**Source code for Finding Bilateral Symmetry Planes**

September 2016

We are interested in creating computer-based tools to help design engineers during the first stage of the design process, known as conceptual design. Hence, we develop Sketch-Based Modelling (SBM) systems. Our approach for SBM relies on finding cues or regularities, which are those sketch properties which reveal properties of the three-dimensional object depicted in the sketch. In this context, symmetry planes are cues. In particular, the *finding-symmetry* algorithm we consider here (SYM) is aimed at finding circuits of lines that represent contours of symmetry planes of the polyhedral shapes depicted by wireframe models of polyhedral objects.

Attached is a C++ implementation of the code of the new SYM algorithm.

Although most of the cues are mutually related, we intend to detect every cue using minimal information. For SYM to work, the input sketch must have been previously vectorised, so as to convert strokes of the sketch into lines of the line drawing. In addition, faces must have been detected. Hence, the input to the code is a 2D drawing, which includes a list of vertices, edges and faces in the following format:

* Coordinates of every vertex are stored in an instance of the POINT2D class. The set of all vertices is stored in a standard vector (std::vector <POINT2D>Vertex)
  + VertexCount= number of vertex in the line drawing
  + VertexX[i]= X coordinate of the i-th vertex
  + VertexY[i]= Y coordinate of the i-th vertex
* Edges are defined by way of their head and tail vertices. The set of all edge heads is stored in a standard vector (std::vector <long> EdgeU) and the set of all edge tails is stored in another standard vector (std::vector <long> EdgeV):
  + EdgeCount= number of edges in the line drawing
  + EdgeU[i]= Head vertex defining the i-th edge
  + EdgeVl[i]= Tail vertex defining the i-th edge
* Faces are defined by a closed sequence of edges. The information is stored in instances of a FACE class, (std::vector <FACE> Face), where every FACE instance includes:
  + int C= Number of edges/vertices that define the face
  + std::vector <int> E= List of edges that define the face
  + std::vector <int> V= List of vertices that define the face

Reader must note that the vectorisation of the sketch required by this approach includes previous merging of endpoints of all the edges that share nodes or vertices of the polyhedral shape (merging into a single vertex all endpoints that should be perceived by humans as a common junction).

The output is the list of symmetry planes, together with their merits:

* The symmetry planes are stored in nested vectors. Thus SymPlanes[i][j][k] contains the k-th corner of the j-th symmetry plane of the i-th subgraph. The information of symmetry planes is stored through the corners of the sides of the circuits. Corners may be vertices or midpoints. Vertices are stored by their numbers, while midpoints are stored by the number of the edge they belong to switched to negative and decreased by one unit (i.e. midpoint of edge 3 is stored as -4).
* The merits of the symmetry planes are stored in nested vectors. Thus SymMerits[i][j] contains the merit of the j-th symmetry plane of the i-th subgraph.

The approach is encapsulated into a main class CCueSymmetry, which shares the same file with some auxiliary classes.

This approach for finding symmetry is extensively described in:

Plumed R., Company P. and Varley P.A.C. (2016).  
Detecting mirror symmetry in single-view wireframe sketches of polyhedral shapes.  
Computers & Graphics, Vol. 59. pp. 1-12, 2016.

To help understanding how the code works, and in order to offer a test environment too, the following main file is also provided:

* Main.cpp, and its header Main.h.

Finally, we must highlight that the code was written to make it readable. Efficiency never was a goal.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The SYM code is free software. But if you find it useful for your own research,  
please cite our paper:

Plumed R., Company P. and Varley P.A.C. (2016).  
Detecting mirror symmetry in single-view wireframe sketches of polyhedral shapes.  
Computers & Graphics, Vol. 59. pp. 1-12, 2016.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Código Fuente para encontrar planos de simetría bilateral**

Septiembre 2016

Estamos interesados en crear herramientas basadas en computador para ayudar a los ingenieros de diseño durante la primera etapa del proceso de diseño, conocida como diseño conceptual. Por tanto, desarrollamos sistemas de modelado basado en bocetos (SBM por sus siglas en inglés). Nuestra aproximación al SBM se basa en encontrar indicios o regularidades, que son aquellas propiedades del boceto que revelan propiedades del objeto tridimensional representado en el boceto. En este contexto, los planos de simetría son indicios. En concreto, el algoritmo para *encontrar planos de simetría* que consideramos aquí (SYM) está encaminado a encontrar circuitos de líneas que representan contornos de planos de simetría de las formas poligonales representadas mediante modelos de alambres de objetos poliédricos.

Se adjunta una implementación en C++ del código del nuevo algoritmo SYM.

[Usted puede descargar el código de SYM desde aquí](http://www.regeo.uji.es/FindingSymmetry.zip).

Aunque muchos de los indicios están relacionados mutuamente, nosotros intentamos detectar cada indicio usando información mínima. Para que SYM funcione, el boceto de entrada debe haber sido previamente vectorizado, para convertir los trazos del boceto en líneas del dibujo lineal. Además, las caras deben haber sido detectadas. Por tanto, la entrada para el código es un dibujo 2D, que incluye una lista de vértices, ejes y caras en el siguiente formato:

* Las coordenadas de cada vértice se guardan en una instancia de la clase POINT2D. El conjunto de todos los vertices se guarda en un vector estándar (std::vector <POINT2D>Vertex)
  + VertexCount= número de vertices del dibujo lineal
  + VertexX[i]= coordenada X del i-esimo vértice
  + VertexY[i]= coordenada Y del i-esimo vértic
* Las aristas se definen mediante sus vértices de cabeza y cola. El conjunto de todas las cabezas de aristas se guarda en un vector estándar (std::vector <long> EdgeU) y el conjunto de todas las colas de aristas se guarda en otro vector estándar (std::vector <long> EdgeV):
  + EdgeCount= número de aristas del dibujo lineal
  + EdgeU[i]= Vértice de cabeza que define a la i-esima arista
  + EdgeVl[i]= Vértice de cola que define a la i-esima arista
* Las caras se definen mediante una secuencia cerrada de aristas. La información se guarda en instancias de la clase FACE class, (std::vector <FACE> Face), donde cada instancia FACE incluye:
  + int C= número de aristas/vertices que definen la cara
  + std::vector <int> E= lista de aristas que definen la cara
  + std::vector <int> V= lista de vertices que definen la cara

El lector debe notar que la vectorización del boceto requerida para que funcione éste método incluye la fusión previa de los extremos de todas las aristas que comparten vértices de la forma poliédrica (fusionando en un único vértice todos los extremos de aristas que son percibidos por los seres humanos como una esquina o unión común).

La salida es la lista de planos de simetría, juntamente con sus méritos:

* Los planos de simetría se guardan en vectores anidados. Así, SymPlanes[i][j][k] contiene la esquina k-esima del j-esimo plano de simetría del i-esimo sub-grafo. La información de los planos de simetría se almacena a través de las esquina de los lados del circuito. Las esquinas pueden ser vértices o puntos medios. Los vértices se almacenan mediante su número, mientras que los puntos medios se almacenan mediante el número de la aristas a la que pertenecen conmutado a negativo y decrementado en una unidad (por ejemplo, el punto medio de la arista 3 se guarda como -4).
* Los méritos de los planos de simetría se guardan como vectores anidados. Así, SymMerits[i][j] contiene el mérito del j-esimo plano de simetría del i-esimo subgrafo.

El método está encapsulado dentro de una clase principal CCueSymmetry, que comparte fichero con algunas clases auxiliares.

Este método para encontrar planos de simetría se describe extensamente en:

Plumed R., Company P., y Varley P.A.C. (2010). Detecting mirror symmetry in single-view wireframe sketches of polyhedral shapes.  
Computers & Graphics, Vol. 59. pp. 1-12, 2016.

Para ayudar a entender cómo funciona el código, y para ofrecer un entorno de prueba, se suministra también el siguiente fichero:

* Main.cpp, y su fichero de encabezamiento Main.h.

Finalmente, queremos remarcar que el código fue escrito para hacerlo legible. La eficiencia nunca fue un objetivo.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

El código SYM es software libre. Pero si usted lo encuentra útil para su propia investigación, por favor cite nuestro artículo:

Plumed R., Company P. and Varley P.A.C. (2016).  
Detecting mirror symmetry in single-view wireframe sketches of polyhedral shapes.  
Computers & Graphics, Vol. 59. pp. 1-12, 2016.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*