIBRARY_NAME
    lse_vtk
)
SET (SOURCES
    vtkLaplacianSurface.cpp
    MathTools.cpp
)
SET_SOURCE_FILES_PROPERTIES(
    MathTools.cpp
  PROPERTIES
    WRAP_EXCLUDE 1
)
SET (WRAPPED_LIBRARIES
    vtkCommon
    vtkFiltering
    vtkVolumeRendering
)
SET (LIBRARIES
    ${WRAPPED_LIBRARIES}
)
SET (INCLUDE_DIRS
    ${PROJECT_SOURCE_DIR}/${LIBRARY_NAME}
    ${PROJECT_SOURCE_DIR}/include
)
FIND_PACKAGE(VTK REQUIRED)
INCLUDE(${VTK_USE_FILE})
ADD_EXECUTABLE(test_LS test_LS.cpp)
TARGET_LINK_LIBRARIES(test_LS vtkRendering lse_vtk)
INCLUDE(../CMakeCommon.txt)
```

to avoid bad interface components such as templates, enumerators, bool and pointers to non VTK objects. $WRAP\_EXCLUDE$ 1 mentioned in 3.2 can make Python wrapper to ignore some classes with bad interfaces.

So far we create a new algorithm and it can be compiled and invoked cross-platform and cross-language.

4. ALGORITHMS IN VTKMODELING

In this section we will briefly introduce key ideas of several algorithms in vtkModeling. Detail information can be found in references.

4.1. Laplacian Surface Optimization

Laplacian surface is also called differential coordinates. It represents each point as the difference between such point and its neighborhoods. Laplacian Surface Optimization is an algorithm to improve triangle quality of a surface mesh. The inputs are anchor points and an initial surface mesh. The output is an optimized surface mesh.

Here we introduce notations and derive this algorithm. Let the mesh $\mathbb{M}$ be described by a pair $(\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = \{v_1, ..., v_n\}$ describes the geometric positions of the vertices in $\mathbb{R}^3$ and $\mathbb{E}$ describes the connectivity. The neighborhood ring of a vertex i is the set of adjacent vertices $\mathbb{N}_i = \{j|(i,j) \in \mathbb{E}\}$ and the degree d_i of this vertex is the number of elements in $\mathbb{N}_i$. We assume that the mesh is connected. Instead of using absolute coordinates $\mathbb{V}$, the mesh geometry is described as a set of differentials $\Delta = \{\delta_i\}$. Specifically, coordinate i will be represented by the difference between v_i and the average of its neighbors. A uniform weights definition is:

$$\delta_i = v_i - \frac{1}{d_i} \sum_{j \in \mathbb{N}_i} v_j \tag{1}$$

Assume V is the matrix representation of $\mathbb{V}$. The transformation between vertex coordinates V and Laplacian coordinates Δ can be described in matrix algebra. Let N be the mesh adjacency (neighborhood) matrix and $D = diag(d_1, ..., d_n)$ be the degree matrix. Then $\Delta = LV$, where $L = I - D^{-1}N$ for the uniform weights.

Using a small subset $\mathbb{A} \subset \mathbb{V}$ of m anchor points, a mesh can be reconstructed from connectivity information alone [1]. x, y and z positions of reconstructed object ($V'_p = [v'_{1p}, ..., v'_{np}]^T, p \in \{x, y, z\}$) can be solved separately by minimizing the quadratic energy

$$\|LV'_p\|^2 + \sum_{a \in \mathbb{A}} \|v'_{ap} - v_{ap}\|^2 \tag{2}$$

where the v_{ap} are anchor points. The first half try to smooth the object by minimizing the difference, and the second half keeps anchor points unchanged. In practice, with m anchors, the $(n+m) \times n$ overdetermined linear system $AV'_p = b$

$$\begin{bmatrix} L \\ I_{ap} \end{bmatrix} V'_p = \begin{bmatrix} 0 \\ V_{ap} \end{bmatrix} \tag{3}$$

is solved in the least squares sense using the method of normal equations $V'_p = (A^TA)^{-1}A^Tb$. The first n rows of $AV'_p = b$ are the Laplacian constraints, corresponding to $\|LV'_p\|^2$, while the last m rows are the positional constraints, corresponding to $\sum_{a \in A} \|v'_{ap} - v_{ap}\|^2$. I_{ap} is the index matrix of V_{ap}, which maps each V'_{ap} to V_{ap}. The reconstructed shape is generally smooth, with the possible exception of small areas around anchor vertices. The minimization procedure moves each vertex to the centroid of its 1-ring, since the uniform Laplacian L is used, resulting in good inner fairness.

The main computation cost of this algorithm is big matrix multiplication and inverse. Since A is sparse matrix, A^TA is sparse symmetric definite matrix. Conjugate Gradient [10] can be employed to solve the system.

4.2. Laplacian Surface Editing

Laplacian Surface Editing, which is an algorithm for local deformation. The inputs are deformed control points and an initial mesh. In our specific case, control points are model and image landmarks. Model landmarks are moved to image landmarks directly. The deformation of rest points can be calculated by Laplacian Surface Deformation. Note that after Global Deformation process, the displacements of control points are restricted in a local range. The output is the deformed mesh.

Using the same notation as Section 4.1, this time we need to minimize this quadratic error function:

$$E(V') = \sum_{i=1}^{n} \|\delta_i - \delta_i'\|^2 + \sum_{i \in \mathbb{C}} \|v_i - v_i'\|^2 \tag{4}$$

where δ' and v' is Laplacian and Cartesian coordinates after deformation. $\mathbb{C}$ is the set of control points. The first half try to keep the shape according to previous time step, which is δ. The second half can move control points v to deformed positions v'. However, using function 4, no point will be moved except control points $\mathbb{C}$. The main idea of Laplacian Surface Editing is to compute an appropriate transformation T_i for each vertex i which can be plugged into error function 4:

$$E(V') = \sum_{i=1}^{n} \|T_i\delta_i - \delta_i'\|^2 + \sum_{i \in \mathbb{C}} \|v_i - v_i'\|^2 \tag{5}$$

However, if T_i is unconstrained, the natural minimizer for function 5 is a membrane solution, and all geometric detail is lost. Thus, T_i needs to be constrained in a reasonable way. T_i should include rotations, isotropic scales, and translations. In particular, anisotropic scales should not be allowed, as they allow removing the normal component from Laplacian coordinates. Particularly the class of matrices representing isotropic scales and rotation can be written as:

$$T_i = \begin{bmatrix} s & -h_3 & h_2 & t_x \\ h_3 & s & -h_1 & t_y \\ -h_2 & h_1 & s & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

where $(s_i, h_i, t_i)^T$ can be solved by minimize:

$$\|A_i(s_i, h_i, t_i)^T - b_i\| \tag{7}$$

where A_i contains the positions of v_i and its neighbors and b_i contains the position of v_i' and its neighbors. The structure of $(s_i, h_i, t_i)^T$ yields:

$$A_i = \begin{bmatrix} v_{kx} & 0 & v_{kz} & -v_{ky} & 1 & 0 & 0 \\ v_{ky} & -v_{kz} & 0 & v_{kx} & 0 & 1 & 0 \\ v_{kz} & v_{ky} & -v_{kx} & 0 & 0 & 0 & 1 \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \end{bmatrix}, k \in \{i\} \cup \mathbb{N}_i \tag{8}$$

and

$$b_i = \begin{bmatrix} v'_{kx} \\ v'_{ky} \\ v'_{kz} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}, k \in \{i\} \cup \mathbb{N}_i \tag{9}$$

The linear least-squares problem above is solved by:

$$(s_i, h_i, t_i)^T = (A_i^T A_i)^{-1} A_i^T b_i \tag{10}$$

The function 5 can be minimized iteratively by finding T_i and apply it on each vertex coordinates. When v converge, this error minimization problem is solved. Transformation T_i is an approximation of the isotropic scaling and rotations when the rotation angle is small. The main computation cost of this algorithm is function 10 for each point, which is fast since A_i can be obtained offline.

4.3. Meshless Method

This approach to deformable modeling is greatly inspired by so-called mesh free or meshless methods for the solution of partial differential equations, which originated in the FEM community approximately a decade ago. Using spatial hashing, neighborhoods of each point can be calculated without connectivity information. After obtaining proper neighborhoods, meshless continuum mechanics [4] or meshless Laplacian Volume Editing [11] can be applied to calculate deformation.

4.4. Matrix Calculation

Matrix TCL Lite [12] is employed to do matrix calculation. It is easy to use and distribute since it only has one header file and works like Matlab in some way. But it is really slow for big matrix operations. We applied these algorithms on clinical data with around 3,000 vertices. It takes hours for Laplacian Surface Optimization and more than ten minutes for Editing. The Original Matrix TCL is for dense matrix only, without any optimization for sparse one. Thus we added several functions for sparse matrix calculation. Conjugate Gradient [10] is used to solve $Ax = b$, where A is a sparse symmetric definite matrix. We can always change $Ax = b$ to $A^T Ax = A^T b$, where A is sparse and $A^T A$ will be sparse symmetric definite. We put the method in Matrix TCL, named as SSDSolver. To calculate sparse matrix multiplication $A^T A$, firstly we compress the dense matrix to link list, then perform multiplication. After that the link list will be restored to dense matrix format. The algorithms is named as SparseMul. The running time decreases from hours to seconds.

5. FUTURE WORK

More deformation algorithms will be implemented, like Green Coordinates [13], Harmonic Coordinates [14]. We will use this toolkit to solve biomedical and vision problems.

6. REFERENCES

[1] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rossl, and Hans-Peter Seidel, "Laplacian surface editing," in *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 2004, pp. 179–188.

[2] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa, "Laplacian mesh optimization," in *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, New York, NY, USA, 2006, pp. 381–389, ACM.

[3] Scott Schaefer, Travis McPhail, and Joe Warren, "Image deformation using moving least squares," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 533–540, 2006.

[4] M. Mller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa, "Point based animation of elastic, plastic and melting objects abstract," .

[5] "http://www.vtk.org," .

[6] "http://www.cmake.org," .

[7] "http://www.cmake.org/cmake/help/examples.html," .

[8] "http://sourceforge.net/projects/vtkextend/," .

[9] "http://www.python.org," .

[10] Jonathan R Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Tech. Rep.

[11] Xiaoxu Wang, Ting Chen, Shaoting Zhang, Dimitris Metaxas, and Leon Axel, "Lv motion and strain computation from tmri based on meshless deformable models," *the 11th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI2008)*, pp. 636–644, 2008.

[12] "http://www.techsoftpl.com/matrix/matlite.htm," .

[13] Yaron Lipman, David Levin, and Daniel Cohen-Or, "Green coordinates," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1–10, 2008.

[14] Tony DeRose and Mark Meyer, "Harmonic coordinates," Tech. Rep.