**Source code for Finding Perimeters in sketched drawings of polyhedral shapes**

September 2017

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, perimeters of sketched drawings of polyhedral shapes are cues. In particular, the *finding-perimeters* algorithm we consider here (PER) is aimed at finding circuits of lines and junctions (including intersections and T-junctions) of the original line drawing that belong to the contour of the polyhedral shapes depicted by wireframe models of polyhedral objects.

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

[You can download the PER code from here](http://www.regeo.uji.es/FindingPerimeter.zip).

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

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 of the algorithm is a list of ordered lines and corners that belong to the perimeter:

* The edges that belong to the perimeter are sequentially ordered in PerimeterLines.
* The corners (junctions, intersections or T-junctions) that belong to the perimeter are sequentially ordered in PerimeterCorners. Positive numbers in the list of corners refer to junctions (vertices) of the original line drawing, while negative numbers are pointers to a list of intersections or T-junctions that belong to the perimeter. Note that the distinction between intersections and T-junctions is simple: the intersection becomes a T-junction if it is close to one vertex of the outgoing edge.
* The coordinates of the intersections and T-junctions that belong to the perimeter are saved (list TX), as much as the lines that produce each intersection (list TEdges)

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

This approach for finding perimeters is extensively described in:

Company P., Varley P.A.C., and Plumed R. (2017).  
Perimeter detection in sketched drawings of polyhedral shapes.  
STAG: Smart Tools and Applications in Graphics (2017).  
A. Giachetti, P. Pingi and F. Stanco (Editors).

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

Company P., Varley P.A.C., and Plumed R. (2017).  
Perimeter detection in sketched drawings of polyhedral shapes.  
STAG: Smart Tools and Applications in Graphics (2017).  
A. Giachetti, P. Pingi and F. Stanco (Editors).

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

**Código Fuente para encontrar perímetros en dibujos bocetados de formas polihédricas**

Septiembre 2017

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 perímetros de los dibujos bocetados de formas poliédricas son indicios. En concreto, el algoritmo para *encontrar perímetros* que consideramos aquí (PER) está encaminado a encontrar circuitos de líneas y uniones (incluyendo intersecciones y uniones en T) del dibujo líneas original que pertenecen al contorno de las formas poliédricas representadas mediante modelos de alambres de objetos poliédricos.

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

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

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

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 ordenadas de líneas y esquinas que pertenecen al perímetro:

* Las aristas que pertenecen al perímetro están ordenadas en secuencia en PerimeterLines.
* Las esquinas (uniones, intersecciones o uniones en T) que pertenecen al perímetro están ordenadas en secuencia en PerimeterCorners. Los números positivos en la lista de esquinas se refieren a uniones (vértices) del dibujo lineal original, mientras que los números negativos son punteros a una lista de intersecciones y uniones en T que pertenecen al perímetro. Note que la distinción entre intersecciones y uniones en T es simple: las intersecciones devienen en uniones si están suficientemente cerca de alguno de los vértices de la arista saliente.
* Las coordenadas de las intersecciones y uniones en T que pertenecen al perímetro se salvan (lista TX), tanto como las líneas que producen cada intersección (lista TEdges).

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

Este método para encontrar perímetros se describe extensamente en:

Company P., Varley P.A.C., and Plumed R. (2017).  
Perimeter detection in sketched drawings of polyhedral shapes.  
STAG: Smart Tools and Applications in Graphics (2017).  
A. Giachetti, P. Pingi and F. Stanco (Editors).

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

Company P., Varley P.A.C., and Plumed R. (2017).  
Perimeter detection in sketched drawings of polyhedral shapes.  
STAG: Smart Tools and Applications in Graphics (2017).  
A. Giachetti, P. Pingi and F. Stanco (Editors).

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