FiberQ / QGIS Terrain Analysis

Two separate testing guides: with FiberQ and without FiberQ

Proof-of-concept workflow using a DEM and bilinear elevation interpolation

Guide 1 - Testing with the FiberQ QGIS plugin

Purpose: test terrain-adjusted route and cable length inside a normal FiberQ project. The script can analyse a FiberQ Route, an Aerial cable or an Underground cable. For FiberQ cables it also reads the existing slack/reserve stored in the cable attributes.

1. Prerequisites

Item	Requirement
CRS	Use a projected CRS with units in metres (for example EPSG:3909 in Serbia).
DEM	Load a DEM GeoTIFF covering the test area and rename the raster layer exactly to DEM.
FiberQ	A project with Route and/or Aerial cables / Underground cables.
Python Console	QGIS: Plugins > Python Console > Show Editor.

Important: the DEM does not have to use the same CRS as the FiberQ layer. The script transforms coordinates automatically.

2. Download and load the DEM

1. Open the OpenTopography Global Raster Data page: https://portal.opentopography.org/raster
2. For a quick test, choose SRTM GL1 (about 30 m resolution), or another DEM that covers your location.
3. Find the test area on the map, click Select a region, and draw a rectangle covering the complete Route/cable, with a small margin around it.
4. Set Data Output Format to GeoTIFF.
5. Submit/run the raster job and, when it is complete, download the Download Compressed Raster file.
6. Extract the downloaded archive. It should contain a DEM .tif file.
7. In QGIS, go to Layer > Add Layer > Add Raster Layer, select the .tif file, and click Add.
8. In the Layers panel, rename the loaded raster exactly to DEM.
9. The DEM does not need to use the same CRS as the Route/cable layer. If QGIS asks you to choose a coordinate transformation, select one whose Area of Use matches the test location.
10. If you move to a different geographic area, load a DEM that covers the new location; a DEM from the previous location will not provide elevation values outside its extent.

3. Test A - FiberQ Route

11. Create a route with FiberQ using Create route manually.
12. Make sure the route lies inside the area covered by the DEM.
13. Click the Route layer in the Layers panel so it is the active layer.
14. Select one Route feature.
15. Open the QGIS Python Console, paste the script below into the editor and run it.
16. Check the console output and the elevation profile window.

Expected mode: Generic route / line. Typical outputs: Geometry 2D length, Terrain-adjusted length, Terrain additional length, terrain percentage, min/max elevation, elevation gain/loss and the elevation profile.

4. Test B - FiberQ Cable + Optical Slacks

17. Create or use a FiberQ Route.
18. Lay an Aerial cable or Underground cable along the route.
19. Add Optical Slack points in FiberQ (for example 20 m at each end).
20. Open the cable attribute table and confirm that slack_m and total_len_m were updated by FiberQ.
21. Click the relevant cable layer (Aerial cables or Underground cables) so it is the active layer.
22. Select the cable feature and run the same script.
23. Compare FiberQ current total with TERRAIN-AWARE TOTAL.

For a FiberQ cable the script uses this calculation:

terrain-adjusted geometric length + FiberQ slack_m = TERRAIN-AWARE TOTAL

Example from the Kopaonik test	Result
Geometry 2D length	888.71 m
Terrain-adjusted length	894.61 m
Terrain additional length	+5.90 m
FiberQ slack/reserve	+40.00 m
FiberQ current total	928.79 m
Terrain-aware total	934.61 m

5. Expected DEM error

If the selected route/cable is outside the loaded DEM, the script should not calculate a false result. It should print:

```
FEATURE 1
ERROR: Insufficient DEM data along the selected line.
```

6. FiberQ testing script

The script is read-only: it does not modify cable geometry, Route geometry or FiberQ attributes.

```
from qgis.core import (
    QgsProject, QgsCoordinateTransform, QgsPointXY,
    QgsWkbTypes, NULL
)
import matplotlib.pyplot as plt
import math

DEM_LAYER_NAME = "DEM"
SAMPLE_DISTANCE = 5.0

project = QgsProject.instance()
layer = iface.activeLayer()

if layer is None:
    raise Exception("No active layer. Select a line or cable layer first.")

if QgsWkbTypes.geometryType(layer.wkbType()) != QgsWkbTypes.LineGeometry:
    raise Exception(f"Layer '{layer.name()}' is not a line layer.")

if layer.crs().isGeographic():
```

```
    raise Exception(
        "The active layer uses a geographic CRS. "
        "Use a projected CRS with units in metres."
    )

features = layer.selectedFeatures()
if not features:
    raise Exception("Select at least one line or cable.")

dem_layers = project.mapLayersByName(DEM_LAYER_NAME)
if not dem_layers:
    raise Exception(
        "DEM layer not found. The elevation raster must be named 'DEM'."
    )

dem_layer = dem_layers[0]
provider = dem_layer.dataProvider()
transform = QgsCoordinateTransform(layer.crs(), dem_layer.crs(), project)

field_names = {f.name() for f in layer.fields()}
fiberq_fields = {"duzina_m", "slack_m", "total_len_m"}
is_fiberq_cable = fiberq_fields.issubset(field_names)
mode = "FiberQ cable" if is_fiberq_cable else "Generic route / line"


def safe_float(value, default=0.0):
    if value is None or value == NULL:
        return default
    try:
        return float(value)
    except Exception:
        return default


def bilinear_elevation(point):
    extent = dem_layer.extent()
    width = provider.xSize()
    height = provider.ySize()
    if width <= 0 or height <= 0:
        return None

    pixel_x = extent.width() / width
    pixel_y = extent.height() / height
    col_f = (point.x() - extent.xMinimum()) / pixel_x - 0.5
    row_f = (extent.yMaximum() - point.y()) / pixel_y - 0.5
    col0 = math.floor(col_f)
    row0 = math.floor(row_f)
    dx = col_f - col0
    dy = row_f - row0

    vals = []
    for ro in (0, 1):
        row_vals = []
        for co in (0, 1):
            col = col0 + co
            row = row0 + ro
            x = extent.xMinimum() + (col + 0.5) * pixel_x
            y = extent.yMaximum() - (row + 0.5) * pixel_y
            value, ok = provider.sample(QgsPointXY(x, y), 1)
            if not ok or value is None:
                return None
            try:
                value = float(value)
                if math.isnan(value):
                    return None
            except Exception:
                return None
            row_vals.append(value)
```

```
        vals.append(row_vals)

    z00, z10 = vals[0]
    z01, z11 = vals[1]
    top = z00 * (1 - dx) + z10 * dx
    bottom = z01 * (1 - dx) + z11 * dx
    return top * (1 - dy) + bottom * dy


print("\n" + "=" * 76)
print("FIBERQ UNIVERSAL TERRAIN ANALYSIS")
print("=" * 76)
print(f"Layer:            {layer.name()}")
print(f"Mode:             {mode}")
print(f"Selected:         {len(features)}")
print(f"Sampling:         {SAMPLE_DISTANCE:.1f} m")
print("=" * 76)

for number, feature in enumerate(features, start=1):
    geom = feature.geometry()
    if geom.isEmpty():
        print(f"\nFEATURE {number}  |  ID {feature.id()}")
        print("ERROR: The selected feature has no geometry.")
        continue

    length_2d = geom.length()
    distances, elevations, points = [], [], []

    d = 0.0
    while d <= length_2d:
        point_geom = geom.interpolate(d)
        if not point_geom.isEmpty():
            p = point_geom.asPoint()
            dem_point = transform.transform(QgsPointXY(p.x(), p.y()))
            z = bilinear_elevation(dem_point)
            if z is not None:
                distances.append(d)
                elevations.append(z)
                points.append((p.x(), p.y(), z))
        d += SAMPLE_DISTANCE

    if not distances or distances[-1] < length_2d:
        point_geom = geom.interpolate(length_2d)
        p = point_geom.asPoint()
        dem_point = transform.transform(QgsPointXY(p.x(), p.y()))
        z = bilinear_elevation(dem_point)
        if z is not None:
            distances.append(length_2d)
            elevations.append(z)
            points.append((p.x(), p.y(), z))

    if len(points) < 2:
        print(f"\nFEATURE {number}  |  ID {feature.id()}")
        print("ERROR: Insufficient DEM data along the selected line.")
        continue

    terrain_length = 0.0
    elevation_gain = 0.0
    elevation_loss = 0.0

    for i in range(1, len(points)):
        x1, y1, z1 = points[i - 1]
        x2, y2, z2 = points[i]
        horizontal = math.hypot(x2 - x1, y2 - y1)
        dz = z2 - z1
        terrain_length += math.sqrt(horizontal ** 2 + dz ** 2)
        if dz > 0:
            elevation_gain += dz
```

```
        elif dz < 0:
            elevation_loss += abs(dz)

    terrain_extra = terrain_length - length_2d
    terrain_percent = (
        terrain_extra / length_2d * 100 if length_2d > 0 else 0
    )

    print(f"\nFEATURE {number}  |  ID {feature.id()}")
    print("-" * 76)
    print(f"Geometry 2D length:         {length_2d:.2f} m")
    print(f"Terrain-adjusted length:    {terrain_length:.2f} m")
    print(f"Terrain additional length:  +{terrain_extra:.2f} m")
    print(f"Terrain increase:           +{terrain_percent:.3f} %")

    if is_fiberq_cable:
        fiberq_length = safe_float(feature["duzina_m"])
        slack = safe_float(feature["slack_m"])
        fiberq_total = safe_float(feature["total_len_m"])
        terrain_total = terrain_length + slack

        print("-" * 76)
        print(f"FiberQ stored length:       {fiberq_length:.2f} m")
        print(f"FiberQ slack/reserve:       +{slack:.2f} m")
        print(f"FiberQ current total:       {fiberq_total:.2f} m")
        print(f"TERRAIN-AWARE TOTAL:        {terrain_total:.2f} m")
        print(f"Difference vs FiberQ:       {terrain_total - fiberq_total:+.2f} m")

    print("-" * 76)
    print(f"Lowest elevation:           {min(elevations):.2f} m")
    print(f"Highest elevation:          {max(elevations):.2f} m")
    print(f"Elevation range:            {max(elevations)-min(elevations):.2f} m")
    print(f"Total elevation gain:       +{elevation_gain:.2f} m")
    print(f"Total elevation loss:       -{elevation_loss:.2f} m")
    print(f"DEM samples:                {len(elevations)}")
    print("-" * 76)

    plt.figure()
    plt.plot(distances, elevations)
    plt.title(f"Terrain Profile - {layer.name()} [ID {feature.id()}]")
    plt.xlabel("Distance along line (m)")
    plt.ylabel("Elevation (m)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

print("\n" + "=" * 76)
print("ANALYSIS COMPLETE")
print("=" * 76)
```

Guide 2 - Testing in QGIS without the FiberQ plugin

Purpose: verify the terrain-length method using only standard QGIS features. No FiberQ layers or FiberQ attributes are required.

1. Prerequisites

Item	Requirement
QGIS line layer	Any LineString layer created in QGIS.
CRS	Use a projected CRS with units in metres.
DEM	Load a DEM GeoTIFF covering the line and rename it exactly to DEM.
Python Console	QGIS: Plugins > Python Console > Show Editor.

2. Create a test line using standard QGIS tools

24. Go to Layer > Create Layer > New Temporary Scratch Layer (or create a GeoPackage/Shapefile line layer).
25. Set Geometry type to LineString.
26. Choose a projected CRS in metres for the test location.
27. Enable editing for the line layer.
28. Use Add Line Feature and draw a line over the terrain you want to test.
29. Save the edit and select the line feature.
30. Click the line layer in the Layers panel so it is the active layer.

3. Download and load the DEM

31. Open the OpenTopography Global Raster Data page: https://portal.opentopography.org/raster
32. For a quick test, choose SRTM GL1 (about 30 m resolution), or another suitable DEM.
33. Find the location on the map, click Select a region, and select an area covering the complete test line.
34. Set Data Output Format to GeoTIFF, submit the job, and download the Download Compressed Raster file when processing is complete.
35. Extract the archive and locate the .tif file.
36. In QGIS, go to Layer > Add Layer > Add Raster Layer, select the .tif file, and click Add.
37. Rename the raster layer exactly to DEM.
38. The DEM may use a different CRS from the line layer; the script transforms coordinates automatically. If QGIS asks for a coordinate transformation, select one appropriate for the test location.
39. When testing a new location, load a DEM that actually covers that area.

4. Run the test

40. Open Plugins > Python Console and click Show Editor.
41. Paste the generic QGIS terrain script below.
42. Run the script while the line layer is active and the line feature is selected.
43. Read the console output and inspect the elevation profile.

Output	Meaning
Geometry 2D length	Normal map/plan length of the selected line.
Terrain-adjusted length	Estimated 3D length following the DEM elevation profile.

Terrain additional length	Difference between 3D terrain length and 2D geometry length.
Terrain increase	Terrain correction expressed as a percentage.
Elevation gain/loss	Accumulated climbing and descending along the sampled line.
Elevation profile	Graph of DEM elevation versus distance along the line.

5. Generic QGIS testing script

```
from qgis.core import (
    QgsProject, QgsCoordinateTransform, QgsPointXY, QgsWkbTypes
)
import matplotlib.pyplot as plt
import math

DEM_LAYER_NAME = "DEM"
SAMPLE_DISTANCE = 5.0

project = QgsProject.instance()
layer = iface.activeLayer()

if layer is None:
    raise Exception("No active layer. Select a line layer first.")

if QgsWkbTypes.geometryType(layer.wkbType()) != QgsWkbTypes.LineGeometry:
    raise Exception(f"Layer '{layer.name()}' is not a line layer.")

if layer.crs().isGeographic():
    raise Exception(
        "The active layer uses a geographic CRS. "
        "Use a projected CRS with units in metres."
    )

features = layer.selectedFeatures()
if not features:
    raise Exception("Select at least one line feature.")

dem_layers = project.mapLayersByName(DEM_LAYER_NAME)
if not dem_layers:
    raise Exception(
        "DEM layer not found. The elevation raster must be named 'DEM'."
    )

dem_layer = dem_layers[0]
provider = dem_layer.dataProvider()
transform = QgsCoordinateTransform(layer.crs(), dem_layer.crs(), project)


def bilinear_elevation(point):
    extent = dem_layer.extent()
    width = provider.xSize()
    height = provider.ySize()
    if width <= 0 or height <= 0:
        return None

    pixel_x = extent.width() / width
    pixel_y = extent.height() / height
    col_f = (point.x() - extent.xMinimum()) / pixel_x - 0.5
    row_f = (extent.yMaximum() - point.y()) / pixel_y - 0.5
    col0 = math.floor(col_f)
    row0 = math.floor(row_f)
    dx = col_f - col0
```

```
    dy = row_f - row0

    vals = []
    for ro in (0, 1):
        row_vals = []
        for co in (0, 1):
            col = col0 + co
            row = row0 + ro
            x = extent.xMinimum() + (col + 0.5) * pixel_x
            y = extent.yMaximum() - (row + 0.5) * pixel_y
            value, ok = provider.sample(QgsPointXY(x, y), 1)
            if not ok or value is None:
                return None
            try:
                value = float(value)
                if math.isnan(value):
                    return None
            except Exception:
                return None
            row_vals.append(value)
        vals.append(row_vals)

    z00, z10 = vals[0]
    z01, z11 = vals[1]
    top = z00 * (1 - dx) + z10 * dx
    bottom = z01 * (1 - dx) + z11 * dx
    return top * (1 - dy) + bottom * dy


print("\n" + "=" * 72)
print("QGIS TERRAIN ANALYSIS")
print("=" * 72)
print(f"Layer:    {layer.name()}")
print(f"Selected: {len(features)}")
print("=" * 72)

for number, feature in enumerate(features, start=1):
    geom = feature.geometry()
    if geom.isEmpty():
        print(f"\nFEATURE {number}: ERROR - The feature has no geometry.")
        continue

    length_2d = geom.length()
    distances, elevations, points = [], [], []
    d = 0.0

    while d <= length_2d:
        pg = geom.interpolate(d)
        if not pg.isEmpty():
            p = pg.asPoint()
            dem_point = transform.transform(QgsPointXY(p.x(), p.y()))
            z = bilinear_elevation(dem_point)
            if z is not None:
                distances.append(d)
                elevations.append(z)
                points.append((p.x(), p.y(), z))
        d += SAMPLE_DISTANCE

    if not distances or distances[-1] < length_2d:
        pg = geom.interpolate(length_2d)
        p = pg.asPoint()
        dem_point = transform.transform(QgsPointXY(p.x(), p.y()))
        z = bilinear_elevation(dem_point)
        if z is not None:
            distances.append(length_2d)
            elevations.append(z)
            points.append((p.x(), p.y(), z))
```

```
    if len(points) < 2:
        print(f"\nFEATURE {number}")
        print("ERROR: Insufficient DEM data along the selected line.")
        continue

    terrain_length = 0.0
    elevation_gain = 0.0
    elevation_loss = 0.0

    for i in range(1, len(points)):
        x1, y1, z1 = points[i - 1]
        x2, y2, z2 = points[i]
        horizontal = math.hypot(x2 - x1, y2 - y1)
        dz = z2 - z1
        terrain_length += math.sqrt(horizontal ** 2 + dz ** 2)
        if dz > 0:
            elevation_gain += dz
        elif dz < 0:
            elevation_loss += abs(dz)

    terrain_extra = terrain_length - length_2d
    terrain_percent = terrain_extra / length_2d * 100 if length_2d else 0

    print(f"\nFEATURE {number}  |  ID {feature.id()}")
    print("-" * 72)
    print(f"Geometry 2D length:         {length_2d:.2f} m")
    print(f"Terrain-adjusted length:    {terrain_length:.2f} m")
    print(f"Terrain additional length:  +{terrain_extra:.2f} m")
    print(f"Terrain increase:           +{terrain_percent:.3f} %")
    print(f"Lowest elevation:           {min(elevations):.2f} m")
    print(f"Highest elevation:          {max(elevations):.2f} m")
    print(f"Total elevation gain:       +{elevation_gain:.2f} m")
    print(f"Total elevation loss:       -{elevation_loss:.2f} m")

    plt.figure()
    plt.plot(distances, elevations)
    plt.title(f"Terrain Profile - {layer.name()} [ID {feature.id()}]")
    plt.xlabel("Distance along line (m)")
    plt.ylabel("Elevation (m)")
    plt.grid(True)
    plt.tight_layout()
    plt.show()

print("\n" + "=" * 72)
print("ANALYSIS COMPLETE")
print("=" * 72)
```

6. Notes and limitations

- The quality of the result depends on the DEM resolution and accuracy. A 30 m SRTM DEM is suitable for a proof of concept but cannot reproduce small terrain details.
- Bilinear interpolation makes the elevation profile smoother, but it does not create terrain detail that is absent from the source DEM.
- For underground cables following a road or trench, a surface DEM can provide a useful estimate. Tunnels, directional drilling, bridges and other structures that do not follow the terrain require a different model.
- For aerial cables, this terrain calculation estimates the ground-profile effect only. Exact pole attachment heights and cable sag would require additional data and calculations.
- The script is intended for testing and estimation, not survey-grade measurement.