Resolution Adaptive Vector Rasterization in Discrete Global Grid Systems

Amirhossein Mirtabatabaeipour, John Hall and Faramarz Samavati

February 25, 2022

Abstract

Discrete Global Grid Systems (DGGS) are relatively new globe-based digital earth systems capable of integrating large geospatial datasets. In this report, we describe a new algorithm for rasterizing vector data to DGGS that relies on the hierarchical property of DGGS and adaptively changes to coarser rasterization to reach to target resolution high-resolution girds.

1 Introduction

Throughout human history, we have always longed for more information about our surroundings. Nowadays with trillions of sensors deployed all over the Earth, we gather hundreds of terabytes of data every single day on a wide variety of Earth features, creating massive datasets about these features of the Earth. These datasets are comprised of a few well known data formats, one of which is vector data. Vector data is a series of points connected in some fashion (often geodesic arcs) that form a closed curve, usually on the surface of the Earth. Examples of vector data include borders of countries, and the boundary of natural phenomenon like lakes, rivers and forests. Any type of data, including vector data needs its own specific algorithms and data structures to be fully utilized to solve real world problems.

Geographic Information Systems (GIS) are designed to store, perform analysis on and visualize different geographical datasets like vector datasets. An important process that can greatly help with analysis of vector datasets is discretization of the vector data. This data is discretized to, for example, predict weather using finite element methods [1] and

simulation of bodies of water [2]. While it may be tempting to solve problems like this in a continuous setting, it's often computationally infeasible to do so and instead we use a discretization of the space to solve these problems. This means that we need a method for transforming continuous data into a discretized representation (i.e rasterization).

The problem of how to convert vector data to raster data has been around since the advent of raster displays and it has been an integral part of computer graphics [3]. In computer graphics, we usually have vector data that we wish to render on a raster display. The scanline algorithm [4] and it's later iterations like [5] are to this day some of the main approaches for transforming scene data to raster color. To utilize the power of grid based algorithms in GIS systems, we need a similar grid system to map our vector data onto. With the emergence of Discrete Global Grid Systems (DGGS)s and multiresolution grids [6, 7, 8] as the new generation of globe-based and grid-based GIS systems, creating novel and efficient methods for converting vector to multiresolution grids is an important requirement.

DGGSs are data storage, management and analysis platforms that discretize Earth's surface to create a hierarchy of regular or semi-regular spherical or ellipsoidal cells [9]. Each cell has an index associated with it for better accessibility to that cell's data [10]. This provides a system that allows for working with discrete data defined across the surface of the Earth. However, continuous data needs to be discretized to take advantage of this platform.

Unfortunately, the scanline algorithm can not be used directly on a DGGS grid, as it's designed to work on a rectilinear, regular quad grid. In contrast, a DGGS grid is curved as it follows the curvature of the Earth. It is also common for DGGS cells to not be quadrilateral shaped and on top of that are not always regular. This means that in order to transform vector data to DGGS systems, it is not possible to rely on these regular rasterization algorithms and we need to create new algorithms to deal with this problem.

In this report, we take advantage of DGGS properties to efficiently raster vector data in DGGS. We introduce raster adaptive vector discretization (RAVD) to convert vector data to a set of cells in various resolutions. For a given target resolution, we efficiently identify the boundary and internal cells of the input data. This is done by exploiting the hierarchical structure of DGGS through creating more compact structures in intermediary steps. Our method starts off by finding an initial cell on the boundary of the vector data and then conducting a local search on the most recently found boundary cell to find a new one until we find all of the boundary cells. From here, we first detect the perimeter of internal and external regions that are created by this set of boundary cells. Then starting from a cell at one of the internal regions, we adaptively change the resolution

in a region growing method that finds all internal cells of that region without traversing them all at the target resolution. By repeating the same process for all internal regions, we find internal cells of the given vector feature.

Our novel resolution adaptive method finds a natural representation of all vector features in DGGS. Our experimentation shows that it is 400 times to 44900 times faster in comparison to a direct method.

2 Methodology

2.1 Required Grid Properties and Operations

For this method to work we need certain properties of the underlying grid and it is assumed certain grid operations exist on the chosen grid. For grid properties, we assume the grid allows for congruent subdivision and is hierarchical, spanning the entirety of the part of the Earth containing the vector data of interest. We also need the following features to be implemented:

- Point to cell: A point to cell query to find the cell that includes any given point at a given resolution.
- Arc cell intersection: The ability to find the intersection of an arc between two points of the vector data and a given cell of the grid (if there is any).
- Neighboring cells: The ability to find the neighboring cells of any particular cell. This means that we want the cells that share an edge with our given cell at the same resolution as the given cell.
- Parent cell: The ability to find the parent of a given cell at a coarser resolution of the grid. As we are working with congruent subdivision, each cell at any resolution (other than the coarsest resolution) have a unique parent cell at one resolution above it.
- Children cells: The ability to find the children of a given cell at a finer resolution. Again, since we are working with congruent subdivision, each cell has a finite number of children cells at a finer resolution that partition it.

2.2 RAVD Input

The RAVD method takes the following data as input for discretization of vector features:

1. Hierarchical grid: A hierarchical grid with the above properties and operations defined.
2. Vector feature: A list of points on the surface of the Earth (sometimes given in latitude/longitude coordinates) implicitly connected by arcs across the surface of Earth (often great circle arcs). These points must define a closed loop with no self intersections so that the area enclosed by the vector feature is well defined.
3. Target Resolution: The grid resolution at which the vector feature is to be discretized.

2.3 RAVD method

Given a hierarchical grid, a vector feature and a target resolution, we now present the RAVD method for discretizing the vector feature onto the grid at the target resolution. To perform this discretization, the first step in our method is to find the cells of the grid that intersect the boundary of our vector feature (see Figure 1). To do this, we take the first vertex (p_0) in our vector feature and use the point to cell query on it (see Figure 2). This gives us an initial cell (c_0) that is on the boundary of the vector feature. We then find the intersection of the arc p_0p_1 with the boundary of c_0. Note that if this arc does not intersect with the boundary of c_0, it must be fully contained within the cell and we can simply test the next arc (p_1p_2) in the vector feature for intersection. Now that we have the edge of the cell that the arc intersects with, we determine the neighbouring cell c_1 that shares an edge with c_0 and is on the boundary of our region of interest (see Figure 3). By finding the neighbours of c_1 and determining which ones are intersecting with p_0p_1, we can find c_2. Repeating this process on the neighbours of c_2 and we eventually find all cells that intersect with p_0p_1. The last cell found is the cell that contains p_1, called c_0'. Using c_0' like c_0 and p_1p_2 like p_0p_1 we can find all cells that intersecting with p_1p_2. Repeating this process for all edges will give us the boundary cells of the vector feature (see Figure 4).

Next we must construct a list of interior cells for our region of interest. To do this, we first consider a list of all edges of the cells on the boundary of our region of interest. From the list of these edges we construct a set of vector features that form the boundary and internal regions. To find the internal regions, we test for containment of these regions

within the external vector feature using winding number (see Figure 5). Then, for each internal region, we find a single neighbor that shares an adjoining edge with the boundary of the region (see Figure 6). Using this cell as a seed, we use a region growing algorithm to find all the other cells internal to the region (see Figure 7). For this region growing, we first find the parent of our seed cell. If this parent does not contain a boundary cell, we find the parent of the parent and so forth until we find an intersection with the boundary (see Figure 7). Once we have found an intersection with the boundary, go to the last parent cell we found that did not have any boundary cell inside it and find their neighbors (see Figure 8). These become the new seed cells for the region growing and we repeat aforementioned steps. This repeats until we have all of the cells internal to the region (see Figure 9).

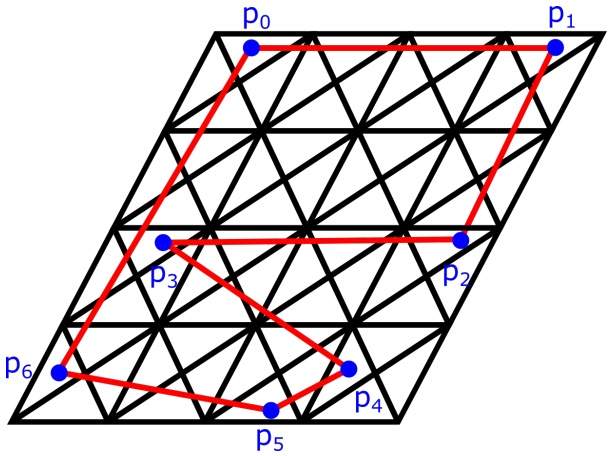

Figure 1: Triangle grid subdivided to resolution 3. In this example, resolution 3 is the target resolution. An example vector feature is shown with red arcs and blue vertices $p_0, p_1, p_2, p_3, p_4, p_5, p_6$.

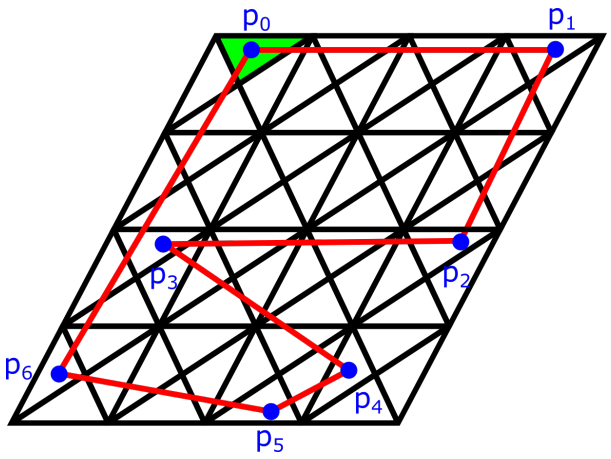

Figure 2: Using the point-to-cell query on vertex p_0 gives us the green cell c_0.

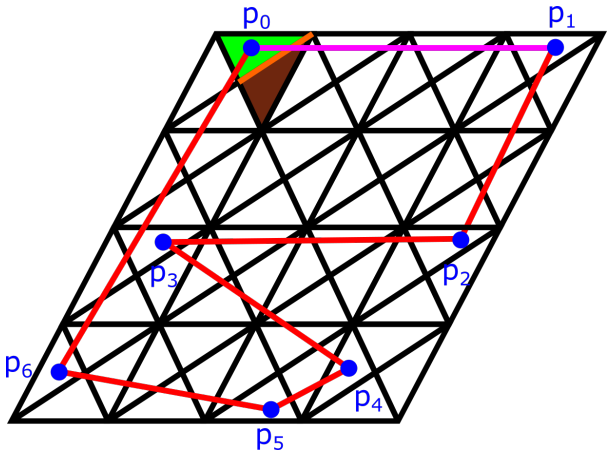

Figure 3: From the intersection of arc p_0p_1 with neighbours of c_0, then next cell c_1 is determined.

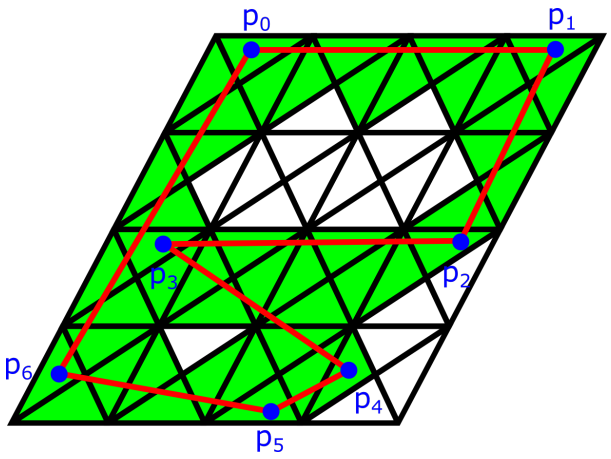

Figure 4: Continuing to intersect arcs of the vector feature with the boundary of cells (repeating the procedure in Figure 3 with the cell labeled in brown as the new green cell) allows us to construct a list of all the cells labeled in green.

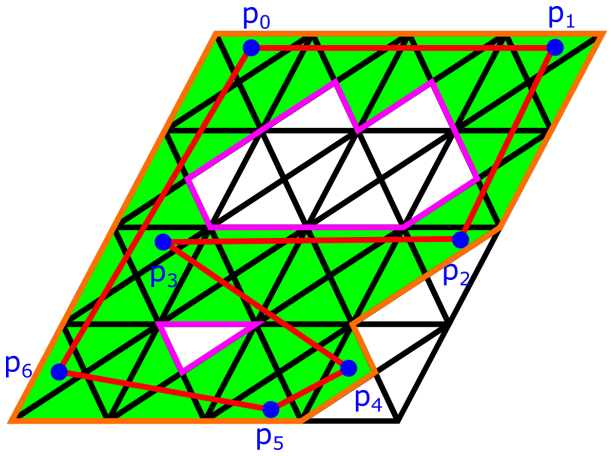

Figure 5: The edges in orange and magenta are constructed by noticing that they comprise all of the edges of the cells in green minus the edges that intersect the red arcs. However, the edges that intersect the red arcs were already found in the previous steps. To find out which edges should be labeled orange and which magenta, first construct these three vector features by associating edges that share a vertex. To find out which vector features are interior (magenta), calculate the winding number of a single vertex on any of these three with respect to any other.

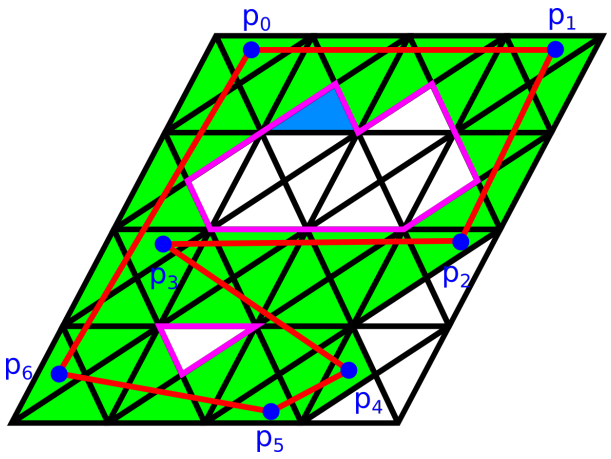

Figure 6: To find the index of the cell in brown, notice that it shares a magenta edge with a cell labeled in green. We use the blue cell as a seed for region growing to find the cells enclosed in the magenta region.

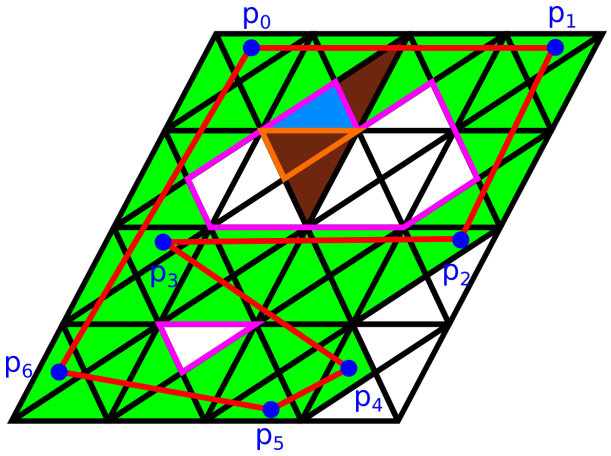

Figure 7: To find the remaining cells enclosed by the magenta region, we find the parent of the cell labeled with blue (same cell from Figure 6). If this parent (labeled in brown) contains a cell previously labeled in green, we know the brown parent is on the boundary of the vector feature. In this case, we check the neighbours of the blue cell, attempting to find one not previously labeled green. In this example, the cell outlined in orange will be discovered.

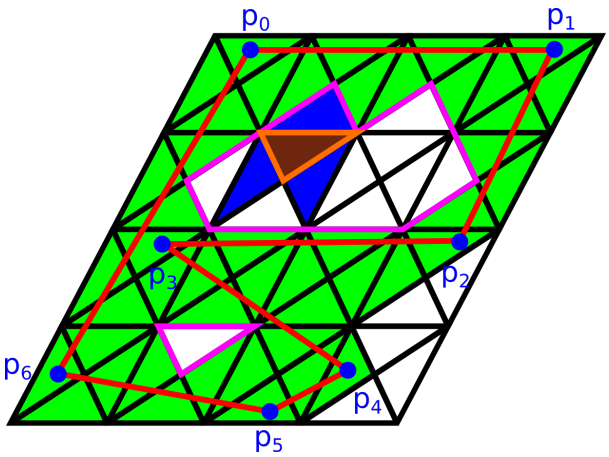

Figure 8: We now consider the cell outlined in orange and find their neighbors (labeled in dark blue). Again we check if these neighbors contain a cell labeled in green or has they previously designated as interior cell. If not, we have found more cells in the interior of the magenta region and we repeat the step in Figure 6 on these new cells.

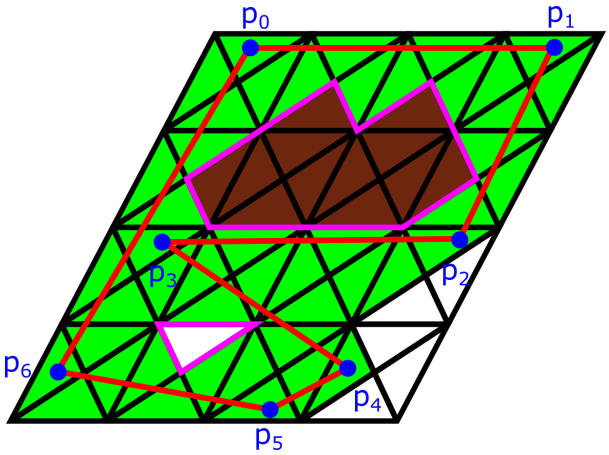

Figure 9: We repeat the steps in Figures 7 and 8, until we have found all the cells enclosed by the magenta region. We repeat this procedure for all the magenta regions enclosed in the vector feature.

procedure RAVD($polygon, resolution$)
  for $i \leftarrow 0;\ i < length(polygon) - 1;\ i++$ **do**
    $border.add(GrowEdge(polygon[i], polygon[i+1])$ ▷ See figures 1, 2, 3, 4
  end for
  $perimeter\_list \leftarrow CreatePerimeter(border)$
  $outer\_perimeter \leftarrow FindOuterPerimeter(perimiter\_list)$
  $interior \leftarrow FindInteriorCells(border, outer\_perimeter)$
end procedure

```
procedure FindInteriorCells(border, outer_perimiter)
    for b_cell ∈ border do
        neighbours ← FindNeighbours(b_cell)
        borderWithParents ← FindAllParents(border)
        for n_cell ∈ neighbours do
            if !n!n_cell.hasEdge(outer_perimeter) then
                ResolutionAdaptiveCellGrow(n_cell, borderWithParents, interior)
            end if
        end for
    end for
end procedure

procedure ResolutionAdaptiveCellGrow(cell, border, interior)
    interior.add(cell.parent)
    if !cell.growParent then
        cell.growParent ← TRUE
        ResolutionAdaptiveCellGrow(cell.parent, border, interior)
    else if !cell.growNeighbour then
        cell.growNeighbour ← TRUE
        for n_cell ∈FindNeighbours(cell) do
            if !HasInteriorParent(n_cell, interior) &&!n_cell.growNeighbour then
                ResolutionAdaptiveCellGrow(n_cell, border, interior)
            end if
        end for
    else
        for c_cell ∈FindChildren(cell) do
            ResolutionAdaptiveCellGrow(c_cell, border, interior)
        end for
    end if
end procedure
```

3 Results

We ran RAVD against the direct method of testing each cell at target resolution against all line-segments that makes up the border of the given polygon to determine if it intersects with it or is it inside the polygon or outside. We compare them on three provinces of

Alberta, Ontario and Quebec. The RAVD algorithm shows significant improvement in comparison with the direct method as it produces the results 400 to 44900 times faster than the direct method (See Table 1). Testing has been done on an AMD 5800X with 32GB of RAM. Both codes are running on a single thread and no paralization has been used to optimize neither algorithm's run-time.

	Alberta		Quebec		Ontario	
Resolution	Direct	RAVD	Direct	RAVD	Direct	RAVD
1	1352.67 ms	1.76 ms	62452.07 ms	16.42 ms	9310.80 ms	7.35 ms
2	5699.66 ms	2.41 ms	252752.00 ms	24.44 ms	37234.91 ms	10.45 ms
3	24406.96 ms	2.86 ms	942273.18 ms	26.31 ms	154754.17 ms	12.10 ms
4	99773.58 ms	3.74 ms	3765001.71 ms	28.82 ms	618862.83 ms	13.76 ms

Table 1: Run time of Direct and RAVD algorithm on the borders of three provinces in Canada.

References

[1] Colin J Cotter and Andrew TT McRae. "Compatible finite element methods for numerical weather prediction". In: *arXiv preprint arXiv:1401.0616* (2014).

[2] Nuttapong Chentanez and Matthias Müller. "Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid". In: *ACM Trans. Graph.* 30.4 (July 2011). ISSN: 0730-0301. DOI: 10.1145/2010324.1964977. URL: https://doi.org/10.1145/2010324.1964977.

[3] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics, Fourth Edition*. 4th. USA: A. K. Peters, Ltd., 2016. ISBN: 1482229390.

[4] C. Wylie et al. "Half-tone perspective drawings by computer". In: *AFIPS '67 (Fall)*. 1967.

[5] Juan Pineda. "A Parallel Algorithm for Polygon Rasterization". In: *SIGGRAPH Comput. Graph.* 22.4 (June 1988), pp. 17–20. ISSN: 0097-8930. DOI: 10.1145/378456.378457. URL: https://doi.org/10.1145/378456.378457.

[6] Matthew B. J. Purss et al. "The OGC® Discrete Global Grid System core standard: A framework for rapid geospatial integration". In: *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. 2016, pp. 3610–3613. DOI: 10.1109/IGARSS.2016.7729935.

[7] Ben Bondaruk, Steven Roberts, and Colin Robertson. "Discrete Global Grid Systems: Operational Capability of the Current State of the Art". In: 7.6 (2019).

[8] John Hall et al. “Disdyakis Triacontahedron DGGS”. In: *ISPRS International Journal of Geo-Information* 9.5 (2020). ISSN: 2220-9964. DOI: 10.3390/ijgi9050315. URL: https://www.mdpi.com/2220-9964/9/5/315.

[9] Troy Alderson et al. “Digital Earth Platforms”. In: *Manual of Digital Earth*. Ed. by Huadong Guo, Michael F. Goodchild, and Alessandro Annoni. Singapore: Springer Singapore, 2020, pp. 25–54. ISBN: 978-981-32-9915-3. DOI: 10.1007/978-981-32-9915-3_2. URL: https://doi.org/10.1007/978-981-32-9915-3_2.

[10] Ali Mahdavi-Amiri, Troy Alderson, and Faramarz Samavati. “A Survey of Digital Earth”. In: *Computers & Graphics* 53 (2015), pp. 95–117. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2015.08.005. URL: https://www.sciencedirect.com/science/article/pii/S0097849315001399.