An Object-oriented Framework for Spatial Interpolation

Yo-Ming Hsieh[1], and Mao-Sen Pan[2]

[1]Department of Construction Engineering, National Taiwan University of Science and Technology, No. 43, Sec. 4, Keelung Rd., Taipei, Taiwan; PH(886) 2-2730-1056; FAX(886) 2-2737-6606;
e-mail: ymhsieh@mail.ntust.edu.tw
[2]Department of Construction Engineering, National Taiwan University of Science and Technology, No. 43, Sec. 4, Keelung Rd., Taipei, Taiwan; e-mail: D9505503@mail.ntust.edu.tw

Abstract

Interpolation is an important operator in numerical methods for solving partial differential equations and in geospatial applications. There are many interpolation methods proposed in the past. In this work, a unified software framework is proposed through the use of design-patterns in object-oriented programming. By using this framework, little effort is necessary to implement different interpolations algorithms when commonality with implemented algorithms can be found. Furthermore, through this framework, it becomes easy to compare the performance of different algorithms because of the unified application interface..

1. Introduction

Spatial interpolation has wide applications in the field civil engineering, e.g. solving PDE (partial differential equations) and GIS (geographical information systems). Currently, there is neither standard nor widely adopted API (application programming interface) for interpolation. As a result, it is necessary for developers of aforementioned applications to implement their own interpolation operators or adapt their applications to some developed codes. If ten interpolation methods is to be evaluated for understanding its applicability or its performance, the developer suffers because they need to read 10 different documents, they may have different data interfaces, and they may even conflict in names. Therefore, there is a dare need for a standard interface for performing interpolation.

A software framework for interpolation is proposed in this work to unify implementations of different interpolation algorithms. The unification is made possible by the use of encapsulation, inheritance and *polymorphism* characteristics of OOP (object-oriented programming). The design is guided by design patterns, a concept pioneered by Gamma(1991) to achieve low coupling between different classes.

It is believed the proposed software framework can potentially benefit developers of both 1) new interpolation algorithms by encouraging code reuse; and 2) applications in need of interpolation by having a unified interface to multiple interpolation algorithms.

2. Methodology

This study follows the flowchart shown in Figure 1. It is seen in the flowchart that this study consists the following steps: survey, common feature extraction, core class design, system interface design, and evaluation.

First step of this study was a survey of interpolation methods interested. In this survey, few methods for interpolation commonly used in solving PDE were included. These methods were selected solely because these were needed by other studies in author's research group, but authors believed the generality of the proposed software framework should not be affected much by the limited scope of the survey.

Common procedures and requirements were then analyzed and extracted from the surveyed interpolations algorithms. This step is necessary in order to identify necessary common function call interfaces and common data interfaces. This step contributes to the design of a general software framework that unifies the implementation of interpolation algorithms.

From the identified common requirements between interpolation algorithms, abstract core classes were then designed. The abstract core classes define the common interface, including both data interface and API interface, for implementing interpolation algorithms. The data interface was carefully designed with great

generality to ensure it satisfies all the needs of surveyed algorithms; the API interface or class collaboration were designed with aids from design patterns to lower the coupling between classes in order to maximize extensibility while minimizing changes needed when adding new algorithms.

In order to make the framework for interpolation easy to use for application developers, a wrapper class or system interface class is then designed to wrap up the above mentioned core classes to provide a simple to use application interface for application developers. More details about the system interface class are given at later paragraphs.

Once all classes were designed, the designed classes were validated and evaluated by implementing surveyed algorithms in order to make sure these classes: 1) can implement all the surveyed methods, and 2) encourage good code-reuse.

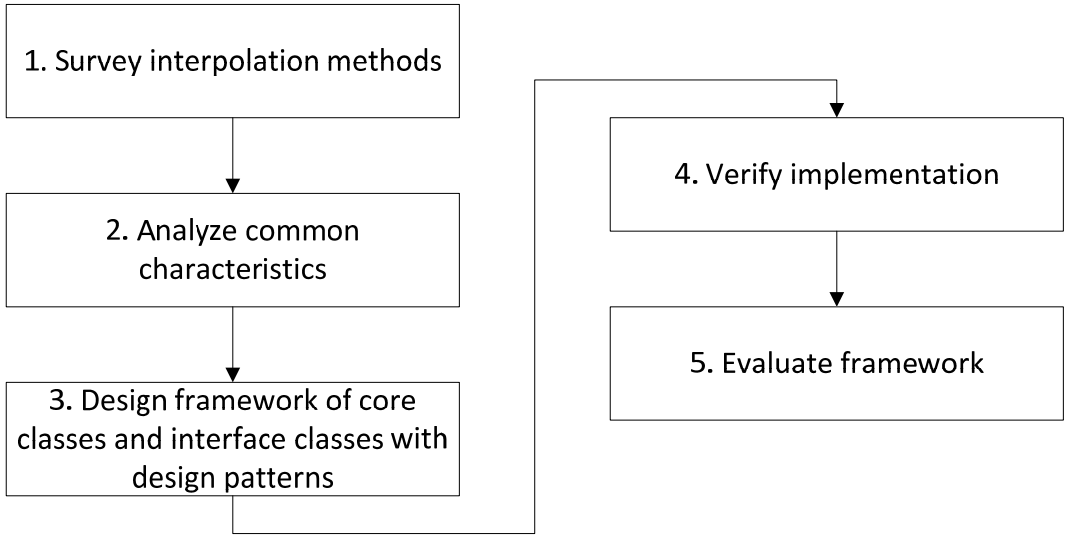


Figure 1: The overall flowchart of this study

3. Core Interpolation Framework

Before the proposed software framework is described, interpolation is first defined. In this work, spatial interpolation is defined as an evaluation of a quantity of interest at a given spatial location based on the quantities of the same type that exists on a set of known points.

Based on surveyed interpolation algorithms, it was determined the evaluation of one interpolation operation often involves evaluations of: 1) weighting, which controls the how influential each known point is to the point of interest; 2) basis function, which controls how the quantity to be evaluated is distributed continuous in space; and 3) neighboring points, which finds surrounding points for a given point.

Accordingly, four abstract classes were designed to define interfaces for the aforementioned four evaluations. These classes need to be derived or "sub-classed" to define concrete implementation of these evaluations. The collaboration of these classes is shown in Figure 2, and is discussed in the following paragraphs.

The *interpolation* class is the main "driving" class for performing interpolation. Applications in need of interpolation will create a concrete object derived from this type. This interface class defines the common interface for initializing data required to perform interpolation and performing interpolation for all interpolation algorithms. Figure 2 shows an example of implementing *MLS* (moving least square, Lancaster and Salkauskas 1981) interpolation algorithm. The *MLS* class is a concrete class inherited from *interpolation* class, and the concrete class overrides data initialization method and provides an implementation of MLS interpolation algorithm. In a way, the interpolation class is similar to a template class that defines the collaboration between *weightFunction*, *basisFunction*, and *searchAlgorithm* abstract classes. The class itself does not assume or use any concrete class. Therefore, this design ensures great extensibility or flexible for implementing interpolation algorithms. Furthermore, if a particular interpolation method does not use *weightFunction*, *basisFunction*, or *searchAlgorithm* classes, the implementer can simply ignore references to the aforementioned three abstract classes and use only the defined interfaces by *interpolation* class.

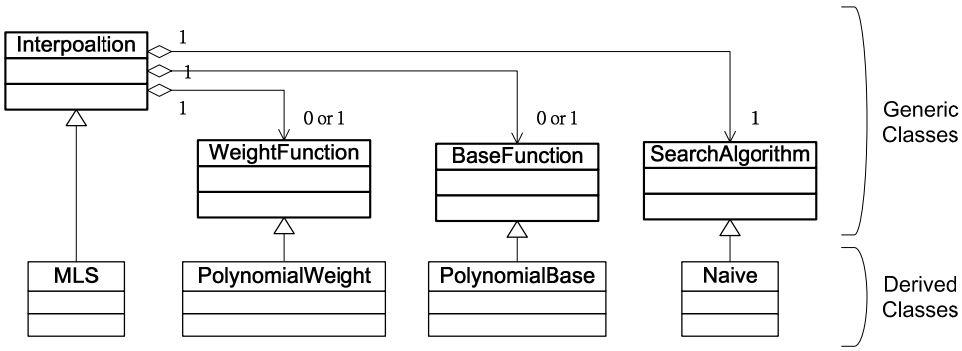


Figure 2: Class collaboration for performing interpolation

The *weightFunction* class, similar to the *interpolation* class, is an abstract class that defines universal interface for determining weights of existing points with known quantities to the point to be interpolated. There were many weighting functions proposed in the past, e.g. Liu(2003), Li et al(2004). If a weighting function is to be implemented based on the proposed framework, they need to sub-class the abstract *weightFunction* class to ensure that they can be easily incorporated or "bridged" into any interpolation algorithm.

The *basisFunction* class defines the common interface for basis functions (also known as interpolation function, shape function, kernel function, etc.), such as polynomials and radial basis functions (Liu 2003; Liu and Gu 1999; 2001; 2005; Wang and Liu 2000). Any implementation of basis function evaluation needs to sub-class from *basisFunction* class and implements the virtual functions for calculating local interpolation.

Finally, the *searchAlgorithm* class is a helper abstract class for searching a specified number of neighboring points from a given set of points. It is considered as a helper class because interpolation algorithms usually do not concern how to find neighboring points, but most interpolation algorithms do need to find surrounding points for a given spatial location. It should be noted the efficiency of search algorithm tends to dictate the interpolation efficiency, and efficient search algorithms is necessary in order to get performing interpolations.

Two design patterns, bridge and strategy, were used in the design of class collaborations shown in Figure 2. The bridge design pattern "*decouples abstraction from implementation so that implementation and abstraction can be varied independently*" (Gamma et al. 1995). Derived classes of *interpolation* are to be implemented by using abstract interfaces of other three classes that are aggregated to the *interpolation* class. Therefore, any concrete implementation of *interpolation* can use any combination of concrete classes derived from *weightFunction*, *basisFunction*, and *searchAlgorithm*. Furthermore, the *interpolation* class itself is an abstract class, which can be refined further to develop yet another abstract class if desired.

The other design pattern applied in the design is the strategy design pattern. The strategy design pattern is a behavior design pattern that allows algorithms to be swapped at runtime. This is achieved by defining a virtual function that needs to be implemented by all sub-classes of abstract core classes such as *interpolation*, *weightFunction*, etc. Therefore, applications are allowed to choose which concrete sub-classes of *weightFunction*, *basisFunction*, and *searchAlgorithm* are to compose the *interpolation* class.

The core interpolation framework benefits developers of interpolation algorithms. With loose-coupling between concrete classes, it is easy to reuse existing classes and vice-versa. Efforts of implementing new interpolation algorithms that shares common weight functions or basis functions can be reduced. By using the framework, many different interpolation algorithms can be implemented by different composition of weighting, basis, and interpolation classes. On the other hand, the implementation of search algorithm dictates the efficiency of interpolation and should not be overlooked.

4. Interface Class

The core interpolation framework benefits developers for interpolation algorithms by encouraging code-reuse with well organized program structure. However, to ensure the maximum flexibility of the framework, it was decided that instantiation of concrete classes in the core framework is the responsibility of the program which uses the framework. This may be inconvenient for application developers who solely want

to perform interpolation without knowing too much detail regarding interpolations. Therefore, an interface class following the "Façade" design pattern is designed to ease the use of core interpolation framework.

Figure 3 shows the composition of this interface class with the core interpolation framework. The interface class is responsible for instantiating concrete classes of *interpolation*, *weightFunction*, *basisFunction*, and *searchAlgorithm*, and then initializes each of these classes properly. One of the most important responsibilities of this class is to make sure the "right" concrete classes of *weightFunction*, *basisFunction*, and *searchAlgorithm* are assigned to the concrete *interpolation* class.

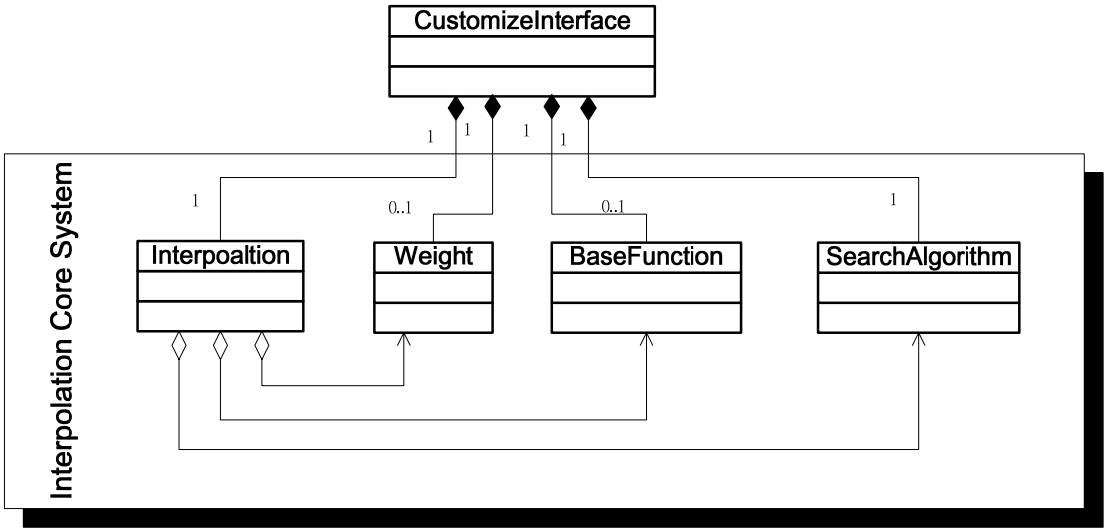


Figure 3: Interface class for the core interpolation framework

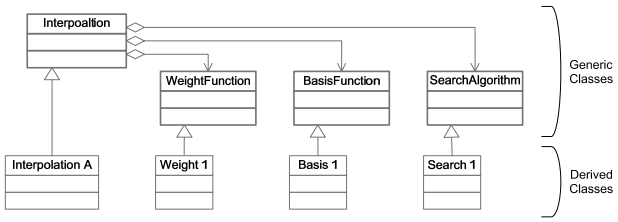
By introducing the interface class, application programs only need to know and use the interface class without knowing it is supported by four classes (*interpolation*, *weightFunction*, *basisFunction*, and *searchAlgorithm*). Therefore, the interface class benefits application developers who need simple interpolation operations without knowing details of interpolation algorithms.

5. Evaluation

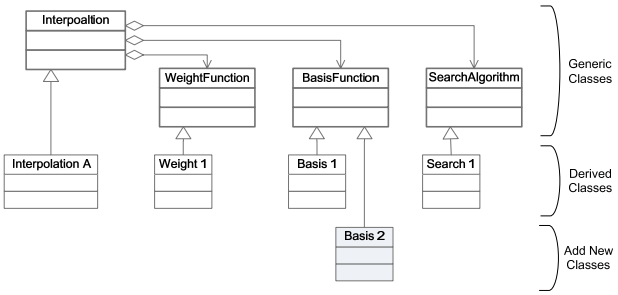
The designed software framework is evaluated from two perspectives: interpolation algorithm developers and application developers. For interpolation algorithm developers, the framework should reduce the effort of programming; for application developers, the framework ought to provide easy-to-use interfaces.

Figure 4 shows the benefit of the software framework for interpolation algorithm developers. Assuming an implementation for interpolation method A has been completed, as in Figure 4(a). One may create another interpolation method B, as in Figure 4(b), by 1) implementing a new concrete *basisFunction* class and 2) creating another concrete *interface* class that aggregates new *basisFunction* class with old concrete *weight interpolation*, and *searchAlgorithm* classes. Similarly, different interpolations can be realized by substituting weight functions, as in Figure 4(c). Other interpolation methods are possible by using the same weight and basis functions but different interpolation procedures, Figure 4(d). Finally, if one is unsatisfied with the performance of interpolation, he or she may try to improve the performance by introducing better search algorithms, as in Figure 4(e). This great flexibility or extensibility is attained by *encapsulation* and *inheritance*, two important characteristics of OOP.

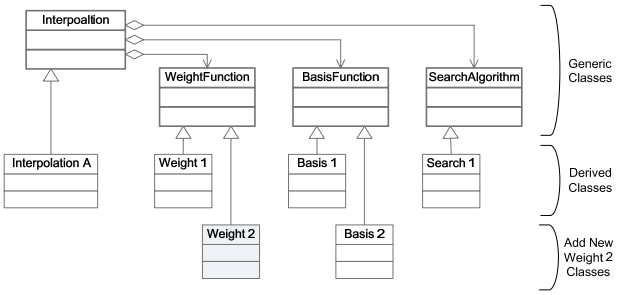
Figure 5(a) shows the ease of maintaining interface class. User can also define the interface class himself. Basically, once a new interpolation algorithm is implemented, a new interface class is added by 1) programming constructor to instantiate four concrete classes of appropriate combination; and 2) writing an inline method that calls to the calculation method of the concrete interpolation class. It may be noted that identical combination of core interpolation classes with different "default" parameters may be programmed into a different interface class with different parameter list under the same method name – an example of using *polymorphism*.



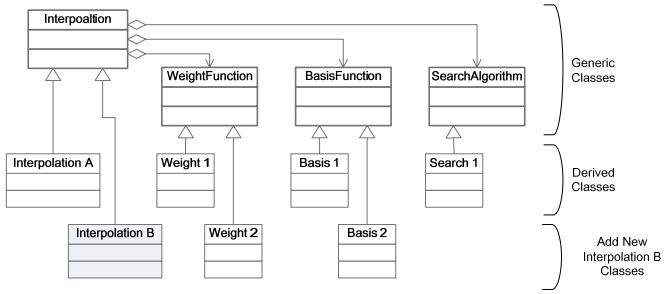
(a) Assuming interpolation method A has been completed



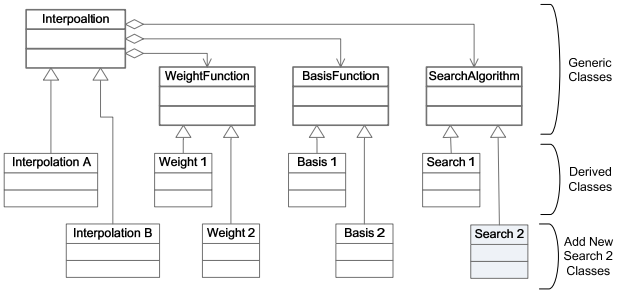
(b) New interpolation algorithm with new basis function



(c) New interpolation method with new weight function



(d) New interpolation method by new interpolation procedures



(e) Gaining better interpolation efficiency by implementing efficient search

Figure 4: Benefits for interpolation algorithm developers

Figure 5(b) shows the effort required for application developers who need performing interpolation, assuming core interpolation classes have been implemented and bundled as a class library. It is seen that the effort involves 1) preparing data, 2) instantiating a concrete interface class, and 3) calling the calculation

method of the interface class. These efforts are necessary regardless the practice of programming. Therefore, it is considered by authors that the design of the interface class is essential to help application developers who do not concern about details of interpolation algorithms.

Figure 5(c) shows the ease of switching to different interpolation algorithms using the interface class. This is sometimes necessary because application developers do not necessarily know beforehand which interpolation algorithm suits his application the most. By changing one line of code, which calls to the constructor of a particular constructor of a concrete interface class, the interpolation algorithm can be completed altered – achieved through *inheritance*.

```
//====================//
//  Interpolation interface header  //
//====================//

//In this area include necessary header file. (interpolation method, weight function …)
#include <xxxx.h>
…

enum WeightType {_noWeight, _weight1=1, _weight2, … };
enum BasisType {_noBasis, _basis1=1, _basis2, …};
enum SearchType {_noSearch, _search1=1, _search2, …};
enum InterpolationType {_noInterpoaltion, _interpoaltion1=1,
                                          //Enumeration all types methods for weight,
                                          basis, search, and interpolation methods. It
                                          needs to update enum parameters if new
                                          algorithms be added.

class InterpolationInterface {
protected:
    //basic field  parameters
    int dimension;
    …

    WeightFunction   *weightFunction;
    BasisFunction     *basisFunction;
    SearchAlgorithm  *search;
    Interpolation      *interpolation;
                                          //Programming maintainer use these pointer to get
                                          default or selection algorithms objects.

    void createInterpolation(InterpolationType I…
    void createWeight(WeightType weight);
    void createBaseFunction(BasisType base);
    void createSearch(SearchType search);
                                          //Programming maintainer use these functions to realize
                                          objects. These Functions get parameter which selected
                                          from enumeration types, then create objects. It also needs
                                          to update if new algorithms be added.

    void InterpolationBuilder();
                                          //Combining appropriate algorithms in this function.
public:
    // step 1. Constructor and Destructor
    InterpolationInterface(basis parameter1, basis parameter2, …);
    ~InterpolationInterface ();

    // step 2.User select one of below method to build interpolation object
    void create(parameter1, parameter2, …);  //create by advance use
    void createInterpolationAlgorithm1();    //default method
    …
                                          //Programming maintainer provide
                                          interface for user create interpolation.
                                          User can use default method function
                                          to create easily or give selection
                                          parameters by advance function to
                                          create.
    // 3. User use this function to calculate and get interpolation result
    void calcField(calc parameter1, calc parameter2, …);

};
```

(a) Maintenance of interface class

Figure 5: Benefits of proposed framework for application developers

```
// step 1: creating use of interface class
    InterpolationInterface  *myInterpolation = new InterpolationInterface(parameter1, …);

// step 2: choosing one of style method for building interpolation object.
    myInterpolation -> createInterpolationAlgorithm1();

// step 3: calling the calculation method of use interface class.
    myInterpolation -> calcField(calc parameter1, calc parameter2, …);
```

(b) Use of interface class in applications

```
// step 1: creating use of interface class
    InterpolationInterface  *myInterpolation = new InterpolationInterface(parameter1, …);

// step 2: choosing one of style method for building interpolation object.
    myInterpolation -> create(InterpolationAlgorithm1, basis1, weight1, searc
                                                                              //Switching to different
// step 3: calling the calculation method of use interface class.               interpolation algorithms jus
    myInterpolation -> calcField(calc parameter1, calc parameter2, …);        change creation function of
                                                                              step2.
```

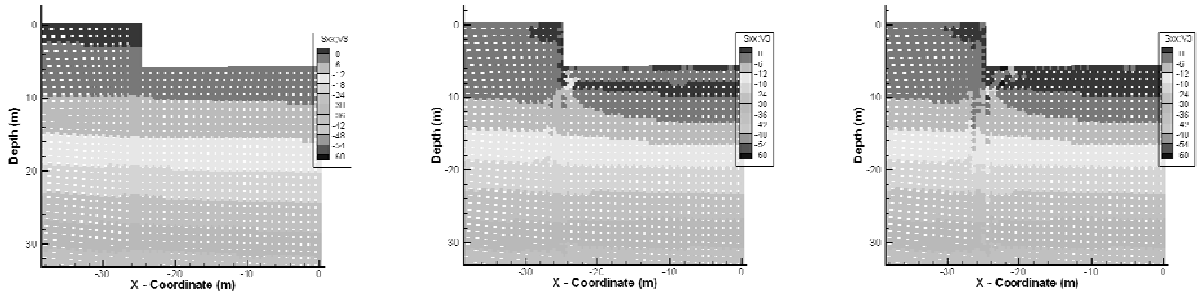
(c) Changing interpolation algorithms in applications

Figure 5 (cont'd): Benefits of proposed framework for application developers

6. Applications

The interpolation library developed based on the described framework was applied to evaluate various interpolation algorithms for stress interpolation in excavation analyses. These interpolations are necessary for various purposes such as solving partial differential equations using mesh-free methods and post-processing analysis results for producing contours, etc.

Figure 6 shows interpolations of horizontal stresses produced by an excavation analyses. It is seen that different algorithms may produce different results due to differences in algorithms, proximities to boundaries, etc. Therefore, it is necessary to have the proposed interpolation framework to assist evaluations of various interpolation algorithms to find the most-suitable interpolation algorithms for one's own purpose.



(a) Least-Square (b) Moving Least-Square (c) PIM

Figure 6. Different stress-distributions by different interpolation algorithms

7. Summary

A software framework for implementing interpolation algorithms is proposed based on object-oriented programming and well-known design patterns. The framework is considered general and should be capable of unifying all implementations of interpolation algorithms. The proposed framework benefits implementers of interpolations by providing great extensibility, good code-reuse and code-management; it also benefits application developers by the ease of use and ease of switching to different algorithms.

Acknowledgement

This work is partially supported by NSC 95-2221-E-011-110 of National Science Council of Taiwan.

References

[1] Gamma, E. (1991). "Object-oriented software development based on ET++: design patterns, Class Library, Tools." PhD thesis, University of Zurich Institut für Informatik.

[2] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). "Design Patterns Elements of Reusable Object-ienting Software." Addison-Wesley, USA, ISBN 0-201-63361-2.

[3] Lancaster, P. and Salkauskas, K. (1981). "Surfaces Generated by Moving Least Squares Methods." Math. of Comp., 37(155), 141-158.

[4] Li, H., Wang, Q. X. and Lam, K. Y. (2004). "Development of a novel meshless Local Kriging (LoKriging) method for structural dynamic analysis." Comp. methods in Appl. Mech. And Engrg., 193, 2599-2619.

[5] Liu, G. R. and Gu, Y. T. (1999). "A point interpolation method." Proceedings of 4th Asia-Pacific Conf. on Comp. Mech., Singapore, 1009-1014.

[6] Liu, G. R. and Gu, Y. T. (2001). "A point interpolation method for two-dimensional solids." Int. J. Muner. Meth. Engng, 50, 937-951.

[7] Liu, G. R. (2003). "Mesh Free Methods – Moving beyond the Finite Element Method." CRC Press, ISBN 0-8493-1238-8.

[8] Liu, G. R. and Gu, Y. T. (2005). "An introduction to meshfree methods and their programming." Springer, ISBN 1-4020-3228-5.

[9] Wang, J. G. and Liu, G. R. (2000). "Radial point interpolation method for lastoplastic problems." Proc. of the 1st Int. Conf. On Structural Stability and Dynamics, Taipei, Taiwan, 703-708.