**Source code for Detecting Vertices in sketched drawings of polyhedral shapes**

November 2018

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, junctions of sketched drawings are cues which depict vertices of polyhedral shapes. In particular, the *detecting-vertices* algorithm we consider here (JUNC) is aimed to merge the 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).

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

[You can download the JUNC code from here](DetectVertices.sln).

Although most of the cues are mutually related, we intend to detect every cue using minimal information. The input sketch comprise only discrete strokes, sequences of points obtained between a pen down and a pen up. For JUNC to work, tips or endpoints of each segment line need to be previously filtered by applying some corner detection approach. Some of these known approaches are:

* IStraw (implemented by Xiong Y., LaViola J.: *A shortstraw-based algorithm for corner finding in sketch-based interfaces.* Computers and Graphics 34(5) (2010), 513–27);
* Shortstraw (Wolin A., Eoff B., Hammond T.*: Shortstraw: a simple and effective corner finder for polylines.* SBIM 2008 (2008), 33–40);
* Sliding strips (proposed by Masood A., Sarfraz M*.: Corner detection by sliding rectangles along planar curves.* Computers and Graphics 31 (2007), 440–48).

Hence, the input to the code is a 2D drawing, which includes a list of tips of line segments 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 vertices in the drawing
  + Vertex[i].x= X coordinate of the i-th vertex
  + Vertex[i].y= Y coordinate of the i-th vertex
* Edges are defined by means of their head and tail vertices, which correspond with the endpoints of each stroke. The set of all edge heads is stored in a standard vector (std::vector <long> EdgeHead) and the set of all edge tails is stored in another standard vector (std::vector <long> EdgeTail):
  + EdgeCount= number of edges in the line drawing
  + EdgeHead[i]= Head junction defining the i-th edge
  + EdgeTail[i]= Tail junction defining the i-th edge

Reader must note that examples attached (corresponding to the examples of the technical report) have been previously filtered by the IStraw approach, to find the corners of the strokes which define the endpoints of each segment line.

The output of the algorithm is a list of vertices, which depict junctions in the 3D model, and a list of edges:

* For each vertex, the algorithm returns its coordinates and the figure of merit that estimates how likely a vertex is to be perceived as a junction by human beings.
* The list of edges with their endpoints updated according to the new vertices.

The approach is encapsulated into a main class CueJunctions. Two more files containing auxiliary classes and operations are required for the approach to work:

* Tools\_Geometry.cpp, and its corresponding header Tools\_Geometry.h.
* Tools\_Vector.cpp, and its corresponding header Tools\_Vector.h.

This approach for detecting junctions is extensively described in:

Company P., Varley P.A.C., Plumed R. and Camba J.D. (2018).  
Detection of vertices in sketched drawings of polyhedral shapes.  
CIARP: 23rd Iberoamerican Congress on Pattern Recognition (2018).

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.

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

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

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

Company P., Varley P.A.C., Plumed R. and Camba J.D. (2018).  
Detection of vertices in sketched drawings of polyhedral shapes.  
CIARP: 23rd Iberoamerican Congress on Pattern Recognition (2018).

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

**Código Fuente para detectar vértices en dibujos bocetados de formas poliédricas**

Noviembre 2018

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, las uniones de los dibujos bocetados de formas poliédricas son indicios que describen los vértices de figuras poliédricas. En concreto, el algoritmo para *detectar vértices* que consideramos aquí (JUNC) está encaminado a fusionar los extremos de todas las aristas que comparten nodos o 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)

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

[Usted puede descargar el código de JUNC 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. El boceto de entrada comprende sólo trazos discretos, secuencias de puntos adquiridos entre los movimientos de bajada y subida de un lápiz. Para que JUNC funcione, los extremos de cada línea de segmento deben ser previamente filtrados aplicando algún método de detección de esquinas. Algunos de estos métodos conocidos son:

* IStraw (implementado por Xiong Y., LaViola J.: *A shortstraw-based algorithm for corner finding in sketch-based interfaces.* Computers and Graphics 34(5) (2010), 513–27);
* Shortstraw (Wolin A., Eoff B., Hammond T.*: Shortstraw: a simple and effective corner finder for polylines.* SBIM 2008 (2008), 33–40);
* Sliding strips (propuesto por Masood A., Sarfraz M*.: Corner detection by sliding rectangles along planar curves.* Computers and Graphics 31 (2007), 440–48).

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 vértices se guarda en un vector estándar (std::vector <POINT2D>Vertex)
  + VertexCount= número de vértices del dibujo lineal
  + VertexX[i]= coordenada X del i-ésimo vértice
  + VertexY[i]= coordenada Y del i-ésimo vértice
* 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> EdgeHead) y el conjunto de todas las colas de aristas se guarda en otro vector estándar (std::vector <long> EdgeTail):
  + EdgeCount= número de aristas del dibujo lineal
  + EdgeHead[i]= Vértice de cabeza que define a la i-ésima arista
  + EdgeTail[i]= Vértice de cola que define a la i-ésima arista

El lector debe notar que los ejemplos adjuntos (correspondientes a los ejemplos del informe técnico) han sido filtrados previamente con el método IStraw para encontrar las esquinas de los trazos que definen los extremos de cada línea de segmento.

La salida del algoritmo es una lista de vértices, que describen uniones en el modelo 3D, y una lista de aristas:

* Para cada vértice, el algoritmo devuelve sus coordenadas y la figura de mérito que estima la probabilidad de que un vértice sea percibido como una union por los seres humanos.
* La lista de aristas con sus extremos actualizados de acuerdo a los nuevos vertices.

El método está encapsulado dentro de una clase principal CCuePerimeter. Otros dos ficheros contienen clases y operaciones auxiliares requeridas para el funcionamiento del método:

* Tools\_Geometry.cpp, y su correspondiente cabecera Tools\_Geometry.h.
* Tools\_Vector.cpp, y su correspondiente cabecera Tools\_Vector.h.

Este método para detectar vértices se describe extensamente en:

Company P., Varley P.A.C., Plumed R. and Camba J.D. (2018).  
Detection of vertices in sketched drawings of polyhedral shapes.  
CIARP: 23rd Iberoamerican Congress on Pattern Recognition (2018).

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

* Main.cpp.

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

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

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

Company P., Varley P.A.C., Plumed R. and Camba J.D. (2018).  
Detection of vertices in sketched drawings of polyhedral shapes.  
CIARP: 23rd Iberoamerican Congress on Pattern Recognition (2018).

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