

Source code for finding Parallel edges by vanishing points (PEVP)

August 2014

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, parallel edges are cues. In particular, the *parallel-edges* algorithm we consider here (PEVP) is aimed at grouping lines of the sketch that depict edges that are thought to be parallel in the 3D model represented by the sketch.

Attached is a C++ implementation of the new PEVP algorithm.

[You can download the PEVP code from here.](#)

Although most of the cues are mutually related, we intend to detect every cue using minimal information. For FF 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
 - Vertex[i].x= X coordinate of the i-th vertex
 - Vertex[i].y= 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> 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 vertex defining the i-th edge
 - EdgeTail[i]= Tail vertex defining the i-th edge

Reader must note that the vectorisation of the sketch required by this approach is minimal, since endpoints of different lines that should be perceived by humans as defining a common junction are not necessarily merged in a single vertex. Hence, it may well happen that every edge endpoint defines a different vertex.

The output is a list of groups of lines of the drawing parallel to each other:

- The information is stored in a standard vector of vectors (`std::vector <std::vector <long>> ParallelEdges`), where `ParallelEdges[i][j]` contains the number of the j-th edge of the i-th group of parallel edges.

The approach is encapsulated in two classes:

- CCueParallelEdges class is intended to be called from the external application.
- CCueVanishingPoints class is called from CCueParallelEdges, although it can also be directly called from the external application, in case only vanishing points approach is used.

The first class includes two different approaches to detect parallel edges. The first one is our own implementation of the Angular Distribution Graph (ADG) proposed by Lipson and Shpitalni in the following reference:

H Lipson, M Shpitalni. (1996). Optimization-based reconstruction of a 3D object from a single freehand line drawing. Computer-Aided Design, 28(8) 651-663.

The second one is our approach for finding vanishing points, described in:

P. Company, P.A.C. Varley, R. Plumed (2014). An algorithm for grouping lines which converge to vanishing points in perspective sketches of polyhedral. LNCS 8746, pp. 1-19.

The third approach is automatic, and uses ADG for normalon objects (also known as “Manhattan like” objects or scenes) depicted in parallel projection, and uses Vanishing Points otherwise. This automatic approach considers that a drawing depicts a normalon shape if (a) three and only three prevailing angles exist, and (b) all the edges fit in one of the three main directions.

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.

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

- MainParallelEdges.cpp, and its header MainParallelEdges.h.

Finally, we must highlight that the code was written to enhance readability. Efficiency never was a goal.

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

P. Company, P.A.C. Varley, R. Plumed (2014) An algorithm for grouping lines which converge to vanishing points in perspective sketches of polyhedral. LNCS 8746, pp. 1-19.

Código Fuente para encontrar aristas paralelas mediante puntos de fuga (PEVP)

Agosto 2014

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 aristas paralelas son indicios. En concreto, el algoritmo de *aristas-paralelas* que consideramos aquí (PEVP) está encaminado a agrupar líneas del boceto que representan aristas que se cree que son paralelas en el modelo 3D representado por el boceto, en representaciones de objetos poliédricos.

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

[Usted puede descargar el código de FF desde aquí.](#)

Aunque muchos de los indicios están relacionados mutuamente, nosotros intentamos detectar cada indicio usando información mínima. Para que PEVP 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 ejes 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 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-esima arista
 - EdgeTail[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 es mínima, porque los extremos de diferentes líneas que serían percibidas por los humanos como una esquina o unión común no necesitan estar fusionados en un único vértice. Por tanto, puede ocurrir que cada extremo de arista defina un vértice diferente.

La salida es una lista de grupos de líneas del dibujo paralelas entre sí.

- La información se guarda en un vector estándar, (std::vector <std::vector <long>> ParallelEdges), donde ParallelEdges[i][j] contiene el número de la j-esima arista del i-esimo grupo de aristas paralelas.

El método está encapsulado en dos clases:

- La clase CCueParallelEdges está prevista para ser llamada desde la aplicación externa.

- La clase CCueVanishingPoints debe ser llamada desde CCueParallelEdges, aunque también puede ser llamada directamente desde la aplicación externa, en el caso de que sólo se use el método de puntos de fuga.

La primera clase incluye dos métodos diferentes para detectar aristas paralelas. La primera es nuestra propia implementación del Grafo de Distribución Angular (ADG) propuesto por Lipson y Shpitalni en la siguiente referencia:

H Lipson, M Shpitalni. (1996). Optimization-based reconstruction of a 3D object from a single freehand line drawing. Computer-Aided Design, 28(8) 651-663.

El segundo es nuestro propio método para encontrar puntos de fuga, descrito en:

P. Company, P.A.C. Varley, R. Plumed (2014). An algorithm for grouping lines which converge to vanishing points in perspective sketches of polyhedral. LNCS 8746, pp. 1-19.

El tercer método es automático, y utiliza ADG para objetos normalones (también conocidos como objetos o escenas "Manhattan like") dibujados en proyección paralela, y utiliza Puntos de Fuga en el resto de casos. El método automático considera que el dibujo representa una forma normalon si (a) existen tres y solo tres ángulos prevalentes, y (b) todas las aristas encajan en alguna de las tres direcciones principales.

Para funcionar, el método requiere otros dos ficheros que contiene clases y operaciones auxiliares:

- Tools_Geometry.cpp, y su correspondiente encabezamiento Tools_Geometry.h.
- Tools_Vector.cpp, y su correspondiente encabezamiento Tools_Vector.h.

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

- MainParallelEdges.cpp, y su fichero de encabezamiento MainParallelEdges.h.

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

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

P. Company, P.A.C. Varley, R. Plumed (2014) An algorithm for grouping lines which converge to vanishing points in perspective sketches of polyhedral. LNCS 8746, pp. 1-19.

