Conversions between 3D representations and formats

Lesson 7.1*

Contents

This lesson describes different conversions between 3D representations and formats that a geomatics engineer might have to perform. It does not claim to be an overview of all potential conversions, but it rather offers insights about the algorithms and methods most commonly used, and points out pitfalls to be aware of.

The lesson is divided into two distinct parts:

Fields: conversions that are performed when we are dealing with a field, let it be the temperature or the concentration of a certain chemical in the air (modelled as a 3D volume). In most cases, the data that are collected to study a field will be sample points in 3D space (x, y, z) to which an attribute is attached. Voxels are usually what is exchanged and analysed.

Objects: when we are dealing with data (points, surfaces, and volumes) that represent the boundaries of objects in our environment. These can be sample points from lidar or dense matching of images, or the b-rep of some buildings (which have been reconstructed from different acquisition methods).

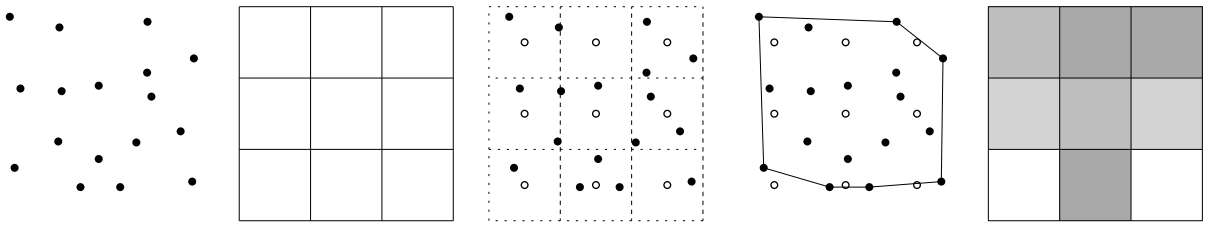


Figure 1: **(a)** input sample points. **(b)** size/location of output grid. **(c)** 9 interpolations must be performed (at locations marked with ○): at the middle of each cell. **(d)** the convex hull of the sample points show that 2 estimations are outside, thus no interpolation. **(e)** the resulting raster.

1 Conversions for fields

1.1 Points to voxels

The conversion from scattered points to grid is trivial: simply interpolate at regular locations in three dimensions (which represent the centre of each voxel) and output the results in the appropriate format (grids can be stored in many ways). Figure 1 shows the process in two dimensions.

All the interpolation methods discussed during GEO1015 (Chapters 4 and 5) generalise to three dimensions.

Nearest neighbour. Already discussed in Lesson 2.2.

Linear interpolation in tetrahedra Already discussed in Lesson 2.2.

Natural neighbour interpolation. Already discussed in Lesson 2.2.

Inverse distance weighting (IDW). The generalisation of this method to three dimensions is straightforward: a searching *sphere* with a given radius is used. The same problems with the one-dimensionality of the method will be even worse because the search must be performed in one more dimension. The method has too many problems to be considered has a viable solution for fields as found in geosciences: the interpolant is not guaranteed to be continuous, especially when the dataset has an anisotropic distribution (and anisotropy is very frequent in 3D samples, see Lesson 2.2), and the criterion has to be selected carefully by the user. Note that the implementation problems are also similar to the ones encountered with the previous method, and an auxiliary data structure must be used to avoid testing all the points in a dataset.

Kriging. All of the most common kriging varieties generalise to three dimensions without major changes, including simple kriging and ordinary kriging. In the simplest case, covariance functions, experimental variograms and fitted functions work exactly the same as in 2D but are computed using distances in 3D.

However, the vertical direction has a much weaker correlation than the horizontal directions in many fields, eg temperature, pressure and humidity. Anisotropy is thus a much more significant factor in 3D and almost always has to be modelled. A minimal solution is a custom distance function that scales the vertical direction. A better (but still simple) solution involves computing multiple experimental variograms: two (for the horizontal plane x, y and for the vertical direction z) or three (for x, y and z).

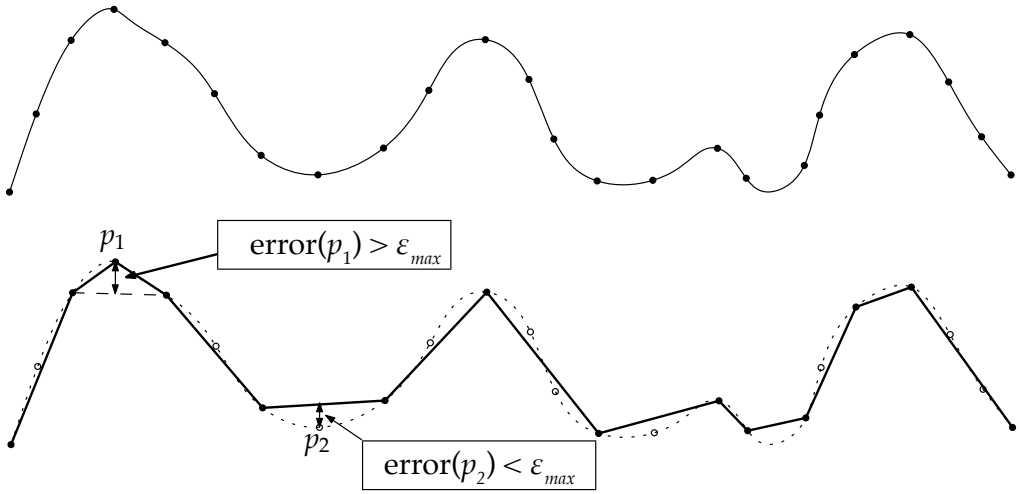


Figure 2: The importance measure of a point can be expressed by its error. When this error is greater than a given threshold ϵ_{max}, the point is kept (p_1), else it is discarded (p_2).

1.2 Voxels to points

The conversion of a voxel to a set of scattered points is not a simple operation. Given a three-dimensional grid, it is possible to create one data point at the centre of each voxel. Notice however that potentially a lot of the neighbouring points will be the same value, and thus a lot of redundancy is stored.

A better approach to this problem is to consider it as a simplification problem. Given a set S of points in $\mathbb{R}^3$ representing a field f (where each point p in S as an attribute a attached to itself), the aim is to find a subset R of S which will approximate f as accurately as possible, using as few points as possible. The subset R will contain the 'important' points of S, ie a point p is important when a at location p can not be accurately estimated by using the neighbours of p.

The two algorithms described in the GEO1015 book (Section 8.3) can in theory be generalised; Figure 2 shows the idea for the 1D case. Both strategies (decimation and refinement) can be implemented.

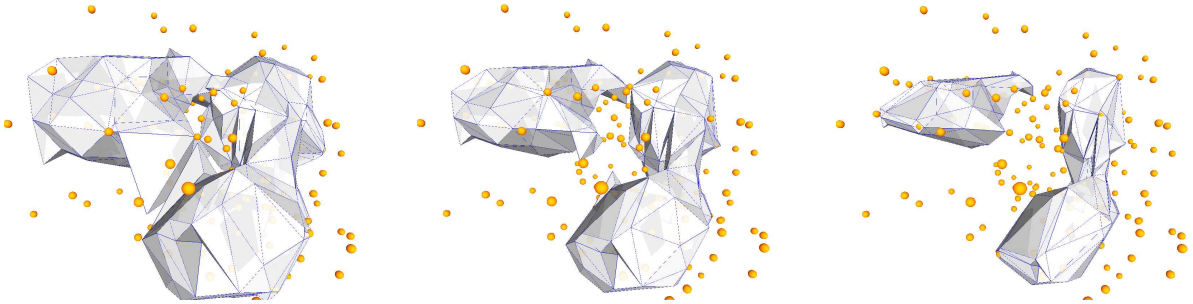
The error associated with each point p, denoted error(p), is calculated by interpolating at location p after p has been temporarily removed from the field, and comparing the value obtained with the real attribute a of p, thus error(p) $= |a - estimation|$. As shown in Figure 2 for a one-dimensional case, when the error is more than ϵ_{max} then the point must be kept, if it is less then the point can be discarded.

The method for 2D fields in GEO1015 uses linear interpolation in triangles, ie after p has been temporarily deleted from DT(S), the triangulation is updated and the estimation is obtained with the triangle containing location p. However, since mentioned earlier, using the DT in 3D for interpolation is not advised (because they contain slivers). As an alternative, one could use for instance the natural neighbour interpolation, and each error is calculated by interpolating in the field at the location and comparing the real and the estimated value.

1.3 Conversion to isosurfaces

Given a trivariate field $f(x, y, z) = a$, an isosurface is the set of points in space where $f(x, y, z) = a_0$, where a_0 is a constant. Isosurfaces, also called *level sets*, are the three-dimensional analogous concept to isolines (also called contour lines), which have been traditionally used to represent the elevation in topographic maps. Figure 3 shows one concrete example.

In two dimensions, isolines are usually extracted directly from a TIN or a regular grid. The idea is to compute the intersection between the level value (eg 200m) and the terrain, represented for instance with a TIN. Each triangle is scanned and segment lines are extracted to form an approximation of an isoline.



(a) (b) (c)

Figure 3: An example of an oceanographic dataset where each point has the temperature of the water, and three isosurface extracted (for a value of respectively 2.0, 2.5 and 3.5) from this dataset.

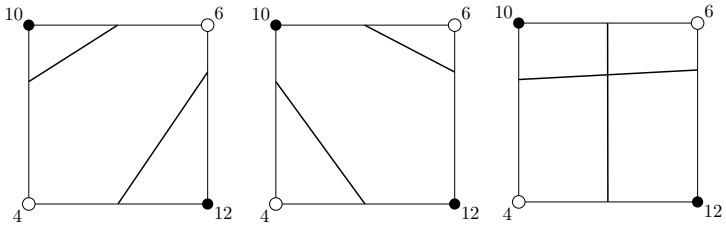


Figure 4: Ambiguous extraction of an isoline where the attribute is 8.

In three dimensions, for a trivariate field, the same idea can be used to extract surfaces.

From voxels: Marching Cubes. The principal and most known algorithm for extracting an isosurface form a voxel dataset is the *Marching Cubes*. The isosurface is computed by finding the intersections between the isosurface and each voxel/cube of the representation. Linear interpolation is used along the edges of each cube to extract 'polygonal patches' of the isosurface. There exist 256 different cases for the intersection of a surface with a cube (considering that the value of each of the eight vertices of a cube is 'above' of 'under' the threshold), although if we consider the symmetry in a cube that comes down to only 15 cases. The major problem with the marching cubes algorithm is that the isosurface may contain 'holes' or 'cracks' when a cube is formed by certain configurations of above and under vertices. The ambiguities are shown in Figure 4 for the two-dimensional case when two vertices are above the threshold, and two under, and they form a 'saddle'. The three-dimensional case is similar, with many more cases possible.

From tetrahedral mesh: Marching Tetrahedra. Although it is possible to fix the ambiguities, as is the case in two dimensions, the simplest solution is to subdivide each cell into simplices (cubes into tetrahedra in 3D). The so-called *Marching Tetrahedra* algorithm is very simple: each tetrahedron is tested for the intersection with the isosurface, and triangular faces are extracted from the tetrahedra by linear interpolation on the edges. The resulting isosurface is guaranteed to be topologically consistent (ie will not contain holes), except at the border of the dataset. But again, if a big tetrahedron is used where the vertices are assigned to a value lower than the minimum value of the field, then all the isosurfaces extracted are guaranteed to be 'watertight'. The nice thing about the algorithm is that only three cases for the intersection of the isosurface and a tetrahedron can arise:

1. the four vertices have a higher (or lower) value. No intersection.
2. one vertex has a higher (or lower) value, hence the three others have a lower (or higher) value. Three intersections are thus defined, and a triangular face is extracted. See Figure 5(a) on the left.
3. two vertices have a higher (or lower) value and the others have a lower (or higher) value. Four intersections are thus defined. To ensure that triangular faces are extracted (better output for

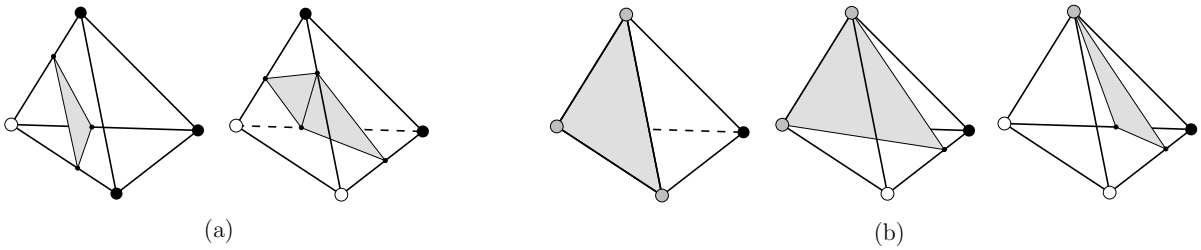


Figure 5: **(a)** Two cases for the intersection of an isosurface and a tetrahedron: either 3 or 4 intersections. **(b)** Three cases when some vertices (the grey ones) have exactly the same value as the isosurface.

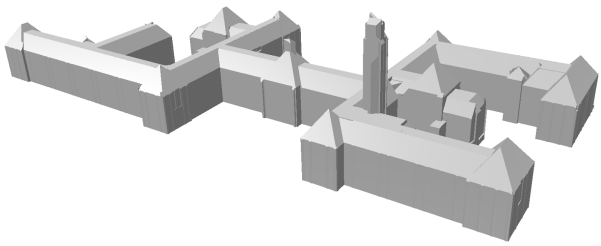


Figure 6: b-rep model of BK-City, from Lesson 2.1

graphics cards), the polygon can be split into two triangles, with an arbitrary diagonal. See Figure 5(a) on the right.

The only degenerate cases possible are when one or more vertices have exactly the same value as the isosurface. These cases are handled very easily, and the intersection is simply assumed to be at the vertices themselves (see Figure 5(b)). Notice that the case when three vertices have exactly the same value, then the complete face of the tetrahedron must be extracted to ensure topological consistency.

2 Conversions for objects

2.1 Points to b-rep

In the context of the built environment, this would most likely mean that from a point cloud, a LoD2 model of the buildings, and eventually of other objects such as trees and bridges, are reconstructed.

See Lesson 2.1.

2.2 Points/surfaces/volumes to voxels

The conversions of points, curves, surfaces and volumes to voxels are covered in Lesson 3.1.

2.3 b-rep to mesh

For the purposes of this lesson, a *mesh* is a collection of simplices that define the (3D) shape of an object (eg a building, a tree, or a bridge).

If we take the 3D model of a building (say BK-City, see Figure 6), this model is formed of several planar faces (hopefully) forming a closed 2-manifold.

In practice, if someone wants the mesh of this b-rep, it could mean two different structures:

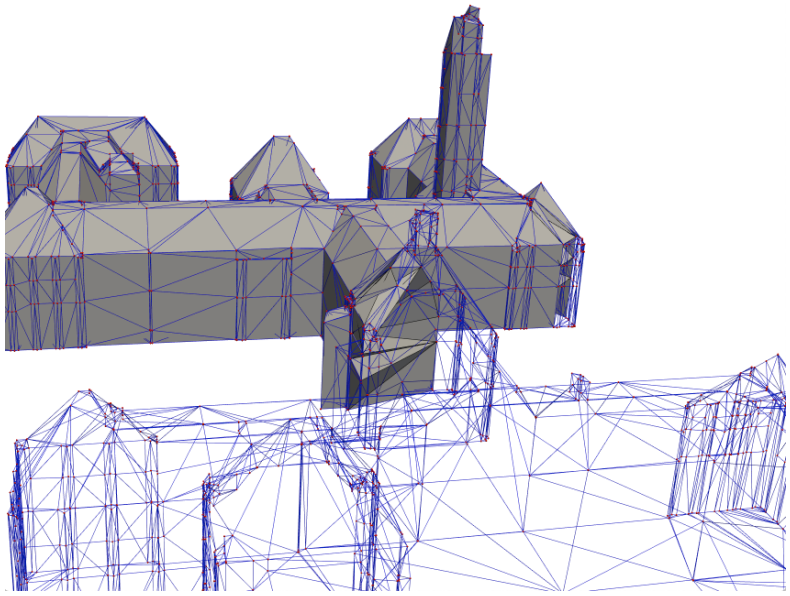


Figure 7: BK-City LoD2 b-rep tetrahedralised.

2D triangulation of each surface: the constrained Delaunay triangulation, or simply an arbitrary constrained triangulation, for each of the polygon can be created. These are independently performed for each surface, and involve transforming the 3D coordinate of the vertices of the surface to a 2D system; this coordinate system is on the plane defined by the surface. Notice that this assumes that all input surfaces of the b-rep are planar, if it is not the case then finding a projection that preserves the topology of the polygon might not be possible.

tetrahedralisation of the volume defined by the surfaces the constrained tetrahedralisation of the volume defined by the b-rep; see Section 5 of the Lesson 2.2.

Figure 7 shows one example, notice that the whole 2-manifold is tetrahedralised, but here only some tetrahedra (in grey) are shown. The surfaces of the b-rep also get meshed (each surface is triangulated) in the same process.

2.4 IFC to/from CityGML

The conversion between IFC and the CityGML data model (in either direction) is a very actual topic (many organisation would like to be able to realise it) but it is also riddled with problems caused by the differences in semantics, in data formats, and in the way geometries are modelled. Automatic conversion with commercial software, eg FME or ArcGIS, will often "work", but because of the complexity of some formats, information will often be lost in the conversion. Be aware.

The following scientific paper summarises the issues and proposes one solution. This solution is (mostly) based on the methods and algorithms we have studied so far in this course.

It should be noticed that this paper is a summary of the MSc thesis of Sjors Donkers, who studied MSc Geomatics in 2014–2015. This MSc thesis gives you an idea of what a (very good) thesis should look like, in content and in scope.

3 Notes and comments

Lorensen and Cline (1987) first describe the Marching Cubes algorithm to extract isosurfaces from voxels.

To read or watch

Donkers S, Ledoux H, Zhao J, and Stoter J (2016). Automatic conversion of IFC datasets to geometrically and semantically correct CityGML LOD3 buildings. *Transactions in GIS*, 20(4):547–569

PDF: `https://3d.bk.tudelft.nl/hledoux/pdfs/16_tgis_ifcitygml.pdf`

Full MSc thesis: `http://resolver.tudelft.nl/uuid:31380219-f8e8-4c66-a2dc-548c3680bb8d`

4 Exercises

1. Converting samples points to voxels require totally different algorithm if the samples point represent a field or an object. Discuss why.
2. If the b-rep model of BK-city contains intersecting surfaces and has gaps/holes, will it be possible to mesh the model?
3. For terrains, linear interpolation in a TIN is very popular and used. Why is it less popular for trivariate fields?
4. In the methodology of Donkers et al. (2016), why are the dilation and erosion operators used? Are they always necessary? Can you think of a simple dataset where they could be skipped?

References & further reading

Donkers S, Ledoux H, Zhao J, and Stoter J (2016). Automatic conversion of IFC datasets to geometrically and semantically correct CityGML LOD3 buildings. *Transactions in GIS*, 20(4):547–569.

Lorensen WE and Cline HE (1987). Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 4:163–168.