3D GIS: An Application

Group:14
Anant Gauniyal (220132) Aayush Sidana (220023) Amipriya Anand (220122)

CE432: Geographic Information System
Instructor: Dr. Salil Goel

1. Introduction

1.1. Brief Overview

Geographic Information Systems (GIS) have become essential tools in spatial analysis, enabling the visualization, analysis, and interpretation of geographic data to form relationships, patterns, tabulate and store data. Traditionally, GIS has been limited to two-dimensional (2D) representations, which, even though useful, often fail to capture the full complexity of the real world. The scope of improvement has consistently led to the development of 3D GIS, which is still evolving today.

With the increasing demand for more realistic and detailed spatial models, the development and adoption of three-dimensional (3D) GIS has gained momentum. 3D GIS extends the previously implemented application in GIS by integrating height and depth information, allowing for the representation and analysis of data in a more in-depth and comprehensive manner. According to Stoter and van Oosterom, 3D GIS research is intensive and covers all aspects of collecting, storing, and analyzing real-world phenomena" [2]. This added dimension has led to better visualization of models, such as terrain simulation, digital elevation models, and line-of-sight applications. It plays a central role in applications that involve urban planning, environmental modeling, archaeology and cultural heritage modeling, mining, geology, and disaster risk analysis such as flood or earthquake simulation. As cities grow vertically and underground infrastructure becomes more intricate, the need for 3D spatial data becomes more pronounced.

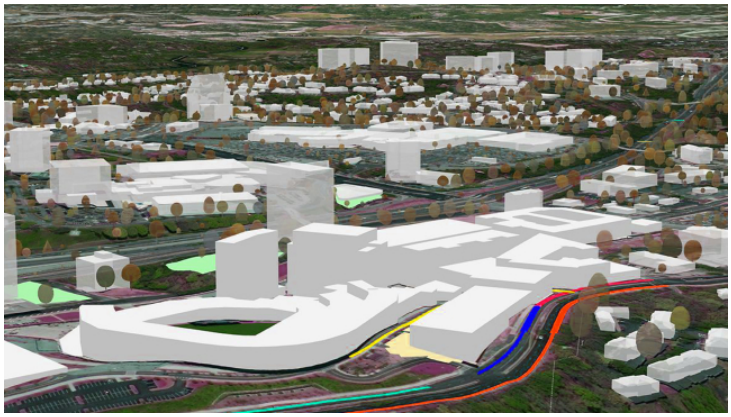


Figure 1: *3D GIS model (source:ESRI)*

1.2. Objective

This paper aims to explore the application of 3D GIS in practical scenarios, focusing on how this technology is transforming spatial analysis and planning in the real world. The subsequent sections cover the basic concepts of 3D GIS, its implementation tools, real-world applications, a coding demonstration, and future directions. It also aims to provide a comprehensive review of the evolution and current state of 3D terrain visualization in GIS, categorizing the techniques and software tools developed over time.
This paper proposes to fulfill the objective of:

1. Flood Risk Identification Using Elevation Data
2. Show other applications of 3D models

Figure 2: *3D GIS-based city model visualization illustrating multi-level building data and spatial relationships(source:ESRI).*

2. Background/Literature Review

Traditional Geographic Information Systems (GIS) have long been used to analyze and visualize spatial data in two dimensions, primarily using vector and raster data models. However, the real world is inherently three-dimensional, and the limitations of 2D GIS in representing vertical and volumetric features became increasingly apparent with the growth of urban environments, complex infrastructure, and environmental modeling.

This has led to the evolution of 3D GIS, a system capable of representing, storing, analyzing, and visualizing spatial data with an additional vertical dimension, overcoming the limitations of 2D analysis.

- Stoter and van Oosterom (2006) [2]. Their work provided a foundational understanding of how 3D GIS systems differ structurally and functionally from their 2D counterparts.
- Jedlička (2018) further examined the conceptual shift from 2D cartographic principles to 3D GIS representations, noting that the third dimension allows for better representation of urban morphology, underground utilities, and terrain interactions. [3].
- A comprehensive review by Ruzinoor et al. (2012) highlighted the major visualization techniques for 3D terrain and classified existing software based on their capabilities and limitations [1].

3. Methodology

The methodology to identify flood-risk prone regions in India.The methodology adopted for this study involved both GIS-based and programming-based workflows to analyze elevation data and simulate flood risk. The process was divided into two complementary parts:

3.1. 1. DEM-Based Flood Simulation using QGIS

1. **Data Acquisition:** Digital Elevation Model (DEM) data was downloaded from the Bhuvan NRSC portal in GeoTIFF format.

Figure 3: *Login screen of Bhoonidhi*

On logging in, we get the full map of India provided by the Government of India which can be open street or in physical format. The input requires two fulfilment: location and satellite filters(optional).

Figure 4: *Input screen*

The area of interest was selected using the select polygon option and details about the satellite to be used to extract information was input as desired. Specifically, Cartosat-1 DEM tiles at 30-meter resolution were selected. On applying search criteria filters, we move on to selecting the shape of polygon, in our case rectangle and several results with satellite sensors are produced.

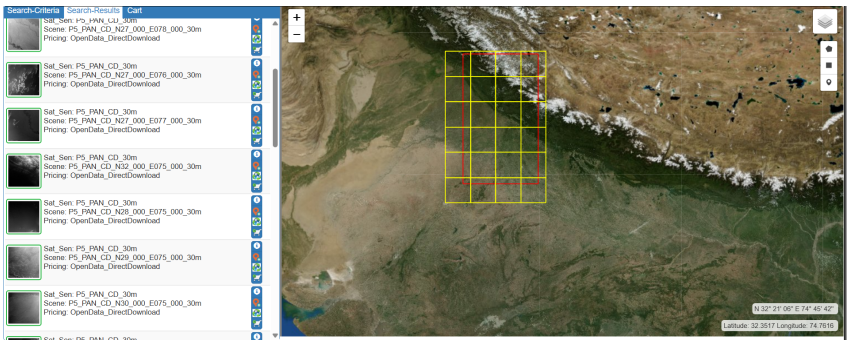
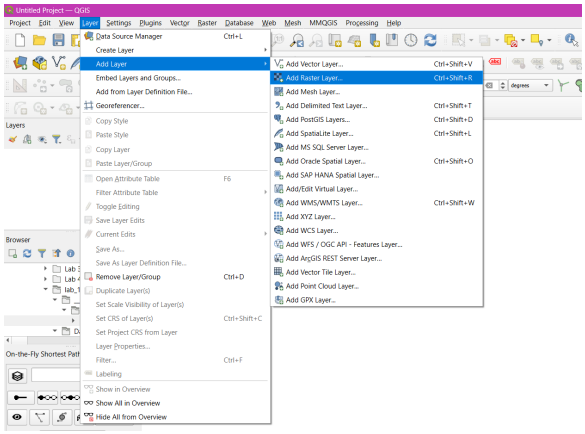


Figure 5: *Polygon of interest further divided into tiles*

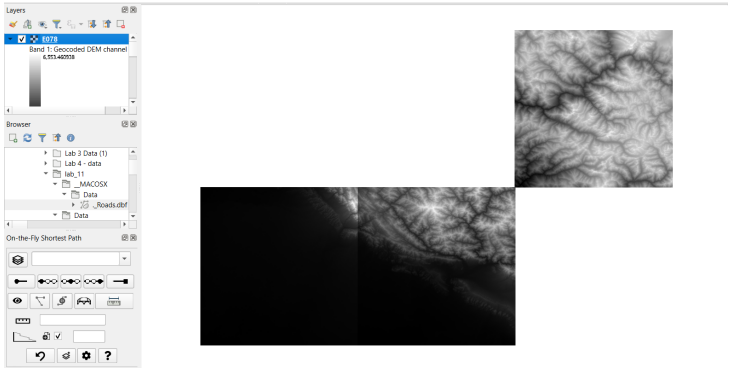
We take atleast three of these files and add them to cart from where they can be downloaded in .tiff format. Our final step of data acquisition includes extracting this data from it's zip file after it is installed.

2. **Data Loading:** The '.tif' DEM files are loaded into QGIS for this setup. Since the tiles selected initially were adjacent to each other, we make sure to combine the rasters before proceeding. We open QGIS and go to Layer → Add Layer → Add Raster Layer.

   This allows us to input or three layers named: **E076.tif, E077.tif, E078.tif**. This gives us the imagery of uploaded tiles.



(a) *Raster tiles input:E076.tif, E077.tif, E078.tif*



(b) *Visual appearance of uploaded DEM tiles*

Figure 6: Loading and viewing adjacent DEM tiles in QGIS

Care is taken to check projection of each layer by going to properties→information of each tab so as to make sure the layers share the same projection to be worked upon. It is confirmed they are referenced in EPSG:4326 - WGS 84 - Geographic reference system.

3. **Merging Rasters:** We go to Rasters→Merge option to merge our three rasters in order to make a continuous terrain which can be utilized for 3D GIS. The layer is named: "Merged.tif".

4. **Checking Maximum and Minimum Elevations:** To proceed further, we first check the maximum and minimum elevations which help us deduce our elevation-risk estimation against flood prone areas. On inspection, the elevation is shown with these details: Our minimum is 0 m while maximum is 6553.4609375 m

Band 1

- STATISTICS_APPROXIMATE=YES
- STATISTICS_MINIMUM=0
- STATISTICS_MAXIMUM=6553.4609375
- STATISTICS_MEAN=908.81685189906
- STATISTICS_STDDEV=1631.3628507603
- STATISTICS_VALID_PERCENT=100

- Scale: 1
- Offset: 0

Figure 7: *Statistical information of elevation*

5. **Flood-Prone Zone Extraction** To identify potential flood-affected areas, a thresholding operation was performed on the merged DEM. Using QGIS's Raster Calculator, elevation values below 500 meters were extracted via the condition:

```
"Merged" < 500
```

This produced a binary raster indicating flood-prone areas (value = 1) and non-prone areas (value = 0). The resulting raster was then styled to visually highlight low-lying zones, and was also used in 3D visualization to examine topographical vulnerability in a spatially meaningful manner. Further, the value is increased from 500 to 2500 to see the changes in our pattern.

Here is a comparison between two raster outputs after putting "500" and "1000" as elevation constraints.

Studies have considered 1000-2500 m range of Himalayas susceptible to flash floods, climate change and cloud bursts resulting in vulnerable regions surrounding the hilly terrain.

Figure 8: *DEM Constraint set as 500*

Figure 9: *DEM Constraint set as 1000*

The white regions are area more susceptible to flooding than black which are at a higher elevation than the suspected flood prone region. Since the threshold is

different, we see different dendrites emerging from each raster confirming as our threshold increases, larger suspected flood prone regions emerge.

We further apply colour symbology changes where now: blue→1 and white→0. This change indicates flood prone regions are shown with blue while safe areas with white.

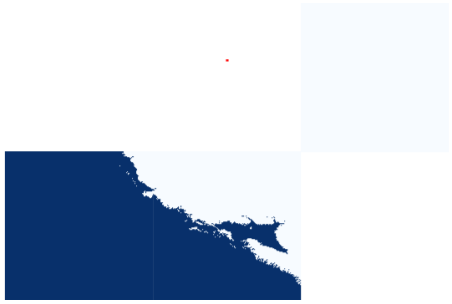


Figure 10: *Symbology of 500 constraint DEM changed to blue=1 and white=0*

The flood-prone region for a particular elevation can now be visually perceived in an easier view.

6. **3D Model extraction** To extract the 3D model, the extracted layer with symbology is used and converted into 3D Model on applying New 3D Map Views from View section. Final output is shown in the result section.

3.2. Coding Demonstration

Apart from QGIS implementation, we would like to implement our 3D model using python code implementation.

To complement GIS-based flood risk analysis, a Python-based visualization was implemented using `rasterio` and `pyvista`. The objective was to programmatically simulate terrain and flood-prone zones from the merged DEM, and render them in 3D.

Source: Given in references

The process involved loading the elevation data, applying a flood threshold (1500 meters, for example), and constructing a structured grid where low-lying zones were marked distinctly. A vertical exaggeration factor of 5 was used to enhance topographic clarity. The flooded regions (elevation less than 1500 m) were highlighted in blue to visually represent vulnerable areas. . The code demonstrates actions taken such as:

1. **Cleaning and Down-sampling:**
   Invalid values such as zeros, NaNs, and extreme spikes (e.g., elevation above 9000 m) were masked. The DEM was then down-sampled (e.g., every 15^{th} row and column) to reduce the data size for efficient 3D visualization.

2. **3D Grid Generation:**
   Using the affine transform parameters obtained from the DEM metadata, real-world X and Y coordinates were generated. A mesh-grid of X, Y, and elevation values (Z) was created using `numpy.meshgrid`.

3. **Visualization with PyVista:**
   The structured grid was constructed using PyVista's `StructuredGrid` class. The 3D terrain was rendered using `add_mesh()` with color maps such as `terrain` and `viridis`. Vertical exaggeration was applied to enhance terrain perception.

4. **Flood Risk Mapping:**
   A threshold (e.g., 1500 m) was applied to identify and visualize potentially flood-prone zones. These regions were highlighted using scalar masks and distinct color mappings in the 3D model.

4. Results

This section provides results gathered from both QGIS based modeling and the python-code implemented model.

4.1. Results from QGIS Visualization

Since we already had the 2D representation of the merged raster in continuous color manner, we now convert it into 3D model after applying View then New 3D view. We do not get our model in 3D format yet.
We apply **overlaying** of the original merged raster and overlay it on top of our DEM model.
Note: We are applying these tools on the merged raster with elevation constraint below **500 m**. Same steps can be followed for any elevation constraint.

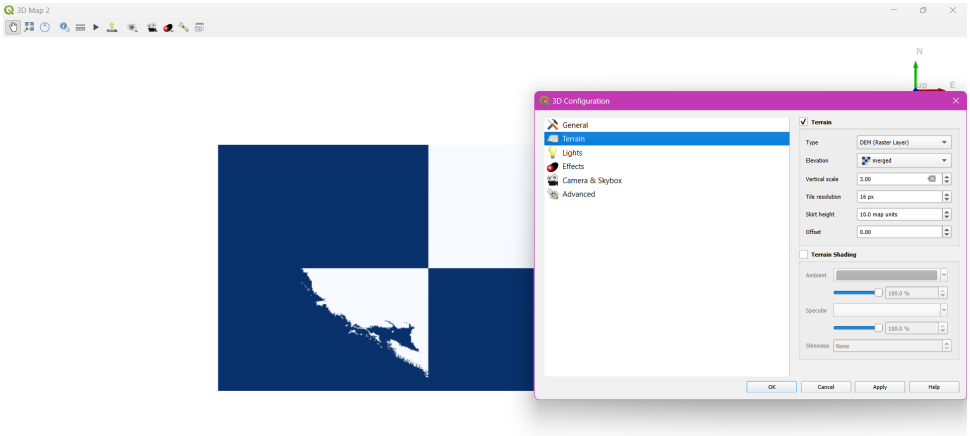


Figure 11: *Inputs given before producing a 3D output*

The original raster was kept under Elevation, this is necessary for providing us with the 3rd dimension, which is the z-axis. To avoid visual defect, the scale is kept at 3.

Figure 12: *Elevation of produced model*

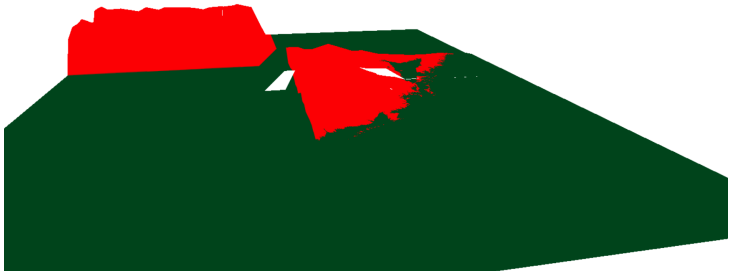


Figure 13: *Same model with different symbology*

Overlaying the merged elevation raster on top of the constraint layer in QGIS 3D view allows simultaneous visualization of terrain and vulnerability, enhancing spatial interpretation of risk-prone regions.

The added third dimension gives us a more profound look and depth into our study of flood-risk estimations. The QGIS template is attached along with the report.

4.2. Results from Python-Based Coding Demonstration

We apply Python-based approach on the constraint set as **1500 m** In the Python-based approach, the same merged DEM was down-sampled and used to generate a 3D terrain mesh using PyVista. After scaling the spatial coordinates using the affine transform and applying vertical exaggeration, the terrain was visualized in a 3D environment with appropriate color maps.The code utilises various features and libraries of python.

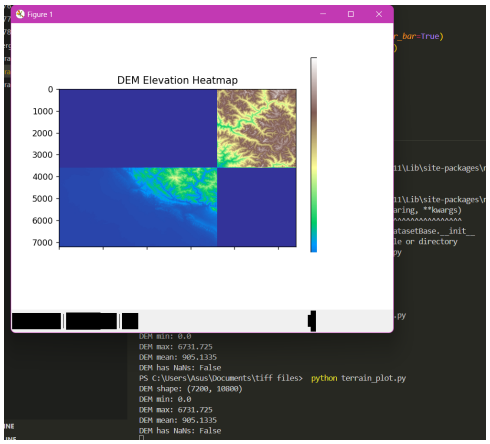


Figure 14: *2D-heatmap plotted using matplotlib library*

The resulting 3D model clearly showed variations in terrain elevation and slope, and additional threshold-based masking was used to highlight flood-prone areas in blue. Screenshots were exported for use in documentation and analysis.

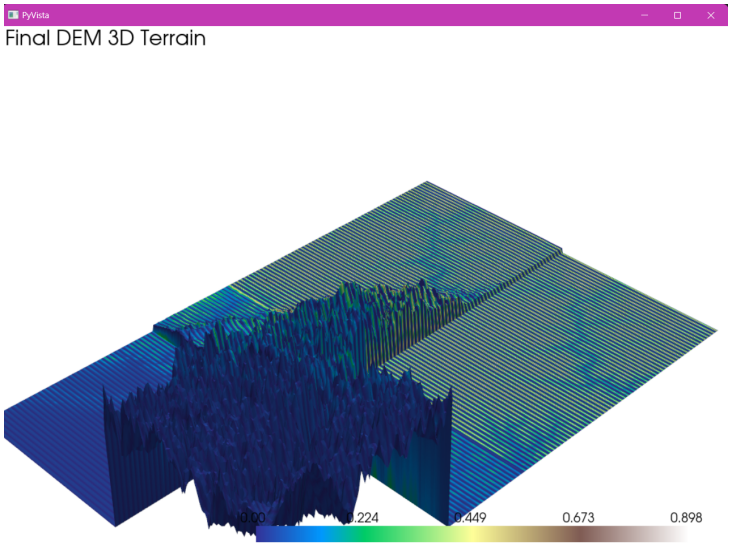


Figure 15: *3D-Mapping utilising python libraries*

The blue peaks we're seeing are elevations above 1500 meters, and everything rising toward green, yellow, and brown is lower than that implying Deep Blue → Higher elevation, hence tending towards 0 in a binary mask. On comparing these results with 1500m elevation model found in QGIS, we get the same result. All models have been included in the zip along with the report.

Coding implemented models with constraints set as 500 m,1000 m,1500 m,2000 m are also included in the zip.

5. Discussion

5.1. Interpretation of Results

The processed DEM tiles covering the selected Himalayan foothill region revealed significant elevation variation across the terrain. In the QGIS-based results, raster classification using elevation thresholds helped identify low-lying flood-prone areas effectively. Regions below 1500 meters appeared in white, while higher terrains were masked or filtered out, allowing us to isolate areas vulnerable to flooding under extreme rainfall or reservoir overflow conditions.

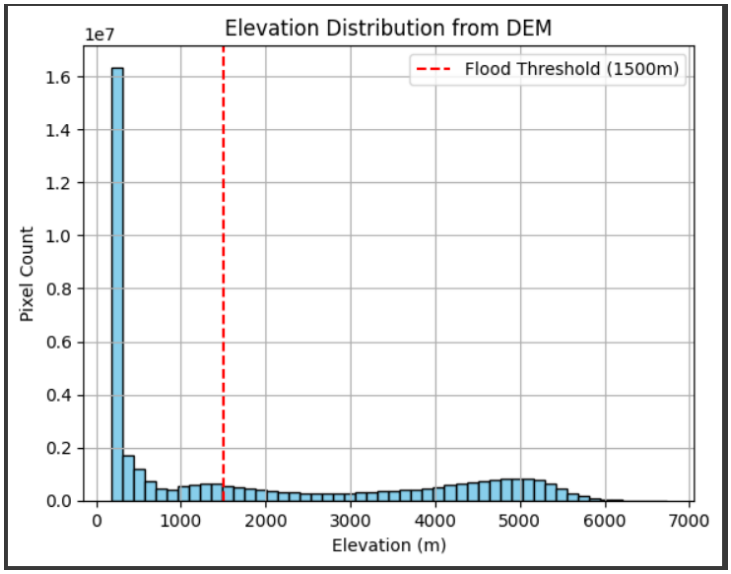


Figure 16: *Histogram showing distribution of elevation across study area for elevation constraint of 1500m*

A huge concentration of pixels are below 500 m, meaning much of our area is low-lying. The majority of terrain lies below 1500 m, hence susceptible to flooding. A significant drop in frequency as elevation increases beyond 1500 m, making those higher zones relatively safer.

In the coding-based results using Python and PyVista, a structured 3D surface plot was generated from the merged and down-sampled DEM. Although initial attempts faced issues like incorrect aspect scaling and flat renderings, successive refinements led to a correct and visually interpretable 3D model of the elevation landscape. The final visualization allowed us to observe the morphological layout of valleys and ridges and identify depressions likely to accumulate water.

5.2. Some other applications

Using, the same methodology as we did for flood risk elevation analysis, it can also be similarly done for sea-level elevation risks too. We get The data as extracted earlier from Bhoonidhi server and implement using QGIS. We merge two adjacent tiles located near **Western Maharashtra**. The model for sea-level based elevation risks is also attached along with the report.

Figure 17: *2D depiction of sea-level on modeling with QGIS*

Figure 18: *3D depiction of sea-level on modeling with QGIS*

The figures show a binary flood mask where areas susceptible to flooding if sea level rises to 5m are shown in red while safe zones are shown in yellow.

5.2.1 Coding Implementation

The coding implementation done in Python using libraries of rasterio, numpy and Pyvista follows similar pathways as done for flood risk evaluation.

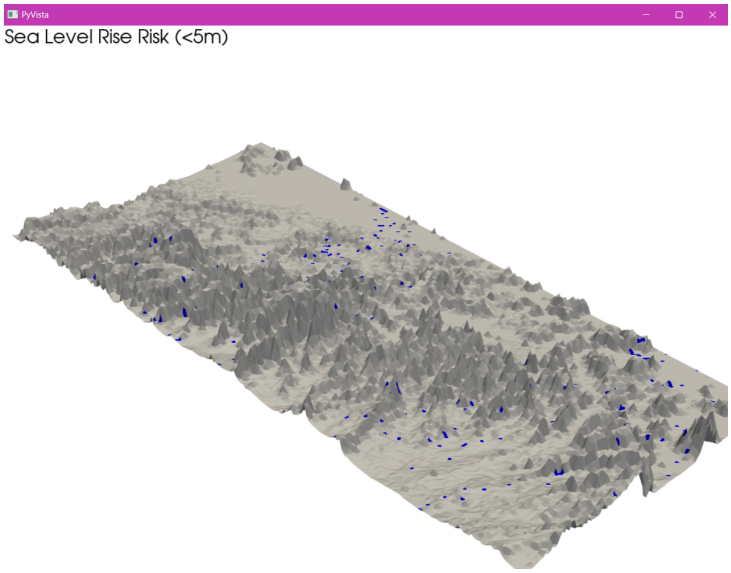


Figure 19: *3D model of sea-risk elevation*

The elevation threshold of 1-5 meters for sea level rise risk has been adopted based on empirical analysis from coastal Western Ghats studies [6].

5.3. Limitations and Observations

Despite the eventual success of the modeling pipeline, there were several limitations and technical challenges:

- The DEM tiles used contained NoData values in surrounding areas, which initially led to misleading visualizations unless explicitly masked or made transparent.
- Resolution mismatches between tiles or projection inconsistencies could affect accurate merging and elevation continuity, especially at the boundaries.
- In the Python-based modeling, the structured grid required careful alignment and scaling of the x, y, z axes. A slight error in transform parameters could distort the terrain or produce a blank (white) rendering.
- Down-sampling was necessary to ensure performance and avoid crashes, which may have resulted in the loss of finer topographic details.

5.4. Significance of Findings

This study demonstrates the practical integration of GIS tools and Python-based 3D modeling for flood risk visualization. The use of QGIS enabled straightforward raster manipulation and masking, while Python offered flexibility and automation for custom visualization pipelines. The workflow also highlighted how digital elevation data can be transformed into meaningful spatial insights, with implications for urban planning, disaster preparedness, and environmental modeling.

6. Conclusion

This study successfully demonstrated a methodology for identifying flood-prone regions using 3D GIS tools and digital elevation models (DEMs). The workflow encompassed both QGIS-based visual analysis and Python-based 3D modeling using PyVista.
This was effectively visualized using binary raster masks and 3D terrain surfaces. Our analysis reveals that low-lying regions, especially those adjacent to river basins and valleys, are at higher risk during flood events. The use of PyVista added a dimension of topographic realism, helping to better interpret spatial elevation patterns and flood potential.

6.1. Future Works

Digital Twin Integration: With additional real-time hydrological and meteorological data, this model could evolve into a digital twin of the region, enabling continuous flood monitoring and simulation. This has immense utility for disaster management, early warning systems, and urban resilience planning.

Contributions

The following outlines the individual contributions of each group member:

- **Anant Gauniyal (220132)**: Data handling and collection, report writing, Python code implementation, QGIS analysis and visualisation.
- **Amipriya Anand (220122)**: Report writing, presentation preparation, QGIS visualization.
- **Aayush Sidana (220023)**: Presentation preparation, data handling, coding implementation.

References

[1] A review on 3D terrain visualization of GIS data: techniques and software

[2] Stoter, Jantien, and Peter van Oosterom. "3D GIS, current status and perspectives." In *Proceedings of the 23rd Urban Data Management Symposium*, 2006.

[3] Jedlička, Karel. "3D GIS in the context of 2D cartographic concepts." In *Proceedings of the 7th ICCGIS*, 2018.

Longley, P. A., Goodchild, M. F., Maguire, D. J., & Rhind, D. W. (2015). *Geographic Information Systems and Science*. Wiley.

[4] ESRI. *3D GIS Overview*. Available: `https://www.esri.com/en-us/arcgis/products/arcgis-3d-analyst/overview`

[5] QGIS Documentation. Available: `https://docs.qgis.org/`

[6] Western Ghats Research: https://www.sciencedirect.com/science/article/pii/S2352938520301920

[7] Himalayan Range Flood Risk Research: https://www.nature.com/articles/s41598-024-53535-w

[8] Lecture Notes available on HelloIITK

[9] Bhuvan – Geoportal of ISRO. Indian Space Research Organisation. Available at: `https://bhuvan.nrsc.gov.in`

[10] Bhoonidhi – Indian National Data Repository. Indian Space Research Organisation. Available at: `https://bhoonidhi.nrsc.gov.in`

[11] Visual Studio Code (VSCode). Microsoft. Available at: `https://code.visualstudio.com`

[12] Google Colaboratory (Colab). Google Research. Available at: `https://colab.research.google.com`

[13] Indian Space Research Organisation (ISRO). Official Website. Available at: `https://www.isro.gov.in`

[14] PyVista examples: https://docs.pyvista.org/examples/00-load/create-surface.html

[15] Rasterio and Numpy Demonstration: https://rasterio.readthedocs.io/en/stable/ https://numpy.org/doc/

A. Python Code Implementation

```python
import numpy as np
import rasterio
import pyvista as pv

# Loading our merged DEM
with rasterio.open("merged.tif") as src:
    dem = src.read(1)
    transform = src.transform

# Cleaning invalid values
dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)

# Downsampling
step = 30
dem_ds = dem[::step, ::step]
z = np.nan_to_num(dem_ds, nan=0)

# Flood mask: elevation < 500 m
flood_mask = (z < 500).astype(int)

# Coordinate grid
rows, cols = z.shape
x = np.arange(cols) * transform.a * step + transform.c
y = np.arange(rows) * transform.e * step + transform.f
x, y = np.meshgrid(x, y)

# Scaling Z
x_range = np.max(x) - np.min(x)
z_range = np.max(z) - np.min(z)
scale_factor = x_range / z_range if z_range != 0 else 1
z_scaled = (z - np.min(z)) * scale_factor * 0.3

# Creating Grid
grid = pv.StructuredGrid(x, y, z_scaled)
grid.point_data["Flood <500m"] = flood_mask.ravel(order="F")

# Plotting
plotter = pv.Plotter()
plotter.set_background("white")
plotter.add_mesh(grid, scalars="Flood <500m", cmap=["white", "blue"],
                 show_scalar_bar=False)
plotter.add_text("Flood Risk Areas (<500m)", font_size=12)
plotter.camera_position = 'iso'
plotter.camera.zoom(1.5)
plotter.show()
```

Listing 1: Terrain Plot 500

```python
import numpy as np
import rasterio
import pyvista as pv

# === Load your merged DEM ===
with rasterio.open("merged.tif") as src:
    dem = src.read(1)
```

```
    transform = src.transform

# === Clean invalid values ===
dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)

# === Downsample ===
step = 30
dem_ds = dem[::step, ::step]
z = np.nan_to_num(dem_ds, nan=0)

# === Flood mask: elevation < 1500 m
flood_mask = (z < 1000).astype(int)

# === Coordinate grid
rows, cols = z.shape
x = np.arange(cols) * transform.a * step + transform.c
y = np.arange(rows) * transform.e * step + transform.f
x, y = np.meshgrid(x, y)

# === Scale Z
x_range = np.max(x) - np.min(x)
z_range = np.max(z) - np.min(z)
scale_factor = x_range / z_range if z_range != 0 else 1
z_scaled = (z - np.min(z)) * scale_factor * 0.3

# === Create Grid
grid = pv.StructuredGrid(x, y, z_scaled)
grid.point_data["Flood <1000m"] = flood_mask.ravel(order="F")

# === Plot
plotter = pv.Plotter()
plotter.set_background("white")
plotter.add_mesh(grid, scalars="Flood <1000m", cmap=["white", "blue"],
                                                show_scalar_bar=False)
plotter.add_text("Flood Risk Areas (<1000m)", font_size=12)
plotter.camera_position = 'iso'
plotter.camera.zoom(1.5)
plotter.show()
```

Listing 2: Terrain Plot 1000

```
import numpy as np
import rasterio
import pyvista as pv

# === Load your merged DEM ===
with rasterio.open("merged.tif") as src:
    dem = src.read(1)
    transform = src.transform

# === Clean invalid values ===
dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)

# === Downsample ===
step = 30
dem_ds = dem[::step, ::step]
z = np.nan_to_num(dem_ds, nan=0)

```

```
# === Flood mask: elevation < 1500 m
flood_mask = (z < 1500).astype(int)

# === Coordinate grid
rows, cols = z.shape
x = np.arange(cols) * transform.a * step + transform.c
y = np.arange(rows) * transform.e * step + transform.f
x, y = np.meshgrid(x, y)

# === Scale Z
x_range = np.max(x) - np.min(x)
z_range = np.max(z) - np.min(z)
scale_factor = x_range / z_range if z_range != 0 else 1
z_scaled = (z - np.min(z)) * scale_factor * 0.3

# === Create Grid
grid = pv.StructuredGrid(x, y, z_scaled)
grid.point_data["Flood <1500m"] = flood_mask.ravel(order="F")

# === Plot
plotter = pv.Plotter()
plotter.set_background("white")
plotter.add_mesh(grid, scalars="Flood <1500m", cmap=["white", "blue"],
                 show_scalar_bar=False)
plotter.add_text("Flood Risk Areas (<1500m)", font_size=12)
plotter.camera_position = 'iso'
plotter.camera.zoom(1.5)
plotter.show()
```

Listing 3: Terrain Plot 1500

```
import numpy as np
import rasterio
import pyvista as pv

# Loading our merged DEM
with rasterio.open("merged.tif") as src:
    dem = src.read(1)
    transform = src.transform

# Cleaning invalid values
dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)

# Downsampling
step = 30
dem_ds = dem[::step, ::step]
z = np.nan_to_num(dem_ds, nan=0)

# Flood mask: elevation < 2000 m
flood_mask = (z < 2000).astype(int)

# Coordinate grid
rows, cols = z.shape
x = np.arange(cols) * transform.a * step + transform.c
y = np.arange(rows) * transform.e * step + transform.f
x, y = np.meshgrid(x, y)

# Scaling Z
```

```
x_range = np.max(x) - np.min(x)
z_range = np.max(z) - np.min(z)
scale_factor = x_range / z_range if z_range != 0 else 1
z_scaled = (z - np.min(z)) * scale_factor * 0.3

# Creating Grid
grid = pv.StructuredGrid(x, y, z_scaled)
grid.point_data["Flood <2000m"] = flood_mask.ravel(order="F")

# Plotting
plotter = pv.Plotter()
plotter.set_background("white")
plotter.add_mesh(grid, scalars="Flood <2000m", cmap=["white", "blue"],
                 show_scalar_bar=False)
plotter.add_text("Flood Risk Areas (<2000m)", font_size=12)
plotter.camera_position = 'iso'
plotter.camera.zoom(1.5)
plotter.show()
```

Listing 4: Terrain Plot 2000

```
import numpy as np
import rasterio
import pyvista as pv

# === Load DEM ===
with rasterio.open("merged_sea_level.tif") as src:
    dem = src.read(1)
    transform = src.transform

# Clean and masking
dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)
sea_rise_threshold = 10  # meters
mask = dem < sea_rise_threshold

# === Downsample ===
step = 30
dem_ds = dem[::step, ::step]
mask_ds = mask[::step, ::step]
z = np.nan_to_num(dem_ds, nan=0)

# Coordinates
rows, cols = z.shape
x = np.arange(cols) * transform.a * step + transform.c
y = np.arange(rows) * transform.e * step + transform.f
x, y = np.meshgrid(x, y)

# Fixing vertical exaggeration
x_range = np.max(x) - np.min(x)
z_range = np.max(z) - np.min(z)
scale_factor = x_range / z_range if z_range != 0 else 1
z_scaled = (z - np.min(z)) * scale_factor * 0.2

#Creating Grid
grid = pv.StructuredGrid(x, y, z_scaled)

# Applying sea level mask
colors = np.where(mask_ds, 1, 0)
```

```
38
39 # Plotting
40 plotter = pv.Plotter()
41 plotter.set_background("white")
42 plotter.add_mesh(grid, scalars=colors, cmap=["lightgray", "blue"],
43                  show_scalar_bar=False)
44 plotter.add_text("Sea Level Rise Risk (<5m)", font_size=12)
45 plotter.camera_position = 'iso'
46 plotter.camera.zoom(1.5)
47 plotter.show()
```

Listing 5: Sea Level Risk 3 m

```
1  import numpy as np
2  import rasterio
3  import pyvista as pv
4
5  # Loading DEM
6  with rasterio.open("merged_sea_level.tif") as src:
7      dem = src.read(1)
8      transform = src.transform
9
10 # Clean and masking
11 dem = np.where((dem <= 0) | (dem > 9000), np.nan, dem)
12 sea_rise_threshold = 5 # meters
13 mask = dem < sea_rise_threshold
14
15 # Downsampling
16 step = 30
17 dem_ds = dem[::step, ::step]
18 mask_ds = mask[::step, ::step]
19 z = np.nan_to_num(dem_ds, nan=0)
20
21 # Coordinates
22 rows, cols = z.shape
23 x = np.arange(cols) * transform.a * step + transform.c
24 y = np.arange(rows) * transform.e * step + transform.f
25 x, y = np.meshgrid(x, y)
26
27 # Fixing vertical exaggeration
28 x_range = np.max(x) - np.min(x)
29 z_range = np.max(z) - np.min(z)
30 scale_factor = x_range / z_range if z_range != 0 else 1
31 z_scaled = (z - np.min(z)) * scale_factor * 0.2
32
33 #Creating Grid
34 grid = pv.StructuredGrid(x, y, z_scaled)
35
36 # Applying sea level mask
37 colors = np.where(mask_ds, 1, 0)
38
39 # Plotting
40 plotter = pv.Plotter()
41 plotter.set_background("white")
42 plotter.add_mesh(grid, scalars=colors, cmap=["lightgray", "blue"],
43                  show_scalar_bar=False)
44 plotter.add_text("Sea Level Rise Risk (<5m)", font_size=12)
45 plotter.camera_position = 'iso'
```

```
plotter.camera.zoom(1.5)
plotter.show()
```

Listing 6: Sea Level Risk 5 m