International Conference on Computational Science, ICCS 2011

Agent-oriented image processing with the hp-adaptive projection-based interpolation operator

Marcin Sienieka, Piotr Gurgul*, Marcin Skotniczny, Krzysztof Magiera, Maciej Paszyński

*Department of Computer Science, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland

Abstract

In this paper we discuss applications and design of the agent-oriented, hp-adaptive projection-based interpolation technique. We describe the use of the mesh adaptation process to produce the most faithful representation of the input image in the Finite Element space. We discuss the advantages of the agent-oriented application model both in general and in terms of the hp-adaptive application properties. Lastly, we describe a sample problem used as a proof of concept.

Keywords: projection-based interpolation; image processing; computing multi-agent systems; adaptive finite element method

1. Motivation

Interpolants computed using projection based interpolation operator were originally used as convergence estimates for the hp-adaptive Finite Element Method [1, 9, 10]. However, the spectrum of their applications is much wider. In this paper we discuss their use as a basis for approximation of geometry. More specifically, our input is a set of monochromatic bitmaps and we are going to change their representation into the linear combination of the Finite Element basis functions. This transformation is necessary to prepare the input image for future computations (where it is treated as a material function) using the Finite Element Method as well as to generate initial mesh in accordance with material function variation. This application can be useful in certain material science problems as well as Magnetic Resonance (used in medical industry) scans processing.

2. Projection based interpolation operator description

2.1. Definition of a finite element

A finite element can be defined as a triple \( (K, X(K), \psi_j, j = 1, \ldots, N) \), where \( K \subset \mathbb{R}^n, n = 1, 2, 3 \) is a domain, \( X(K) \) is a finite-dimensional space of element basis functions, embedded in some infinite dimensional
functional space $\mathcal{X}$, $N = \dim X(K)$ and $\psi_j : \mathcal{X} \to \mathbb{R}, j = 1, ..., N$ is a set of $N$ linearly independent linear functionals defined on $\mathcal{X}$ (the element's degrees of freedom).

In this paper we focus on the 2D case of the Finite Element Method and all further definitions and theorems will be limited to that dimension.

![Fig. 1 - description of a Finite Element's nodes](image)

2.1.1. *Master element vs. parametric element*

In order to simplify the computations on different elements, there is identified a master element that holds all the operations from real, parametric elements. We skip the general case that can be found and proven in [4] but focus on the special case that is used in our work. Namely, the master element is a 1 x 1 square formed by points (0,0), (0,1), (1,0) and (1,1). To interpolate a function on a physical element we map it to the master element, perform the interpolation there and transfer the result back to the physical element. This approach allows us to store the basis functions and compute its derivatives and integrals for the master element only.

2.2. *Interpolation*

In order to interpolate an input image with the Finite Element Method basis functions, we create an arbitrary initial mesh on it and perform a selection of h- and p- adaptations [1] as long as the error rate between original and interpolated image remains above the desired level. We leverage an automated hp-adaptation algorithm described below and discuss projection-based interpolation formulas for a single element limited by $[0;1] \times [0;1]$.

We are using further the following notation:

- $u_i = a_i \phi_i$ - i-th interpolant,
- $\{\phi_i\}$ - base functions,
- $\{a_i\}$ - coefficients,
- $U(x,y)$ - interpolated function,
- $u = \sum_i u_i$ - interpolating function,

Different kinds of the basis functions (and though interpolants) on a single element are marked as follows:

- $u_v : i \in \{1..4\}$ - vertex approximation functions associated to basis functions,
- $u_e : i \in \{5..8\}$ - edge approximation functions associated to basis functions,
- $u_f$ - face approximation functions associated to basis functions.

2.2.1. *Interpolated bitmap assumptions*
Since we use ordinary monochromatic bitmaps as the processing input, there are several assumptions to follow in order to treat them as ordinary approximated functions. Firstly, the input data is discrete, while the interpolated function is assumed to be non-discrete. As the result we have to compute interpolated function values for the non-integer arguments, based on the pixels of the input image. It is done by computing the weighted average of values of the closest neighbors (pixels). Besides, we approximate first derivatives using differential quotients.

The formulas below are used to compute the interpolation coefficients. These concrete formula sets are specific to all hierarchical shape functions of order up to 2, but can be generalized. Even for higher orders there are only several local systems of equations to be solved and there is no global system of equations.

2.2.2. Interpolation at vertices

Thanks to the locality and conformity conditions (described in [10]) this is the simplest case – both interpolating and interpolated function must match at mesh vertices (they are considered interpolation nodes):

\[ \forall_{(x,y) \in \text{MV}} u(x, y) = U(x, y) \]

where MV stands for mesh vertices.

Having this constraint we can easily obtain the linear interpolants \( u_1, u_2, u_3, u_4 \) for all master element’s vertices.

\[
\begin{align*}
    a_1 &= \frac{U(0,0)}{\varphi_1(0,0)} \\
    a_2 &= \frac{U(1,0)}{\varphi_2(1,0)} \\
    a_3 &= \frac{U(0,1)}{\varphi_3(0,1)} \\
    a_4 &= \frac{U(1,1)}{\varphi_4(1,1)}
\end{align*}
\]

(1)

2.2.3. Projecting over edges

Projecting over edges does not produce the exact solution (unless \( U \) is from the finite element space). Thus, for an edge \( E_i \) we are looking for \( u_i \) such that:

\[
\min_{u_i} \left\| U - u_1 - u_2 - u_3 - u_4 \right\|_{U_i(E_i)} \rightarrow \min
\]

where \( u_i : i \in \{5..8\} \) is one of the edge basis functions. The difference \( (U - u_1 - u_2 - u_3 - u_4) \) produces a function which vanishes at each of element’s vertices.

By rewriting the norm we obtain (for the case of a vertical edge):

\[
\int_{E_i} \left( \frac{dU}{dy} - \sum_{j=1, j \neq i}^{4} \frac{du_j}{dy} \right) \frac{d\varphi_i}{dy} dy = 0
\]

but: \( u_i = a_i \varphi_i \) and thus

\[
a_i \int_{E_i} \frac{d\varphi_i}{dy} \frac{d\varphi_i}{dy} = \int_{E_i} \left( \frac{dU}{dy} - \sum_{j=1}^{4} \frac{du_j}{dy} \right) \frac{d\varphi_i}{dy} dy
\]

and finally:

\[
a_i = \frac{\int_{E_i} \left( \frac{dU}{dy} - \sum_{j=1}^{4} \frac{du_j}{dy} \right) \frac{d\varphi_i}{dy} dy}{\int_{E_i} \left( \frac{d\varphi_i}{dy} \right)^2 dy}
\]

(2)

Horizontal edges can be processed similarly using \( dx \) differential and integration over \( dx \).

2.2.4. Projecting over face

This time we are trying to minimize the difference between \( U \) decreased by all interpolants of lower orders and the face interpolant.
\[
\|(U - u_1 - u_2 - u_3 - u_4) - u_5 - u_6 - u_7 - u_8) - u_0\|_E(t) \rightarrow \min
\]
\[
\int_{\Gamma} \nabla((U - \sum_{i=1}^{8} a_i \varphi_i) - u_0) \circ \nabla \varphi_0 \, dxdy = 0
\]
And since \( u_0 = a_0 \varphi_0 \)
\[
\int_{\Gamma} \left( \frac{d}{dx} \left( U - \sum_{i=1}^{8} a_i \varphi_i - a_0 \varphi_0 \right) \frac{d \varphi_0}{dx} + \frac{d}{dy} \left( U - \sum_{i=1}^{8} a_i \varphi_i - a_0 \varphi_0 \right) \frac{d \varphi_0}{dy} \right) \, dxdy = 0
\]
\[
a_0 \int_{\Gamma} \left( \frac{d \varphi_0}{dx} \frac{d \varphi_0}{dy} + \frac{d \varphi_0}{dx} \frac{d \varphi_0}{dy} \right) \, dxdy = \int_{\Gamma} \left( \frac{dU}{dx} \frac{d \varphi_0}{dx} + \frac{dU}{dy} \frac{d \varphi_0}{dy} + \left( \sum_{i=1}^{8} a_i \varphi_i \right) \frac{d \varphi_0}{dx} + \frac{d}{dy} \left( \sum_{i=1}^{8} a_i \varphi_i \right) \frac{d \varphi_0}{dy} \right) \, dxdy
\]
Finally
\[
a_0 = \frac{\int_{\Gamma} \left( \frac{d \varphi_0}{dx} \right)^2 + \left( \frac{d \varphi_0}{dy} \right)^2 \, dxdy}{\int_{\Gamma} \left( \frac{d \varphi_0}{dx} \right)^2 + \left( \frac{d \varphi_0}{dy} \right)^2 \, dxdy}
\]

3. HP Mesh refinements and its role in projection-based interpolation

The quality of the interpolation can be improved, as usually, by the expansion of the interpolation base. In FEM terms, this could be done thanks to some kind of mesh adaptation.

We consider two methods of adaptation:

3.1. P-adaptation – increasing polynomial approximation level

One approach is to increase order of the shape functions on the elements where the error rate is higher than desired. More shape functions in the base means smoother and more accurate solution but also more computations and the use of high-order polynomials.

3.2. H-adaptation – refining the mesh

Another way is to split the element into two in order to obtain finer mesh. This idea arose from the observation that the domain is usually non-uniform and in order to approximate the solution fairly some places require more precise computations than others, where the acceptable solution can be achieved using small number of elements. The crucial factor in achieving optimal results is to decide if a given element should be split into two (or, respectively four) parts or not. We are going to present the automated algorithm that decides after each iteration for the element if it needs h- or p-refinement or not. The refinement process is fairly simple in 1D but the two-dimensional case enforces a few refinement rules to follow:

3.3. Automated hp-adaptation algorithm

Neither the p- nor the h-adaptation guarantees error rate decrease that is exponential with a step number. This can be achieved by combining together these two methods under some conditions, which are not necessarily satisfied in our case. Still, in order to locate the most sensitive areas at each stage dynamically, and improve the solution as much as possible, we employed the self-adaptive algorithm that decides if the given element shall be still refined or
it is fine enough for the satisfactory interpolation, in an analogical manner to the algorithm for Finite Elements adaptivity described by L. Demkowicz in [1].

1: function adaptive_pbi( \( \text{mesh}_{\text{initial}}, \text{err}_{\text{desired}} \) )
2: \( \text{mesh}_{\text{coarse}} = \text{mesh}_{\text{initial}} \)
3: repeat
4: \( \text{u}_{\text{coarse}} = \text{compute interpolation on } \text{mesh}_{\text{coarse}} \)
5: \( \text{mesh}_{\text{fine}} = \text{copy } \text{mesh}_{\text{coarse}} \)
6: divide each element of \( \text{mesh}_{\text{fine}} \) into two new elements
7: increase order of shape functions on each element of \( \text{mesh}_{\text{fine}} \) by 1
8: \( \text{u}_{\text{fine}} = \text{compute interpolation on } \text{mesh}_{\text{fine}} \)
9: for each element \( K \) of \( \text{mesh}_{\text{fine}} \) do
10: \( \text{err}_K = \text{compute error decrease rate on } K \) //discussed below
11: end do
12: \( \text{mesh}_{\text{adapted}} = \text{copy } \text{mesh}_{\text{coarse}} \)
13: for each element \( K \) of \( \text{mesh}_{\text{adapted}} \) do
14: if \( \text{err}_K > \text{threshold} \times \text{err}_{\text{max}} \) then //see below
15: divide \( K \)
16: end if
17: end do
18: enforce \( \text{mesh}_{\text{adapted}} \) integrity
19: \( \text{mesh}_{\text{coarse}} = \text{mesh}_{\text{adapted}} \)
20: until \( \text{err}_{\text{max}} < \text{err}_{\text{desired}} \)
21: return \( \text{mesh}_{\text{fine}} \)
22: end function

Alg. 1 hp-adaptive PBI pseudocode

4. Agent-based approach

4.1. Justification for particular design decisions
As one can see the presented algorithm acts mostly locally on subsequent parts of the domain. Communication is required only by:

- mesh manipulation → to preserve irregularity rules – see mesh description
- max error computation → could be estimated by maxima local to a computational node, or easily accumulated globally
- long range mesh dependencies → in some uncommon cases it might happen that a degree of freedom which contributes to the interpolation on a given element is distant in terms of computing nodes – this, however, is still pretty straightforward to solve, provided that the environment is able to localize it

On the other hand the method is computation-intensive (when combined with a FEM solver) so some parallelization is needed. What is more, as the hp-adaptive algorithm can produce very unbalanced meshes we decided to leverage Agent-oriented paradigm to increase locality, robustness and open the way for load balancing based on the local information only (e.g. Diffusional Algorithm in [12], [13] and [14]).

Due to the fact that Finite Elements computations which follow the interpolation step in our research workflow tend to generate a stiffness matrix not suitable for iterative solvers, we were forced to accommodate direct methods, Gaussian elimination precisely. In order to keep the algorithm distributed, we decided to employ a special form of Gaussian elimination for mesh problems [7]. It performs elimination on each element locally and then merges the solution with each neighbor. Such process forms a binary tree and the time complexity is logarithmic. The concepts, data structures and the implementation of this algorithm in an agent-oriented environment were described in [3]. This use case has a significant impact on the design of the application described below.

### 4.2. Agents

We have parallelized computations using domain decomposition. Each agent performs PBI on its own slice of the mesh and is capable to divide and delegate the task to another agent when needed. Effectively we managed to introduce only three different types of agent's roles in our application:

- Slave Agent - performs the actual computations
- Master Agent - manages an interface between its children
- Root Agent - manages the highest-level interface

Such distinction is related to the fact that these entities have different tasks in the subsequent step of computations the application is intended for: the PDE solver.

### 4.3. Interactions between agents

#### 4.3.1. Interactions on the constrained computational mesh

To use the above approach to mesh distribution, we had to develop an efficient method of mesh refinement that is run on all agents simultaneously. That is why, we divide mesh into submeshes that will be later distributed among different agents. The division is made along some path of edges. Optimal division would create two meshes of similar amount of faces using the smallest possible number of edges on the division path. The latter condition stems from the fact that the amount of communication necessary to synchronize the submeshes is proportional to the amount of edges on the border. Since it is very hard to find an optimal path in reasonable computing time, heuristic algorithms have to be used.
The main influence that has driven us during the development of mesh distribution algorithm is the choice of base shape functions. Some of them span across multiple faces and during the mesh division process we will have to divide across some of them. To deal with that limitation, the division path is chosen so that all the split shape functions will be based on an edge or vertex of that path. In the case of image processing problem this will enable us to calculate the coefficients for each base function separately on both submeshes and their values will be the same.

To keep the set of base shape functions simple we have to enforce a restriction we call $\tau$-constraint – two neighboring elements cannot differ more than one $h$-adaptation degree. This is equivalent to the constraint that on each edge of an element there can be no more than one additional vertex (we call it $\tau$-vertex) which belongs to a smaller, more refined neighboring element. This constraint requires that after a refinement step there has to be another step of its enforcement – we will divide recursively elements that do not conform to it. This leads to the main obstacle with synchronization – the information which faces were divided has to be passed to neighboring submeshes.

In the agent-based approach, each agent represents some part of the whole mesh. Depending on the role of an agent it can contain a single submesh (slave agent) or represents an interface between submeshes of two different agents (master agent). In the latter case, the subagents can also have role of master and create a binary tree. The master is responsible for routing synchronization data from both subagents to each other and up the tree.

In case of distributed mesh refinement, the main information that has to be synchronized is the list of border edges (edges that were on the division path) that will be split during the refinement process. During the refinement process, each border edge split information is send to the master agent above. The master agent decides if the border split is internal – if so, it is forwarded to the other subagent – or external – it is again send up the tree.

4.3.2. Distributed algorithm implementation

For the reasons described in [3], the agents behavior is expected to change several times in runtime. That is why each agent's task is defined in general as playing an assigned Role. Each role's steps are called Actions and are switched according to the role-specific ActionSwitcher.

4.3.2.1. Interpolation

The PBI algorithm implementation is pretty straightforward once you have a functional distributed mesh implementation. The formulas 1,2,3 are applied to mesh nodes in the following order:

- vertices (apart from $\tau$-vertices which don't provide a distinct degree of freedom)
- constrained edges - those incidental with $\tau$-vertices
- unconstrained edges
- faces

This action corresponds to lines 4 and 8 of Alg. 1. This is performed solely in Slave agents. Masters and the Root yield.
4.3.2.2. Refinement

The mesh in each Slave agent is unconditionally refined. Note that mesh irregularity rules compliance is not affected here, so there is no need to negotiate any forced refinements here. Corresponds to lines 5-7 of Alg. 1. Masters and the Root still yield.

4.3.2.3. Error Estimation

This is with reference to lines 9-11 in Alg. 1. Slave agents are responsible for computing their local error decrease rates, Masters - for accumulating local maxima and the Root - for computing the globally maximal error decrease rate based on the information from its children.

The error decrease rate is usually understood as a $H^1_{0}$ norm of a difference between the fine and coarse solution (relative error decrease rate) or as a difference between absolute errors ($H^0_{0}$ norm of the difference between an interpolation and the interpolated function). In some cases (i.e. non-existent interpolated function derivatives) it might make sense to use $L^2$ norm instead.

4.3.2.4. Adaptation

Implements lines 12-19 of Alg. 1. Initiated by the Root propagating the maximal error decrease rate to its children, passed down by Masters, the core activity is performed, as always, on the bottom of the hierarchy. As noted above (see Fig. 2), on this step, by contrast, there is a need for enforcement of the mesh regularity (line 19 of Alg. 1).

5. Exemplary problem and numerical results

As the proof of concept for our application we decided to use the image presented in Figure 5 as the input for interpolation. The picture represents a single step of the austenite-ferrite phase transformation [12].
The desired accuracy was reached after the 6th iteration:

Fig. 5 - image representing the interpolated function

Fig. 6 - PBI results for subsequent steps (0-5) of mesh adaptation, light green marks the element borders

Fig. 7 - interpolation computed after seven iterations of PBI

Fig. 8 - error decrease rate with iteration number
The adaptation was targeted to minimize locally maximal relative $L^2$ error. The decrease of global relative $L^2$ error is shown on Fig. 8.

6. Conclusions and future work

There are multiple conclusions related to our work. Firstly, there are still many unknowns and the method must be further developed until it will be mature enough to be compared with the existing solutions. On the other hand there exist many potential fields where it can be leveraged and it is challenging to choose the most prospective one.

Acknowledgements

The work reported in this paper has been supported by Polish MNiSW grants NN 519 447739 and NN 501 120 836.

References