FACET DETECTION AND VISUALIZING LOCAL STRUCTURE IN GRAPHS

Dana Vrajitoru
Intelligent Systems Laboratory
Computer and Information Sciences
Indiana University South Bend
South Bend, IN 46634, USA
email: danav@cs.iusb.edu

ABSTRACT
In this paper we present an algorithm for facet detection in graphs. We define the facets as simple cycles in the graph minimizing the length. Our algorithm detects the local structure of simple cycles intersecting in a particular vertex of the graph. For this purpose, we reduce the problem to an instance of the Traveling Salesman in an induced co-cyclic graph and use a method inspired from the minimum spanning tree to construct a circuit solution. Our algorithm found a correct solution for most of the cases we tested it on.

KEY WORDS
graph visualization, facet detection, traveling salesman

1 Introduction

In this paper we present an algorithm for facet detection in a graph, defined in terms of a set of simple cycles intersecting in a particular vertex of the graph. The goal of the algorithm is primarily to emphasize the local structure in the graph concerning a particular vertex, which can provide a better understanding of the graph by means of visualization. Other applications are also foreseeable for this algorithm, for example, computing the total surface of the graph.

Facet detection algorithms have many applications. Recently, they have been employed for detecting and tracking features in 2D images and in animations [10], and for automatic object recognition [5].

It is often the case that surfaces are initially defined as clouds of points [1], in particular when they represent implicit surfaces or when they are captured by a 3D scanner. The most often, a mesh is reconstructed from the cloud of points which can take an intermediate step in the form of a graph [3]. In this study, a cloud of points obtained from a 3D scanning device is first transformed into a graph. A second step constructs the surface by a facet detection algorithm.

A related problem is the edge detection in images [11], which is often based on pixel analysis. The facet detection methods consider the facets as collections of edges with particular properties, for example, convexity in terms of polygons. Zheng and Tian [15] for example, start with an edge detection algorithm, followed by facet detection using a combination of gradient based parametric facet detection and zero crossing for the edges. In a related study [9], features are extracted by hysteresis thresholding followed by a step consisting in reconstructing a minimum spanning graph to emphasize contours for 2D and 3D data. In this study, the cycles represent an important criterion to recognizing features.

The idea of feature extraction is closely related to the notions of facet and polygon. Iqbal and Aggarwal [7] propose detecting structures in two dimensional images by extracting line segments and then building a cotermination graph. The method employs graph theory to detect polygons in the image as fundamental circuits in the graph. A maximum spanning tree is used for the detection, making use of the spatial representation of the graph.

In this paper we propose a method for extracting local structures in graphs. We start from the premise that the graph represents a solid object with a well-defined surface. We assume that the surface itself is not known, and that we do not possess information about the geometry of the graph. We are interested in the intrinsic local structure of the graph outside of the geometric considerations.

The algorithm we propose follows the general idea that a facet is a simple cycle in a graph with some properties of minimality. In a related paper [3], Azernikov and Fischer start from an edge to detect a facet, using properties of convexity of the resulting polygon and minimizing the length of the cycle. We follow a different approach, going from all the simple cycles intersecting in a vertex to selecting those that might define the actual surface. Our approach differs from that of related work in the sense that we do not use the geometric representation of the graph, but only the internal structure in terms of edge connections.

Our algorithm focuses on one vertex at a time and detects the local polygons that the vertex belongs to using a minimal cycle method. The set of all such cycles are meant to represent the surface of the object represented by the graph. They constitute a minimal neighborhood of the object that is topologically open. In the current form we do not consider information about the weights in the graph, but the algorithm can be extended to find the minimal cycle set in terms of weights.

The main application of this algorithm is to emphasize the local structure in the graph for purposes of visualization. As a potential additional application, it can be used to compute the total surface of the graph. Efficient solutions have been proposed for meshes [14], starting from the assumption that the polygons composing the surface have already been defined. Our algorithm can be used as a preliminary step for this problem.

2 Cycle Detection and Co-Cyclic Graph

Definition. *Two vertices in a graph are* **co-cyclic** *if they belong to a common simple cycle.*

We use the notion of co-cyclic vertices in the sense most often used in chemistry and not in relation to the notion of cocyle in a graph [13].

We will start by analyzing the subset of the cycle space containing all the simple cycles that intersect in a given vertex of the graph, or origin. From this subset we can construct a weighted co-cyclic graph where an edge is drawn between two vertices if they are both adjacent to the origin and co-cyclic. The weight of the edge will be the length of the smallest cycle containing both vertices.

The co-cyclic graph can be used to extract the cycle structure or the facets that intersect in the origin.

2.1 Cycle Detection

The issue of detecting a cycle in a graph seems rather trivial, but it is still worth mentioning for the sake of presenting a complete algorithm. We will consider the case of an undirected graph. For directed graphs the cycle detection is more complex and requires a topological sorting algorithm.

Let $G = \{V, E\}$ be an undirected graph with n vertices and m edges.

Existence

The test for the existence of a cycle in a graph is linear over the number of vertices n. In an acyclic undirected graph, $m \leq n - 1$. This is a necessary but not sufficient condition for the graph to be acyclic. In the case where this condition is true, we still need to verify the existence of cycles with a breadth-first search, which is linear over the number of edges m. The necessary condition insures thus that the test is linear over n.

Detection Algorithm

The cycle detection algorithm requires a small modification to a classic breadth-first search (BFS) [12]. The BFS explores the vertices in the graph by moving from a vertex to its neighbors in the order of the distance to the origin of the search in terms of number of edges. A queue is used to store the visited vertices that may still have unprocessed neighbors. If we need to store the path from the origin to a destination vertex in the graph, then for each vertex we record its predecessor in the search. Traditionally, the BFS marks the visited vertices to avoid cycles. The same mark can be used to detect the cycles in our case.

Let us denote by o the origin of the BFS. Let us consider a particular step of the BFS where the current vertex is x and y is one of its neighbors. The following three cases can occur:

1. y has not been marked yet. In this case y is a vertex not yet visited and the BFS can continue the search by adding it to the queue and marking it.

2. $y = predecessor(x)$ or $x = predecessor(y)$. In this case the algorithm has found a cycle by retracing one of its steps. Since this would not be a simple cycle, we can discard the entry y.

3. Otherwise y is a vertex that has been encountered before on a different path. Thus, by concatenating these two paths by which we reached y, we close a cycle.

If our goal is to detect the existence of non-trivial cycles in the graph, then encountering the third case described above is a necessary and sufficient condition.

To construct the actual cycle, we first construct the path from x to o by following the predecessors backward to the origin o and by reversing the order. Then we continue this path with y and the entire path from y to o. The cycle can then be described as $o \rightarrow x - y \rightarrow o$. If the cycle obtained this way is not simple, a simple cycle containing o can easily be extracted from it.

2.2 Induced Co-Cyclic Graph

Definition. *Let $o \in V$ be a particular vertex in the graph for which we want to extract the facet structure. Let $C_s(o)$ be the set of all the simple cycles containing o. Let $H_o = \{A_o, E_o, W_o\}$ be the weighted* **co-cyclic graph** *induced by the vertex o defined the following way:*

$$A_o = \{v \in G \mid ov \in E\} \quad (1)$$

$$E_o = \{vw \mid \exists c \in C_s(o), v \in c, w \in c\} \quad (2)$$

$$W_o(vw) = min\{length(c) \mid c \in C_s(o)\} \quad (3)$$

To determine the structure of this graph, first we can notice that the relation defining the edges of H_o in Equation 2 is transitive. For this, let $v, w, y \in V$ such that v and w are co-cyclic, as well as w and y. Let us denote the two cycles starting from o and containing these vertices by $o\,v\,C_1\,w\,o$ and $o\,w\,C_2\,y\,o$. Let us suppose that $v \notin C_2$ and $y \notin C_1$, otherwise we would already have found a simple cycle containing v, y, and o.

Then the cycle obtained by merging these two cycles, $o\,v\,C_1\,w\,C_2\,y\,o$ contains both v and y. If the cycle is not simple, any subcycle inside the region $C_1\,w\,C_2$ can be removed to make the cycle simple, since neither o, nor v or y are present on that path. Figure 1 illustrates the configuration we described.

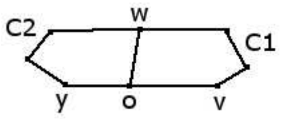

Figure 1. Two cycles that can be merged

Thus, we have shown that

$$vw \in E_o \ \wedge \ wy \in E_o \ \Rightarrow vy \in E_o$$

The consequence of the transitivity for the co-cyclic graph H is that its connected components are complete subgraphs.

Co-Cyclic Graph Closure

The cycle detection algorithm presented in Section 2.1 does not always detect the complete subspace of simple cycles $C_s(o)$. An intersection of edges in a vertex y as described in the algorithm generates a single simple cycle in the algorithm. Thus, the algorithm detects at most $m - (n - 1)$ cycles. The number of simple cycles o belongs to can be larger. o can have as many as $n - 1$ neighbors, and thus $(n - 1)(n - 2)/2$ incident cycles. If the graph is complete, then the BFS can detect all of these cycles, otherwise the number of detected cycles will be lower than the total number of them.

Example

Let us take the case of a graph with the structure of a hypercube and the origin O as shown in Figure 2 left. This graph is interesting from the point of view of our algorithm because it represents a four-dimensional geometric object in which there is no obvious simple cycle fan that surrounds each vertex.

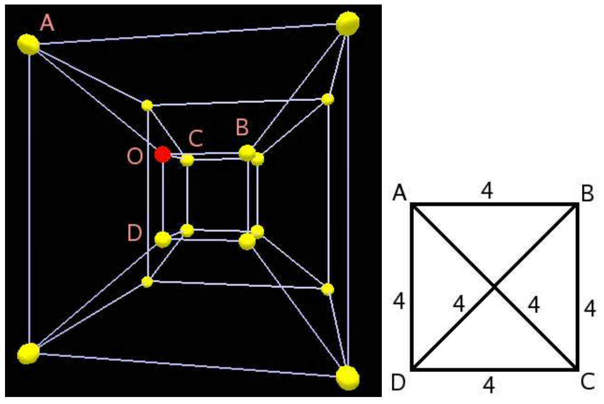

Figure 2. A hypercube, the vertices adjacent to O (left) and the induced co-cyclic graph (right)

There are four vertices in the graph adjacent to O and they are labeled A, B, C, D. Each of them is connected to all of the others by a simple cycle of length four representing a facet of the hypercube. The co-cyclic graph H_o for this particular vertex is shown in Figure 2 right.

2.3 Cycle Structure

Starting from the assumption that we have constructed the co-cyclic graph H_o successfully, let us examine what the cycle structure in the vertex o entails in terms of the co-cyclic graph. For this step, each connected component of the graph will be treated separately. Let us consider for now that the graph H_o is complete. There are two possible constraints to the solution. The first one involves not using an edge more than twice. The second one involves using each outgoing edge from o and minimizing the total length of the considered cycles.

Definition. *A* **circuit solution** *for the facet problem is one where each outgoing edge from o is present in exactly two cycles in the structure and for which the total length of the cycles is minimal.*

The constraint that each outgoing edge should appear twice means that the subset of cycles constitutes a Hamiltonian circuit in the graph H_o, a simple cycle containing all the vertices. The constraint about minimizing the total length of the cycles translates in terms of the graph H_o as a Hamiltonian circuit minimizing the total cost. This means that the problem can be reduced to a symmetric instance of the Traveling Salesman Problem (TSP) in the co-cyclic graph.

TSP is NP-complete, but we have not shown a complete equivalence to this problem. In fact we can observe that the co-cyclic graph satisfies the triangle inequality. More precisely, if $v, w, y \in V_o$, then based on the demonstration of the transitivity of the co-cyclic property, the weight of the edges vw, wy and vy in the co-cyclic graph satisfy the inequality

$$W_o(vy) \leq W_o(vw) + W_o(wy) - 2$$

There are many studies of TSP in the case where the graph has particular properties [8], especially Euclidean [2]. The major algorithm for the case where the triangle inequality is satisfied was introduced by Christofides in [4]. He proved that there exists a polynomial algorithm that produces a solution not worse than twice the optimal cost. His algorithm uses the idea of the minimum spanning tree.

For example, for the hypercube with co-cyclic graph presented in Figure 2, a circuit solution could be the one shown in red in Figure 3 left. The corresponding sequence of cycles in the original graph is shown in the same figure to the right. We show the 3D object in this figure using a different perspective that makes it easier to see the solution.

Definition. *A* **spanning tree solution** *for the facet problem requires each outgoing edge to be present at least once and for the total length of the cycles to be minimal.*

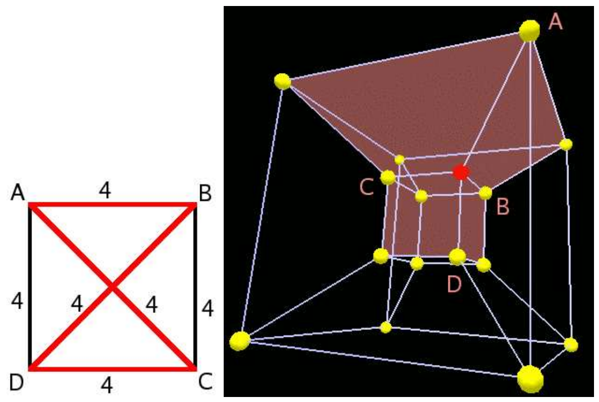

Figure 3. A circuit solution in the hypercube

The relaxation of the constraints means that the object satisfying these properties in the graph H_o is a minimum spanning tree [12].

For example, for the hypercube and the co-cyclic graph presented in Figure 2, a spanning tree solution could be the one shown in red in Figure 4 left; the corresponding sequence of cycles in the original graph is shown in the same figure to the right.

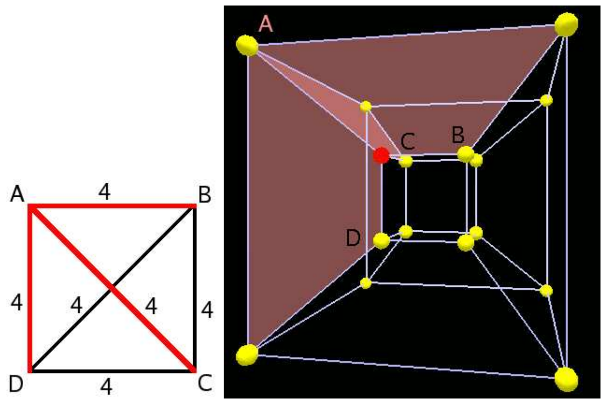

Figure 4. A spanning tree solution in the hypercube

3 Facet Definition and Detection

In this section we present the facet detection algorithm, starting with the existence and detection of simple cycles in the graph, followed by a selection and classification of the simple cycles intersecting in a particular vertex in the graph.

The definition of a facet is closely related to that of a feature. Basically a feature is usually composed of several facets, so facet detection can be seen as an intermediary step in feature detection.

A *feature* is part of an object that is localized, meaningful, and detectable. Its definition is often subjective, making reference to what a human eye can recognize [9]. Examples include edges, corners, chains, line segments, parameterized curves, regions, surface patches (3D), closed polygons [6].

A *facet* is usually defined as part of an object or of an image that is flat, planar, linear, quadratic, or cubic. Considering that a graph represents objects with straight edges and planar surfaces, a facet is a closed and planar polygon, preferably convex.

3.1 Facet Detection

Instead of focusing locally on each of the facets, our algorithm detects them by using the entire structure of cycles surrounding each vertex in the graph. Starting from the assumption that the graph represents a solid surface without holes, we aim to construct a fan of minimal polygons centered in each vertex such that they can have a local planar representation without overlapping. For this purpose, every vertex in the graph that is adjacent to the origin, must belong to two and only two cycles of the fan.

The algorithm shown bellow starts by detecting all the simple cycles intersecting in o, then sorting them by length. After that it follows the idea of a minimum spanning tree, adapted to construct a Hamiltonian circuit.

Algorithm

```
for every vertex o in the graph {
  build a list of all simple cycles
         containing o;
  sort the list of cycles by cost;
  Fo = empty;
  for every cycle c in this list and
      while size(Fo) < degree(o)-1 {
    v, w = the two vertices in c
           adjacent to o;
    if ((each of v and w appear each in
         at most one other cycle in Fo)
         and ((c does not close
               a cycle of cyles in Fo)
               or c is the last cycle
                    we need))
      add c to Fo;
  }
}
```

Complexity

If n is the number of vertices and m the number of edges, then in the worst case the cycle-detection algorithm will have a complexity of $m = O(n^2)$. The operation of merging two cycles and eliminating sub-cycles is linear over n. Computing the closure of the co-cyclic graph can be done with a greedy algorithm of complexity in the worst case $O(n(n(n-1)/2-m)^2) = O(n^5)$. By computing only the cycles necessary to close the circuit at the end of the algorithm, we can reduce this step to n^3. This can be done by a simple search of a path in the co-cyclic graph. Sorting the cycles by length is $O(m \log m) = O(n^2 \log n)$.

The most costly operation in the algorithm above is checking that the cycle c does not close a cycle of cycles. This can be done efficiently by storing the neighbors of the vertex o in a union/find data structure, in which case its complexity will be $O(\log n)$ at each step, making the total complexity $O(m \log n) = O(n^2 \log n)$.

In conclusion, our algorithm has a complexity of $O(n^2 \log n)$ without closure of the co-cyclic graph, and of $O(n^3)$ with it.

3.2 Examples

We present here some examples of results of our algorithm for a variety of graph configurations. In each of the following figures the origin vertex is shown in light green, while the cycles in the structure identified by the algorithm are shown in shades of red. This allows us to visually distinguish between the cycles composing the structure.

First, Figure 5 shows the hypercube configuration that we discussed earlier in this paper. We can see in the left image that the algorithm found the predicted cycle structure for the vertex we analyzed before. It also found an equivalent correct solution for a vertex on the exterior cube, shown in the image to the right.

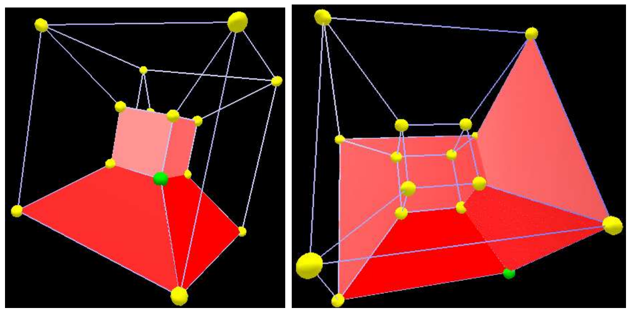

Figure 5. Solutions found by the algorithm in a hypercube

Figure 6 shows the cycle structure for a regular dense fan. Figure 7 shows the solution found by our algorithm in a grid both for a vertex on the outer border of the grid (left) and for a vertex inside the grid (right). Figure 8 shows the structure for a vertex on a torus graph. For all these graphs the algorithm was able to identify the cycle structure precisely.

The closure of the co-cyclic graph based on the transitivity of the co-cyclic relation can have both positive and negative impact on the solution. For example, in the case of the vertex on the border of the grid in Figure 7 left, the BFS cycle detection algorithm failed to capture all the simple cycles around the origin vertex (in green in the figure). As a result, the circuit of cycles was not completed. It is a Hamiltonian path but not circuit. For this particular case, this emphasizes the structure better than a circuit because the grid represents an open surface.

For more complex local structures in the graph, not

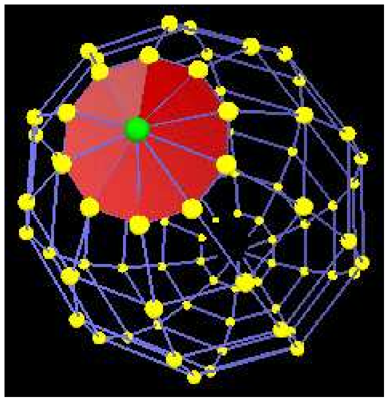

Figure 6. A vertex in the center of a fan on an ellipsoid graph

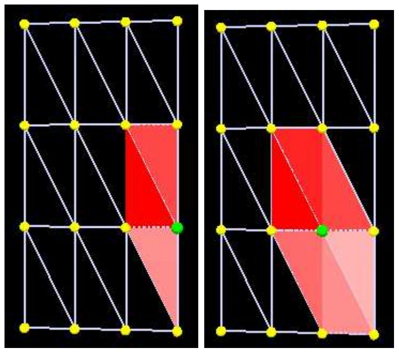

Figure 7. A vertex on the border of a grid and inside a grid

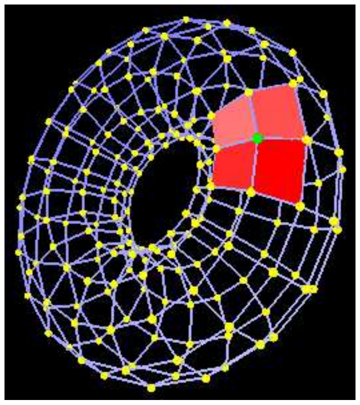

Figure 8. A vertex on a torus graph

closing the co-cyclic graph can lead to incomplete solutions. Such an example is shown in Figure 9.

Last, we have shown in Section 2 that the problem can be expressed as an instance of the Traveling Salesman in the co-cyclic graph. TSP is NP-complete, and even in the case where the graph satisfies the triangle inequality, as in our case, the best polynomial algorithm cannot guarantee an optimal solution, but only one that is no more than

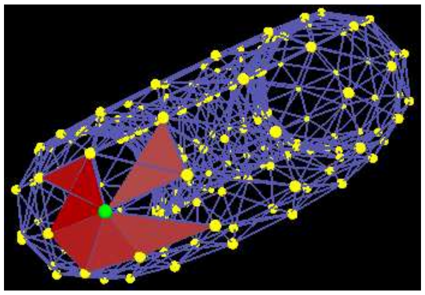

Figure 9. An incomplete solution without closure of the co-cycle graph

a given percentage more costly than the optimal one. Figure 10 shows an example where the solution found by the algorithm is not optimal.

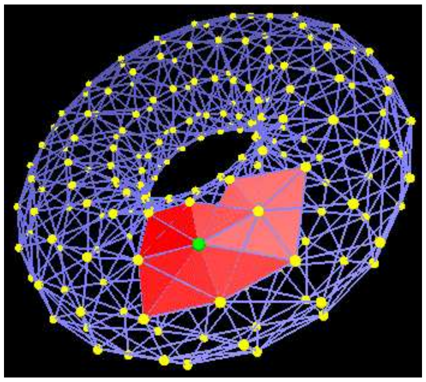

Figure 10. A non-optimal solution on a torus graph

4 Conclusion

In this paper we presented an algorithm to detect facets in a graph and emphasize the local structure of the graph around a particular vertex for the purpose of visualization.

Section 2 presented a cycle detection algorithm and introduced the notion of induced co-cyclic graph. We showed that our problem can be reduced to an instance of the Traveling Salesman in the co-cyclic graph. Section 3 introduced the facet detection algorithm inspired from the method constructing the minimum spanning tree and discussed its complexity.

Our method detected the correct solution for most of the graphs that we examined and is of polynomial complexity. It is the most efficient when the graph is a representation of a regular mesh extracted from the surface of an object.

References

[1] N. Amenta and Y.J. Kil. Defining point-set surfaces. In *Proceedings of ACM SIGGRAPH*, pages 264–230, 2004.

[2] S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of the 37th IEEE FOCS*, pages 2–11, 1996.

[3] S. Azernikov and A. Fischer. Efficient surface reconstruction method for distributed cad. *Computer-Aided Design*, 36(9):799–808, 2004.

[4] N. Christofides. Worst case analysis of an algorithm for the traveling salesman problem. Technical Report 388, Carnegie Mellon University, Graduate School of Industrial Administration, 1976.

[5] C. D'Souza, K. Sivayoganathan, D. Al-Dabass, and V. Balendran. Simulation of object recognition by a robot. In *Proceedings of the UKSIM Workshop, SIMULATION '99*, pages 23–26, UCL, London, 1999.

[6] R. Haralick and L. Shapiro. *Computer and Robot Vision*. Addison-Wesley, 1993.

[7] Q. Iqbal and J. K. Aggarwal. Image retrieval via isotropic and anisotropic mappings. In *IAPR Workshop on Pattern Recognition in Information Systems (PRIS 2001)*, pages 34–49, Setubal, Portugal, July 6-8 2001.

[8] M. Karpinski P. Berman. 8/7-approximation algorithm for (1,2)-tsp. In *Proceedings of the 17th ACM-SIAM SODA*, pages 641–648, 2006.

[9] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. *Computer Graphics Forum*, 22(3):281–290, 2003.

[10] A. Shahrokni, L. Vacchetti, V. Lepetit, and P. Fua. Polyhedral object detection and pose estimation for augmentedreality applications. In *Proceedings of Computer Animation*, pages 65–69, Geneva, Switzerland, 2002.

[11] M.C. Shin, D.B. Goldgof, and K.W. Nikiforou and. Comparison of edge detection algorithms using a structure frommotion task. *IEEE Transactions on Systems, Man and Cybernetics*, 31(4):589–601, 2001.

[12] M.A. Weiss. *Data Structures & Algorithm Analysis in C++*. Addison Wesley Longman, 1999.

[13] R.J. Wilson. *Introduction to Graph Theory*. Prentice Hall, 1996.

[14] C. Zhang and T. Chen. Efficient feature extraction for 2d/3d objects in mesh representation. In *Proceedings of IEEE International Conference on Image Processing*, pages 935–938, 2001.

[15] S. Zheng, J. Tian, and J. Liu. Efficient facet-based edge detection approach. *Optical Engineering*, 44(4):47202–47211, 2005.