GPU-Accelerated Scalable Geocomputation for Large-Scale Lidar-derived Road Elevation Models

Yan Liu, Ph.D.
Computational Scientist
Computational Sciences and Engineering Division
Oak Ridge National Laboratory

Objectives

1. Create a road lidar dataset as a baseline dataset for developing the road elevation model
   - Only lidar points on road surface are included
   - Organized by counties or by TxDOT maintenance sections
2. For each road shape (polygon or centerline), add the elevation value *z* to each (*x*,*y*) coordinate on the road shape: 2D → 2.5D
   - Road shape sources: Ecopia, TxDOT road inventory, …
   - GIS format: Polygon → Polygon Z; LineString → LineString Z

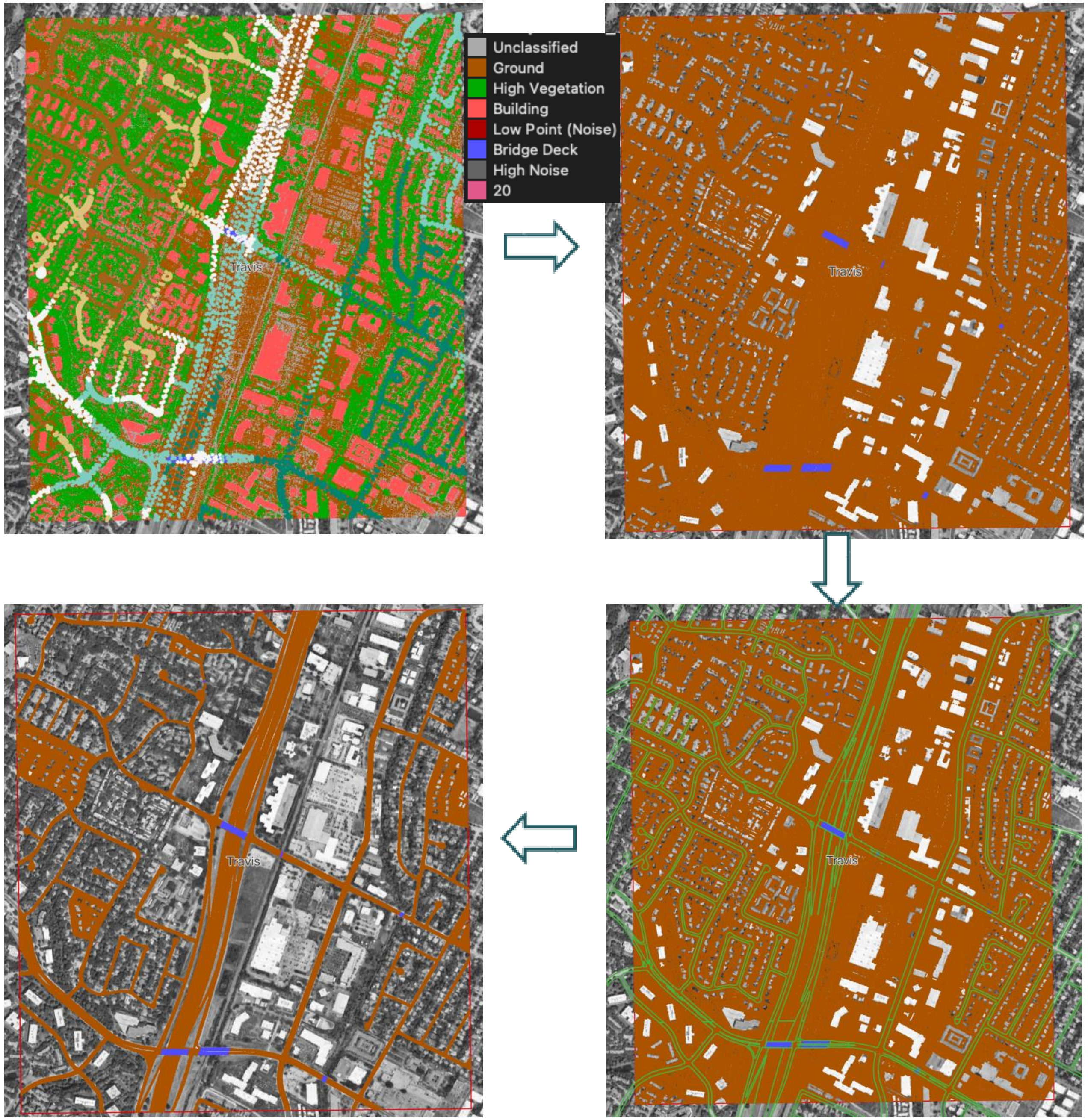
- Status
  - Project started in May 2023; the Austin District is processed in October 2023
    - 3D road shapes are sent to Dr. Maidment (proprietary data, not published)
    - Road lidar data:
      - By counties: https://web.corral.tacc.utexas.edu/nfiedata/road3d/austin_district/AustinCounties_H_epsg6343_V_epsg5703/
      - By maintenance sections: https://web.corral.tacc.utexas.edu/nfiedata/road3d/austin_district/AustinMaintenanceSections_H_epsg6343_V_epsg5703/
  - Software
    - Took longer than expected to develop due to the complexity of lidar data and high-performance computing requirements
    - The workflow software is being polished for open source release

GIS Processing
- road lidar construction

1. Load lidar tile
2. Extract ground and bridge points
3. Load road polygons
4. Point-in-polygon (PIP) test for road lidar points

- Computational intensity
  - Step 4:

num_pip_tests = num_gb_pnts * num_polygons



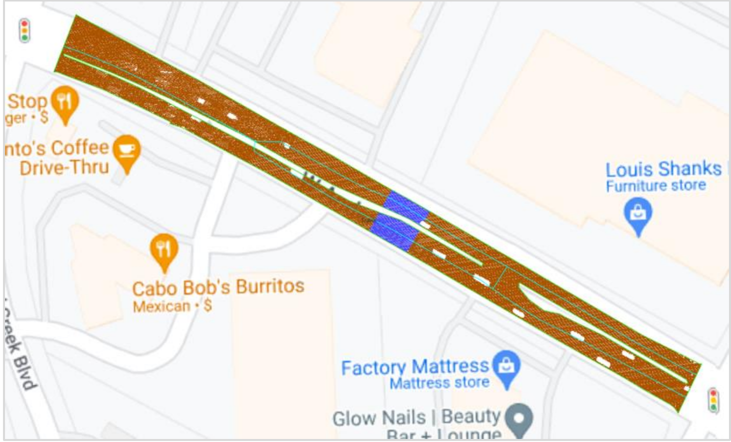
GIS Processing
– z interpolation on road shapes

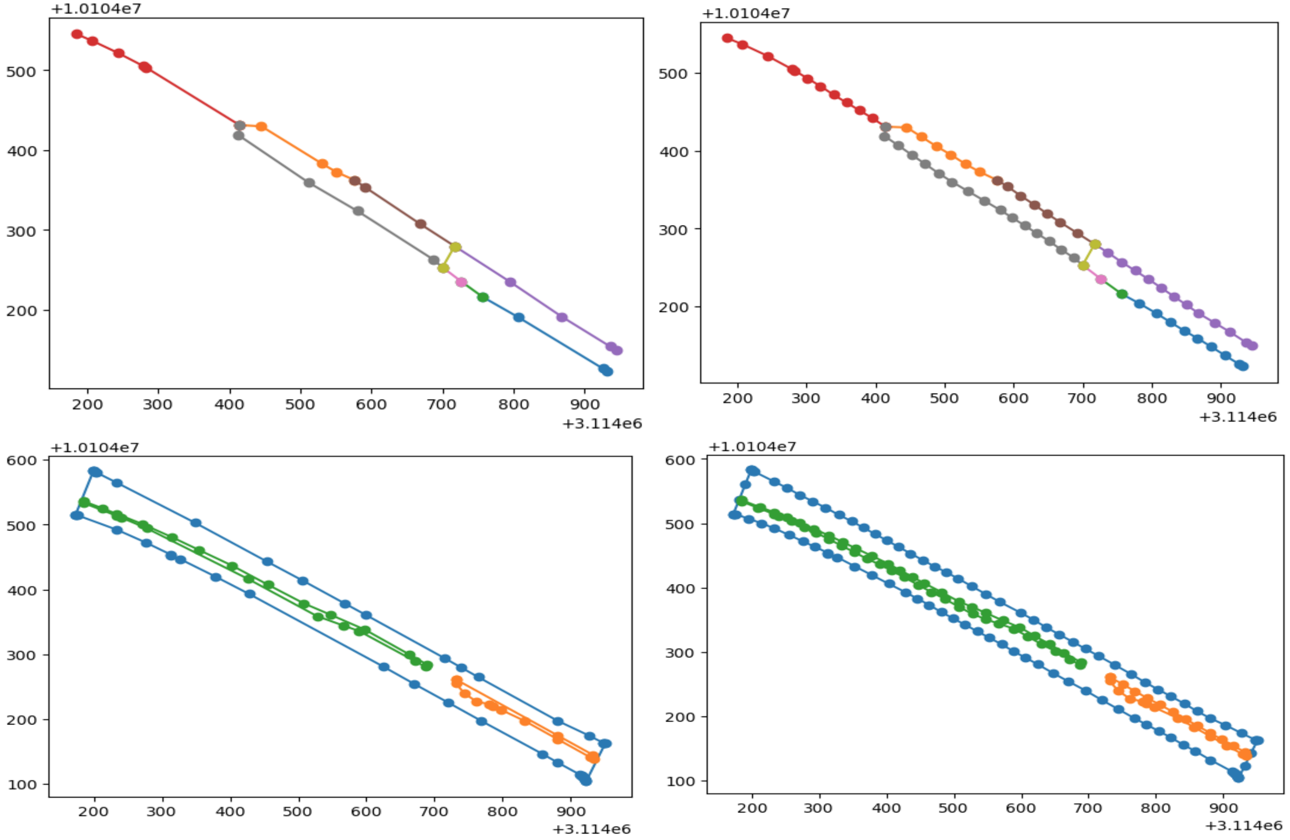
1. Load road lidar tile
2. Load road polygons and centerlines
3. Evenly space each line segment → {*query_points*}
4. For each query point, search neighboring lidar points → z sample
   – Point-in-polygon tests
5. Interpolate z of the query point from the sample

- Computational intensity
  – Step 4:

num_pip_tests = num_qp * num_road_lidar_pnts

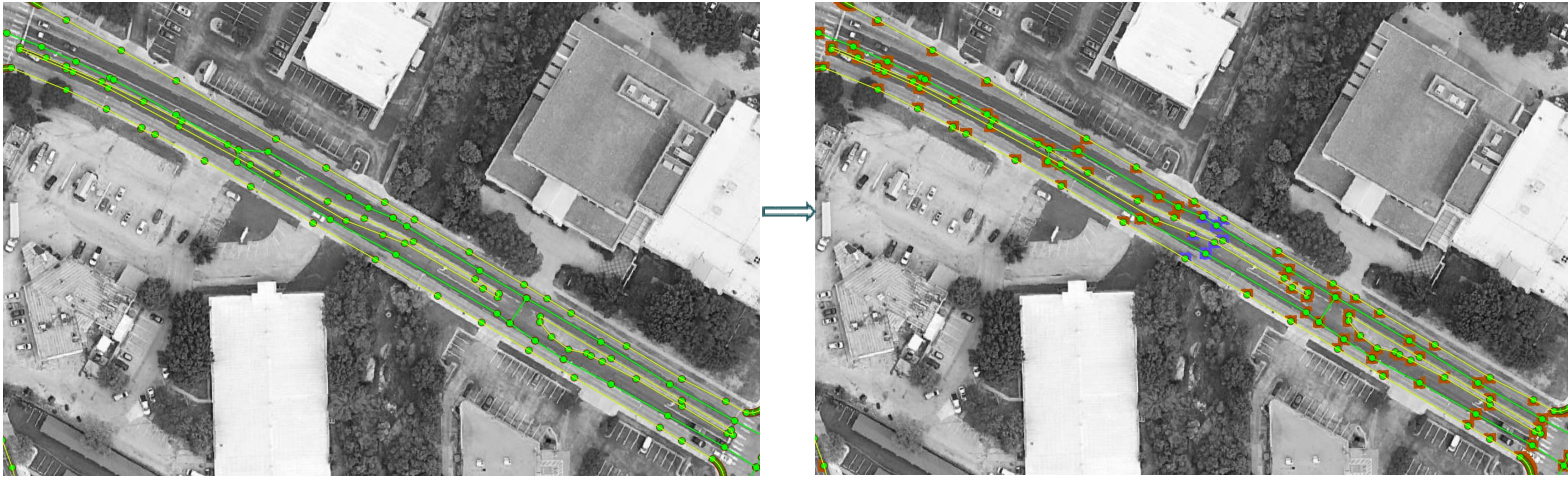
Shoal Creek and Anderson Ln





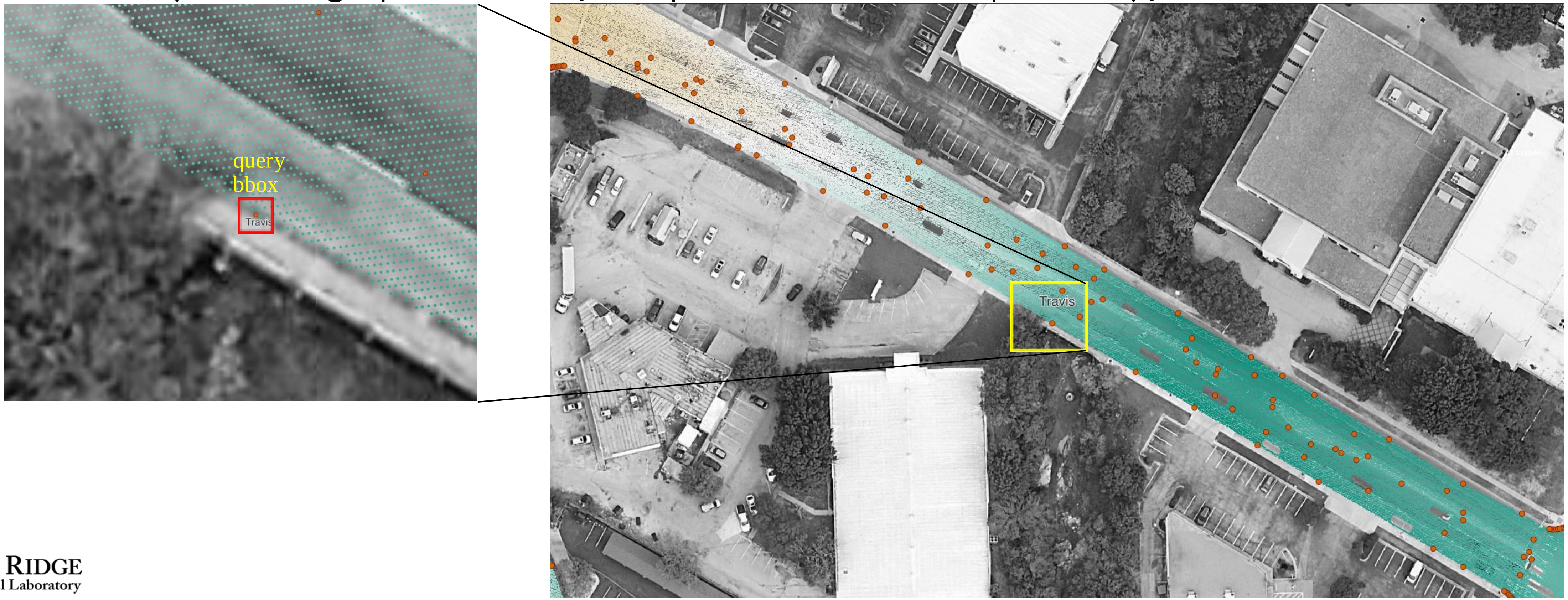
GIS Processing
- radius search for neighboring lidar points

- For each point on a road shape, search for neighboring lidar points within a radius
- Point in polygon/circle search is computationally intensive



GIS Processing
- z Interpolation Algorithm

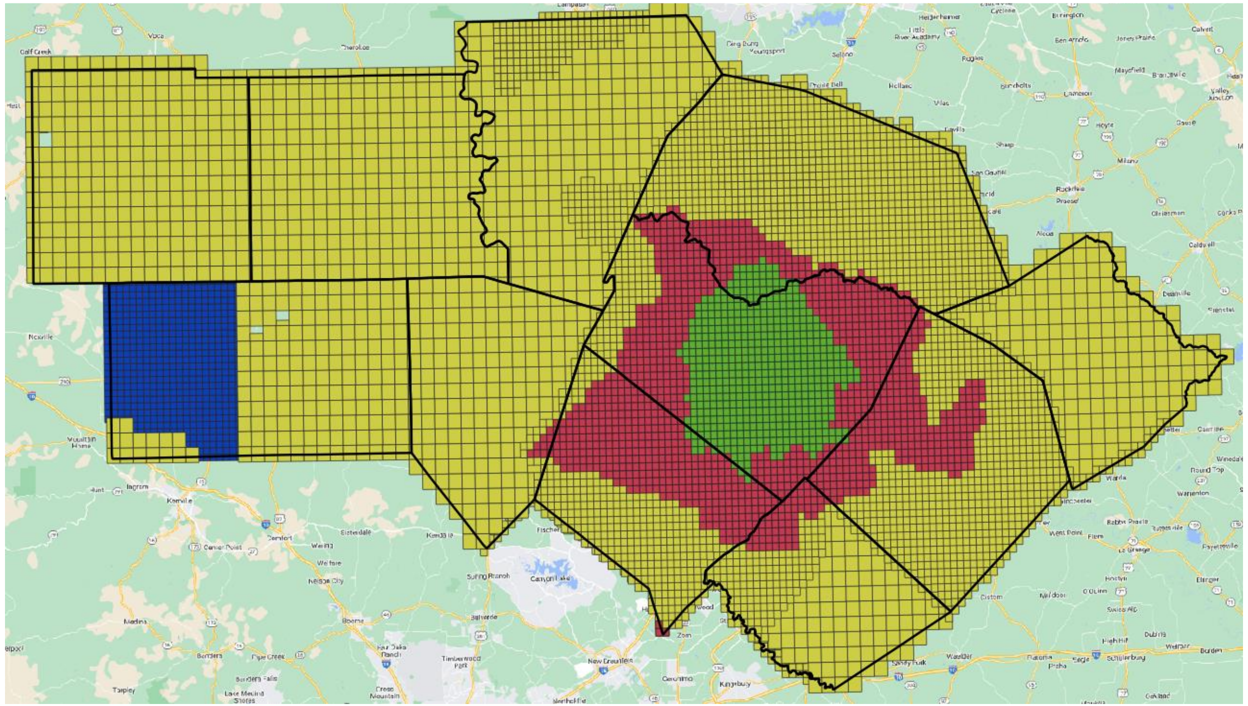
- If num of bridge points > num of neighboring points * 15%, use bridge points only
- Iteratively, smooth the sorted (by *Z*) points if | *Zmean* – *Zmedian* | > 1ft:
  - If *Zmean* – *Zmin* > *2ft,* get rid of the first point (w/ min *Z*)
  - If *Zmean* – *Zmin* > *2ft,* get rid of the last point (w/ max *Z*)
- Otherwise (not enough points left, or | *Zmean* – *Zmedian* | <= 1ft), return *Zmean*



Data Volume and Computing Environment

	#units	Size	Tile extent & size	Projection
Central Texas 2017	5811 tiles	608GB	~1.5km x 1.5km; ~15KB-1.8GB (145M points)	EPSG:26914, NAD83 / UTM zone 14N EPSG:6343, NAD83(2011) / UTM zone 14N EPSG:3721, NAD83(NSRS2007) / UTM zone 14N EPSG:6369, Mexico ITRF2008 / UTM zone 14N
Bexar-Travis 2021	516 + 3157 = 3673 tiles	857GB	28cm: ~15ft x 16ft; ~163MB - 1.3GB 50cm: ~5k ft x 5.7k ft; ~11MB - 350MB	EPSG:6578, NAD83(2011) / Texas Central (ftUS) EPSG:6588, NAD83(2011) / Texas South Central (ftUS)
South Central Texas 2018	528 tiles	30GB	~1.5km x 1.5km; ~40-90MB	EPSG:6343, NAD83(2011) / UTM zone 14N
Total	**10,012**	**1.5TB**		

Ecopia road data: 75,855 polygons, 285,607 centerlines. Projection: EPSG:32614, WGS 84 / UTM zone 14N



High-performance computing environment

- Oak Ridge Research Cloud
- HPC machine specification
  - 96 CPU cores, Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz
  - 800GB memory
  - 100TB storage
  - 4 NVIDIA V100S GPUs
  - Network: Globus transfer on high-speed network between ORNL and TACC
- Future processing may use TACC Lonestart6

Computational Strategies

- Define the basic computing element to enable parallel computing paradigms
  - Each tile is a basic computing element, not a road polygon or centerline
    - To avoid visiting a lidar tiles for multiple times
- Maximize the use of vectorized processing
  - CPU: Numpy vectorized operations
  - GPU: customized batch processing using Rapids tools (*cupy, cudf, cuspatial*)

Assumptions

- Projection
  - Each input/output projection has an EPSG number
  - Only meter ←→ foot conversion is needed
  - Tile processing uses the native projection of the lidar tile → reprojection is needed
- Lidar input
  - Ground | bridge classification among different las point formats is exclusive
    - No conflict: ground = 2 in point format *a* and ground = 4 in point format *b*
- Single output projection
  - Horizontal EPSG:6343 (UTM 14N)
  - Vertical EPSG:5703 (NAVD88 height)

Computing Workflow

1. [S] Create R-tree road polygon index
2. [S] Create lidar tileset table with processing priority
3. [PG] Batch-process all lidar tiles for 3D road shape construction
    1. [S] Filter road polygons and centerlines in the tile
    2. [G] Crop lidar tile to road tile (point-in-road-polygon tests on GPU)
    3. [S] Space road polygons and centerlines; create query points and query bbox
    4. [G] Radius search for each query bbox on GPU
    5. [S] Z-interpolation for each query point (CPU acceleration via vectorization)
    6. [S] Aggregate 3D query points and associate them with road polygon/centerline
4. [PG] check and re-run failed tiles
5. [S] Generate XYZ road polygons and centerlines
6. [P] Reproject road tiles to the output projection using *pyproj*
7. [PG] Crop overlapping tiles ordered by priority
    1. [S] Create tile bbox R-tree
    2. [G] For each tile, skip non-overlapping and completely covered tiles; crop intersected tiles
8. [P] Generate copc tiles using *untwine*
9. [P] Generate road tiles by counties and maintenance sections using *lasmerge*
10. [P] Cleanup

```
After the 1st run: 109 failed tiles,
100 of them are lcra07 old tiles that
have version 1.0, which is no longer
supported. Other 9 were caused by GPU
memory contention, resolved by simply
re-running them with less parallelism
(=2).
```

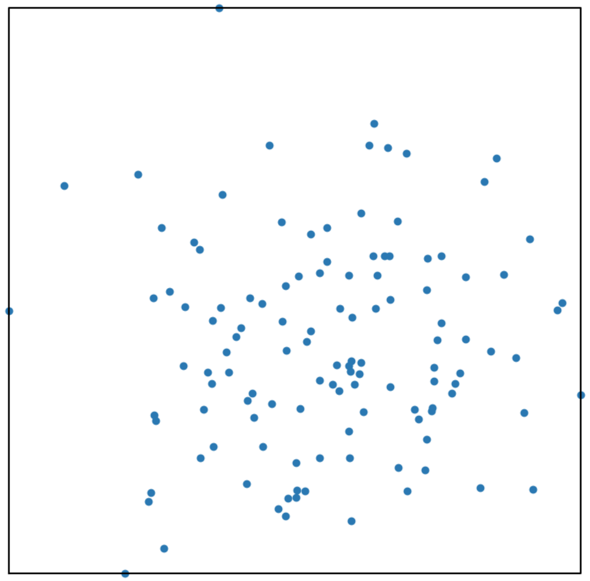
```
Step 7:
  6030 non-overlapping
  1111 completely covered
   811 cropped
```

Step 6-9: road tile processing

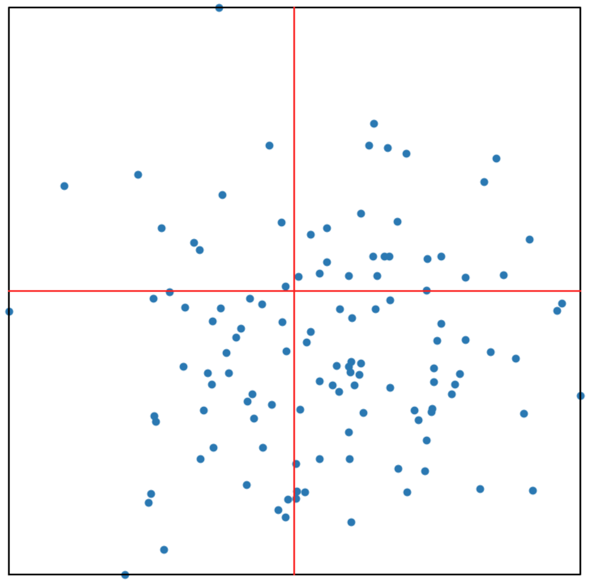
S: sequential computing; **P**: CPU parallel; **G**: GPU parallel

Quadtree Indexing of Road Lidar Points

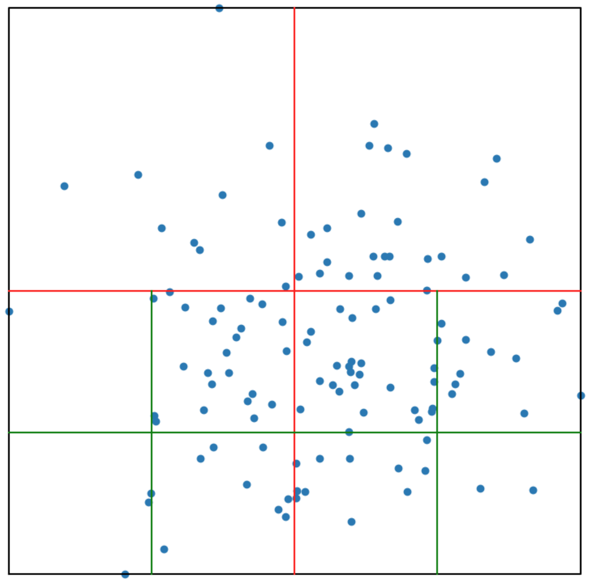
max_size/quad = 30



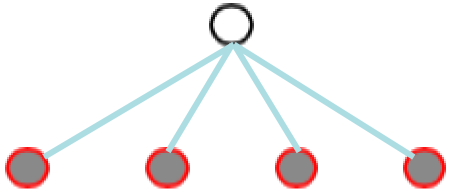
point data

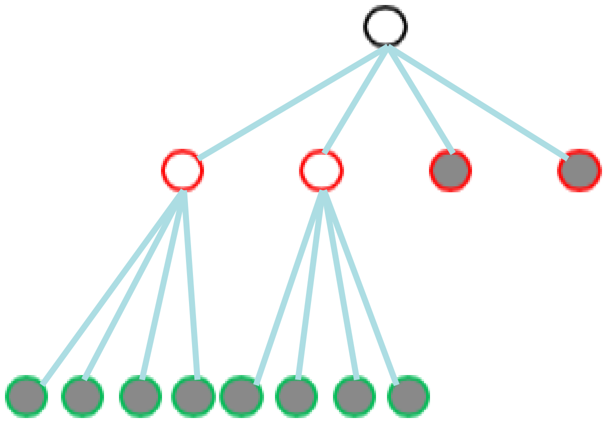


level 0 quadtree



level 1 quadtree





cuSpatial code

```
key_to_point, quadtree = cuspatial.quadtree_on_points(
    points, minx, maxx, miny, maxy, scale, max_depth, max_size
)
```

quadtree data table

num of indexed points: 120

	key	level	is_internal_node	length	offset
0	0	0	True	4	5
1	1	0	True	4	9
2	2	0	False	14	86
3	3	0	False	19	100
4	8	0	False	1	119
5	0	1	False	4	0
6	1	1	False	12	4
7	2	1	False	5	16
8	3	1	False	24	21
9	4	1	False	7	45
10	5	1	False	1	52
11	6	1	False	28	53
12	7	1	False	5	81

Radius Search Using Quadtree

- Intersection of query bounding box and quads
  - Which quads are to be searched next?

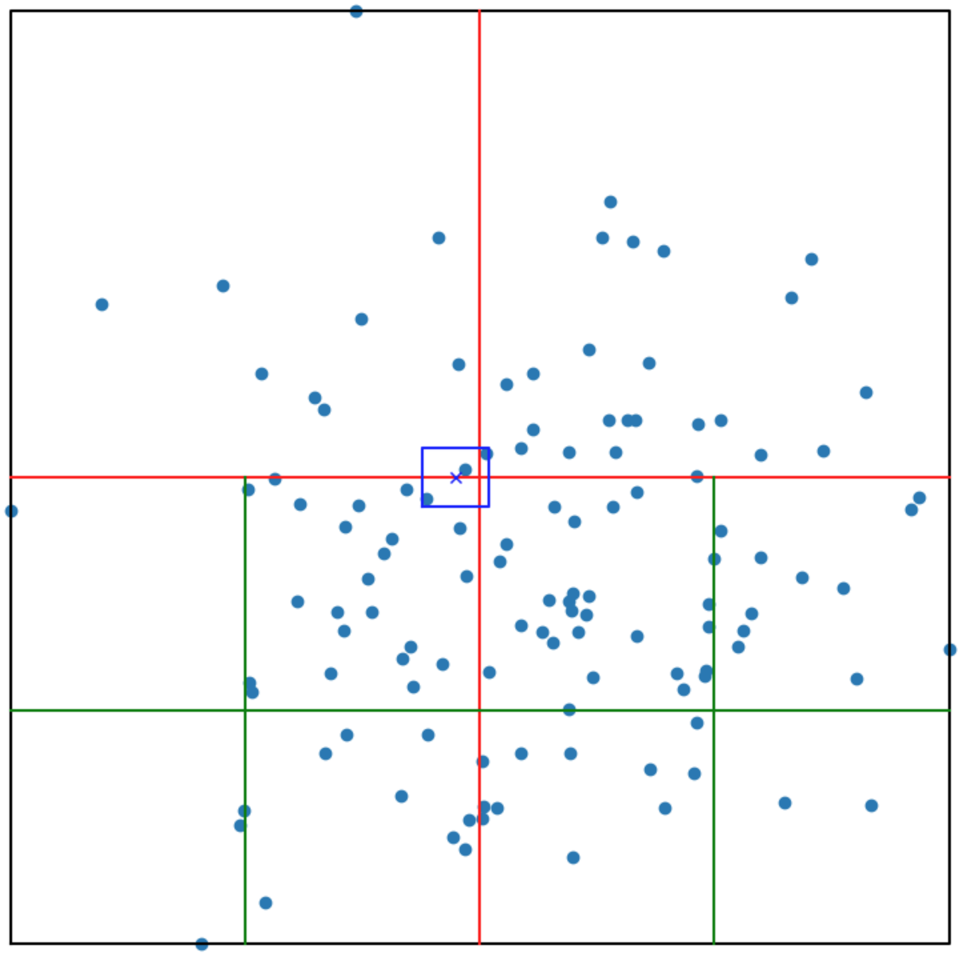
```
intersections = cuspatial.join_quadtree_and_bounding_boxes(
    quadtree=quadtree,
    bounding_boxes=gpu_bboxes,
    x_min=minx, x_max=maxx, y_min=miny, y_max=maxy,
    scale=scale,
    max_depth=max_depth
)
```

	bbox_offset	quad_offset
0	0	8
1	0	11
2	0	2
3	0	3

- Point-in-polygon using quadtree
  - Test if a lidar point in an intersected quad is within the query bounding box

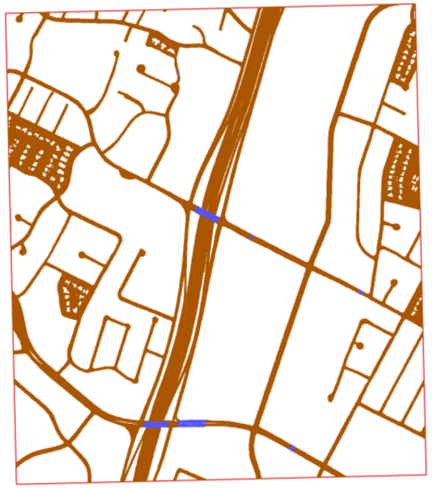
```
polygons_and_points = cuspatial.quadtree_point_in_polygon(
    poly_quad_pairs=intersections,
    quadtree=quadtree,
    point_indices=key_to_point,
    points=points,
    polygons=cuspatial.GeoSeries(pnt_bbox_geoms)
)
```

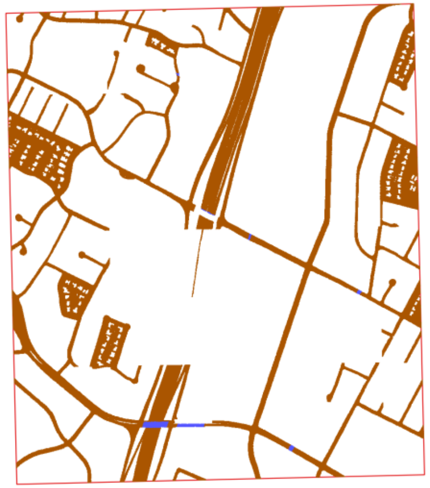
	polygon_index	point_index
0	0	43
1	0	91
2	0	93

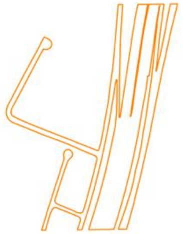


Technical Exploration for Lidar Point Cropping and Radius Search

- PDAL cropping + PCL octree neighborhood search
  - Issue: PCL octree search slows down dramatically on large lidar data. Search on the 1.8GB lidar data could not finish within 30min
- PDAL cropping
  - If the input MultiPolygon is complex, PDAL does not produce correct results
  - Cropping polygon by polygon + lasmerge works, but too slow. 1hr for the 90m-point lidar tile covering Shoal Creek and Anderson Ln







- Laspy + SciPy cKDTree
  - Load lidar points using *laspy;* build a KD-tree; do radius search
  - It works and it is fast (1m for loading data; subseconds for search)
  - Issue: it uses a lot of memory (2GB lidar uses 46GB memory). Not practical for parallel computing of 13k lidar tiles
- GPU: cuSpatial point in polygon for cropping
  - It's super fast (a few seconds to load data, subseconds for search)
  - cuSpatial is part of NVIDIA RAPIDS
    - CUDA memory error when lidar data or polygons are too large

GPU-Acceleration for 3D Road Shape Processing

- Programmability on GPU in Python
  - `cupy – numpy`
  - `cudf – pandas`
  - `cuSpatial – shapely`
    - Not really, but cuSpatial has point-in-polygon test and quadtree search
    - Vector standard - GeoArrow
  - Difficulties
    - No memory management support
    - New → poor documentation
      – Source code reading is necessary
    - Code can crash without useful error messages
- Data parallel computing
  - Desirable for massive data stream processing
  - One GPU card can be shared by multiple processes
  - Multi-GPU computing model is straightforward in our use case
    - Embarrassingly parallel

Accelerated	Solution	CPU/GPU alternatives
Find road shapes in lidar tile	• R-tree search (CPU, <1s) Efficiency: 99.99%	Too slow without indexing
Road lidar cropping	• *laspy* filter (CPU, 1.3s) • point-in-polygon (GPU, 2.4s)	*pdal* pipeline (CPU, 3m) - cannot handle complex polygons
Radius search	• quadtree search (GPU, 5.4s, 0.4GB GPU mem) • search result aggregation (CPU, 0.5s)	• KDtree search (CPU, 4.8s, 46GB mem) • PCL Octree C++: not scalable • aggregation via pandas (5m34s) • aggregation on GPU (>10m)
Z interpolation	numpy vectorization (CPU, 0.6s)	numpy iteration (CPU, 4.2s)

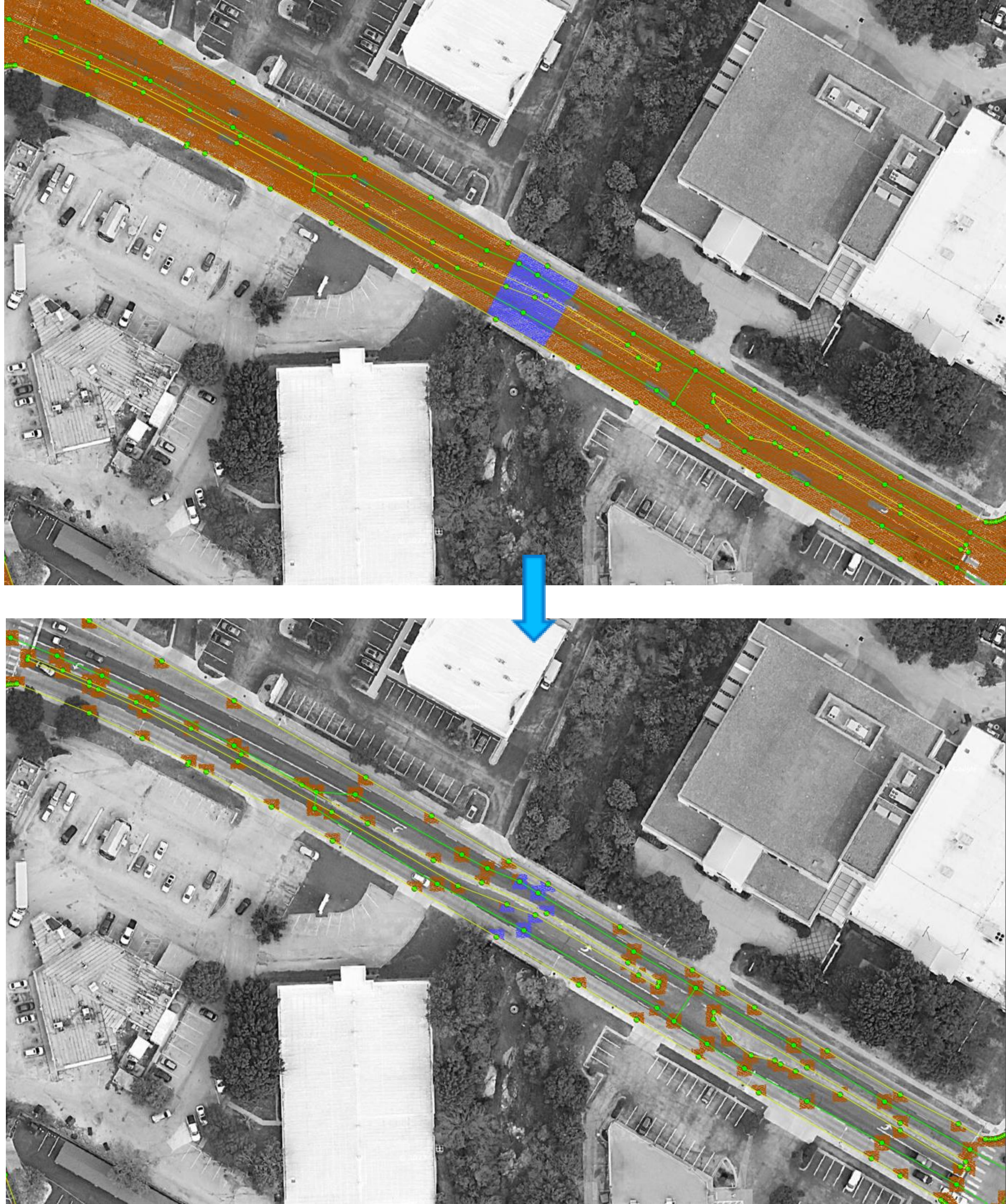
image source: https://www.servethehome.com/inspur-nf5488m5-review-a-unique-8x-nvidia-tesla-v100-server/

Benchmark data: 90m lidar points

GPU Batch Processing

- *cuSpatial* limitations
  - GPU memory limitation on how many lidar points can be quadtree-indexed at a time
  - Point-in-polygon search for multiple query bounding boxes is supported, but GPU memory limits how many query points can be served at a time
- Batch processing of radius search
  - Batch construction for both lidar points and query bounding boxes
    - Quadtree search on a batch of lidar points generates generates a subset of lidar points within the radius. Aggregation needed to get the final result
  - Pseudo code

```
• for each batch of lidar points
  – construct quadtree
  – for each batch of query bboxes
    • intersect bboxes and quads
    • point-in-polygon test for each lidar
      point in intersected quads
• For each query point
  – aggregate lidar points within the radius
```

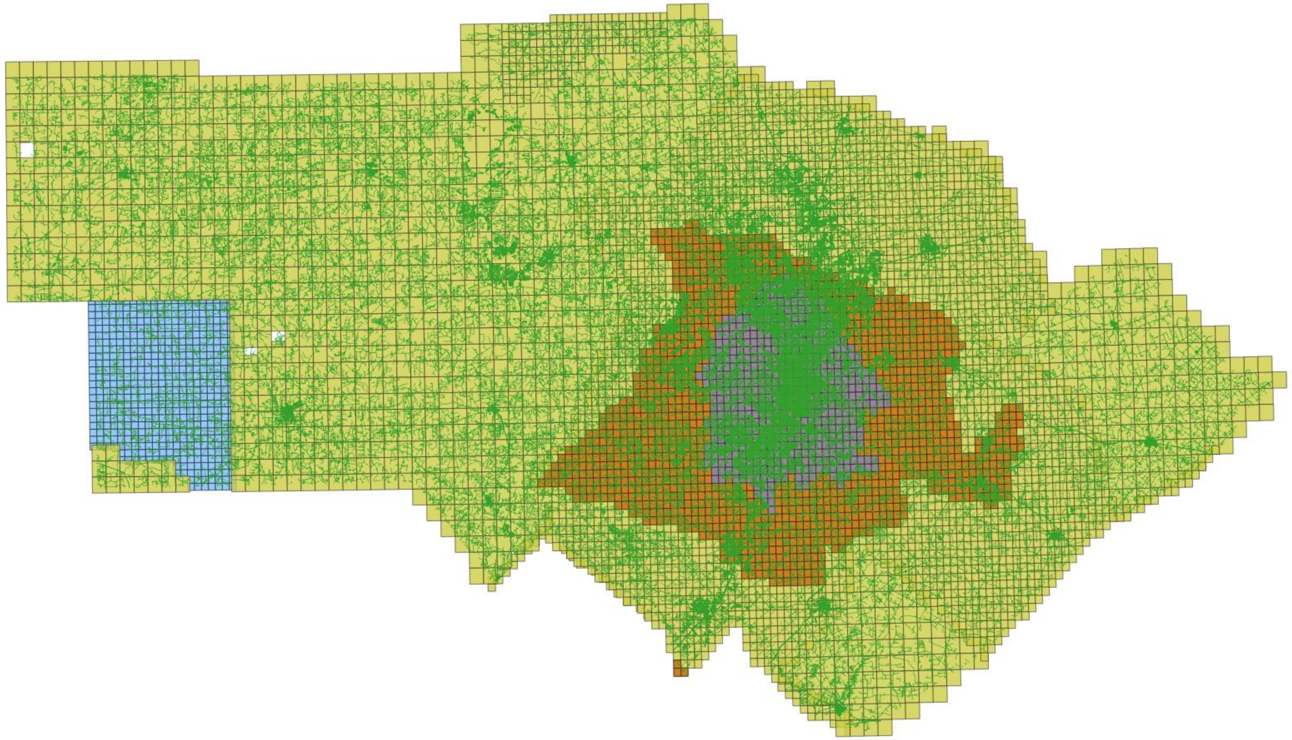


Oak Ridge National Laboratory

Results: 3D Road Shapes

Austin District Output

- Road lidar dataset
  - 3.85 billion points
- Road shape datasets
  - 3D LineString Z: 285,558 / 285,607
  - 3D Polygon Z: 75,833 / 75,855



Computational Performance

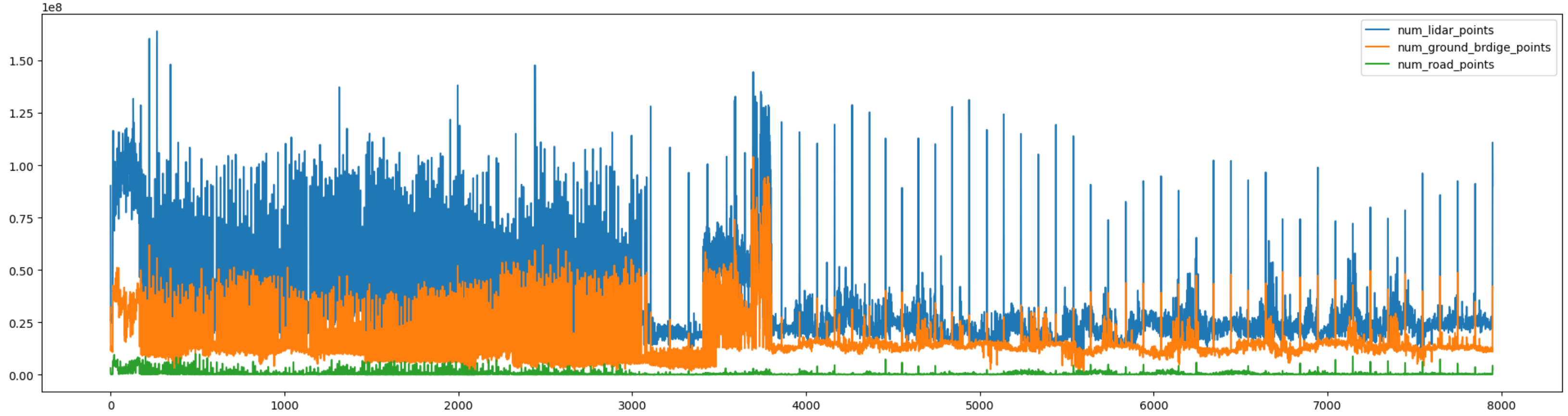
computing	4 GPUs, parallelism/gpu=6
road shape R-tree indexing	6 minutes
10,012 lidar tiles	skipped: 2060 (Bexar tiles) processed: 7952 success: 7843 failed: 109 • 3: GPU quadtree search error • 6: GPU memory access error • 100: old lcra0 las V1.0 not supported
tile computing time	3 hours 20 minutes GPU parallelism: 6
rerun of failed tiles	success: 109, time: 12 minutes GPU parallelism: 1
road shapes aggregation	8 minutes 32 seconds
road lidar aggregation	3 hours (to double check)

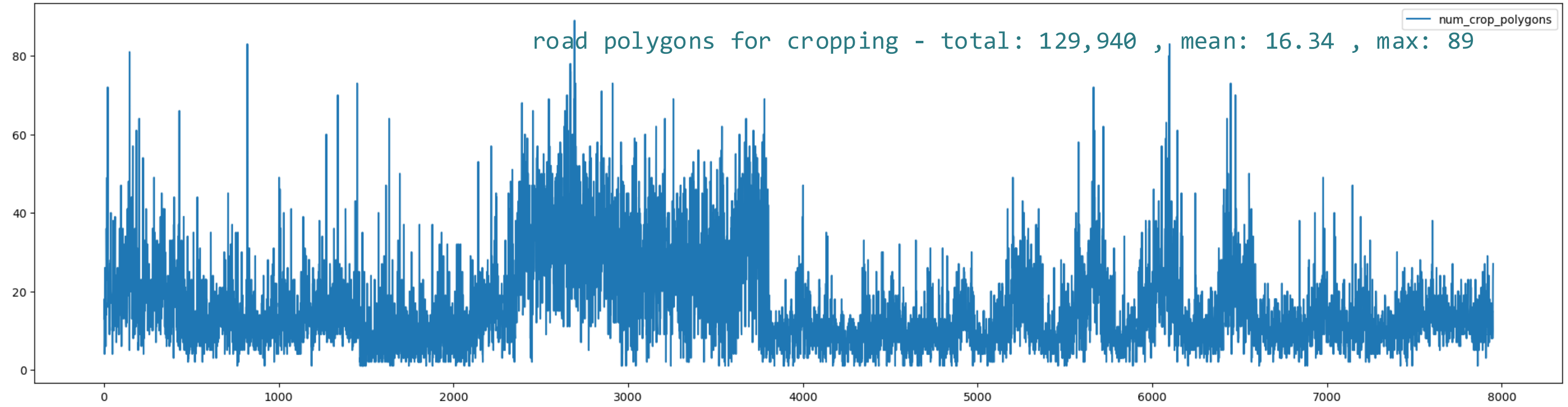
Oak Ridge National Laboratory

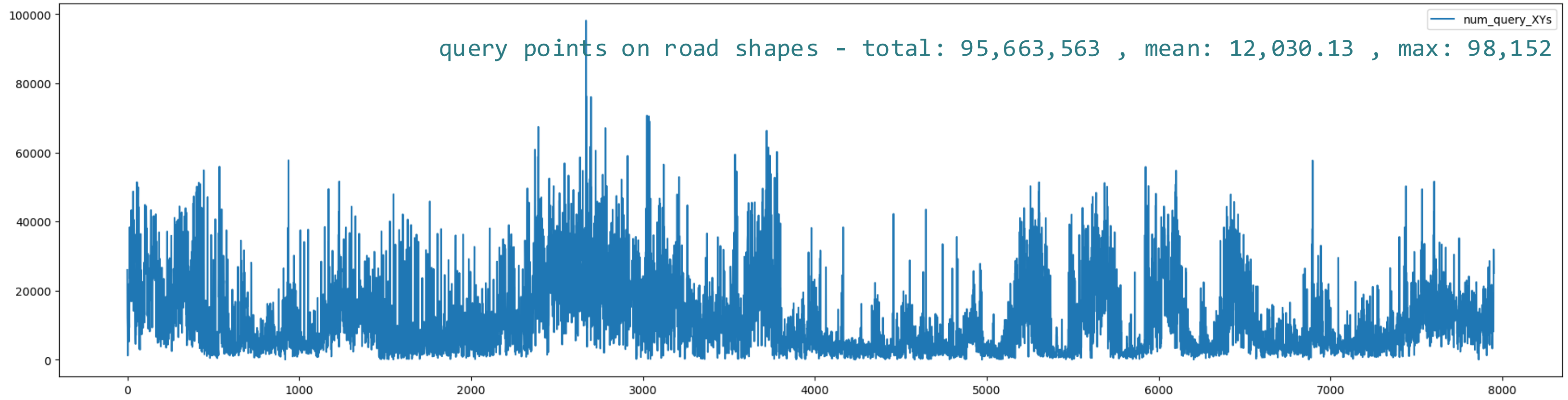
Statistics

- 230 billion lidar points in 7,952 lidar tiles are scanned
- 3.86 billion road lidar points are extracted
- 1.96 trillion point-in-polygon tests
  - point-in-polygon tests after R-tree search for tile-road polygon intersection
  - road lidar has 1.67% of original lidar points, on average
- number of radius search operations w/o quadtree
  - 369,015 trillion
  - quadtree search significantly reduced this number
- 95.6 million Z values are added to road centerlines and polygons
  - evenly spaced. more points than in the original Ecopia data

lidar points - total: 230,420,449,907 , mean: 28,976,414.73 , max: 163,961,040
ground/bridge points - total: 112,709,640,901 , mean: 14,173,747.6 , max: 103,955,779
road points - total: 3,857,431,762 , mean: 485,089.51, max: 9,888,605



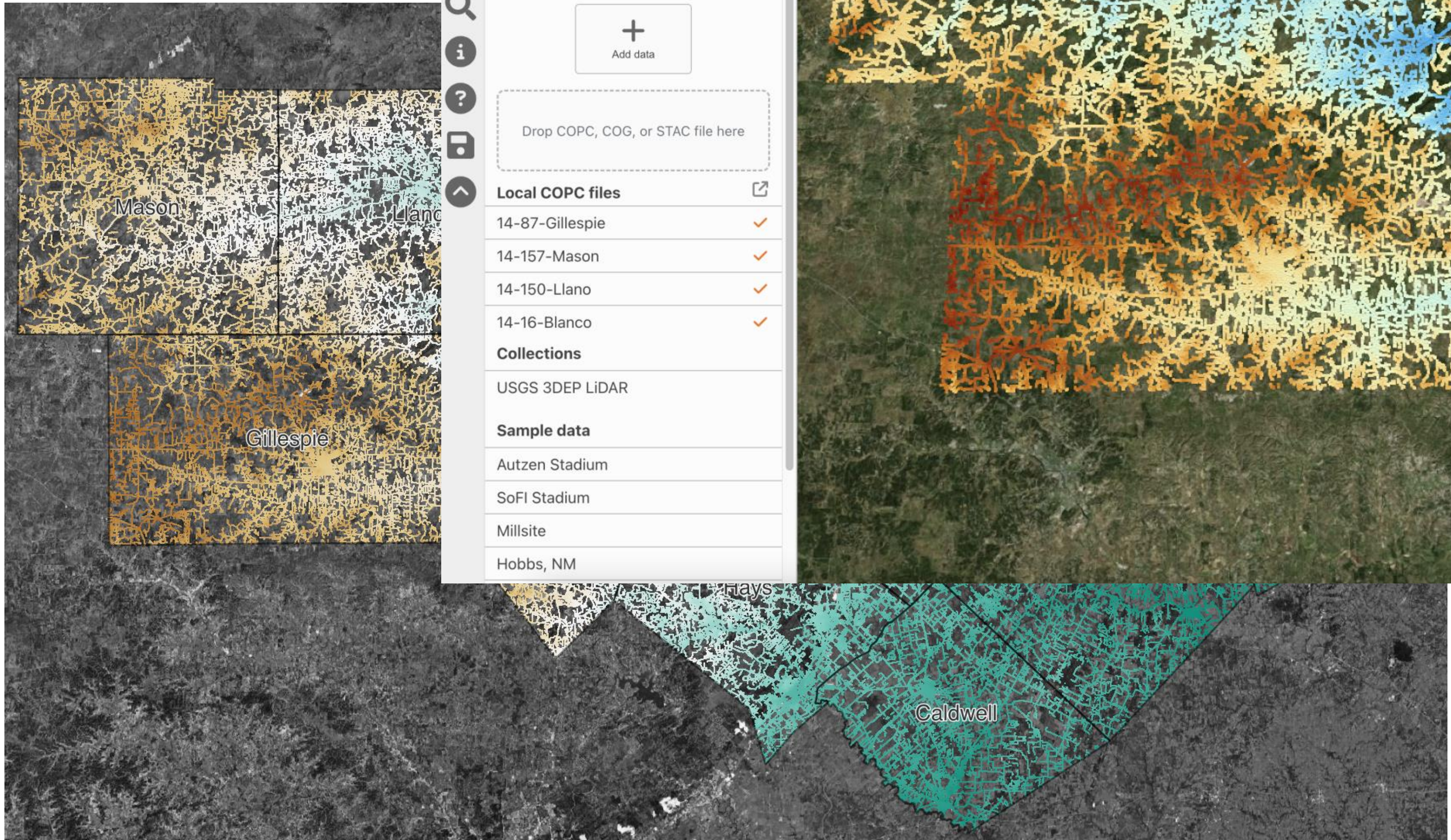


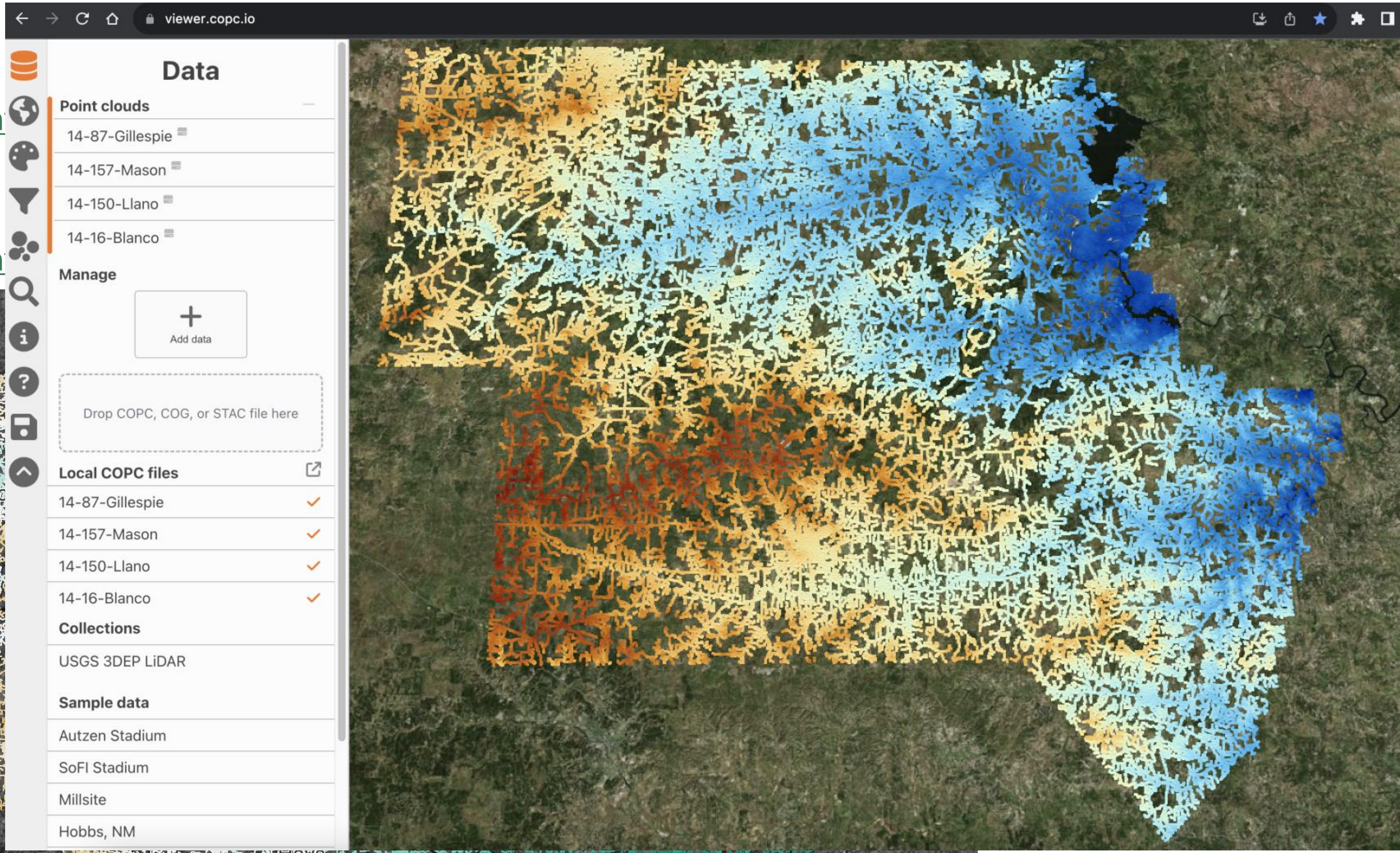


OAK RIDGE National Laboratory

Results: road lidar points for the Austin District

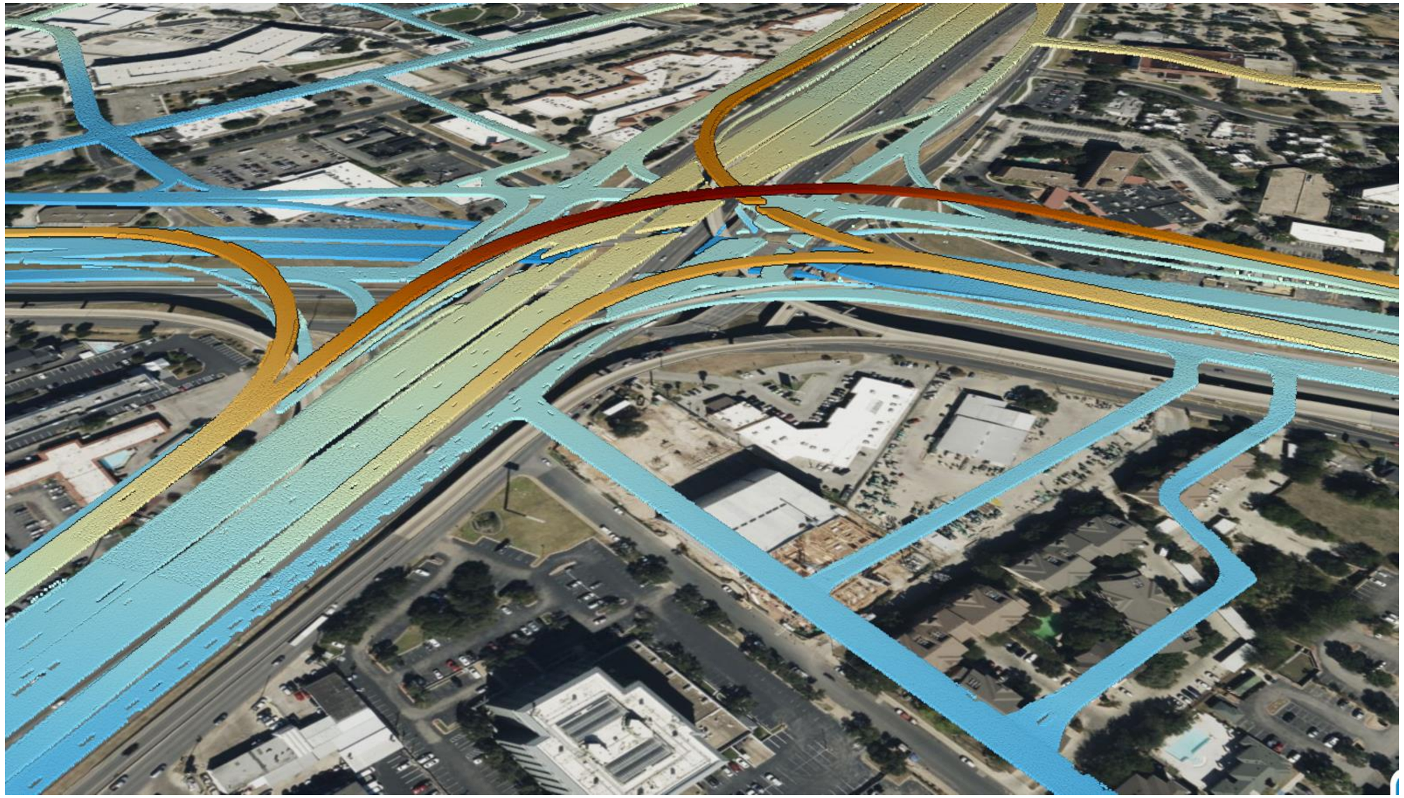
- By counties:
  - https://web.corral.tacc.utexas.edu/n
- By maintenance sections:
  - https://web.corral.tacc.utexas.edu/n





Results: road lidar points for the Austin District - copc

- All the road lidar tiles have the copc version
- Loadable and viewable on https://viewer.copc.io/



Summary: Lidar Data Processing Issues

- Incorrect projection information encoding in las
  - "PDAL: readers.las: Global encoding WKT flag not set for point format 6 - 10."
  - Solution
    - Input handling: find WKT info. in metadata
    - Output handling: header info may be carried over to output las. Explicitly override header info.
- Las version 1.0 in some lidar tiles in the LCRA lidar collections is no longer supported
  - "laspy.errors.FileVersionNotSupported: 1.0"
  - Solution: upgrade to version 1.1 in writing the road lidar tile
- Misclassification of road surface points
  - Current filtering rule: ground (2), bridge (17), and culvert (13 and 14). Some bridge areas may be classified as other classes or “Other”

Conclusion and Discussions

- GPU-acceleration made the computation of the Austin District feasible
- Scaling to all 25 TxDOT districts using the same computing environment is feasible (~200 hours)
- Road tiles can be contributed back to each participating lidar data collection
- Road tiles are published
  - 3D road shapes are proprietary data
- Next steps
  - Processing for all the districts
  - Road surface geometry fitting using road tiles
  - …

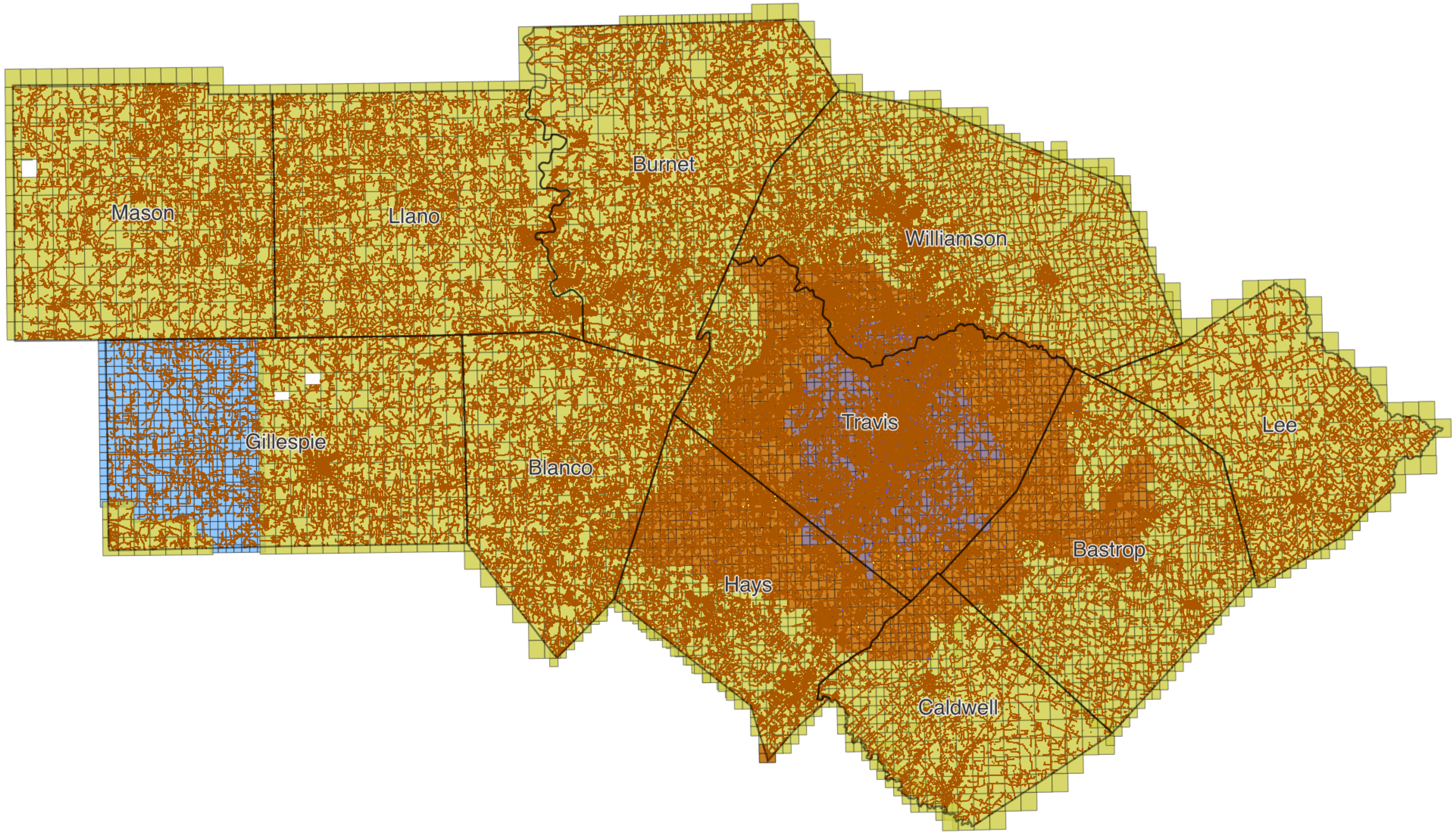
Acknowledgements

This work is sponsored in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725. The research used resources from the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility, as well as the CADES Compute and Data Resources and the Oak-Ridge Research Cloud (ORC) at Oak Ridge National Laboratory.

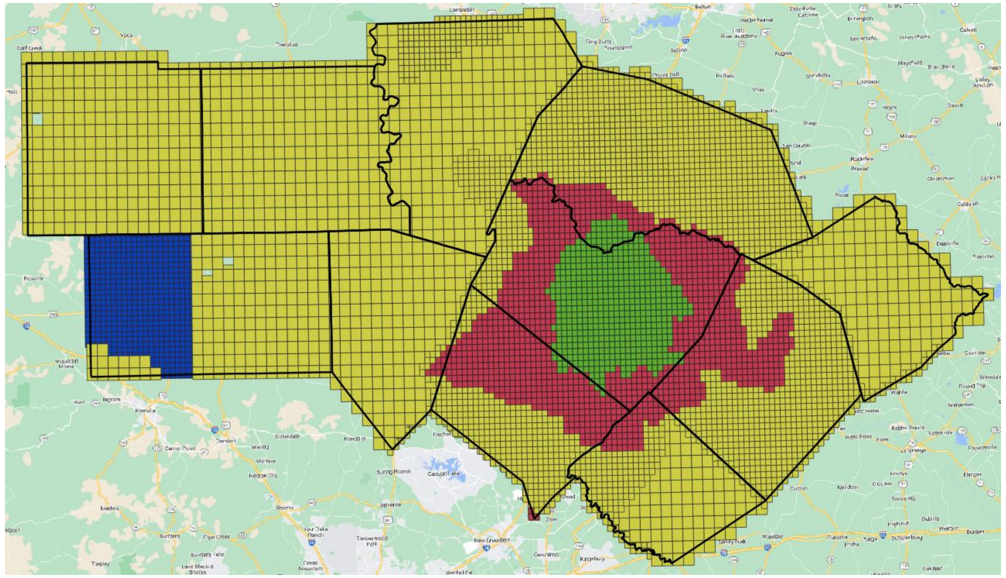
Disclaimer

Backup slides

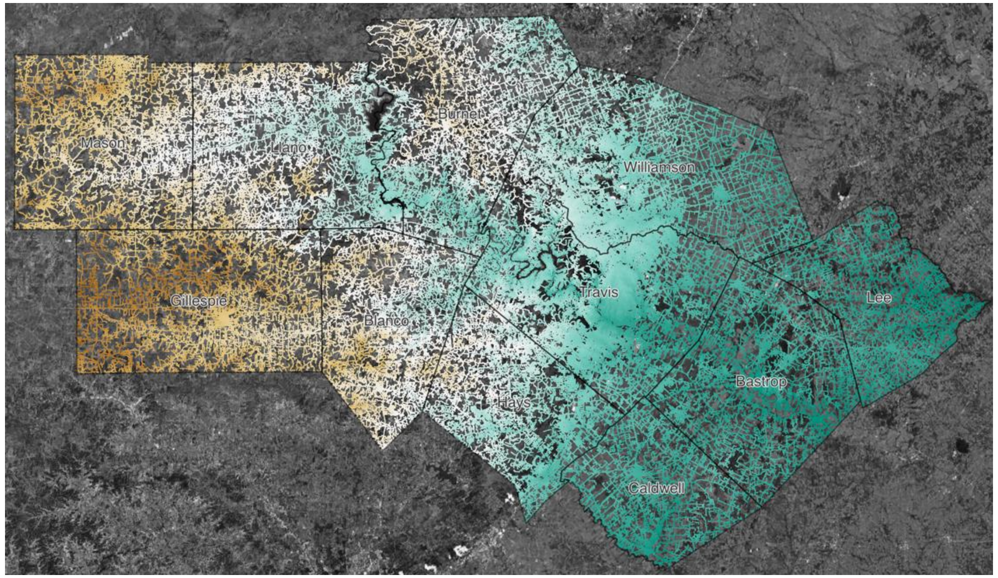
Road lidar points for Austin District overlaid with lidar tile extents



Oak Ridge National Laboratory

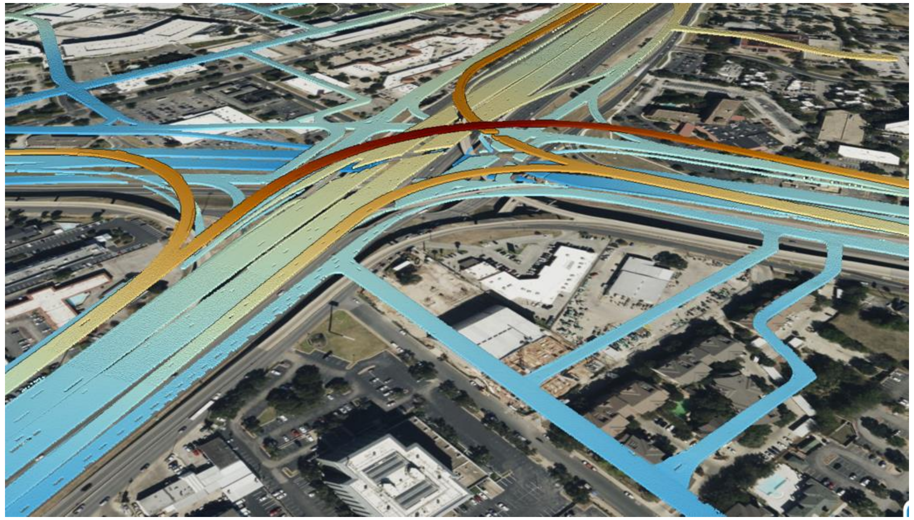


(a) Lidar data coverage



(b) Road lidar data product

(c) Road lidar details



(c) 3-D view of road lidar