

Isoliner3D - 3D viewer, bed and block model

Contents

Introduction	2
Installation	2
Quick start	2
The viewer window	3
The coordinate reference system	4
A raster layer: the settings	4
Bed bodies	6
Vector layers: elevation, boreholes, the section	7
The elevation source	7
The point marker and labels	8
Colour	8
Boreholes	8
The section plane	9
The section drawing on the ribbon	10
Polygons with Z: bodies, outlines, prisms	10
Querying the scene by a click	11
Clipping the scene, markup, views	11
Clipping by elevation	12
The Bed and block model group	12
1.01 Assemble a bed grid	12
1.02 Bed calculator	13
1.03 Bed grid to a block model	13
1.04 Surfaces to 3D (meshes)	14
1.05 Domains to a bed band	14
1.06 Reserve difference (write-off)	14
1.07 Create sample data (demo)	14
Field hints	15
1.08 A map for a texture (demo)	15
Clipping by surfaces	15
The Interpolation in three dimensions group	15
2.01 Demonstration boreholes in three dimensions	15
2.02 Interpolation of points in three dimensions	16
2.03 Cube to a block model	16
2.04 Cube body as voxels	17

2.05 Check of the interpolation	17
2.06 Kriging in three dimensions	17
A cube of values: an isosurface and voxels	18
Several shells at once	18
The coordinate box	19
A wall along a line	19
Cutting a body	19
Shells in a table	20
Cleaning an isosurface	20
Voxels	20
Exporting the scene	20
The neighbouring plugins	21
Typical situations and solutions	21
Appendix. The multiband grid convention	23

Introduction

Isoliner3D is a 3D window of its own for QGIS. It shows project rasters as surfaces, multiband grids as watertight bed bodies, boreholes as coloured stems and polygon layers carrying a Z elevation as volumetric bodies. The module does not depend on the built-in QGIS 3D view or on Qt3D: the render runs on pyqtgraph and PyOpenGL bundled with the plugin, nothing has to be installed.

The second part of the module is the **Bed and block model** tool group in the Processing toolbox: bed grid assembly, reserve calculation, the block model, domains, write-off, mesh export. It has a chapter of its own below.

Interpolation, kriging, isolines, relief and cross-sections stay with the Isoliner plugin, of which Isoliner3D is a companion. Isoliner3D reads ready grids, computes volume and reserves from them and shows the result in three dimensions.

Requirements: QGIS 3.16 or newer, Qt5 or Qt6. No external dependencies.

Installation

Plugins - Manage and Install Plugins - Install from ZIP, pick isoliner3d.zip. No QGIS restart is needed.

An **Isoliner3D** toolbar with a single button appears, together with the **Plugins - Isoliner3D - 3D surface viewer...** menu entry. If pyqtgraph or PyOpenGL are unavailable for any reason, the entry is not created and the reason is written to the QGIS message log, section Isoliner3D.

Quick start

1. Open a project that already contains rasters: roof and bottom grids, relief, multiband bed grids.
2. Press the **Isoliner3D** toolbar button.
3. On the **Layers** tab tick the rasters you need and press **Update the scene**.
4. Raise the **Vertical exaggeration** until the structure reads, 3 to 10 is usually enough for gently dipping beds.
5. Spread the surfaces with **Z spacing** so that the pile unfolds into a stack.
6. Save the frame with **PNG snapshot...**

The viewer window

The layer list is on the left, the scene on the right. The scene rotates with the mouse and zooms with the wheel. Large grids are thinned automatically: the vertex budget is counted for the whole scene and shared between the ticked layers, so a single surface gets more detail than a pile of ten.

The list is one, rasters and vectors together, as in the QGIS layer tree. The layer type is marked in the row, because the set of properties depends on it. The first row is a pinned **Scene**: the scene-wide settings are the same kind of list object as a layer.

The list order follows the map tree, from top to bottom. The drawing priority comes from the same order: the upper map layer is drawn over the lower one where the geometry coincides. The list follows the tree by itself, so layers can be added and reordered without touching the window.

The tick includes a layer in the scene and acts at once, without a rebuild: the items already sit in video memory. The properties open on a double click or with the right button, the properties window is not modal and changes its content when another row is selected.

The scene is computed by the **Rebuild the scene** button, which stands first on the icon panel and is separated from the rest. Ticks and sliders only record what to show. While the shown scene lags behind the settings the button is highlighted and the status line says what to press. The **Update automatically** tick in the scene properties brings back the former behaviour and suits light data.

The vertex budget is set in the scene properties by the **Vertex limit for the scene (thousands)** row and is shared between the layers drawn as bodies. If a layer ran out of budget, the status line names the numbers: how many vertices were taken and what the limit is.

A panel of icons sits over the scene at the top left: rebuilding the scene, top view, parallel projection, contour drawing, markup visibility, clip removal, saving the contour as a layer, copying and saving a snapshot.

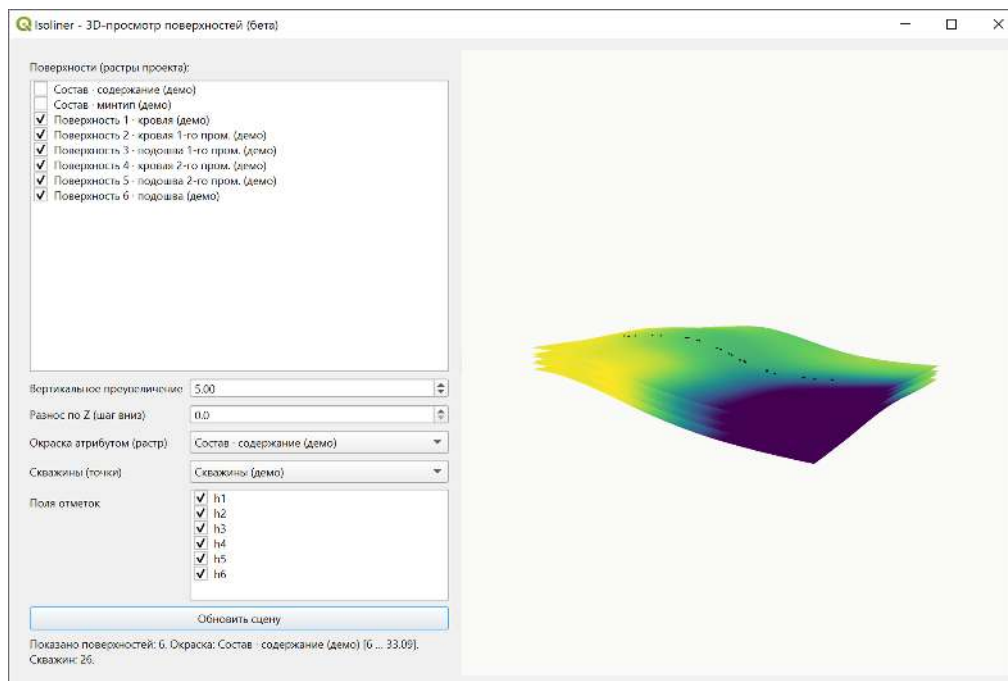


Figure 1: A stack of surfaces coloured by an attribute grid. The scale bar with the range sits under the buttons.

The coordinate reference system

The scene lives in the project CRS, just as the map canvas does. Layers in other systems are reprojected on the fly, and the number of reprojected layers is printed to the status line. The elevation is left alone, it is already in metres.

Changing the CRS of a layer in QGIS does not move the stored coordinates, it changes how they are read. So a layer given the right system will fall into place only after the scene is recomputed.

Raster values are read in the raster's own grid: colouring, the click query and the ray search of the surface take the point back. Raster clipping is also computed on its grid, so the contour is taken into the layer's system.

A raster layer: the settings

The list shows all the project rasters. The **Filter layers...** line narrows the list by a substring, the **All** and **None** buttons check and uncheck the rows visible after the filter. A scene set is assembled by hand in seconds even in a project with dozens of rasters. The checks survive a list refresh, so adding a new layer to the project does not reset a scene that is already composed.

Under the list is the **Layer settings** panel for the selected row. The settings are individual per layer and are remembered for the session.

- **Mode:** Auto (a multiband grid is drawn as a body, a singleband one as a surface), Surface (forced, any band used as heights), Bed body.
- **Elevation band (Z)** - a drop-down of this raster's bands with the names taken from the band descriptions.
- **Colouring** - a single list: Palette, Custom colour, then the layer's own bands by name, then the project rasters. Picking an external raster enables the **Attribute band**, and Custom colour enables the swatch to the right of the list. A click on the swatch opens the colour picker, the colour lives in the layer settings.

A texture is set in one of two ways. The **Project map (texture)** entry takes **all the visible layers of the tree** and renders them as a stack: handy when you need exactly what the map shows, but the extra visible layers get onto the texture too. The **Texture: layer name** entries at the end of the list take exactly one layer, and its visibility in the tree does not matter. The second way is more precise and lets different surfaces wear different maps.

Either way QGIS renders the layers over the extent of the surface itself, and the result becomes a texture. Anything will do, an orthophoto, a basemap, a geological map with all its symbology and labels, a hillshade. Reprojection is handled by QGIS, so the coordinate system of the map layers may differ from that of the grid.

The image resolution is set by **Texture side (pixels)** under the scene-wide settings. It does not depend on the mesh density, so the thinning of large grids does not affect the map detail. The price is video memory: a 4096 by 4096 texture takes 64 MB.

The texture colour is multiplied by the shading from the surface normal. Without it the relief under the map would stop reading, because the light and shade by which the eye recognises form would be gone. The scene grids themselves do not go into the texture: draping a surface onto itself makes no sense.

Separately from that list works the **layer's own ramp**. If the raster is styled with a continuous ramp, a discrete one, an exact one or a palette of classes, the surface is coloured exactly as the raster on the canvas: the same styling is read. Values outside the ramp are clamped to its ends, gaps go grey.

The colouring priority is this: the texture, then the layer's own band, then the external raster, then the layer ramp, then the palette. What was chosen by hand beats the styling. The layer

ramp in turn beats the shared scene scale: the shared one is stretched over all layers at once, and the colouring would drift away from the map exactly where it was asked to match it.

The shared scale is one per scene, the bar with the range appears under the buttons, no-data cells stay grey.

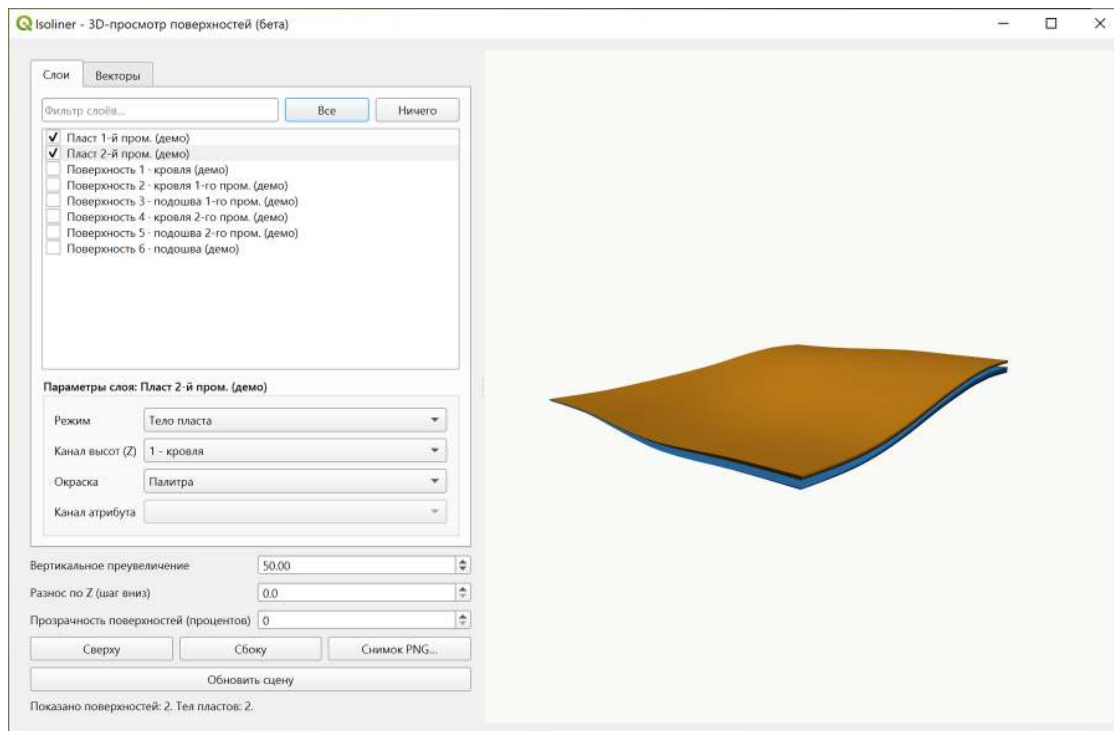


Figure 2: The **Layers** tab: the filter, the **All** and **None** buttons, a set of two bed bodies and the **Layer settings** panel of the selected layer.

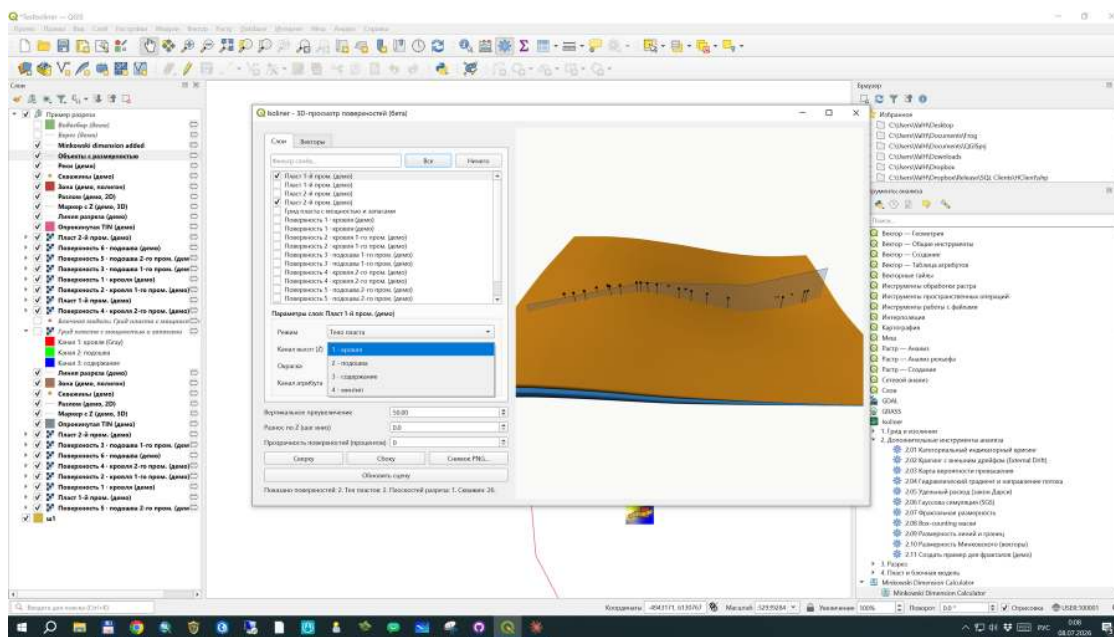


Figure 3: The band lists show the names from the grid descriptions: roof, bottom, content, mineral type.

Bed bodies

In the Auto mode a multiband grid is read as a body by the convention: band 1 is the roof, band 2 is the bottom. The volume is closed by a side skirt along the data boundary, which yields a watertight body that looks correct from any side and is cut correctly by the section plane.

Bed grids assembled by the Isoliner tools show their own band names in the lists: roof, bottom, then the parameter names. Bodies and plain surfaces live in one scene and obey the shared exaggeration and transparency settings.

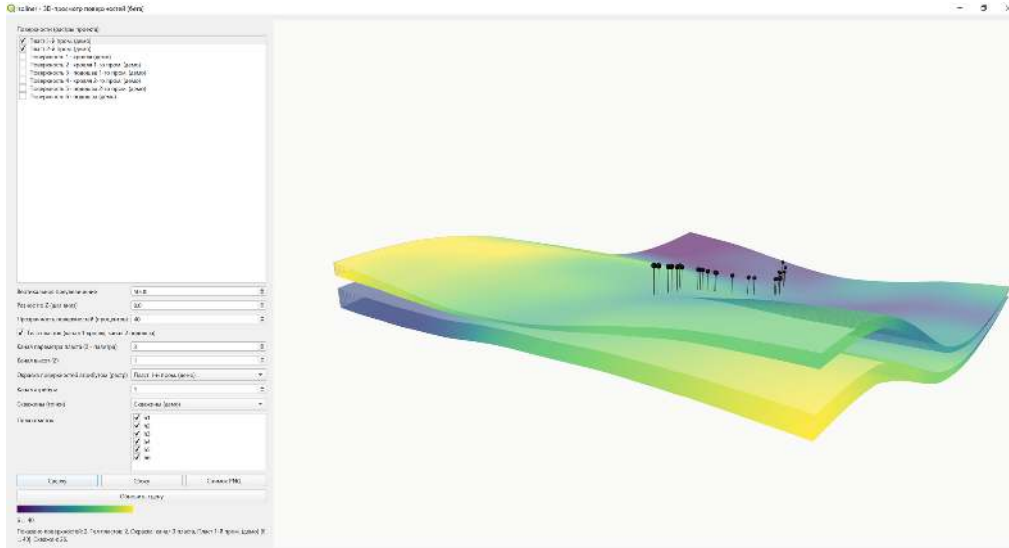


Figure 4: Two bed bodies, each coloured by its own grade band. Boreholes pierce the stack.

Vector layers: elevation, boreholes, the section

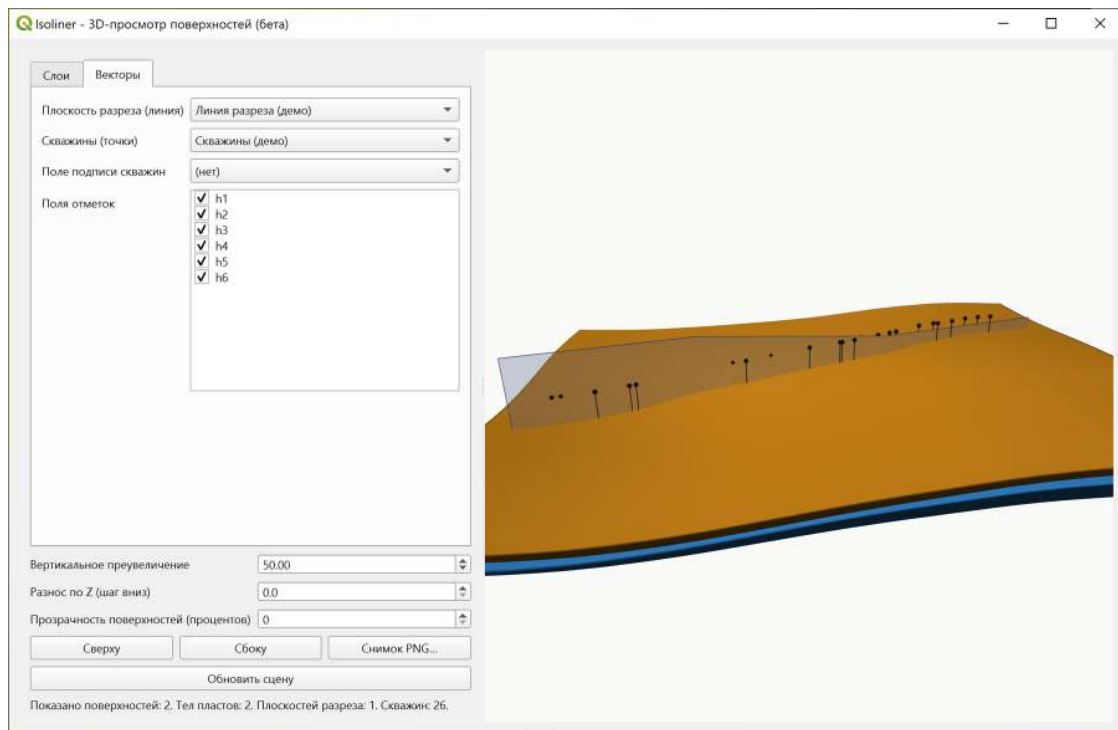


Figure 5: The **Vectors** tab: the section plane, the boreholes, the label field and the elevation fields. The scene shows the section ribbon with boreholes on a bed body.

The set of properties adapts to the layer: a row that does not apply is removed together with its label rather than greyed out. A point layer has no prism rows, lines and polygons have no point-type row, the borehole fields are visible only in borehole mode, the section only for line layers.

The elevation source

Own geometry elevation (Z) takes the value from the vertices. The entry is unavailable to a layer without Z.

Elevation from a field puts the whole feature at one elevation, as befits a contour.

Elevation from a surface takes the value off a chosen raster. It is read at every vertex, so the feature follows the relief instead of standing at one common elevation. Where the surface has no data, the feature is cut away: a point is dropped, a polyline is broken into pieces, and a body triangle with a gap at any vertex is not built at all. Zero will not do here, it is an elevation, not the absence of one. At the very edge of the data the value is filled from the nearest cell: bilinear sampling needs four neighbours and stays silent on the border even where the cell exists.

Flat, at zero puts the layer into the zero plane.

On top of any source works the **Vertical offset, m**. A small lift removes the depth fight when a line lies exactly on the surface it was taken from.

The point marker and labels

Marker shape is chosen from a circle, a square, a diamond, a triangle and a cross. The circle and the rest are built differently, and that is not a matter of taste.

The circle is drawn as an on-screen marker: the size is set in pixels, the marker does not grow when you zoom in, it reads at any distance and costs almost nothing. In exchange it is always drawn over the scene and is never hidden by a surface.

The other shapes lie flat in plan at the elevation of the point, and the size is set in metres. Such a marker is hidden by a surface and goes under a roof, so it shows where the point sits relative to the bed. In exchange it flattens when seen from above. It costs two to four triangles per point: a layer of eight and a half thousand points is seventeen thousand triangles.

The size row follows the shape: pixels for the circle, metres for a flat marker. A zero in the circle size means «from the layer style»: the marker size on the map is set in print millimetres and is converted from the usual two.

Point label field sets what to label with. Labels are thinned: if a labelled point is already near, the text is skipped. The **Labels at most** row limits their number, because every label is a separate drawing item. Zero means «no labels».

Labels are drawn with a halo: the text is outlined in a contrasting colour, as on topographic maps, otherwise it sinks into a busy scene. The size is on screen, the label always faces the camera and stands at an offset from the point.

Colour

A vector layer has no colour of its own. The feature colour comes from the layer styling: categories, graduated classes, rules. Contours coloured by elevation arrive in the scene with their own scale. A class unticked in the layer legend does not reach the scene at all.

Boreholes

Boreholes (points) takes a point layer. The list below holds the numeric fields, in which the horizon elevations have to be ticked. Fields named like h1...h6 are ticked automatically.

Every borehole is drawn as a stem of cylindrical segments between neighbouring elevations. The intervals are coloured by stratigraphic position, that is by the order of the ticked fields, so the same horizon reads in one colour across all the boreholes. A mast with a ball is placed above the collar: it lifts the collar above the roof by two percent of the scene span, and the borehole stays visible even where the stem runs inside an opaque body. The collar balls are drawn while there are no more than five hundred boreholes.

Borehole label field adds text above the masts. Fields named like name and well are guessed automatically, the **(none)** entry switches the labels off. The labels are thinned: if a labelled borehole is already nearby, the text is skipped, and a dense well stock stays readable. The cap is 500 labels per scene.

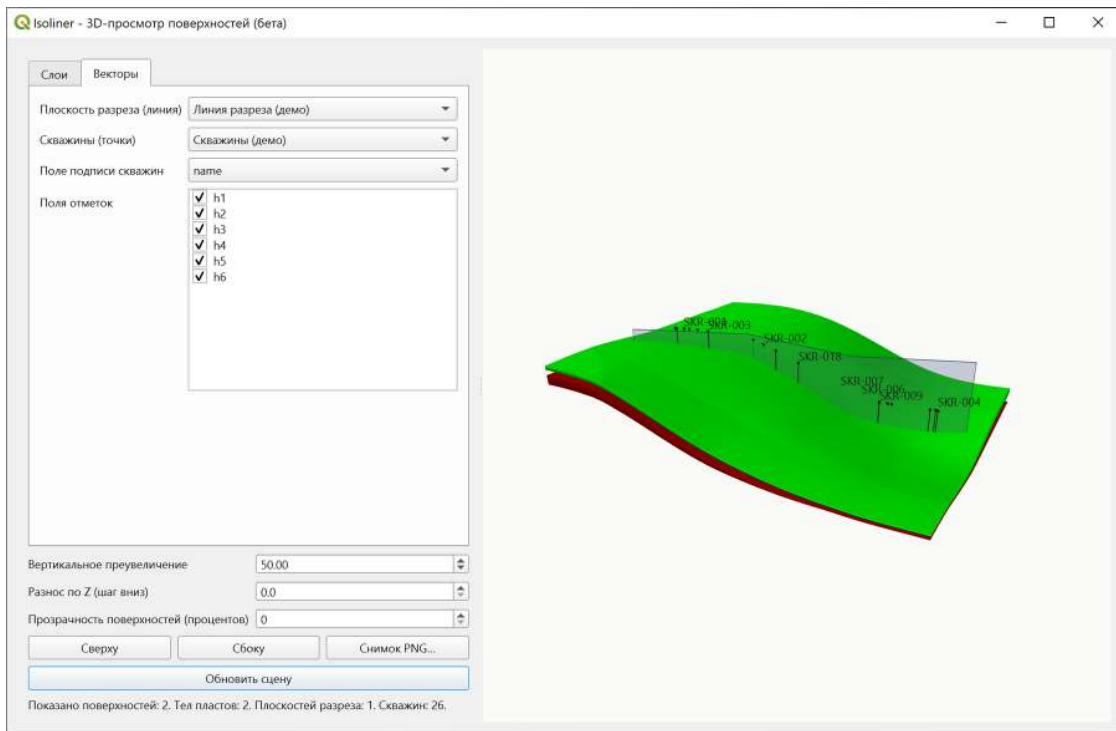


Figure 6: Borehole labels above the masts with automatic thinning. The **Vectors** tab with the label field on the left, the bed bodies coloured with custom colours.

The section plane

Section plane (line) accepts any line layer. The most convenient input is the **Section definition** layer produced by the **Section along a line** tool of the Section group in the Isoliner plugin: the ribbon takes its height range from the *zmin* and *zmax* fields. For an arbitrary line the ribbon is stretched over the scene span with a margin.

Polylines and several lines in one layer are supported, the bends are drawn by the vertices. A bright intersection trace runs along the ribbon over the surfaces, and for the bodies from the **Bodies** tab a section contour is drawn, that is the line where the vertical curtain along the section cuts the body.

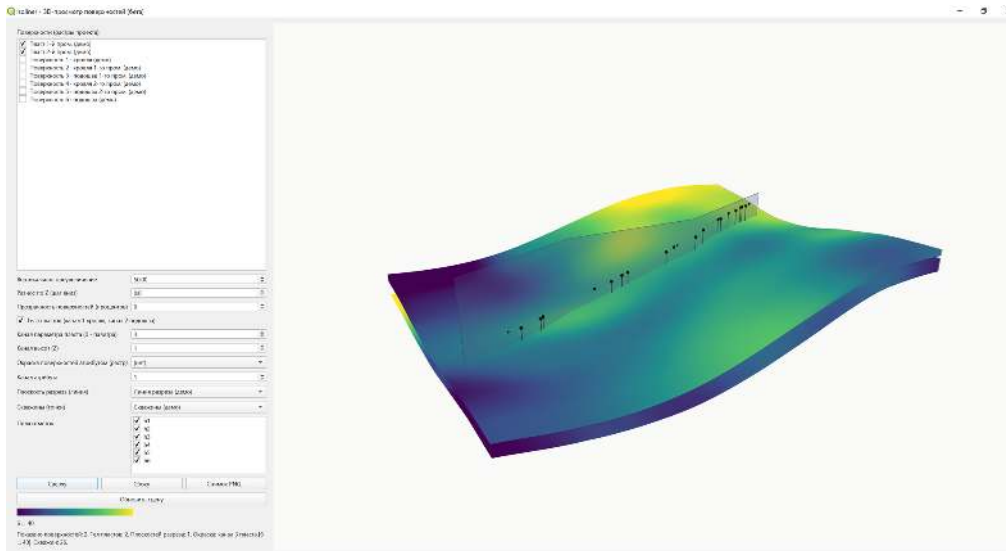


Figure 7: Bed bodies, boreholes and the section plane in one scene: the block model stitched with the section.

The section drawing on the ribbon

The **Section drawing (layer or group)** field dresses the ribbon in a real drawing. The Isoliner section tools build it in «distance along the line by elevation» coordinates, and the ribbon in the scene occupies exactly the same region of space, so the drawing lands where it belongs with no recalculation at all: distance runs along the ribbon, elevation across it.

Either a single layer or a whole group can be picked: a drawing usually has several layers (beds, boreholes, labels), and a group is taken with all its symbology in tree order. Layer visibility does not matter, the drawing lives in its own coordinates and is in the way on the map anyway.

No shading is applied to the drawing: it has to read as a drawing rather than as a lit surface. The resolution is set by the same **Texture side** field.

Several sections are shown at once. The Isoliner tools build the drawings for every line of the layer in one go and lay them out side by side, and every definition line carries its own fields: the section number `sec_id`, the name `sec`, the vertical exaggeration `vex` and the layout offset `ox`, `oy`. That is how the module finds the area of its own section on the drawing layer.

The extent comes from those fields rather than from the bounds of the layer features. A table of distances and azimuths hangs outside the drawing frame, and going by feature bounds would shift the image. What is worse, this would only show up for those who build with the table.

The names of the shown sections are printed in the status line.

When the `ox` and `oy` fields are missing from the definition, the drawing is not draped and the reason goes to the log: the section was built by an older version of Isoliner and only needs rebuilding. When the section number is there but the chosen drawing layers do not contain it, the drawing and the definition come from different builds and the ribbon stays without an image.

Polygons with Z: bodies, outlines, prisms

The **Bodies** tab shows polygon layers carrying a Z elevation as volumetric bodies right in the scene, next to the surfaces and the bed bodies. Polyhedral surfaces, TIN and any MultiPolygon Z layers will do. Tick the layers you want and press **Update the scene**.

The geometry of each feature is broken into triangles as a separate mesh and coloured on its own, so a suite of several beds comes out multi-coloured. The same vertical exaggeration and transparency apply to bodies as to surfaces, so a polyhedron and a stack of horizons read at one scale.

This is the only way to see an overturned fold or an overhang in three dimensions. A grid cannot describe such a form at all, because a grid keeps one elevation per node, while a polyhedron or a TIN keeps a true surface with several Z values above one point of the plan.

Querying the scene by a click

A click on a surface or a body queries the model. A ray is cast from the camera through the cursor, the nearest intersection with the surface is found, and the status line prints the layer name, the point coordinates and the values of all the bands by name, plus the thickness for a bed. The hit is marked with a red ball. It can be cleared in three ways: a click on empty space, the Esc key outside drawing mode, and the clear button on the panel, which removes the clip, the sketches and the point at once.

Dragging is separated from querying: rotating the scene with the mouse works as usual, the query fires only on a click without movement.

Clipping the scene, markup, views

The scene can be cut so that only the part you need is left. The clipping contour is set in the scene properties: any polygon or line layer of the project, or one drawn right in the scene. The **The piece** row sets what to show.

For a polygon it is **Keep what is inside** or **Remove what is inside**: a slice of the cake or the cake without the slice. For a line it is **To the left of the line**, **To the right of the line** and **A corridor along the line**. The corridor takes a band of a given half-width on both sides of the profile, and that is usually more useful than a bare section: the data stay next to the line, and it is visible how the structure changes away from the profile. The half-width is entered on the panel over the scene, next to the drawing icons, and is shown only in the corridor mode.

The edge follows the contour rather than the bounding rectangle, and the holes of the contour stay holes. It is not only the surfaces that are cut: lines break where they leave the piece, points and boreholes are selected by their position, bodies by their centre.

Drawing in the scene. The contour icon turns on the markup mode: a click on the surface adds a vertex, a rubber band follows the cursor, the right button or the neighbouring icon removes the last vertex. From there two ways. The icon with a tick closes the contour and clips an area by it. The polyline icon finishes a line and clips a corridor along it, switching to the right mode by itself.

The vertices are taken from the surface by a ray from the camera, so the markup lies on the relief and stays true when the scene is rotated. When the scene has no raster surfaces, say only isolines are shown, a vertex is taken from the level of the middle of the scene. The plan position of the vertices is what the clipping uses. The markup itself is drawn over the model and does not hide under folds.

The eye icon hides and shows the markup without removing the clip. The icon with a crossed-out contour removes the clip entirely and clears the sketches. The icon with a stack of sheets saves the drawn contour as a project layer, after which the Isoliner tools can use it.

Views. The icon of a frame with a crosshair sets the top view and turns on the parallel projection as well, because a plan with perspective is not a plan. The projection is also switched separately by the cube icon: in a parallel projection the scale is the same across the whole frame, and objects at different heights do not shift relative to each other.

Snapshot. The icon with two rectangles puts a frame of the scene on the clipboard, and so does Ctrl+C. The camera icon saves the frame to a PNG file. The snapshot size equals the size of the scene window, so it is worth maximising the window before shooting.

Clipping by elevation

A contour and a corridor cut in plan only, they have nothing to do with height. A section across a pile of beds is set by elevations, and for that the toolbar carries two fields, $z \geq$ and $z \leq$. The lowest value means «no bound», as the caption shows.

The filter applies to bodies and belts, lines, points, raster surfaces and voxels. The bounds are inclusive: an elevation exactly on the bound stays, otherwise the outermost level of a cube would disappear. Bounds given the wrong way round yield nothing rather than being swapped silently: a typo in a field is better seen at once than hunted for in the data.

The Bed and block model group

Besides the 3D window the module installs an **Isoliner3D** provider into the **Processing** toolbox, with a single group - **Bed and block model**. Its seven tools compute on NumPy and GDAL, they need neither kriging nor isoline building, so they work without the main Isoliner plugin.

The pipeline runs like this. A roof and a bottom are assembled into one multiband bed grid (**1.01**), the calculator derives thickness and reserves from it (**1.02**), the block model unfolds the grid into centroid points (**1.03**), and the difference of two models gives the write-off (**1.06**). Domains (**1.05**) add an area code band to the grid, the mesh export (**1.04**) hands the surfaces over in the 2DM format, the generator (**1.07**) creates demonstration bodies with Z and a test map for the texture.

An assembled bed grid is read by the 3D window as a body: band 1 roof, band 2 bottom. Compute it and look at it right away.

1.01 Assemble a bed grid

Assembles a multiband bed grid by the module convention: band 1 roof, band 2 bottom, bands 3 and on the parameters (grade, mineral type, any other numeric field).

The roof defines the grid of the result. The bottom and the parameters are resampled onto it bilinearly, so the source grids may differ in cell size and extent. The band names are written into the raster band descriptions: roof, bottom, then the names of the parameter layers. That is why the 3D window lists read “content” instead of “band 3”.

Parameter	What it sets
Roof (raster)	the roof grid, defines the output grid
Bottom (raster)	the bottom grid
Parameters (rasters, band 1 is taken)	any number of parameter grids, the field is optional
Roof band, Bottom band	for multiband sources
Bed grid	the result

One run assembles one bed: one roof, one bottom, one output grid. A pile of three beds means three runs, or a single go through the **Run as batch process** button, where every table row sets its own pair of layers.

About the parameter list. The field is optional. Leave it empty and you get a two-band grid, a roof and a bottom. For viewing as a body, for the calculator and for the block model that is already enough.

The list is for the accompanying values tied to the same area: grade, mineral type, recovery factor, any numeric characteristic. The ticked rasters go into the grid as separate bands starting from the third one, band 1 is taken from each.

The point is that after the assembly a bed is described by a single file. Everything else then works off it: the calculator will ask for the grade band and compute the mean grade and the metal tonnage, the block model will lay every band out into its own attribute field, the 3D window will list the bands under their names in **Colouring**. That is exactly why the names come from the layer names: the lists then read “content” instead of “band 3”. It is worth tidying up the layer names in the project before the assembly.

The parameter grids may have their own cell size and their own extent, everything is resampled onto the roof grid bilinearly. It is the roof that defines the output grid, which is worth keeping in mind when choosing what to feed as the first input.

The **Roof band** and **Bottom band** fields live in the advanced section and are only needed when the source grids are themselves multiband. For ordinary singleband ones they stay at one and need no attention.

1.02 Bed calculator

Computes thickness, volume and ore tonnage via the density from a bed grid. When a grade band is given, the thickness-weighted mean grade and the metal tonnage are added.

The summary is taken over the whole bed area or inside a contour: the polygons of a mining block or a domain. Holes in the polygons are honoured. The result is a bed grid with the thickness and the per-cell ore reserve appended as bands, plus an HTML report with the summary.

Cells with a negative thickness, where the bottom ended up above the roof, are zeroed and counted separately. Their number is worth checking in the report: it means the surfaces intersect and the model needs a fix.

Parameter	What it sets
Bed grid	band 1 roof, band 2 bottom
Grade band	empty - compute without a grade
Ore density, t/m³	to convert volume into tonnage
Calculation contour (polygons)	limits the area
Bed grid with thickness and reserves	the result
Report (HTML)	the summary

1.03 Bed grid to a block model

Turns a bed grid into a block model: one centroid point per valid cell. The attributes are the cell row and column, the coordinates, the top and the bottom, the thickness, the volume, the ore tonnage and every parameter band under its name from the descriptions.

From there the ordinary QGIS vector machinery applies: expression filters, joins with external tables, the field calculator. The model grows new attributes without being rebuilt.

The **Vertical layers** parameter splits every column into N blocks between the roof and the bottom. Each block gets its own top and bottom elevations, a layer number and a share of the volume. The grade is copied into the sub-blocks as is, because it is not drilled out vertically. The reserve sum is preserved by the split, which makes a handy self-check: build the model with one layer and with five, the tonnage sums must match to the last digit.

The density comes from the number in the parameters or, when a **Density band** is given, per cell from the grid. The latter is what you need where the ore density varies across the area.

1.04 Surfaces to 3D (meshes)

Exports surface grids into mesh layers of the 2DM format (MDAL). Such layers are understood by the QGIS profile tool, the mesh calculator, the built-in 3D view and third-party software.

A vertical transform is applied to the elevations: an elevation is multiplied by the scale and shifted by the offset. The scale gives the vertical exaggeration, the offset raises or lowers a horizon. **Z spacing** pushes every next grid down by a constant step and turns a stuck-together pile into a readable stack. Thinning reduces the node count on large grids.

The layers are loaded into the project and get a 3D rendering automatically. Cells without data are skipped.

1.05 Domains to a bed band

Rasterises domain polygons into an extra band of the bed grid: every cell gets the code of the domain it falls into, zero outside the domains. The code comes from a numeric field of the layer or, when no field is given, it is the feature order number starting from one. The bands of the source grid are preserved, the domain band is appended last.

From there a domain behaves like any other parameter: the calculator works over the domain contour, the block model is filtered by an expression on the code. The domain contours must be in the same coordinate system as the grid.

1.06 Reserve difference (write-off)

Computes the difference of two block models over the cells with matching row and column: how much reserve was removed between the “before” and the “after” states. The chosen field is subtracted per cell, the ore tonnage by default. The result is points with the before, the after and the difference values. The total write-off is printed to the log.

This is the direct route to operational write-off: the model before the chambers were mined out minus the model after, and the sum of the difference over a contour gives the written-off tonnage that goes into the statement. Both models must be built from the same grid, otherwise the row and column split will not match.

1.07 Create sample data (demo)

Creates demonstration data, so that you can check the display on your own QGIS build without touching the working layers. The variant is set by the **Example** list.

Bodies with a Z elevation. A bed body is a watertight shell of a roof, a bottom and a side skirt. A suite is a stack of folded beds, each loaded as its own layer with its own colour. There are also a cube and a tetrahedron. The plan position and the size come from the extent, vertically the body runs from the base elevation up to the base plus the thickness.

The geometry type is flat, so the elevation is not visible in the 2D view, the Z range is printed to the log. The body itself is best looked at on the **Bodies** tab of the 3D window. A native PolyhedralSurface Z is available from QGIS 3.40, on older builds the output degrades to MultiPolygon Z. The TIN flag yields a triangulated surface.

Map (raster for a texture). A three-band image to check the draping of a texture. The extent is best set by the **Map: by the extent of a grid** field, then the map lands exactly on the surface bounds.

The image is deliberately a test pattern rather than a pretty map. Draping fails in three typical ways, and this map shows each of them at once. A vertical flip shows up in the differing corner marks, a shift or a skew in the graticule, a stretch along one axis in cells that stop being square. On a pretty map none of this is visible.

The check goes like this. Create a map by the extent of your grid, set the grid colouring to **Texture: Map (demo)** and update the scene. If the corner marks are where they belong and the graticule cells are square, the draping works correctly.

Field hints

Every field of every tool carries its own hint, shown right next to the input. The general help sits aside and is read once, while «what do I put here» has to be decided at every field. The hint answers that and says what the other choice costs. The grid step, for instance, says that finer than the distance between places is pointless, because there is no data in between and the node count grows as a square.

1.08 A map for a texture (demo)

Draws a check map with a coordinate grid and bed fields. It is there to see how a texture lands on a surface in the viewer: on a real map skews and stretches show up worse than on squares.

The extent is taken from a ready grid when one is given, otherwise from the extent field: the map then lands exactly on the bounds of the surface.

The map used to be one of the examples in 1.07. Four fields out of thirteen worked for it there, and the rest were not read at all when it was chosen.

Clipping by surfaces

The roof and the floor change across the area, and a flat elevation cannot replace them. Clipping by two surfaces leaves what is between them: that is how everything above the ground surface or outside the bed is removed.

In the scene these are two property rows, any raster of the project. In tool 2.03 the same is done for the block model: the number of removed points goes to the log.

A single surface works too. A point with no surface under it does not stay: letting it through would show data where the clipping did not work.

The Interpolation in three dimensions group

The second group of the Processing toolbox works with a cube of values. A cube is a multiband grid where a band is a horizontal level, and the elevation of the first level and the step live in the Z0 and DZ metadata.

2.01 Demonstration boreholes in three dimensions

Data with a known truth inside: the grade is set by a model and the noise is added separately. Methods of interpolation in three dimensions are compared on such points, because the error is measured against the model rather than by eye.

Deposit type sets the shape of the body. A folded and dipping bed is there to show the main point: cube levels cut the deposit across. A lens is isotropic and the simplest case, a steep vein is the opposite extreme, where the body is nearly vertical.

The borehole grid is jittered, the collars follow the relief, the depths differ and some holes are stopped short. A regular grid of equal depth would give interpolation too easy a task.

Sampling goes by intervals. Layer fields:

field	what it is
hole	the borehole number
from_m, to_m	the sample interval down from the collar, metres
grade	the assay with sampling noise
truth	the grade from the model without noise
zone	one inside the body, zero outside

The noise is lognormal, so no negative grades appear.

For interpolation take grade. The truth field is there to separate the error of the method from the sampling noise: compute the cube from grade and compare it with truth. The hole field is needed in 2.05, to remove a whole borehole.

The same list is printed to the log of 2.01 when it finishes: the explanation is needed where the data has just been created.

The boundary of the body is where the grade falls to half the core value above background. That is the cutoff, and it is printed to the log together with the number of samples and the range of grades.

The site is set by an extent. An empty extent gives a kilometre from the origin, which is written to the log.

2.02 Interpolation of points in three dimensions

The main list holds ten rows and all of them are about the input data. Method tuning has moved to the advanced ones: anisotropy, radius, power, the smallest number of points, sectors.

The elevation source is set apart from the geometry. A flat layer gives a zero Z at every point, and taking it as is would put every sample into one plane. The elevation comes from a field when it has been computed, or as a depth down from a chosen surface: the latter is for soil and similar samples, which record a depth rather than an elevation.

The grid step and the largest number of points are taken from the data if left at zero. The step in plan is a fifth of the distance between places in plan, the vertical step is half the sampling step, and the neighbours are one more than the samples at one place. What was substituted is printed to the log.

Neighbours are gathered by sectors. Without this, under anisotropy all the neighbours land in one borehole, and inverse distances degenerate into nearest neighbour.

Anisotropy is the ratio of the vertical scale to the horizontal one: when drilling with boreholes there is an order of magnitude more data along the vertical, and without anisotropy the interpolation would pull values vertically harder than it should.

Nodes with fewer points within the radius than the given minimum stay a gap.

Distances are computed in blocks of nodes, so the time grows with the number of samples rather than with its square. A thousand samples on a forty by forty by forty six cube take about a second.

2.03 Cube to a block model

One centroid point per occupied cell. A cube as a set of bands is addressable by nothing: a band is a number, not an elevation, and neither an expression filter nor the attribute table works on it. A block model gives the cell back its number, coordinates, size and value.

Fields: bid, lev the level, row and col the grid cell, x, y, z the block centre, dx, dy, dz the block size, vol the volume, val the value, cls the colour interval number, and dens with ore_t when a density is given.

Gaps and cells below the cutoff are not written out, so the model comes out sparse and weighs an order of magnitude less than a full box with empty edges. A contour limits the export to a computation block.

2.04 Cube body as voxels

The same thing the scene shows as voxels, but as a layer: MULTIPOLYGON Z, one feature per colour interval. Fields: `cls`, `vmin` and `vmax` the interval bounds, `faces` the number of faces, `shell` one for a body.

The **Merge neighbouring faces** flag makes the layer light but breaks watertightness. For volume computation the flag is cleared.

Edge pinches are counted separately. A pinch is two cells touching along a single diagonal: it is not a hole and does not spoil the volume, but its edge belongs to four faces and a watertightness check rejects such a body. The **Remove edge pinches** flag fills the corner with one cell, and the contact becomes a face contact. On the demonstration bed of seventeen thousand cells there is exactly one such pinch, and two added cells cure it.

2.05 Check of the interpolation

Removes each sample in turn, computes the value at its place from the rest and compares it with the real one.

This is the only way to learn whether the cube can be trusted: there is nothing to compare the built model with, and to the eye a good model and an invention look equally convincing.

What to remove. Without a borehole field one sample is removed. On an exploration grid that flatters the model: it takes neighbours from the same hole three metres away, and what is measured is continuity along the hole rather than the ability to hit between holes. On the demonstration data the difference is sixfold: the error by samples is 0.17, by holes 1.10. Given a borehole field, the whole hole is removed.

A check by single samples also barely sees the anisotropy: the choice of nearest points does not depend on it while the nearest point is the layer's own down the hole. Such a check cannot decide anything about the plan.

The parameters are the same as in 2.02. By changing them and watching the error one picks the anisotropy, the power and the number of neighbours: there is no right value for them at all, only the best one on this data.

The log gets four numbers. The mean error and the root mean square one tell how large the miss is, the bias tells whether the model leans one way, and the share of the error in the spread of the data puts it in scale: one is a lot on grades up to two and little on grades up to a hundred. Scatter and a one-sided lean look the same and are cured differently, so the bias is kept apart.

Fields of the residual layer: `value` the real value, `model` the computed one, `resid` the difference, `aresid` its absolute value. They show not only how large the miss is but where it happened.

A thousand and a half samples are checked in about a second.

2.06 Kriging in three dimensions

Computes a cube of values by kriging and gives a cube of the estimation variance as a second output.

How it differs from 2.02. Inverse distances weigh by distance alone: they do not care at what distance the connection fades or how much of the scatter is sampling error. Kriging takes its weights from the variogram and knows both. It also accounts for neighbours knowing about each other: two samples side by side carry almost the same thing and are not given a double vote.

When it pays off. Not always, and this is worth knowing in advance. On the demonstration data at different grid densities:

holes	grid step	range	inverse distances	kriging	
16	193 m	243 m	1.331	1.454	−9 %
25	142 m	285 m	1.047	1.055	−1 %
49	109 m	264 m	0.936	0.896	+4 %
100	78 m	294 m	0.643	0.601	+7 %
196	54 m	261 m	0.483	0.446	+8 %

The turn is where the grid step is about half the range. When holes stand farther apart, neighbouring ones know almost nothing about each other, the weights come out nearly equal for any method, and the difference goes into noise. The tool prints the grid step and the range to the log and warns when the grid is sparse.

The estimation variance. The second cube gives what inverse distances lack entirely: zero at a sample, growing away from the data. It is a map of trust, and on a sparse grid it is the only reason to take kriging. It is convenient to look at as voxels: they show at once where the model is guessing.

The variogram is measured on the data itself. The range comes from the plan measurement, the nugget from the vertical one, the anisotropy as the ratio of the ranges. The nugget must not be taken from the plan measurement: in plan there are no pairs closer than the grid step at all, the first interval starts right there, and a nugget from it is a straight line continued to zero through emptiness. Down the hole there are pairs from three metres.

Negative values. Kriging weights can be negative, and the estimate may go outside the range of the samples: on grades that means values below zero, which cannot be. The tool notices and says so. It is cured by the spherical model instead of the gaussian one, or by a raised nugget.

A cube of values: an isosurface and voxels

A multiband grid is read not only as a bed but also as a cube of values: a band is a horizontal level. That is what the result of a volumetric interpolation looks like, say grades from sampling points.

In the properties of a raster layer choose the **An isosurface from a cube** mode and set the **cutoff**. Everything not less than that value goes inside the body. The elevation of the first level and the vertical step are taken from the grid metadata, the Z0 and DZ fields. Without them the count starts from zero with a step of one.

The shell is built by a march over tetrahedra. A tetrahedron is divided unambiguously, so neighbouring cells meet face to face and the body comes out closed: every edge belongs to exactly two faces. That is needed not for beauty: the volume is computed from it and the closed cut at clipping is built from it.

Gaps in the data stay outside the body: emptiness does not attract the shell.

From there such a body lives as any other: it is clipped by a contour and a corridor, coloured, exported to GLB.

Several shells at once

The **Shells at the cutoff** row sets how many shells to build. One is taken at the given cutoff, several are spread from it up to the top of the cube. Each takes its colour from the ramp, and the outer ones are more transparent so the inner ones read through them.

The march walks the boundary cells rather than the whole cube: cells entirely inside and entirely outside give no faces, and they are the vast majority. Vertices are welded, so neighbouring faces share a point. On a hundred by hundred by sixty cube five shells take about a second and weigh eleven megabytes.

The coordinate box

A button on the toolbar draws the edges of the extent, ticks with labels and a north arrow. A scene without ticks gives no sense of size: a body looks the same at a hundred metres and at twenty kilometres. Down the vertical the elevations are labelled, which matters most for a section.

Ticks go by round numbers: the step is one, two or five times a power of ten. The edges of the extent are not labelled, the number there is usually not round.

The grid is set by the scene properties: on which planes to draw and with what step. The floor gives the scale in plan, the walls give it by elevation. Zero as the step takes a round step from the span, and too fine a step is coarsened by itself.

A wall along a line

The third way to show a cube is the **Wall along a line** mode. A shell shows the boundary of a body, voxels show the occupied cells, and a wall shows the field of values itself, where it was drawn.

The line comes from the clip list: draw one with the drawing button or pick a line layer there. There is no separate way of drawing a line for the wall, and none is needed.

The step of the nodes along the line is set by the **Wall step** row. Zero takes the grid step: finer than that there is no data anyway. Down the vertical the cube levels are used.

Values are sampled trilinearly, so the wall comes out smooth rather than stepped. Outside the cube a gap is returned rather than the edge value: extending the edge outwards would mean showing data where there is none. A triangle with even one node without data is not built at all.

A polyline with a bend is walked with one step along its whole length rather than per segment: at the bend the wall does not tear.

The wall is cheap. On a hundred by hundred by sixty cube a three-node polyline at a ten metre step gives seven thousand nodes and fourteen thousand triangles, a third of a megabyte.

Cutting a body

A body is cut by a contour, by a corridor along a line and by a range of elevations, and the cut is closed with a cap: you see the section rather than the inside.

For that the body must be watertight. Build it in 2.04 **with the merging of neighbouring faces cleared**: merging makes the layer many times lighter but tears the boundary with T-junctions, and such a body cannot be capped. The layer comes out twice as heavy — that is the price of being watertight.

Vertical faces are cut along the segment they degenerate into in plan: as a polygon they cannot be cut at all, their area is zero. Without this a wall would stick out past the contour while the neighbouring horizontal face was cut back to it, leaving a slit between them.

The log gets the numbers that show what happened: how many faces are left, the distance from the line to the data, how many bodies are not watertight before the cut, how many boundary edges landed on the cut contour and how many cap polygons were built.

Shells in a table

Shells are set in a table: a row per level, columns for the level, the colour and the opacity, plus a box for showing it. Two lists tied by order slip silently: miss a colour and all the rest land on the wrong levels. Within a row there is nothing to slip.

An empty cell takes the automatic value: the colour by the shell number, the opacity growing outwards. An empty table means the cutoff and one shell. The colour is picked by a double click, a row is removed with the right button, and a new one appears by itself once the last is filled.

Cap where the body meets the cube edge closes the shell where the body runs into the boundary of the cube. Without it the shell is not watertight: the body looks cut open and no volume can be computed. Off by default.

Inner shells are drawn before the outer ones: whatever is drawn first hides what lies behind it, so an outer shell going first would eat all the inner ones.

Cleaning an isosurface

A marching surface goes in steps of the cube cells, and small scraps on it add noise. Two rows in the layer properties: **smoothing** and **drop parts smaller than**.

Small parts are dropped before smoothing: otherwise a scrap pulls its neighbours towards itself, and after its removal a dent is left behind. The edges do not move, because the cap on the cut is built from the boundary edges.

If the threshold removes everything, the surface stays as it was. Smoothing shrinks the body a little, so for volume computation take the unsmoothed one.

Voxels

The second way to show a cube is the **Voxels from the cube** mode. A cell is drawn as a box: the grid step across, the level step down, that is exactly the volume it stands for in the computation. The colour comes from the grade interval, and the number of intervals is set by the **Colour intervals** row.

What has to be counted is faces, not cells. A face between two occupied neighbours is never seen, so it is dropped: on a filled two hundred by two hundred by one hundred cube that is one hundred and twenty six thousand faces instead of twenty four million. Neighbouring faces of one interval merge into a rectangle, and the demonstration bed of four million cells gives thirty four thousand rectangles and two and a half megabytes of scene.

The cost is driven by the surface area, not by the number of cells. A compact deposit is cheap, one scattered by the cutoff gives millions of faces and does not reach the scene: the size is estimated before building, and above the limit the log gets the number and the advice to raise the cutoff or reduce the number of intervals.

Merging costs watertightness: a long rectangle meets two short ones and they share no edge. For display that does not matter, for volume computation the merging is switched off by the **Merge neighbouring faces** tick, and then every edge belongs to exactly two faces.

Clipping voxels is a selection of cells, so no cap has to be built and the cut comes out flat by itself.

Exporting the scene

The icon of a cube with an arrow on the panel over the scene writes what is shown into a GLB file. The format is opened by browser viewers, Blender and Windows, so the model can be sent by mail as a single file.

What is visible is exported, the clipping included, with colours in the vertices. The texture is not exported yet, a surface with a map goes out in a flat colour.

The export asks about the vertical exaggeration. True elevations are right for calculation and for matching other data, the model as on screen is needed for display: a bed kilometres across and tens of metres thick would otherwise flatten into a pancake.

The **coordinate box always goes into the file**, whether it is shown on screen or not: these are different decisions. On screen the box gets in the way of looking, while in the file there is no scale without it.

The edges, ticks and grid are written as lines, the tick labels as ribbons. Line width cannot be set in glTF, the viewer chooses it, and it is usually one pixel that gets lost on a large model. A ribbon gives a real thickness, in metres.

The labels are drawn in a drafting hand: a digit made of strokes. There is no text in glTF — there is either the geometry of the letters or a picture on a plane, and a flat picture turns edge-on and disappears when the model is rotated.

The result of the export is shown in the window: how many bodies, is there a box, was the exaggeration applied, the file size.

The neighbouring plugins

Isoliner3D does not work on its own. Three plugins cover the way from observation points to a volumetric model, and each does its own part.

Isoliner builds grids and isolines: kriging with variograms, minimum curvature, relief and catchments, sections with drawings, a geological bed model. It is the one that prepares the grids and belts shown here. The catalogue: plugins.qgis.org/plugins/grid_isolines.

Topoliner tidies the contours: dangling nodes, self-intersections, gaps and overlaps between adjacent polygons, thinning that preserves topology. It helps before belts and solids are assembled, because a torn topology on the map turns into a torn shell in three dimensions. The catalogue: plugins.qgis.org/plugins/topoliner.

Isoliner3D shows the result in three dimensions, cuts the model by a contour or a corridor, computes reserves from a block model and exports the scene to GLB.

All three work independently: installing the whole set is not required.

Typical situations and solutions

What you see	Cause	Solution
No menu entry and no button	pyqtgraph or PyOpenGL are unavailable, <code>is_available()</code> returned false.	Check the QGIS log, section Isoliner3D. This usually means a broken installation: reinstall the module from the ZIP as a whole, together with the <code>libs</code> folder.
An empty scene and a request to tick a raster in the status line	No layer is ticked.	Tick something on the Layers or the Bodies tab and press Update the scene .
The message about grids that could not be opened	The files of the ticked rasters are unreachable: moved, on a disconnected drive, or the layer was temporary.	Check the layer source in the properties and rebuild the grid if needed.

What you see	Cause	Solution
The surface looks like a flat pancake	The real elevation span is small compared to the extent of the area.	Raise the Vertical exaggeration .
A bed is drawn as a surface instead of a body	The grid is singleband, or the Mode is set to Surface.	Check the band count and set Auto or Bed body.
The body is inside out, the roof below the bottom	The bands are swapped, the convention wants band 1 roof, band 2 bottom.	Reassemble the bed grid with the bands in the right order.
No boreholes are visible	No elevation fields are ticked, or all the elevations are empty.	Tick the numeric elevation fields in the list on the Vectors tab.
Fewer labels than boreholes	The thinning is at work, the cap is 500 labels.	This is by design. Move the camera closer or feed a thinned stock layer.
The surface looks coarser than the source grid	The automatic thinning to 60 thousand nodes has kicked in.	This is by design. For detail on a fragment, cut the piece of the grid you need and feed it as a separate layer.
The section ribbon runs far beyond the pile	The line was fed without the zmin and zmax fields, so the ribbon is stretched over the scene span.	Feed the section definition layer from Isoliner.
A layer sits far away from the rest	The layer CRS is not the one recorded.	Assign the right system and press Rebuild the scene : the status line will report the number of reprojected layers.
Editing a property changes nothing	The scene is computed by the button.	Press Rebuild the scene : while the shown scene lags behind, the button is highlighted.
Bodies are shown incomplete	The vertex budget ran out, the status line names the numbers.	Raise the Vertex limit for the scene in the scene properties.
Contours are now visible, now sunk into the surface	The geometry coincides and the depth fight begins.	Raise the layer in the map tree or give it a Vertical offset, m .
Part of the features vanished with elevation from a surface	The surface has no data there.	That is intended: a gap is not a zero. Check the extent of the elevation grid.
The cube came out in steps by number	The value field is hole.	Take grade: 2.02 substitutes the first numeric field, and that is the borehole number.
Stars and rays in the interpolated field	The sector split on plan samples.	Leave zero in the sectors field: it is taken from the data. For samples in plan there will be no split.
Inverse distances give the same as nearest neighbour	All the neighbours were gathered from one borehole.	Raise the number of search sectors in the advanced parameters of 2.02.
An anomaly with depth smoothed into a flat field	The number of points is larger than the number of samples per place.	Leave zero in the point count field: it will be taken from the data.

What you see	Cause	Solution
The tool refuses, all the points share one elevation	The elevation was taken from the geometry of a flat layer.	Choose the elevation from a field or the depth below a surface.
I do not know whether the cube can be trusted	There is nothing to compare the built model with.	Run 2.05 on the same samples: it removes each in turn and shows how far the model misses.
The error is large but where is unclear	The numbers in the log say nothing about place.	Open the residual layer and colour it by areaid: it shows in which corner of the site the model misses.
Kriging gave values below zero	Kriging weights can be negative.	Take the spherical model instead of the gaussian one, or raise the nugget.
Kriging is no better than inverse distances	The grid step is above half the range.	That is expected. Take kriging for the variance, or make the grid denser.
I do not know where to trust the model	Error numbers say nothing about place.	Look at the variance cube from 2.06 as voxels: where it is large the model is guessing.
The wall is not built and asks for a line	No line is drawn and no layer is picked.	Draw a line with the drawing button or pick a line layer in the clip list.
The wall came out empty	The line is outside the cube.	Check the extent of the cube: beyond its edges there are no values and no wall is built.
Holes in the body after a cut	The body was built with merged faces and is not watertight.	Rebuild it in 2.04 with merging cleared: the log shows the number of bodies that are not watertight.
No box or labels in the export	A build before 0.65.7 is installed.	The build number is in the window title; the export result is printed in the window itself.
Voxels are not built and the log names a face count	The model is larger than the responsiveness limit.	Raise the cutoff, reduce the number of colour intervals, or coarsen the cube.
Points are visible but labels are not	No label field is chosen, or the label count is zero.	Set the Point label field and Labels at most .
A flat marker is hard to see from above	It lies in plan and flattens.	Take the circle: it is on screen and reads from any angle.
A round marker shows through the bed	An on-screen marker is always drawn over the scene.	Take a square, a diamond, a triangle or a cross: they lie in plan and are hidden by a surface.

Appendix. The multiband grid convention

A bed body is assembled from a single raster whose bands are laid out like this:

Band	Meaning
1	the bed roof, absolute elevation

Band	Meaning
2	the bed bottom, absolute elevation
3 and on	parameters: grades, thicknesses, mineral type, any numeric field

The band names are taken from the raster band descriptions, which is why the module lists read “content” instead of “band 3”. Grids assembled by the bed tools in Isoliner set the names themselves. For a third-party GeoTIFF the names can be written by any means that writes band descriptions, for example `gdal_edit.py` or GDAL from Python.

Cells without data must be marked with a nodata value. The module does not fill the gaps: a node without data drops out of the mesh, and the body is closed by a skirt along the boundary of the actual data.