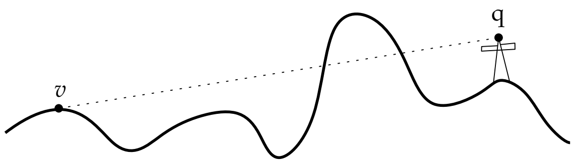
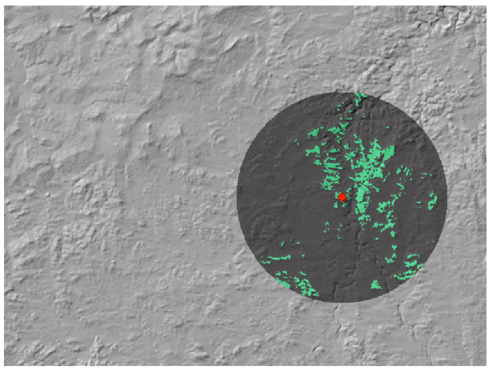
Visibility queries on terrains

Lesson 11[*]



(a)



(b)

Figure 1: **(a)** Line-of-sight between v and q; q is not visible. **(b)** The viewshed at the location marked with a red star (green = visible; maximum view distance (dark grey) is set to 15km).

Several applications using terrains involve *visibility queries*, ie given a viewpoint, which part(s) of the surrounding terrain are visible. Examples of such applications are many: optimal position of telecommunication towers, path planning for hiking (to ensure the nicest views), estimation of the view for scenic drives, estimation of visual damage when trees in a forest are cut, etc. There are also several related problems. One example is the estimation of shadows (position of the sun continually varies, also with seasons). Shadows are important to accurately measure the photovoltaic potential, for determining solar envelopes, for assessing the value of real estate, and for estimating the thermal comfort, among other applications.

When referring to visibility problems, we address the following two fundamental problems:

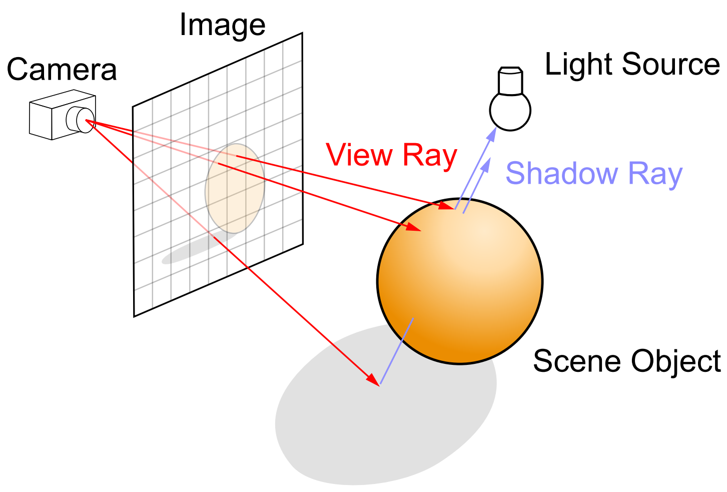


Figure 2: Ray tracing builds the image pixel by pixel by extending rays into the scene. Figure from `https://commons.wikimedia.org/wiki/File:Ray_trace_diagram.svg`.

line-of-sight (LoS): given a viewpoint v and another point q, does v sees q (and vice-versa)?; or, in other words, does the segment vq intersects the terrain? The result is either True or False.

viewshed: given a viewpoint v, which part(s) of the surrounding terrain are visible? The result is a polygon (potentially disconnected) showing the locations and extent of what is visible from v. Usually the extent is limited to a certain "horizon", or radius of visibility. If the terrain is formed of different objects (eg buildings), an object is either visible or not (simple case), or parts of objects can be visible (more complex).

Observe that for both problems, the viewpoint can either be directly on the terrain (at relative elevation 0m) or at a given height (2m for a human, or 30m for a telecommunication tower).

We discuss in this lesson the general problem of visibility as defined in computer graphics, and then discuss how terrains, being 2.5D surfaces, simplify the problem. We discuss how to solve these problems for both TINs and grids.

1 Rendering + ray casting

Rendering is the process of generating images from 2D or 3D scenes. Without going into details, as shown in Figure 2, it involves projecting the (3D) objects in a scene to an image (say 800x800 pixels) and assigning one colour to each pixel. The colour of a pixel is that of the closest object, but to obtain photorealistic images, lighting, shading, and other physics-based functions are often applied (this goes beyond the scope of this course!).

Ray casting is used for each pixel: a ray is defined between the viewpoint p and the centre of the pixel, and the closest object in the scene must be found. The main issue involves finding that closest object, and especially discard the other objects lying behind it.

Read parts of the following chapter. It summarises different methods to determine which surfaces are visible, for the generic cases of objects in 3D space. Read only the following sections: 18.0, 18.1, 18.2, and 18.4.

Salomon D (2011). Visible surface determination. In *Texts in Computer Science*, pages 891–910. Springer London. doi: 10.1007/978-0-85729-886-7_18
PDF: `https://doi.org/10.1007/978-0-85729-886-7_18`

2 For 2.5D terrains, the problem is simpler

The problem is simplified for terrains because a terrain is a 2.5D surface, and we can convert the problem to a 2D one. Furthermore, we can exploit the connectivity and adjacency between the 2D cells forming a terrain to minimise the number of objects to test (for intersections and for determining who is in front of who).

2.1 Visibility in TINs

Read part of the following chapter. Read from Section 1 to Section 3.1; it covers the simplest case for a LoS between 2 points on the surface.

de Berg M (1997). Visualization of TINs. In van Kreveld M, Nievergelt J, Roos T, and Widmayer P, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 79–97. Springer-Verlag, Berlin
PDF: `https://doi.org/10.1007/3-540-63818-0_4`

2.2 Visibility in grids

Solving visibility queries in grids is simpler than with triangles since the topology of the grid is implied (we have direct access to the neighbours of a given cell), and because grid cells are usually small we can assume that a grid cell is visible (or not) if its centre is visible (or not). The same assumption is tricky for triangles, since these can be large; in practice it is often assumed that a triangle is visible if its 3 vertices are visible, but this varies per implementation.

We describe here how both LoS and viewshed queries can be implemented for grids; the same principles could be applied to TINs with minor modifications.

Line-of-sight. A LoS query, between a viewpoint v and another point q, implies reconstructing the profile of the terrain along the vertical projection of vq (let us call it vq_{xy}). It then suffices to follow vq and verify whether the elevation at any (x, y) location along the profile is higher than that of vq. As shown in Figure 3, since the terrain is discretised into grid cells, there are 2 options to reconstruct the profile between v and q:

1. identify all the cells intersected by vq_{xy}, and assign the centre of each cell by projecting it to the terrain profile. This is what is done in Figure 3.
2. consider the edges of the cells, collect all the edges that are intersected by vq_{xy}, and linearly interpolate the elevations. This is far more expensive to compute, and therefore less used in practice.

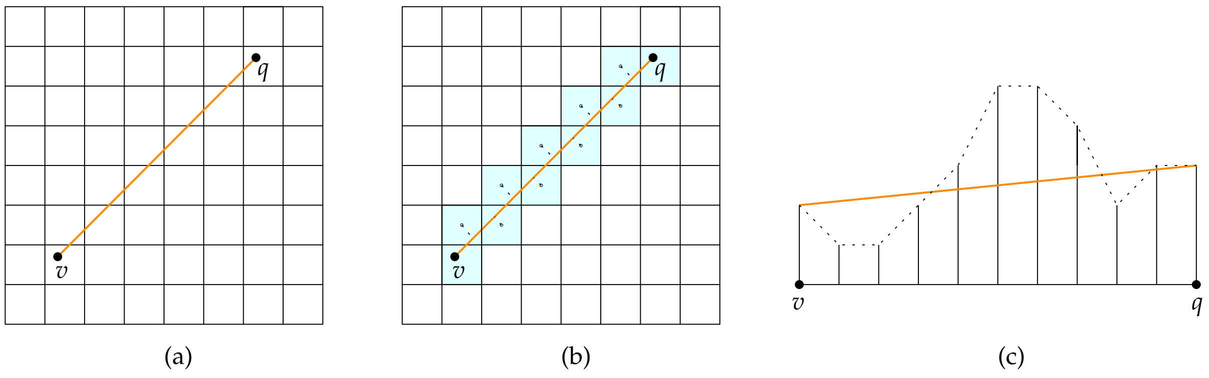


Figure 3: Line-of-sight for between v and q.

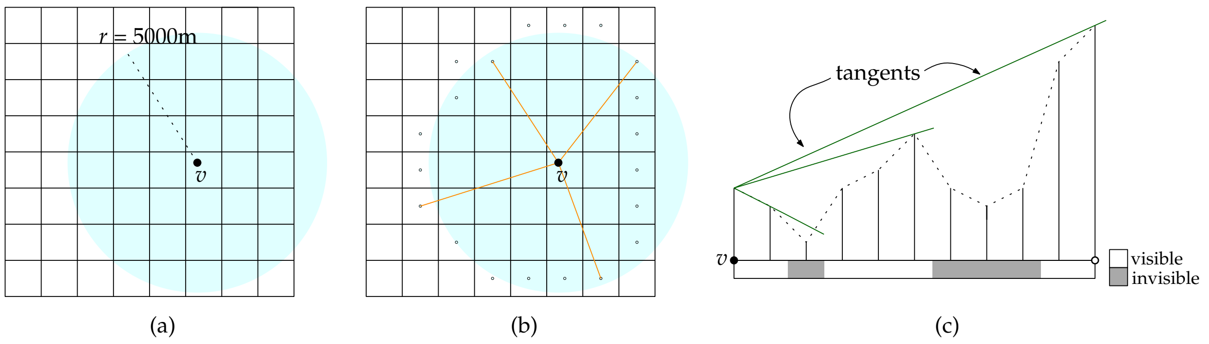


Figure 4: Viewshed for the point v; the blue circle is the radius of the horizon (5000km in this case).

The algorithm is thus as follows. Start at v, and for each pixel c encountered along vq_{xy}, verify whether the elevation value of vq at that location is higher than the elevation of c. If it is, then continue to the next pixel; if not, then there is an intersection and thus the visibility is False. If the pixel containing q is reached without detecting an intersection, then the visibility is True.

Viewshed. As shown in Figure 4, computing the viewshed from a single viewpoint v implies that the LoS between v and the centre of each pixel in a given radius is tested. The result of a viewshed is a binary grid; in Figure 1b True/visible pixels are green, and False/invisible ones are dark grey.

While this brute-force approach will work, several redundant computations will be made, since several of the rays from v will intersect the same grid cells. Furthermore, depending on the resolution, the number of cells in a 5km radius (a reasonable value where humans can see) can become *very* large. As an example, with the AHN3 gridded version (50cm resolution), this means roughly 400M queries ($(\frac{5000\times2}{0.5})^2$).

One alternative solution is shown in Figure 4b: it involves sampling the grid cells intersecting the border of the visible circle (obtaining several centres q_i), and computing the visibility of each of the cells along the line segment vq_i as we 'walk' from v. Observe that, along vq_i, it is possible that a point far from v is visible, while several closer points are not; Figure 4c gives an example.

One solution involves using so-called *tangents*. The current tangent t_{cur} is first initialised as a vector pointing downwards. Then, starting at v, we walk along the ray vq_i, and for each cell intersected its elevation z is compared to the elevation of t_{cur} at that location. If z is lower, then the cell is invisible. If z is higher, then the cell is visible and t_{cur} is updated with a new tangent using the current elevation.

Viewsheds with several viewpoints v_i are also very useful, think for instance of obtaining the viewshed along a road. This can be computed by sampling the road at every 50m and computing the viewsheds from each of the points. Each viewshed yields a binary grid, and it suffices to use a map algebra operator to combine the results into one grid (if one cell is visible from any viewpoint, then it is visible).

3 Notes & comments

The 'tangent algorithm' to compute viewsheds was first described by Blelloch (1990). The description here is inspired by that of De Floriani and Magillo (1999).

References & further reading

Blelloch GE (1990). *Vector models for data-parallel computing*. MIT Press.

de Berg M (1997). Visualization of TINs. In van Kreveld M, Nievergelt J, Roos T, and Widmayer P, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 79–97. Springer-Verlag, Berlin.

De Floriani L and Magillo P (1999). intervisibility on terrains. In Longley PA, Goodchild MF, Maguire DJ, and Rhind DW, editors, *Geographical Information Systems*, volume 1, chapter 38, pages 543–556. John Wiley & Sons.

Salomon D (2011). Visible surface determination. In *Texts in Computer Science*, pages 891–910. Springer London. doi: 10.1007/978-0-85729-886-7_18.