

VIRTUAL ELEMENT METHOD PRE-PROCESSOR SOFTWARE

Paulo Akira Figuti Enabe¹, Rodrigo Provasi¹

¹*Dept. of Structural and Geotechnical Engineering, Polytechnic School, University of São Paulo
Avenida Professor de Almeida Prado tv. 2 n.83, 05508070, São Paulo, Brasil
paulo.enabe@usp.br, provasi@usp.br*

Abstract. The Virtual Element Method (VEM) is a relatively recent method where it is possible to have non-polynomial functions inside the virtual elements (virtual elements are the equivalent of finite elements for VEM), while requiring them to behave like polynomials only at the frontier of the virtual elements. The “virtual” term comes from the fact that the shape functions are computed implicitly using an optimal set of degrees of freedom leading to a stiffness matrix that heavily depends on the choice of the degrees of freedom.

Geometry parameters like coordinate of vertices, polygonal diameter and area are fundamental quantities in VEM because they are related to the choice of the degrees of freedom and, consequently, the stiffness matrix calculation. In this work, a graphical user interface to VEM focused on the two-dimensional case is developed in order to ease input data and guarantee VEM performance by ensuring the optimal choice of the degrees of freedom. The interface is responsible to generate the geometry from a set of coordinates, calculate the area, centroid, and polygonal diameter.

As the main goal of the interface is to make the use of VEM less abstract, a set of requirements were defined for the graphic tool development. Among them, two are crucial: the interface must be user-friendly, guaranteeing that user with no prior experience of the method can use it, and data input must be simple. To keep data input simple, a text file with the coordinates of vertices are used. And to guarantee that the graphic tool is user friendly, C# programming language is employed in a very intuitive and clear construction.

Throughout the article, the requirements and the interface are shown, explaining the choices made during the project phase related to programming language and libraries. To exemplify its use, a simple case of a two-dimensional problem using VEM is done. More specifically, the computation of a local stiffness of a polygon with more than 4 sides is shown.

Keywords: virtual element method, finite element method, graphical user interface

1 Introduction

Virtual Element Method (VEM) is a relatively recent model developed by a group of mathematicians. VEM has a rigorous mathematical formulation requiring a background that is not usual for engineers. To fully understand VEM model it is necessary to be familiarized with some results of Functional Analysis, Measure Theory and Partial Differential Equation Analysis. One of the main goals of this work is to make VEM formulation easier to be understood by a public not much familiarized with pure mathematics concepts but without losing the consistence of the mathematical formulation presented on VEM development. Thus, in the first part of this work, a theoretical background related to the topics mentioned above is presented. Proofs of theorems (and propositions) will be suppressed and all formulation will be constructed as intuitive as possible. The second part is dedicated to present the graphical interface project and construction.

The Virtual Element Method is a generalization of Finite Element Method. The “virtual” term comes from the fact that shape functions are computed implicitly without any quadrature formula method required, even though the shape functions strongly depend on the choice of degrees of freedom. And the choice of degrees of freedom are related to the construction of the virtual element space (the analogous of finite element space for VEM). In the virtual element space exists non-polynomial and polynomial functions. The main requirement for non-polynomial functions is that they behave like polynomials at the frontier elements. To handle with the non-polynomial functions and to build the stiffness matrix, the projection operator Π^{∇} is defined.

Basic aspects of the method are presented on da Veiga et al. [1] and da Veiga et al. [2]. In the first paper,

the theoretical formulation of VEM is described beginning with the Poisson Equation, then the discrete problem is presented and the authors prove that it is well posed. The virtual element space is constructed, the degrees of freedom are chosen and finally, the bilinear form and the load vector are constructed. It is important to mention that the bilinear form with some restrictions will become the stiffness matrix.

The second paper is focused on the practical aspects of VEM. In this sense, the authors give a guideline of how to compute the stiffness matrix from the projector operator Π^∇ and from the degrees of freedom. This matrix can be computed exactly using only the chosen degrees of freedom. Also, some examples of the application of the method are given. In the end, the authors introduce a L^2 -projector related to the construction of mass matrix.

Some implementations of VEM were developed in the past years. In Sutton [3], the method is implemented using *MATLAB*. The implementation is restricted to two dimension problems and only considers the case which order of accuracy is equal to one (order of accuracy will be defined more precisely later in this text). The author uses a native *MATLAB* mesh generator. Ortiz-Bernardin et al. [4] an VEM implementation in C++. In this paper, the algorithm is also restricted to the 2D case but a mesh generator is developed. A performance comparison is made between VEM and FEM. The authors conclude that for a same amount of degrees of freedoms the computational cost and the accuracy in results are the same.

Mengolini et al. [5] enumerate some characteristics of the method and make some comparisons with FEM. According to them, VEM has a good performance with complex geometries because of its pre-processing flexibility and it is robust against mesh distortions (while FEM is highly dependent on mesh quality). According to the authors, in FEM mesh elements are triangles or quadrilaterals but in VEM elements can be any convex or non-convex polygon (a more precise definition of polygons used in VEM will be given later). In 3D case, FEM uses tetrahedra, hexahedra and pyramids while VEM uses any polyhedra.

In da Veiga et al. [1] and da Veiga et al. [2] the presented formulation is very rigorous and consistent but there is a lack of practical applications. On the other hand, Sutton [3], Ortiz-Bernardin et al. [4] and Mengolini et al. [5] are focused on application and some important details of the model are skipped.

As illustrated Virtual Element Method is highly dependent on geometrical characteristics. Therefore, the data input becomes a very important part for a successful application of the method. The main goal of this work is to present a pre-processor software to guarantee a robust data input to a VEM solver in order to minimize input errors and make data visualization simpler.

2 Virtual Element Method (VEM)

On this section, Virtual Element Method mathematical formulation is briefly described. For more detailed formulation consult da Veiga et al. [1] and da Veiga et al. [2]. First some conditions are given to ensure that the discrete problem is well posed. Then, the degrees of freedom will be chosen and the virtual element space will be built. Finally, the projection operator is introduced in order to build the local stiffness matrix and the load vector.

2.1 The Discrete Problem

Let Ω be a polygonal domain. Consider a decomposition τ_h of Ω in simple polygons K . Simple polygons are simply connected open sets in which the frontier is formed by line segments intersecting only on the ends. For each K , take the following definition: $h = \max_{K \in \tau_h} h_K$. The bilinear form can be written as

$$a(u, v) = \sum_{K \in \tau_h} a^K(u, v), \quad (1)$$

with $u, v \in H_0^1$ and where $a^K(u, v) = \int_K \nabla u \cdot \nabla v dx$, for each $K \in \tau_h$. For each polygon K were chosen

$$|v|_{1,K} = a^K(v, v)^{1/2}, \quad (2)$$

for all $v \in V$ as the semi-norm and

$$\|v\|_\Omega = \left(\sum_{K \in \tau_h} |v|_{1,K}^2 \right)^{1/2}, \quad (3)$$

as the norm. It is important to mention that the norm is originated from the Sobolev Space norm and that space will be denoted by H^1 . The polynomial space of dimension k in a space U will be denoted by $\mathbb{P}_k(U)$.

The discrete problem is given by: find $u_h \in V_h$ such that $a_h(u_h, v_h) = \langle f_h, v_h \rangle$, for all $v_h \in V_h$ where a_h is the discrete bilinear form and f_h is a linear functional related to the discrete problem (the linear functional related to the continuous problem will be denoted as $F(v)$).

In da Veiga et al. [1] the following hypothesis are taken:

Hypothesis 1 For each h , we have:

1. $V_h \subset H_0^1(\Omega)$,
2. a symmetric bilinear form $a_h : V_h \times V_h \rightarrow \mathbb{R}$ and a bilinear form $a_h^K : V_h|_K \times V_h|_K \rightarrow \mathbb{R}$ such that

$$a_h(u_h, v_h) = \sum_{K \in \tau_h} a_h^K(u_h, v_h), \quad (4)$$

3. a element $f_h \in V_h'$.

Hypothesis 2 There exists an integer $k \geq 1$ (order of accuracy) such that, for all $K \in \tau_h$:

1. we have $\mathbb{P}_k(K) \subset V_h|_K$, where $\mathbb{P}_k(K)$ is the polynomial space of dimension k in K and $\mathbb{P}_{-1}(K) = 0$,
2. k -consistency: for all $p \in \mathbb{P}_k(K)$ and for all $v_h \in V_h|_K$,

$$a_h(p, v_h) = a^K(p, v_h), \quad (5)$$

3. stability: there exists $C_1, C_2 \geq 0$, independently of h and K , such that:

$$C_1 a^K(v_h, v_h) \leq a_h^K(v_h, v_h) \leq C_2 a^K(v_h, v_h), \quad (6)$$

for all $v_h \in V_h|_K$.

The theorem that guarantees the uniqueness of the solution and its convergence is the following:

Theorem 1 Under the hypothesis mentioned above, we have:

1. the discrete problem has a unique solution,
2. for all approximation $u_I \in V_h$ of u and for all approximation of u_π that is piecewise in $\mathbb{P}_k(K)$,

$$\|u - u_h\|_\Omega \leq \tilde{C}(\|u - u_h\|_\Omega + \|u - u_h\|_\Omega + \tilde{F}_h), \quad (7)$$

where $\tilde{C}(C_1, C_2) \in \mathbb{R}$ and \tilde{F} is the smallest constant such that, for all $v_h \in V_h$, $F(v) - \langle f_h, v_h \rangle \leq \tilde{F}\|v\|_\Omega$.

The proof of the continuity and the coercivity of the bilinear form a_h is done by applying the stability hypothesis and the Cauchy-Schwarz inequality, then using the Lax-Milgram Theorem and concluding the first part of the theorem. The second part of the theorem can be prove by defining $\xi = u - u_I \in V_h$ and then using the linearity property, the k -consistency hypothesis, the stability hypothesis and the triangular inequality.

2.2 Model of Virtual Element Method

Define for each $k \geq 1$, the local virtual element space

$$V_{K,k,h} = \{v \in H^1(K) \mid v|_{\partial K} \in B_k(\partial K), \Delta v_K \in \mathbb{P}_{k-2}(K)\}, \quad (8)$$

where $B_k = \{v \in C^0(\partial K) \mid v|_e \in \mathbb{P}_k(e), \forall e \in \partial K\}$ and e is an edge of the polygon K . In other words, the virtual element space is composed by polynomial and non-polynomial functions. Although it is imposed that the non-polynomial functions must behave as polynomials in the edge of each element.

The choice of the degrees of freedom will imply that the functions $v_h \in V_{h,K,k}$ are well determined. The chosen degrees of freedom are:

1. $\mathcal{V}^{K,k}$: value of v_h on vertices of K ,

2. $\mathcal{M}^{K,k}$: value of v_h on the $k - 1$ midpoint of each edge of K ,
3. $\mathcal{T}^{K,k}$: value of the $k - 2$ internal point (momentum) of K .

By this choice, $\mathcal{V}^{K,k} \cup \mathcal{M}^{K,k} \cup \mathcal{T}^{K,k}$ is unisolvent for $V_{h,K,k}$, i.e, for each set of degrees of freedom, exists a unique $v_h \in V_{h,K,k}$ defined by this set.

The projector operator $\Pi^\nabla : V_{h,K,k} \longrightarrow \mathbb{P}_k(K)$ is directly related with the construction of the bilinear form a_h . As some functions are non-polynomial, this operator is responsible to project these functions on a polynomial space. Thus, the virtual element space can be written as $V_{h,K,k} = \Pi^\nabla(V_{h,K,k}) \oplus (1 - \Pi^\nabla)V_{h,K,k}$.

For the construction of the bilinear form, a naive approach would be to take $a_{K,h}(u, v) = a_K(\Pi^\nabla u, \Pi^\nabla v)$. This choice for the bilinear form satisfies only stability and does not satisfy k -consistency. As a result, it is necessary to introduce a term $S_K : V_{h,K,k} \times V_{h,K,k} \longrightarrow \mathbb{P}_k(K)$ such that $C_3 \cdot a_K(v, v) \leq S_K(v, v) \leq C_4 \cdot a_K(v, v)$, with $C_3, C_4 > 0$ independent of h and K . Therefore the bilinear form is given by eq. 9:

$$a_{K,h}(u, v) = a_K(\Pi^\nabla u, \Pi^\nabla v) + S_K(u - \Pi^\nabla u, v - \Pi^\nabla v). \quad (9)$$

Using eq. 9 it is possible to guarantee k -consistency. Taking the canonical basis $\{\varphi_i\}_{i \in \mathbb{N}}$ of $V_{h,K,k}$ constructed within the Lagrange polynomial, the stiffness matrix is given by eq. 10:

$$a_{K,h}(\varphi_i, \varphi_j) = a_K(\Pi^\nabla \varphi_i, \Pi^\nabla \varphi_j) + S_K(\varphi_i - \Pi^\nabla \varphi_i, \varphi_j - \Pi^\nabla \varphi_j). \quad (10)$$

It is important to mention that at this point it is possible to write $v_h = \sum_i^N \text{dof}_i(v) \varphi_i$ and $u_h = \sum_i^N \text{dof}_i(u) \varphi_i$, where N is the number of degrees of freedom and $\text{dof}_i : V_{h,K,k} \longrightarrow \mathbb{R}$ such that $\text{dof}_i(\varphi_j) = \delta_{ij}$ is the function that gives the value of v and u on each degree of freedom.

For the construction of the load vector, define $P_k^K : V_{h,K,k} \longrightarrow \mathbb{P}_k$ the projection operator using the norm of $L^2(K)$ space. Considering the case $k = 1$, the load vector is given by eq. 11

$$\langle f_h, v_h \rangle = \sum_{K \in \tau_h} P_0^K \frac{1}{N_V} \sum_{i=1}^{N_V} v_h(V_i), \quad (11)$$

where V_i is the i -th vertex of the polygon K and N_V is the vertex number. Now, considering $k \geq 2$ and $f_h = P_{k-2}^K f$ for each $K \in \tau_h$, the load vector for this case is given by eq. 12.

$$\langle f_h, v_h \rangle = \sum_{K \in \tau_h} \int_K f(P_{k-2}^K v_h) dK. \quad (12)$$

3 Graphical Interface for VEM

The software main purpose is to serve as an input and visualization tool for a Virtual Element Method solver. The graphical interface must be simple to use by users with different skill levels and making the use of Virtual Element method less abstract.

According to Sommerville [6], functional requirements are definitions of how the system should work. Non-functional requirements are related to restrictions of system's functionalities. Domain requirements are related to the application domain (e.g. operational system) and reflect the characteristics of the domain. By those definitions, the requirements for the proposed software can be stated as:

- Function requirements
 - allow the user to build a polygonal element,
 - allow input by a text file,
 - allow the user to build an element by entering the coordinates,
 - calculation of centroid, polygonal diameter and area,
 - provide to the user the calculated parameters in a text file,

- provide to the user the local stiffness matrix in text file.
- Non function requirements
 - accessible for users with no experience with software of this nature,
 - make VEM usage less abstract,
 - simplify data input and visualization.
- Domain requirements
 - chosen programming language: C#,
 - chosen graphics API: OpenGL,
 - usage of SharpGL library to integrate C# and OpenGL.

To facilitate the understanding of the selected programming tools, a brief description of each is given in the next paragraphs.

The first version of C# was released on 2001 and accordingly to Sharp [7], the language is a powerful language with a mixture of Java and Visual Basics, also the language uses *Microsoft .NET Framework*. This framework has to main components *Common Language Runtime (CLR)* and *Class Library*. The C# language presents a pre-compilation made by *Common Intermediate Language (CIL)* before the final compilation made by the CLR. This multi-step compilation processes let C# to have a consistent interaction with other programming languages as result the C# becomes more versatile.

The C# language supports the use of *Windows Presentation Foundation (WPF)*. The *WPF* is a very consistent tool for interface creation allowing the developer to program either in native C# or using *XAML (Extensible Markup Language)*. Yosifovich [8] defines *XAML* as a markup language based on *XML* and it has a great variety of markup tools for graphical interface building. With this language programming a text box, a button or a canvas is simple not requiring advanced programming skills. One important feature of *XAML* is the *binding* tool which allows the developer to associate a action on a object considering a action on other object (e.g. associate a variable with the state of a button).

OpenGL is a graphical *API (Application Programming Interface)* of high performance, free and open source. This *API* is implemented in C++ so it is possible to have direct access to graphical resources from the hardware. The most recent versions of *OpenGL* uses a core profile promoting more flexibility for developers once it is possible to access the graphical functions. Although this profile makes the use of the *API* difficult for new users. Through *OpenGL* it is possible to implement textures and illumination elements to the the graphical objects using the hardware resources directly. Since there is not a native integration between *OpenGL* and *WPF*, a library is needed. For this project the *SharpGL* library created was chosen. As result the great majority of *OpenGL* features can be used within *WPF*.

3.1 Calculated parameters

The parameters to be calculated by the software are the area, the centroid, and the polygonal diameter. To calculate the area, we use the Gaussian Area formula stated in the theorem below:

Theorem 2 Let P be a polygon with n vertices and $(x_i, y_i)_{i \in \mathbb{N}}$ the sequence of vertices coordinates of polygon P . The area of the polygon is calculated by:

$$A = \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_i + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right|. \quad (13)$$

The centroid $C = (C_x, C_y)$ can be calculates by:

$$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i), \quad (14)$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i). \quad (15)$$

3.2 Software description

The operation of the software is quite simple, meeting non-functional requirements. The user has the option to choose whether to read a text file with the polygon entries or to place each coordinate manually. In Fig. 1, the interface is shown and the input options are highlighted.



Figure 1. Graphical interface for VEM with two input option highlighted

To complement the description of the software, the five side polygon presented in da Veiga et al. [2] will be used. The format of the input file should be as shown in Fig. 3. It is important to notice the `#BEGIN` line and the `#END` line. The inclusion of these lines is mandatory, since the program uses them as a reference to delimit the beginning and end of the coordinates in the text file.

```

Five Side Polygon.txt - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda
#BEGIN
0.0+++++0.0
3.0+++++0.0
3.0+++++2.0
1.5+++++4.0
0.0+++++4.0
#END

```

Figure 2. Input file of a five side polygon

After entering the text file, the user must press the *Finish* button and the polygon will be drawn on the side panel, highlighting the vertices as shown in Fig. 3. The parameters mentioned in the previous section will be calculated as soon as the *Finish* button is pressed and the user has the option to save them in a text file, selecting this option in the toolbar at the top of the tool. The user can press the *Clear* button to clear this data entry and insert a new one. It is also possible to enter each coordinate manually. Just select the option *Manual Input*. Note that when this option is selected, it is not possible to enter data via text file. Thus, for each coordinate entered, the user must press the *Input* button. At the end, just press the *Finish* button.

For the five side polygon shown in Fig. 3, the following local stiffness matrix was obtained using the software:

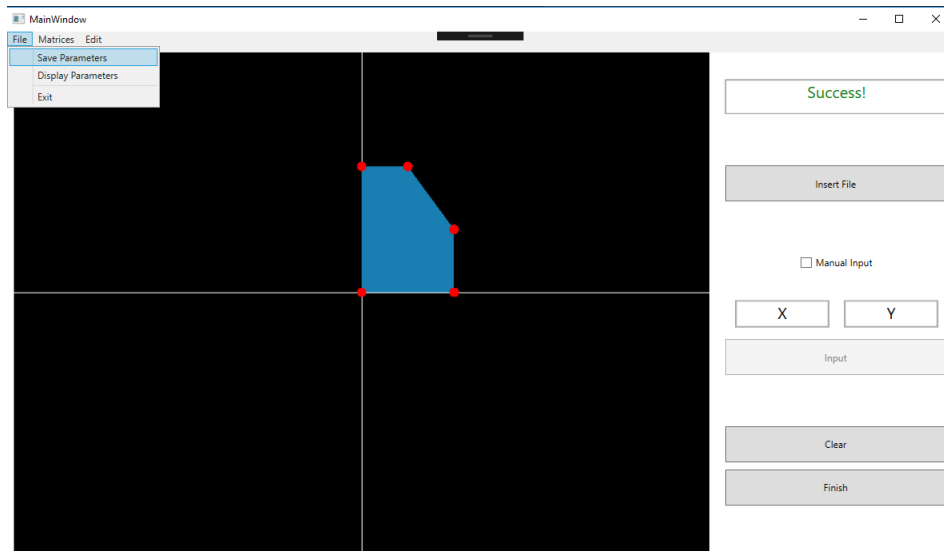


Figure 3. Five side polygons represented on the OpenGL panel

$$\mathbb{K}_{local} = \begin{bmatrix} 0,728331 & -0,20006 & -0,330772 & -0,247439 & 0,0499398 \\ -0,208223 & 0,739271 & -0,332438 & -0,126656 & -0,0719538 \\ -0,34982 & -0,343322 & 0,996123 & 0,0428919 & -0,345873 \\ -0,266486 & -0,13754 & 0,0428919 & 0,871123 & -0,509989 \\ 0,0417766 & -0,0719538 & -0,334989 & -0,499105 & 0,864271 \end{bmatrix}. \quad (16)$$

The geometrical parameters obtained were:

- Area: 10.5,
- Centroid: $C = (1.357, 1.809)$,
- Diameter: 5.

Those values are the same obtained by da Veiga et al. [2].

4 Conclusions

Virtual Element Method is a robust method for complex geometries. The discretization elements should be any simple polygon and no quadrature methods are needed to compute the shape functions. These functions are computed implicitly and are highly dependent of the degrees of freedom. The main goal of the work was to develop a graphical interface to ensure a robust data input for the Virtual Element Method once the method is very sensible considering the geometrical parameters. To build this tool, C# language alongside the graphical API OpenGL were used.

The presented software still in development. The next step consists on implement the Linear Elasticity Theory alongside VEM. In terms of software, it means that material properties will be implemented. Also, a more challenging aspect to be implemented is related to the mesh algorithm. For now, the software do not mesh the geometry. Based on human-computer interaction theory, a greater interaction between the user and the interface should be applied. As result, in the future the user will be able to draw the geometry by interacting directly with the OpenGL panel.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

- [1] da Veiga, L. B., Brezzi, F., Cangiani, A., Manzini, G., Marini, L. D., & Russo, A., 2013. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, vol. 23, pp. 199–214.
- [2] da Veiga, L. B., Brezzi, F., Marini, L. D., & Russo, A., 2014. The hitchhiker’s guide to the virtual element method. *Mathematical Models and Methods in Applied Sciences*, vol. 2, pp. 1541–1573.
- [3] Sutton, O., 2016. The virtual element method in 50 lines of matlab. *Numerical Algorithms*, vol. 75, pp. 1141–1159.
- [4] Ortiz-Bernardin, A., Alvarez, C., Hitschfeld-Kahler, N., Russo, A., Silva-Valenzuela, R., & Olate-Sanzana, E., 2019. Veamy: an extensible object-oriented c++ library for the virtual element method. *Numerical Algorithms*, vol. 82, pp. 1189–1220.
- [5] Mengolini, M., Benedetto, M. F., & Aragón, A. M., 2019. An engineering perspective to the virtual element method and its interplay with the standard finite element method. *Computer Methods in Applied Mechanics and Engineering*, vol. 350, pp. 995 – 1023.
- [6] Sommerville, I., 2016. *Software Engineering*. Pearson.
- [7] Sharp, J., 2013. *Microsoft Visual C# 2013 Step by Step*. Microsoft Press.
- [8] Yosifovich, P., 2012. *Windows Presentation Foundation 4.5 Cookbook*. Packt Publishing.