7 de Maio de 2019

1 Introdução

Um dos grandes objetivo da documentação em programação é o de facilitar a compreensão do código, seja por programadores que não estão familiarizados com este ou pelo próprio autor que está o a revisitar meses (ou anos) depois da sua escrita original. Neste sentido, a documentação deve ser tão detalhada quanto for necessário, mas mantendo-se concisa e sucinta.

Neste documento será descrita uma utilização básica de Doxygen, um software que transforma comentários escritos no próprio código em manuais de documentação. A sua utilização é simples e consiste essencialmente no uso de sintaxe específica que será mais tarde analisada e processada para criar um conjunto de páginas navegáveis.

1.1 O que é o Doxygen?

Doxygen é um gerador de documentação, que opera fazendo parse de comentários com sintaxe específica escritos no próprio código. Como resultado produz manuais navegáveis de utilização em diversos formatos, nomeadamente HTML, LaTeXe man-pages de Linux. Suporta múltiplas linguagens de programação como C/C++/C/Objective-C/PHP/Java e também linguagens de scripting como Python. Ao integrar a documentação diretamente no código, torna-se mais fácil mantê-la atualizada e facilita também a comunicação entre developers. Doxygen é utilizado por mútiplos projectos open-source, como a biblioteca de algebra linear Eigen, a biblioteca multimédia SFML entre muitos outros.

1.2 Instalação

Para instalar o Doxygen recomenda-se seguir as instruções em `http://www.doxygen.nl/manual/install.html` e o uso de binários pré-compilados.

2 Estrutura de comentários

Todos os comentários que se pretende que apareçam na documentação têm de ser escritos com sintaxe específica. Existem diversos estilos aceites pelo Doxygen.

Um dos formatos é o uso de comentários multi-line de C, com dois asteriscos:

```
/**
 * ... text ...
 */
```

Também é possível utilizar o triplo forward-slash:

```
///
/// ... text ...
///
```

E também um ponto de exclamação:

```
//!
//! ... text ...
//!
```

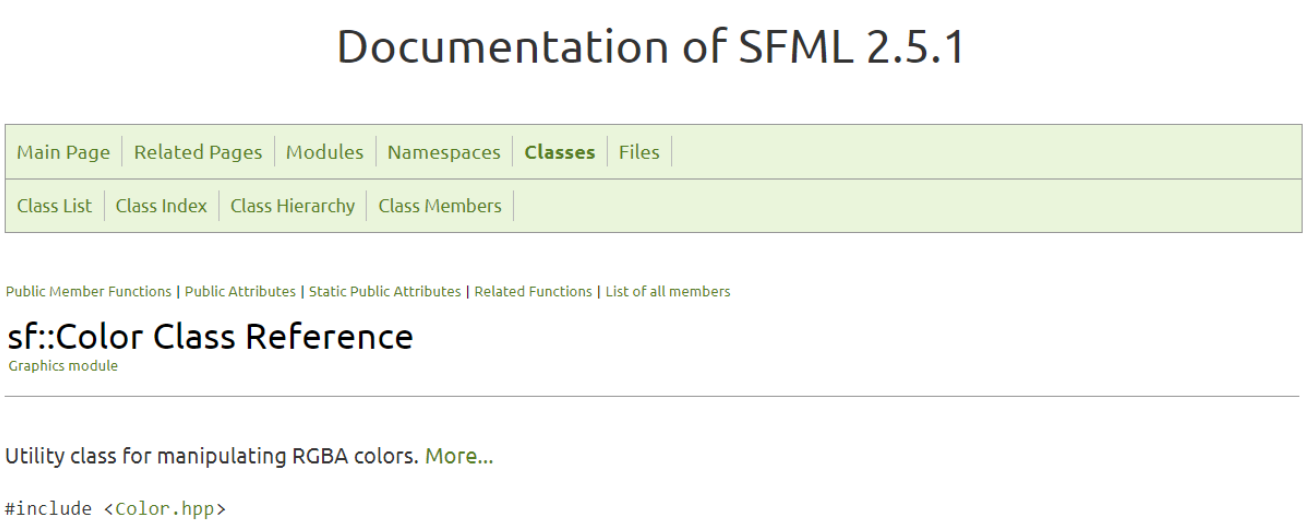
Uma lista completa de estilos encontra-se disponíveis para consulta na documentação oficial. Os comandos doxygen situam-se nestes comentários e são sempre precedidos por arrobas (@) ou back-slashes(\), como se tornará evidente nas próximas secções.

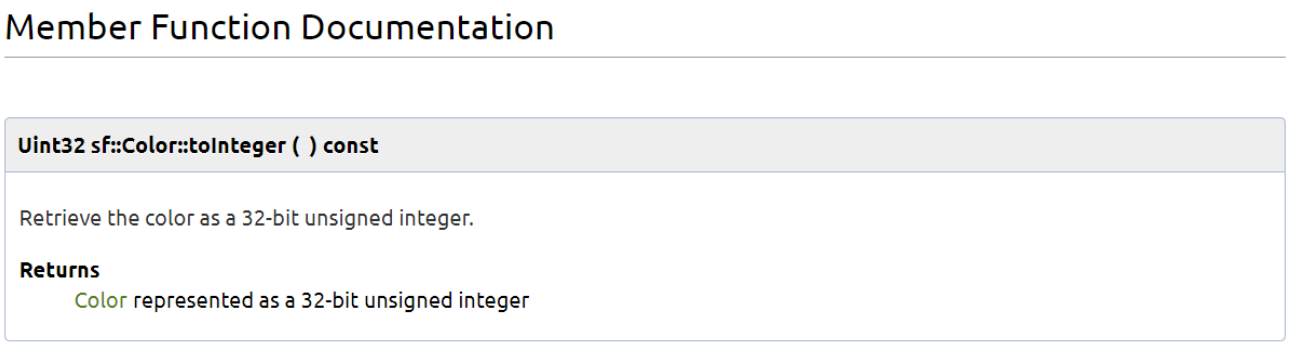
2.1 `brief`

O comando brief é dos mais utilizados e fornece precisamente uma definição breve da estrutura que se segue. Como tal é geralmente aplicado a estruturas de dados e funções. Segue-se como exemplo a simples classe Color da biblioteca SFML.

```
namespace sf
{
////////////////////////////////////////////////////////////
/// \brief Utility class for manipulating RGBA colors
///
////////////////////////////////////////////////////////////
class SFML_GRAPHICS_API Color
{
...
    ////////////////////////////////////////////////////////////
    /// \brief Retrieve the color as a 32-bit unsigned integer
    ///
    /// \return Color represented as a 32-bit unsigned integer
    ///
    ////////////////////////////////////////////////////////////
    Uint32 toInteger() const;
...
}
```

No exemplo da função toInteger() também é caracterizado o valor de retorno, com um comando semelhante `return`. Quando HTML é produzido pelo Doxygen (e é utilizado CSS para definição do estilo visual) obtém-se os seguintes resultados:





3 Execução do Doxygen

O Doxygen é um sistema flexível e altamente configurável. O processamento de geração é feito com base num extenso ficheiro de configuração. Um template deste ficheiro pode ser gerado pelo próprio doxygen com o seguinte comando:

```
$ doxygen -g [config filename]
```

O resultado é um ficheiro de texto com uma sintaxe simples, `CAMPO = VALOR`, pronto a ser ajustado. As configurações mínimas serão o nome do projeto e o caminho para o código fonte, sendo possível especificar ficheiros individuais ou directorias (e o percorrer recursivo).

Estado realizada a configuração podemos executar o doxygen com o seguinte comando.

```
$ doxygen [config filename]
```

Como consequência serão produzidas pastas para os diversos outputs escolhidos (HTML, LaTeX).

No entanto, a forma mais simples de executar o Doxygen é através do seu front-end gráfico, **Doxywizard**. Com este software torna-se mais fácil indicar ao Doxygen onde se encontra o nosso código fonte, optimizar o output para a linguagem de programação pretendida e escolher o tipo de output que pretendemos.

4 Exemplo prático

No exemplo que se segue vamos criar uma mini-biblioteca em C para operações com vetores e fazer a sua documentação com Doxygen.

Na raiz de uma biblioteca de matemática vetorial estará uma simples estrutura para armazenar um vetor, como por exemplo:

```
typedef struct
{
        float x;
        float y;
} vec2;
```

4.1 Anotar o ficheiro

Como primeira tarefa devemos etiquetar o ficheiro com nome do autor e código de versão. Desta forma sabemos quem culpar quando surgem problemas. O código de versão é relevante, pois quando informação se encontrar desatualizada

ficará claramente evidente. Neste caso `file` indica que o bloco de comentários é relativo ao próprio ficheiro e para as informações restantes temos `author` e `version`.

```
/**
        @file vecm.h
        @author Carlos Brito
        @version 1.0

        Defines some fundamental stuctures and functions for dealing with 2D vectors
*/

typedef struct
{
...
} vec2;
```

4.2 Anotar estruturas de dados

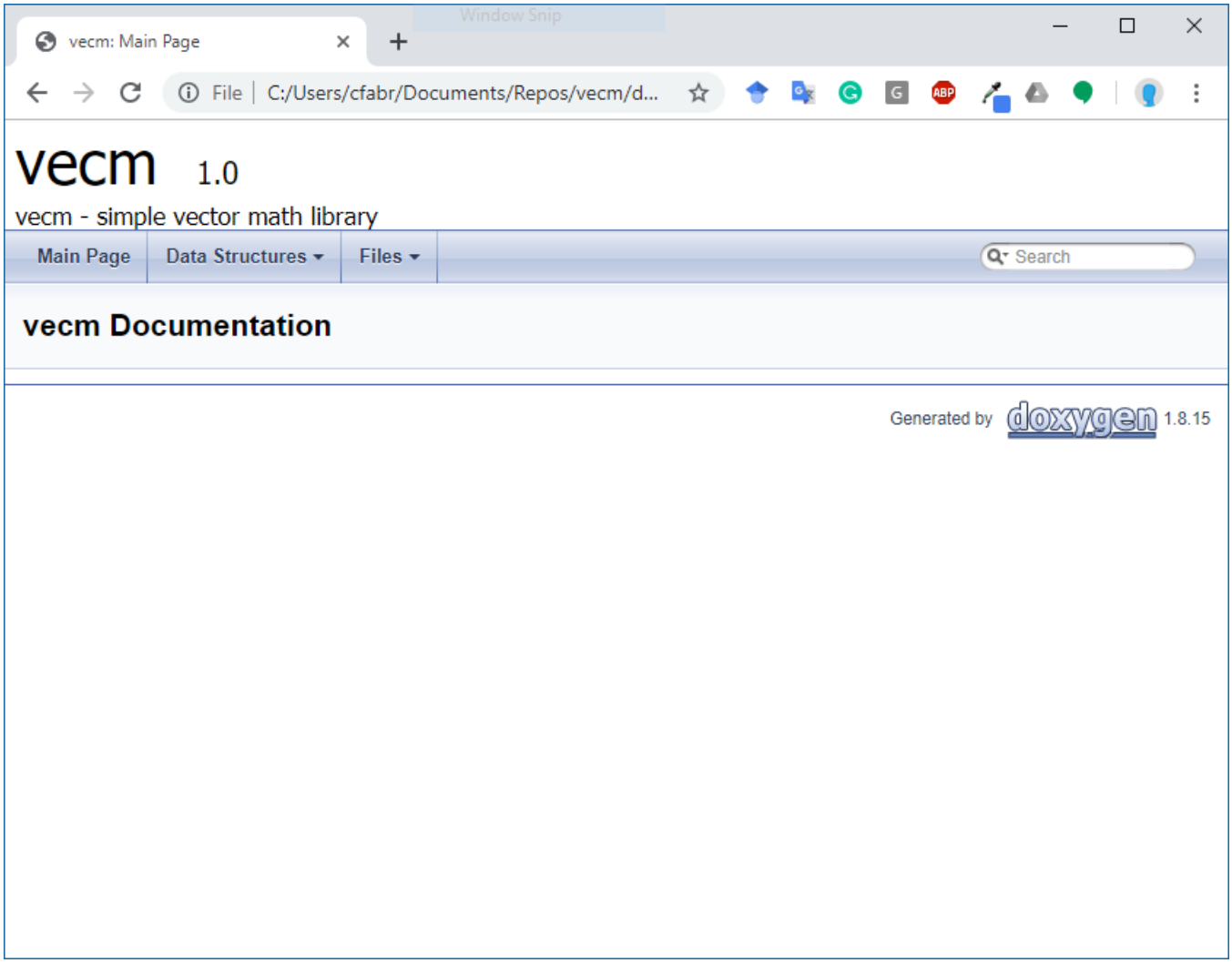
Apesar da class vec2 ser muito simples, vamos utiliza-la para demonstrar a escrita de documentação descritiva. As estruturas de dados ficam associadas aos comentários que as precedem. Usando a tag brief descrita anteriormente, fornecemos informação básica. Ao deixarmos uma linha em branco podemos escrever uma descrição mais detalhada.

```
///
/// @brief This structure encodes a two-dimentional vector
///
/// In 2-dimentions, vectors are described by a pair of coordinates
/// which represent their displacement along each axis
///
typedef struct
{
...
} vec2;
```

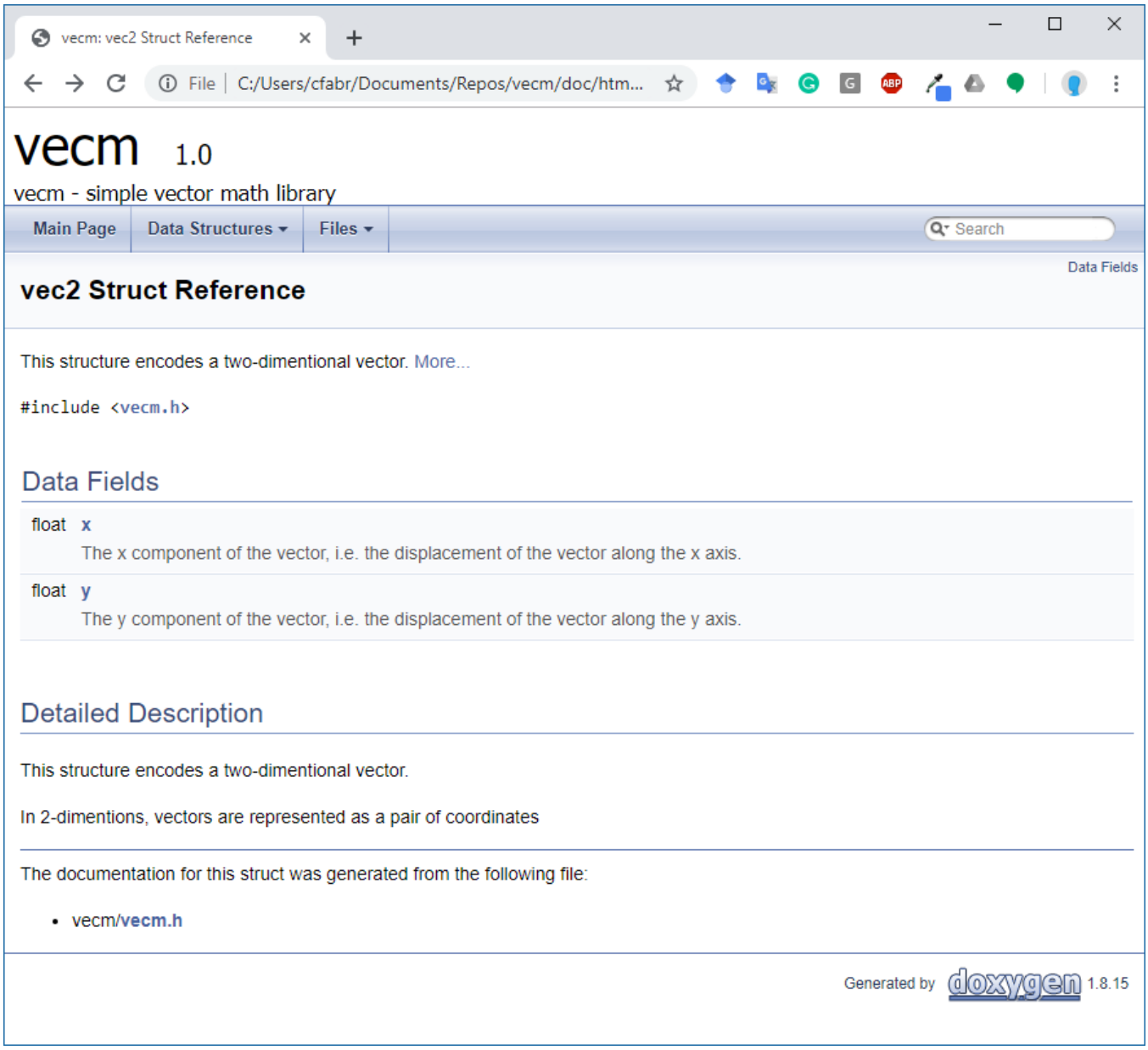
Por último na descrição de uma struct, convém indicar o que representa cada membro de dados, que neste contexto são apenas as coordenadas x e y. A sintaxe para tal consiste é acrescentar um `<` após o comentário Doxygen e situar o comentário na mesma linha ou na linha seguinte. Caso a dimensão do comentário seja restrita a uma linha, é considerada uma descrição breve e será apresentada de forma diferente na documentação final. No entanto uma descrição detalhada também é possível encadeando comentários.

```c
///
/// @brief This structure encodes a two-dimentional vector
///
/// In 2-dimentions, vectors are represented as a pair of coordinates
///
typedef struct
{
        float x; ///< The x component of the vector
        float y; ///< The y component of the vector
} vec2;
```

Executando o Doxygen com configuração mínima (Input optimizado para C) já obtemos uma página HTML navegável.



Acedendo a data-structures, já podemos visualizar as anotações que adicionamos à stuct vec2.



4.3 Anotar funções

Por último vamos acrescentar algumas funções à nossa biblioteca.

```
///
/// @brief This structure encodes a two-dimentional vector
///
/// In 2-dimentions, vectors are represented as a pair of coordinates
///
typedef struct
{
        float x; ///< The x component of the vector
        float y; ///< The y component of the vector
} vec2;

vec2 add(vec2 a, vec2 b);
vec2 sub(vec2 a, vec2 b);

float len(vec2 v);
vec2 normalize(vec2 v);
```

Seguindo o mesmo princípio, a documentação de funções pretende descrever ao programador como deve utilizar a função. Como tal, devem ser descritos os seus vários parâmetros, juntamente com o valor de retorno.

No caso de linguagens como C, em que valores podem ser passados por referência (e alterados) convém realçar quais são argumentos de input (`[in]`) e output (`[out]`). Outro tipo de considerações, como a existência de pré e pós condições devem ser descritas aqui também. Por exemplo, se a função alocou memória que deverá ser libertada, ou se um dado argumento não pode ser nulo.

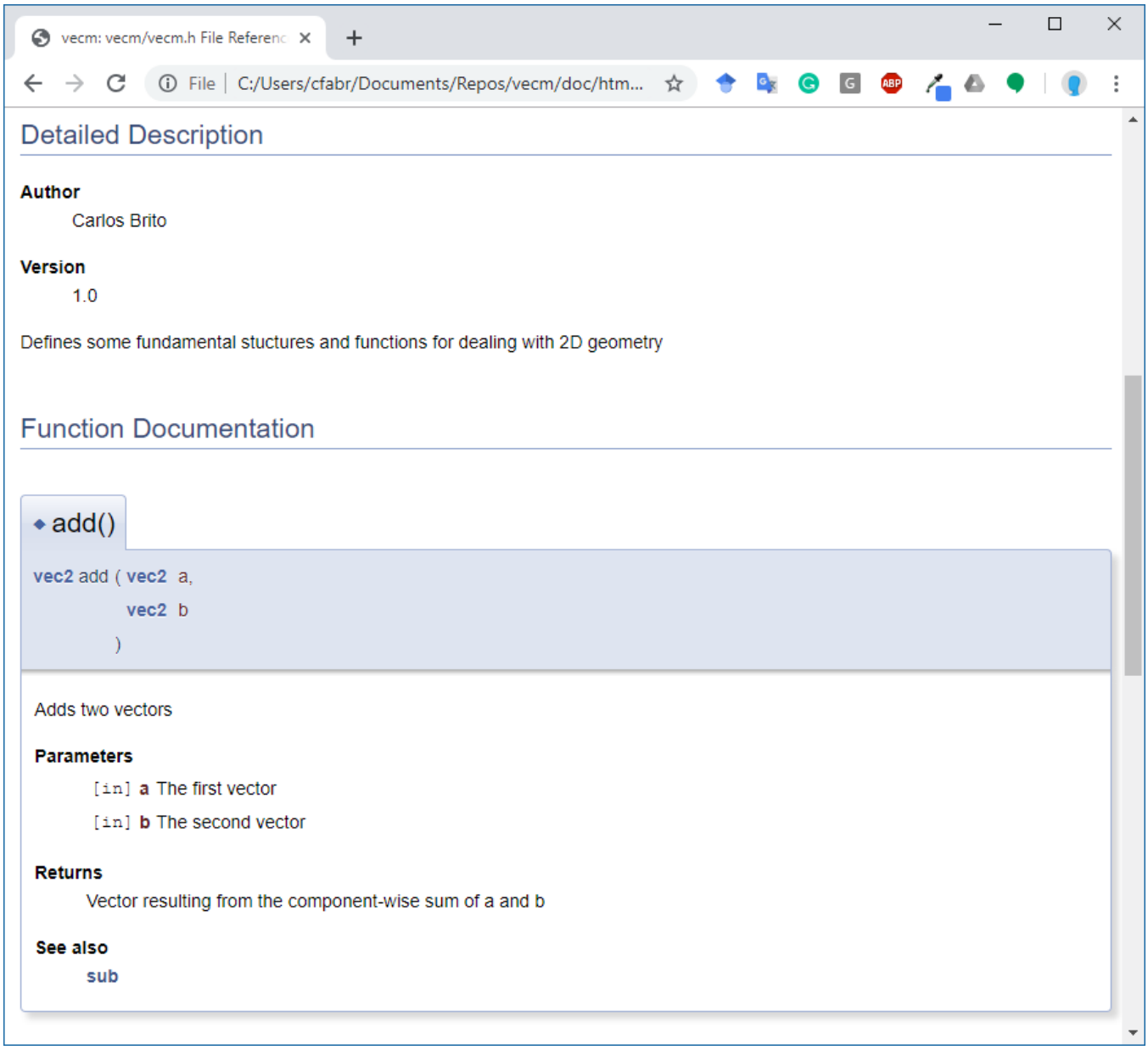
Em termos de sintaxe, são utilizados tags @param combinadas com os modificadores *in*, *out*, ou *in,out*:

```
/**
        @brief Adds two vectors

        @param[in] a The first vector
        @param[in] b The second vector

        @return Vector resulting from the component-wise sum of a and b
        @see sub
*/
vec2 add(vec2 a, vec2 b);
```

Neste exemplo, temos uma descrição breve, uma indicação do tipo e propósito dos argumentos, uma descrição do valor de retorno e também uma nota de "See also"que aponta para a função de subtração. Com estas anotações, pretende-se ligar funções que sejam semelhantes ou que de alguma forma se encontrem relacionadas e possam ajudar a navegar e compreender melhor o código.



Também é possível colocar uma descrição detalhada, sendo necessário deixar uma linha em branco:

```
/**
        @brief Calculates the length of a vector

        Employs Pythagoras' theorem to calculate the length
        or norm of an input vector.

        @param [in] v The vector

        @return The length of the vector
        @see norm
*/
float len(vec2 v);
```

Por último também é possível anotar funções com bugs descobertos e funcionalidade por implementar, com as tags `@todo` e `@bug`.

4.4 Diagramas

Outra feature do Doxygen é a geração de diagramas. Isto é particularmente útil em C++ quando existe uma hierarquia de classes, no entanto também é possível gerar grafos de dependência de forma a visualizar quais ficheiros incluem e são incluídos por outros. Para um exemplo é possível consultar a documentação do Doxygen aqui.

5 Conclusão

As abordagens à escrita de documentação são muitas, variadas e em grande parte subjetivas. No entanto, será relativamente consensual entre todos os programadores que como primeira linha de documentação está o próprio código. Só o uso de nomes explicativos para variáveis e funções já facilita tremendamente a compreensão. Por outro lado, quando o código é complexo, devemos utilizar comentários para explicitar o raciocínio por detrás da sua escrita. Por muito elegante e legível que esteja o código, nunca será mais simples ler e processar mentalmente um bloco extenso de código do que ler uma linha de texto que resume o seu processo.

Regra geral comentários devem conter o porquê do código, ou em inglês *"comments should say "why", not "what"*. Porque o "what"é muitas vezes óbvio, considere-se o exemplo extremo

```
// Guarda o resultado
int r = 1;

// Iterar n vezes
for(i=1; i<=n; ++i)
{
    r *= i;
}

return r;
```

Considere-se a alternativa

```
// 0! = 1
int factorial = 1;

// n! = n * (n-1)!
for(i=1; i<=n; ++i)
{
    factorial *= i;
}

return r;
```

Claro que este exemplo é fabricado, mas ilustra o risco de escrever comentários que são inúteis e apenas servem para poluir o código, e que o nome de uma variável pode ser o suficiente para explicitar o que o código está a fazer. Por outro lado, comentários podem poupar muito trabalho ao leitor, por exemplo[1]:

```
/**
 * Returns the temperature.
 */
int get_temperature(void) {
    return temperature;
}
```

Considere-se a alternativa

```
/**
 * Returns the temperature in tenth degrees Celsius
 * in range [0..1000], or -1 in case of an error.
 *
 * The temperature itself is set in the periodically
 * executed read_temperature() function.
 *
 * Make sure to call init_adc() before calling this
 * function here, or you will get undefined data.
 */
int get_temperature(void) {
    return temperature;
}
```

Analisando o código iriamos eventualmente concluir o mesmo que se encontra no comentário, mas ao termos este comentário poupamos esse tempo e ginástica mental.

[1] Retirado de `https://hackaday.com/2019/03/05/good-code-documents-itself-and-other-hilarious-jokes-you-shouldnt-tell-yourself/`