Visibility polygon traversal algorithm

Izaki, Åsmund [1] Derix, Christian [1]

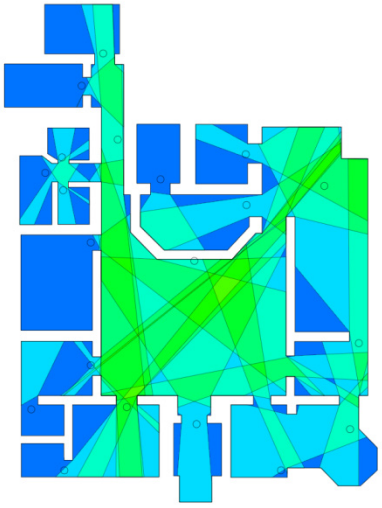
Keywords: visibility in polygon with holes; query-based algorithm; spatial analysis

Abstract The determination of visibility relations within polygonal environments has applications in many different fields; In the current context it is primarily investigated as a basis for spatial architectural analysis, and as a design driver for architectural design from a user and occupant perspective, but it is equally applicable to a wide range of engineering problems and it's a well-established topic in computational geometry.

We introduce a new query-based algorithm for traversing over the visible convex regions of a polygon with holes from any point inside the polygon or from any of its vertices, where each query runs in $O(f' h' + \log n)$ for a polygon with n vertices, f' visible convex partitions, and h' visible holes, with a preprocessing stage that runs in $O(n \log^* n)$ with $O(n)$ space. The log n component of the query only applies to internal points. The visibility polygon traversal algorithm is applicable to a varied set of visibility problems, including the construction of visibility polygons (isovists), and visibility graphs of polygon vertices and/or points inside the polygon.

The algorithmic findings are linked to spatial architectural analysis by representing the regions of architectural or urban plans that are permeable or visually open as polygons with holes. Two applications of the algorithm are presented, which have enabled more responsive and dynamic user interactions with architectural plans. These consist of an interactive isovist tool and a tool for calculating shortest paths, flows and distances within a multi-storey layout.

Fig. 1 An interactive application of the algorithm that constructs a number of visibility polygons in real time. Here used to manually work out an approximation to the art gallery problem (Honsberger, 1976) in Louis Sullivan's National Farmer's Bank of Owatonna



1. Computational Design & Research, R&D, Aedas, London, United Kingdom

Visibility methods in architecture and computational geometry

The work presented in this paper borrows methodologies and concepts from spatial architectural analysis as well as computational geometry, and is an attempt to reconcile some of the shared concepts and methods in these fields.

The term isovist was developed first by Tandy (1967) for landscape analysis and originally referred to the field of view from a particular location, where visibility was understood literally in terms of human perception. In spatial architectural analysis isovists and isovist fields were formally introduced as a tool for the analysis and description of architectural space by Benedikt (1979) who developed a set of mathematical measures of isovists in order to quantify a range of visual and experiential aspects of space.

Isovists have since become one of the basic methods of analysis used in space syntax. Hillier (1996) superimposed isovists of a set of convex polygons to show how the squares in Rome created a network of visually connected nodes. Turner and Penn (1999) continued to investigate the interconnectivity of points and introduced isovist integration analysis. The method which now is commonly referred to as visibility graph analysis (VGA) (Turner et al., 2001) consisted of creating a graph of mutually visible points on which measures such as mean depth could be computed. As these developments have taken place, the initial idea of visibility as a perception-based concept, then towards describing experience of space, seem now to have been replaced by one describing movement. In VGA the input typically represents permeability rather than visibility, where depth describes how many turns are needed in order to reach another location. Sheep and Ruth Dalton (2009) point out the visibility accessibility problem that occurs when there are discrepancies between visual and permeable links, and propose a layered graph to capture these two relations explicitly.

The notion of geometric visibility in computational geometry, which seems to have been developed almost in parallel to the architectural analysis research described so far, is regarded as one of the most fundamental concepts and forms the foundation of a range of seemingly unrelated problems such as polygon partitioning and robotic motion planning. Here two points, *u* and *v,* are said to be mutually visible if the line segment bounded by *u* and *v* does not intersect any of the line segments of the polygon in which they are either vertices of, or that they are contained within. Point visibility polygons, or visibility regions are similar to isovists as they have been generally applied in spatial analysis (as a 2D section rather than a 3D volume, with some exceptions (Derix et al., 2008)), and the term visibility graph is used similarly in both fields, with the distinction that it in computational geometry the points usually refers to vertices of a polygon, while it in space syntax is typically assumed to be a grid of points inside a polygon. For more detailed definitions of geometric visibility and its related problems see O'Rourke (1987).

Some visibility algorithms are output-sensitive meaning that their time complexity primarily depends on the size of the output rather than the input. A group of these, which are similar to ours in that they have pre-processing stage for building data structures on which they can perform queries, are based on visibility decomposition which decomposes a polygon into visually stable regions, a similar concept to the e-partitions proposed by Peponis et al. (1997). Lu et al. (2011) introduced a query-based algorithm for finding the visibility of a polygon with holes that first partitions the polygon into simple polygons. It runs in $O(v + h + \log^2 m + h \log (n / h))$ time, where v is the size of the output polygon, m is the size of the simple polygon containing the viewpoint. The preprocessing stage has a time complexity of $O(n^2 \log n)$.

For visibility graphs for a set of obstacles (equivalent of just considering the holes of the polygon) there exist a number of output-sensitive algorithms with lower time complexity than the one presented here. Pocchiola and Vegter (1996) runs in time $O(n \log n + k)$, where k is the number of edges, and uses $O(n)$ space. Also see Overmars and Welzl (1988), and Gosh and Mount (1991).

A query-based algorithm for traversing visible convex regions

The visibility polygon traversal algorithm introduced here offers a traversal of visible convex regions of a polygon from a point. The algorithm provides a depth-first traversal, in counter-clockwise order. We use a polygon with holes P or a convex partition mesh $M = (V, E, F)$ describing P with a point vp inside the polygon, or any polygon vertex $pv \in P$ or mesh vertex $mv \in M$ as the source of the visibility calculation. These options can be used when a mesh representation already exists for other purposes (e.g. for rendering or as a modeling approach) and for making any number of subsequent queries on the same polygon from different viewpoints.

The algorithm is based on a modified depth-first search (DFS) through the face connectivity graph of the mesh. The traversal keeps left and right constraints for each new face it visits, which is continuously narrowed by the vertices of the edges it has traversed thus far, and consists of the rightmost left point l, and the leftmost right point r from vp in the direction of the search. An adjacent face is only visited if the common edge falls within these constraints. The algorithm differs from a standard DFS in that we do not keep track of already visited faces. This is not necessary due to the left and right constraints, and in certain cases a face will be correctly visited several times with different constraints.

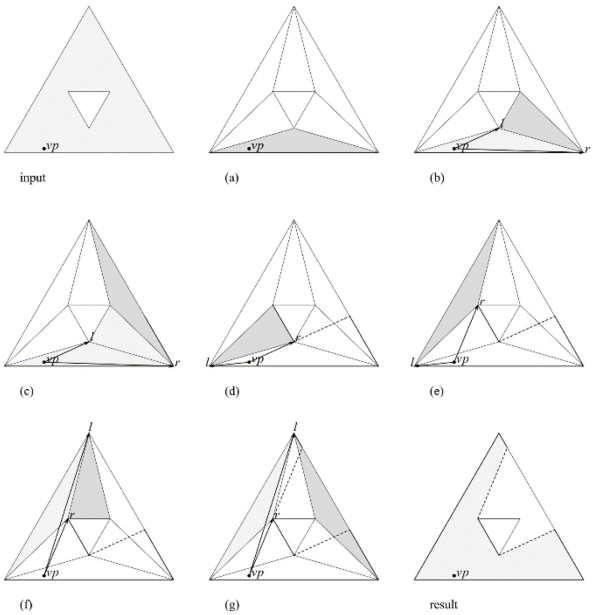


Fig. 2 The execution of the visibility polygon traversal algorithm. The input is a polygon with holes and the point *vp*. Before the execution of the traversal the polygon is triangulated and the starting triangle in which *vp* is contained is found. In **(a)-(g)** the darker shade indicate the current triangle and the brighter shade indicate the internal stack. Here, the traversal algorithm is used to build a representation of the visibility polygon that distinguishes between open (permeable) edges and boundary (solid) edges, but the same mechanism can be used to build other structures, such as the visibility graph

Two configurable operations are explicitly written out in the pseudocode [1] to support extensions that solve specific visibility determination problems.

[1] The pseudocode conventions used here follow the rules described in Cormen et al. (2001, p. 19).

```
VPT-INSIDE(P, vp)
1  M ← TRIANGULATE(P)
2  f ← FIND-FACE(M, vp)
3  VPT-INSIDE(M, f, vp)

VPT-INSIDE(M, f, vp)
1  VPT-VISIT(M, f, NIL, vp, NIL, NIL)

VPT-VERTEX(P, pv)
1  M ← TRIANGULATE(P)
2  mv ← FIND-VERTEX(M, pv)
3  VPT-VERTEX(M, mv)

VPT-VERTEX(M, mv)
1  vp ← position[mv]
2  for each f ∈ F[mv]
3      do r ← position[NEXT(f, mv)]
4         l ← position[PREVIOUS(f, mv)]
5         VPT-VISIT(M, f, NIL, vp, r, l)

VPT-VISIT(M, f, eParent, vp, r, l)
1  VPT-FACE-OPERATION(M, f, eParent, vp, r, l)
2  for each e ∈ EDGES(f, eParent)
3      do eRight ← pos[start[e]]
4         eLeft ← pos[end[e]]
5         if EDGE-IN-VIEW(vp, r, l, eRight, eLeft)
6            then fAdj ← ADJACENT(f, e)
7                 if fAdj = NIL
8                    then VPT-BOUNDARY-EDGE-OPERATION(M, f, e, vp, r, l)
9                    else if eRight ≠ vp and eLeft ≠ vp
10                             then rAdj ← LEFTMOST(vp, r, eRight)
11                                  lAdj ← RIGHTMOST(vp, l, eLeft)
12                                  VPT-VISIT(M, fAdj, e, vp, rAdj, lAdj)

EDGE-IN-VIEW(vp, r, l, eRight, eLeft)
1  if r = NIL or l = NIL
2     then return TRUE
3  if TEST(vp, l, eRight) = LEFT or TEST(vp, r, eLeft) = RIGHT
4     then return FALSE
5  return TRUE

LEFTMOST(vp, r, eRight)
1  if r = NIL or TEST(vp, r, eRight) = LEFT
2     then return eRight
3     else return r

VPT-FACE-OPERATION(M, f, eParent, vp, r, l)
1  ► A configurable operation. The visible convex region is bounded by the input parameters

VPT-BOUNDARY-EDGE-OPERATION(M, f, e, vp, r, l)
1  ► A configurable operation. The visible edge is bounded by the input parameters
```

The pseudocode specifies both the cases of visibility from a point within a polygon or mesh (VPT-INSIDE), or from one of the vertices of a polygon or a mesh (VPT-VERTEX). In the case of using a polygon as an input, it is first partitioned into convex polygons. Any triangulation which does not insert additional vertices can be used, but Seidel's algorithm provides the fastest running time of $O(n \log^* n)$ (Seidel, 1991) and it has been shown to work for polygons with holes (Lamot et al., 2000).

A half-edge mesh (Weiler, 1988) or equivalent data structure is used that allows constant time access to adjacent faces and can iterate over the edges of the face in a counter-clockwise order. ADJACENT(*f*, *e*) is included in the interface to the mesh data structure, and returns the adjacent face over an edge, or if the edge is on the boundary it returns a null pointer. The EDGES(*f*, *e*) operation returns the edges of a face except *e*, in a counter-clockwise order starting from the edge adjacent to *e*, to ensure a counter-clockwise order of the whole traversal. In the same manner $e \in E[f]$ refers to the edges of *f* in a counter-clockwise order, and $f \in F[mv]$ refers to the faces adjacent to *mv* in a counter-clockwise order.

Depending on the application we may need to either locate the mesh vertex corresponding to the polygon vertex (FIND-VERTEX), or to locate the partition a point is contained in (FIND-FACE). For mesh vertices we can create references when the mesh is created, and they can be retrieved in O(1) time. The same is true when viewpoints inside the mesh are determined from the faces themselves (for example for a uniform grid). However if *vp* is arbitrary we need to use a planar point location algorithm. The algorithm by Kirkpatrick (1983) which finds the triangle in $O(\log n)$ time can be used with our triangulation *M*, which results in additional preprocessing that runs in only $O(n)$, and does not change the time complexity order of the preprocessing stage.

Finally, the algorithm relies on a geometric half-plane test. TEST returns RIGHT, LEFT OR STRAIGHT depending on the orientation of the third point in relation to the line defined by the two first. LEFTMOST and RIGHTMOST takes three points as parameters and simply return a reference to which of the two last point is further left, or right, in relation to the first points, given that the angle between the points is always smaller than 180°.

Additional constraints are straightforward to include in the algorithm. For example a view triangle with a limited field of view can be set by changing the initialisation of the query, or a maximum metric traversal distance can be defined by changing the return condition of the recursion.

We see from this preliminary analysis that each query takes $O(f' h' + \log n)$ time where *f'* are the number of visible faces and *h'* are the number of visible holes (meaning partly or fully visible from *vp*), and *n* is the number of vertices in the original polygon. Each visible convex partition is visited for the number of different visible convex regions of *f* which is bounded from above by *h'*. Furthermore we see that the $\log n$ component can be discarded for the cases where we already know which partition *vp* lies in or when *vp* is one of the vertices of the original polygon.

The traversal has an output sensitive running time; meaning that the running time is dependent on the number of visible convex partitions. The running time of the query is not guaranteed to be proportional to the number of vertices in the output visibility polygon or number of edges in the visibility graph, since some of the visited regions may not contain visible boundary edges (only internal edges), nor visible vertices.

Isovist tool

As case studies we have integrated the algorithm in two existing tools at Aedas R&D. The first tool calculates the visibility polygon of one or more points and allows the user to interactively place and move isovists around. Because of the fast execution time very large environments, such as whole cities, can be evaluated in real-time, together with read-outs of its properties.

We extend the edge operation to calculate the visibility polygon.

```
VPT-BOUNDARY-EDGE-OPERATION(M, f, e, vp, r, l)
1  veStart ← PROJECT-START(M, e, vp, r)
2  veEnd ← PROJECT-END(M, e, vp, l)
3  if current = NIL
4     then first ← veStart
5     else if EQUALS(veStart, current) = FALSE
6             then ADD_EDGE(current, veStart, OPEN)
7  ADD_EDGE(veStart, veEnd, BOUNDARY)
8  current ← veEnd
```

Here, PROJECT-START and PROJECT-END return the intersection of the line defined by two points and a line segment (if the intersection is outside the segment it will return the nearest point), and EQUALS returns TRUE if the points have the same coordinates. *Current* is the current point of the visibility polygon during construction, initialised as NIL. A simple interface is provided to construct the visibility polygon: ADD_EDGE adds a new edge to the visibility polygon and marked as BOUNDARY or OPEN. Finally, after the query and therefore not shown above, if *current* and *first* are not EQUAL and not NIL, and edge from *current* to *first* closing the visibility polygon is added and marked as OPEN.

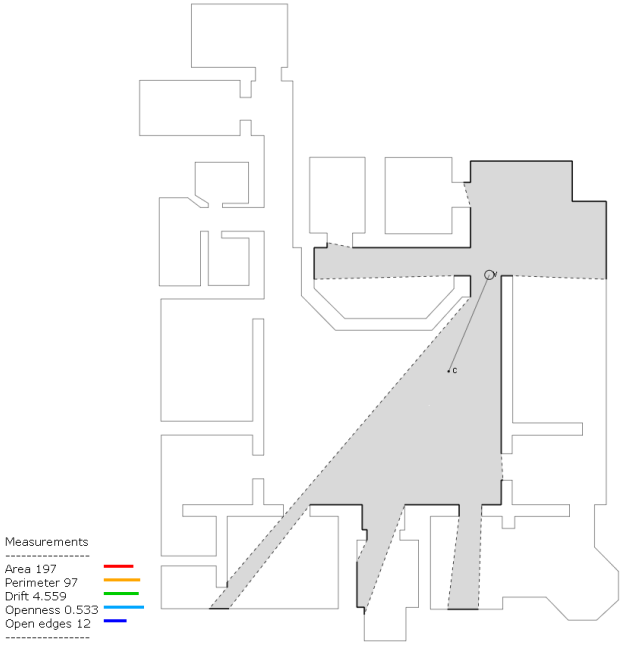


Fig. 3 A screenshot from the interactive isovist tool showing the isovist from point *v* with centroid *c* in National Farmer's Bank. The open edges are drawn as dashed line segments and the boundary edges as thicker solid line segments

Path and flow tool

The second application shows the method applied to the problem of determining shortest paths and flows within and between a layered set of polygons with holes. New functionality was added to enable or disable parts of the circulation to visualise the effects they had on the movement structure across the building: in other words what would be the result of a movement passage being temporarily blocked in the case of maintenance work, or by omitting it from the

design altogether. 2D Boolean operations on the polygons from the JTS Topology Suite open source library enabled the implementation of blocking off certain regions of the circulation areas. The user could enable or disable the subtractions from within the running application.

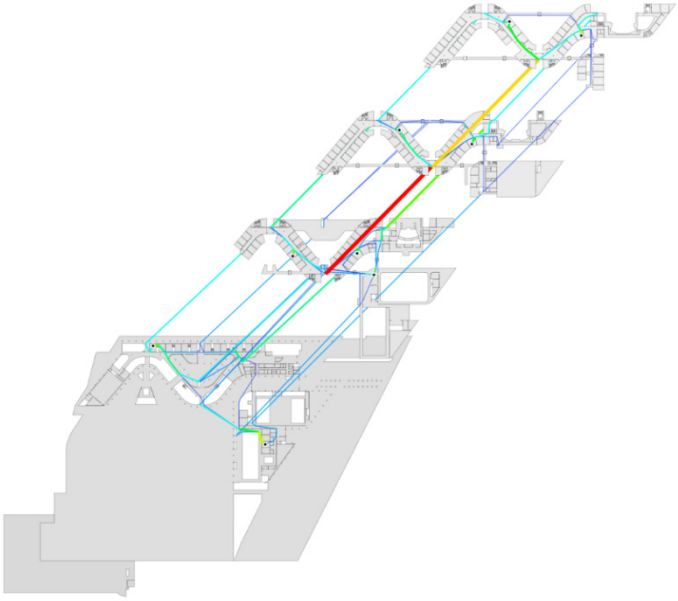


Fig. 4 Screenshot of path and flow tool for an internal Aedas project; "GEMS Academy". Colours and line widths show the amount of flow. The problem consisted of investigating likely movement paths and flows in a school building, as well as alternative routes

Conclusion

The algorithm proposed in the paper responds to the current lack of fast and generic solutions that can address a number of different visibility determination problems. Other query based visibility algorithms typically have a preprocessing stage that is slower than the one we have shown, but that guarantee a lower time complexity for each query. In our case we have preprocessing stage with a low upper bound running time allowing large polygonal environments to be dynamically modified in real-time. The algorithm is simple to implement using well-known primitives such as half-edge meshes and fundamental geometric tests which are readily available in many computational geometry libraries or frameworks.

Many visibility related concepts overlap between spatial architectural analysis and computational geometry. We believe that there are significant applications of computational geometry algorithms that has not yet found its way into spatial architectural analysis, as well as problems and structures put forward in spatial analysis that have not yet been investigated in the field of computational geometry. Further research is needed to test the traversal method on more visibility problems related to structures used in space syntax, and to compare its performance with other algorithms from computational geometry with a set of different architectural input sets.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007- 2013) under grant agreement number 242497.

References

Benedikt, M.L. (1979). To take hold of space: isovists and isovist fields. *Environment and Planning B: Planning and Design*, *6*, 47–65.

Cormen, T.H., Leiserson, C. E., Rivest, R.L., Stein, C. (2003). *Introduction to Algorithms*. The MIT press.

Dalton, S., Dalton, R. (2009). Solutions for visibility, accessibility and signage problems via layered graphs. In D. Koch, L. Marcus, J. Steen (Eds.), *Proceedings to the 7th International Space Syntax Symposium*, *9-11 June 2009*, *Stockholm*.

Derix, C., Gamlesæter, Å., arranza, P.M. (2008). 3D Isovists and Spatial Sensations: Two Methods and a case Study. In S. Haq, C. Hölscher, S. Torgrude (Eds.), *Movement and Orientation in Built Environments: Evaluating Design Rationale and User Cognition*: *Proceedings of EDRAMOVE & SFB TR8 conference on Spatial Cognition*, *28 May 2008* (67-72).

Ghosh, S.K., Mount, D.M. (1991). An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, *20*, 888–910.

Hillier, B. (1996). *Space is the Machine: A Configurational Theory of Architecture*. Cambridge: Cambridge University Press.

Honsberger, R. (1976). Chvátal's Art Gallery Theorem. In *Mathematical Gems II* (104-110). Mathematical Association of America.

Kirkpatrick, D (1983). Optimal search in planar subdivisions. *SIAM Journal on Computing 12* (1), 28-35.

O'Rourke, J. (1987). *Art Gallery Theorems and Algorithms*. New York: Oxford University Press.

Overmars, M.H., Welzl, E. (1988). New methods for computing visibility graphs. In *Proceedings of the 4th Annual ACM Symposium on Computational Geometry* (164–171).

Peponis, J., Wineman, J., Rashid M., Hong Kim S., Bafna S. (1997). On the description of shape and spatial configuration inside buildings: convex partitions and their local properties. *Environment and Planning B: Planning and Design, 24*, 761-781.

Pocchiola, M., Vegter, G. (1996). Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete & Computational Geometry*, *16*, 419–453.

Seidel, R. (1991). A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, *1*, 51-64.

Tandy, C.R.V. (1967). The isovist method of landscape survey. *Methods of Landscape Analysis*, 9-11.

Turner, A., Penn, A. (1999). Making isovists syntactic: Isovist integration analysis. In *Proceedings of the 2nd International Symposium on Space Syntax*, *April 1999* (1–9). Brasilia: Universidad de Brasil.

Turner, A., Doxa, M., O´Sullivan, D., Penn, A. (2001). From isovists to visibility graphs: a methodology for the analysis of architectural space. *Environmental and Planning B: Planning and Design*, *28*, 103–121.

Weiler, K. (1988). The radial edge structure: a topological representation for non-manifold geometric boundary modeling. In *Geometric modeling for CAD applications* (3-36). Amsterdam: Elsevier Science Publish.

Žalik, B., Lamot, M. (2000). A contribution to triangulation algorithms for simple polygons. *Journal of Computing and Information Technology*, *8* (4), 319-331.