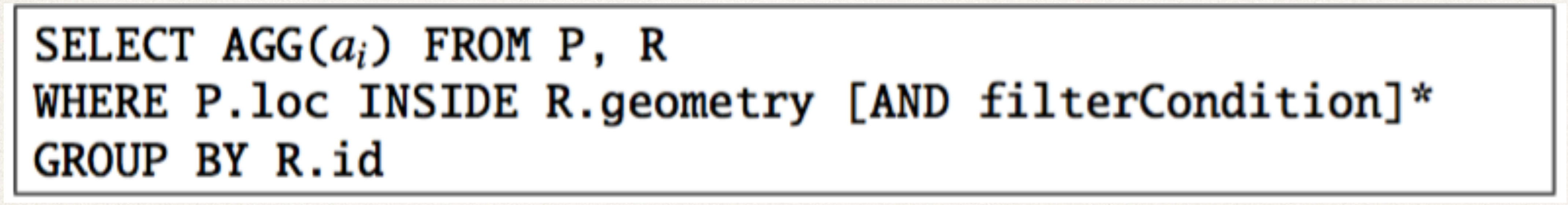
GPU Rasterization for Real-Time Spatial Aggregation over Arbitrary Polygons

Outline

- Problem Statement and Challenges
- Graphics Pipeline
- Raster Join
- Bounded Raster Join
- Accurate Raster Join
- Experimental Evaluation

Problem Statement



```
SELECT AGG(a_i) FROM P, R
WHERE P.loc INSIDE R.geometry [AND filterCondition]*
GROUP BY R.id
```

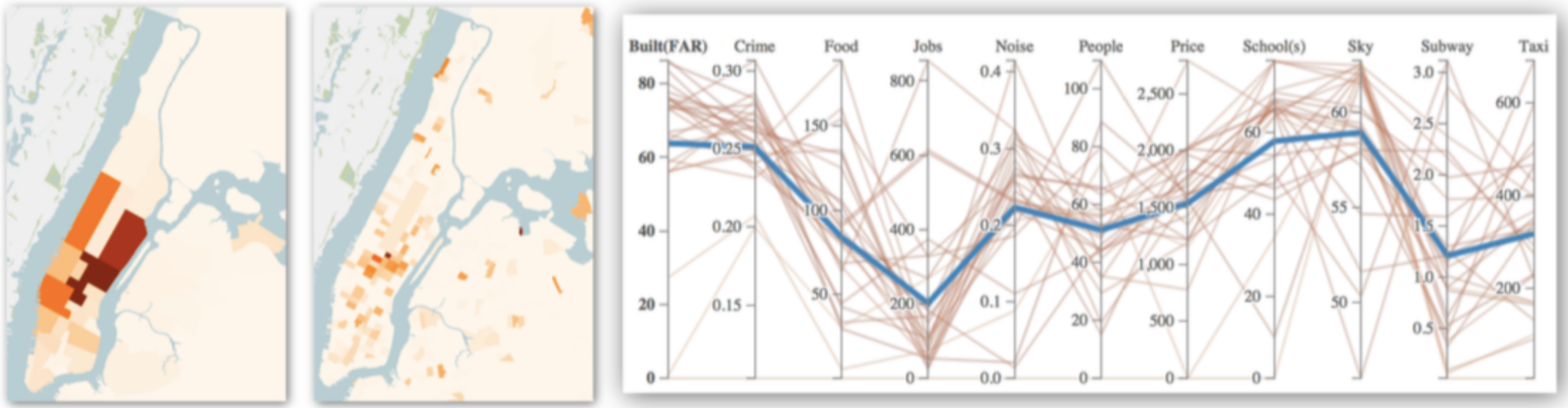
- A set of points: P(location, attribute 1, attribute 2, …)
- A set of regions: R(id, geometry), can be arbitrary polygon
- Aggregation: AGG (include the count of points and average of the attribute a_i)

Application

- **Distribution of NYC taxi pick- ups (*data set*) in the month of June 2012 using a heat map over two resolutions: neighborhoods and census tracts.**

```
SELECT AGG(aᵢ) FROM P, R
WHERE P.loc INSIDE R.geometry [AND filterCondition]*
GROUP BY R.id
```

Setting *P* as pickup locations of the taxi data; R as either neighborhood (a) or census tract (b) polygons; AGG as COUNT(*); and filtered on time (June 2012).



(a) (b) (c)

Figure 1: Exploring urban data sets using Urbane: (a) visualizing data distribution per neighborhood, (b) visualizing data distribution per census tract, (c) comparing data over different neighborhoods. The blue line denotes the NYC average for these data.

Application

Interactive Urban Planning

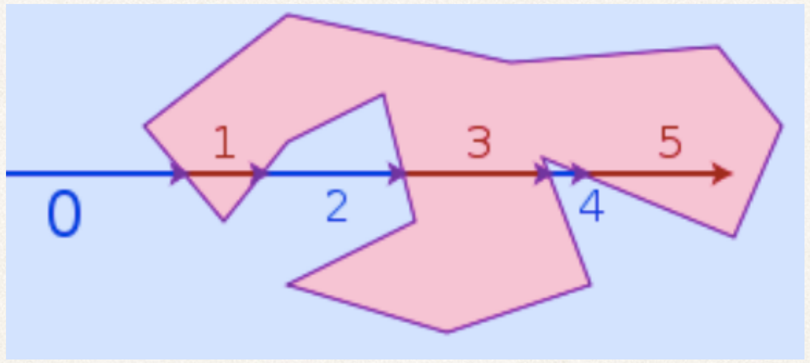
- **Change polygon boundaries or various laws (e.g., new construction rules, building policies for different building types), view how the other aspects of the city (represented by urban data sets) vary with the new change, and inspect the aggregation of the data sets until a particular configuration is satisfied.**

- **Place new resources (e.g., bus stops, police stations), and inspect the coverage with respect to different urban data sets. The coverage is commonly computed by using a restricted Voronoi diagram [1] to associate each resource with a polygonal region, and then aggregating the urban data over these polygons.**

- [1]M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edition, 2008.

Challenges

- **Find which polygons contain each point - spatial join**
  - The join is first solved using approximations (e.g., minimum bounding rectangles - MBRs, bounding boxes), this returns a set of possible matches.
  - False matches are removed by comparing the geometries (e.g., Point-in-polygon (PIP) tests), which is a computationally expensive task.



- **Finally, the aggregates are computed over the materialized join results.**

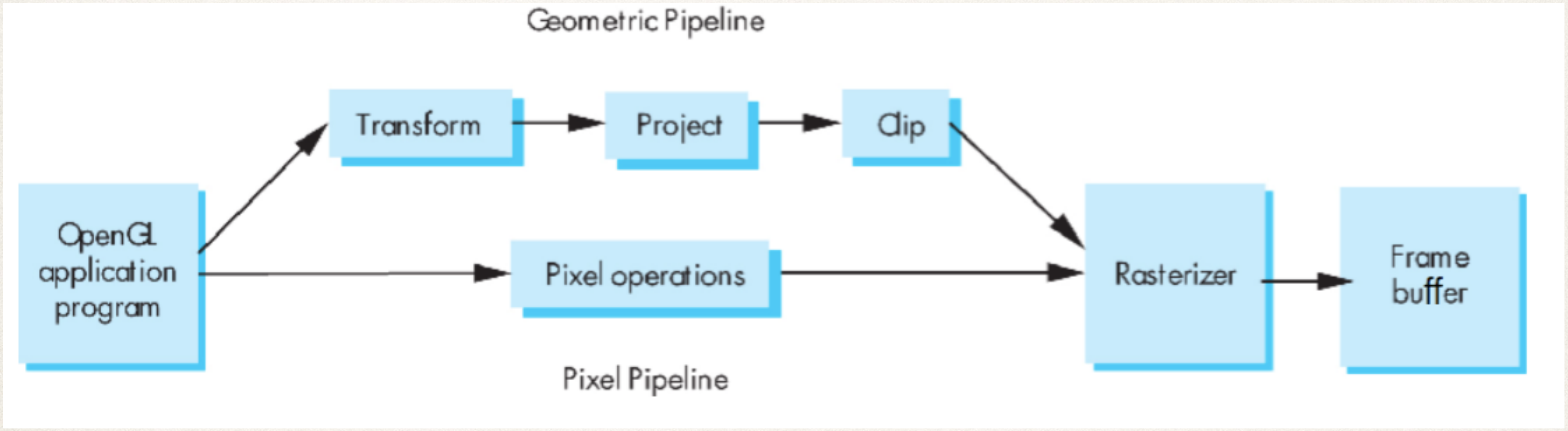
GPU Rasterization-based Methods

- *Insight 1:* It is not necessary to explicitly materialize the result of the spatial join since the final output of the query is simply the aggregate value;
- Advantage: No memory needs to be allocated for storing join results.

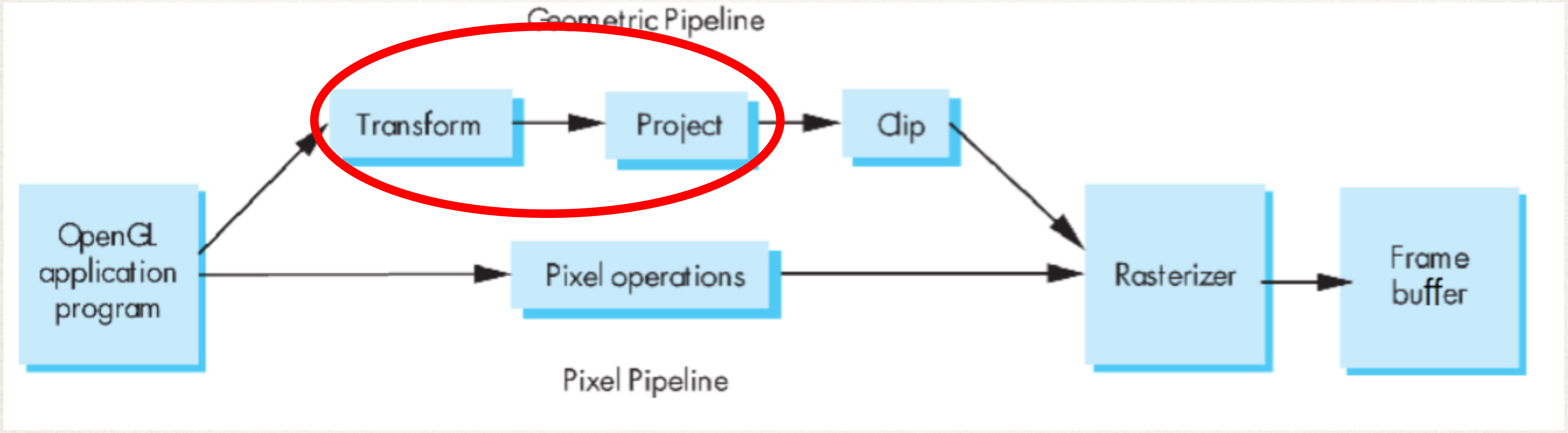
- *Insight 2:* A spatial join between two data sets can be considered as "drawing" the two data sets on the same canvas, and then examining their intersections;
- Advantage: Exploit the *rasterization* operation, which is highly optimized for GPU.

- *Insight 3:* When working with visualizations, small errors can be tolerated if they cannot be perceived by the user in the visual representation.
- Advantage: Enables a mechanism to completely avoid the costly point-in-polygon tests.

Graphics Pipeline

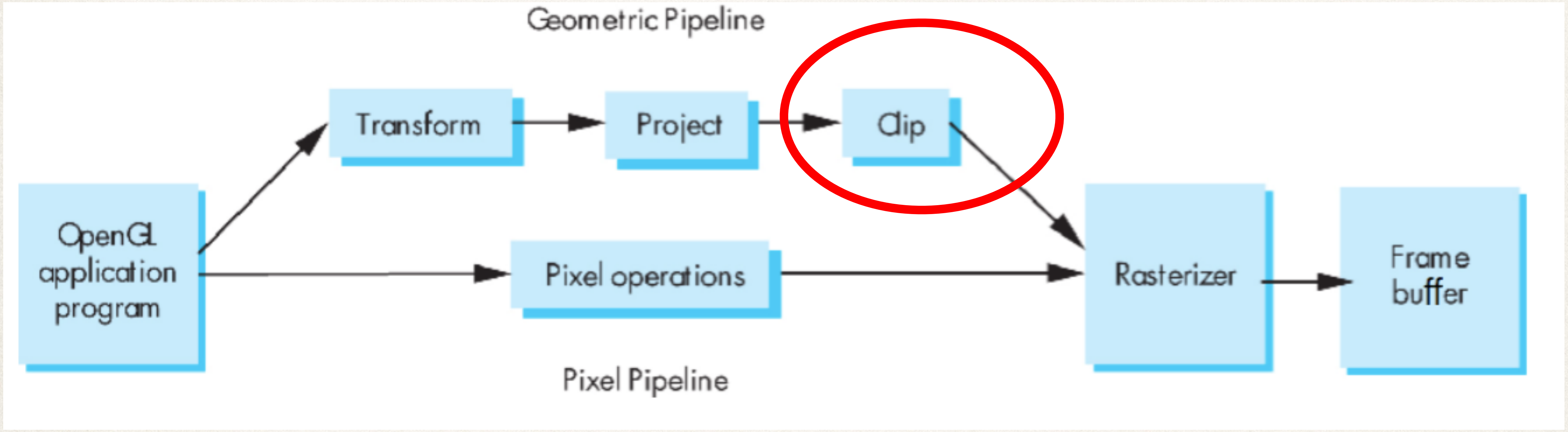


Graphics Pipeline



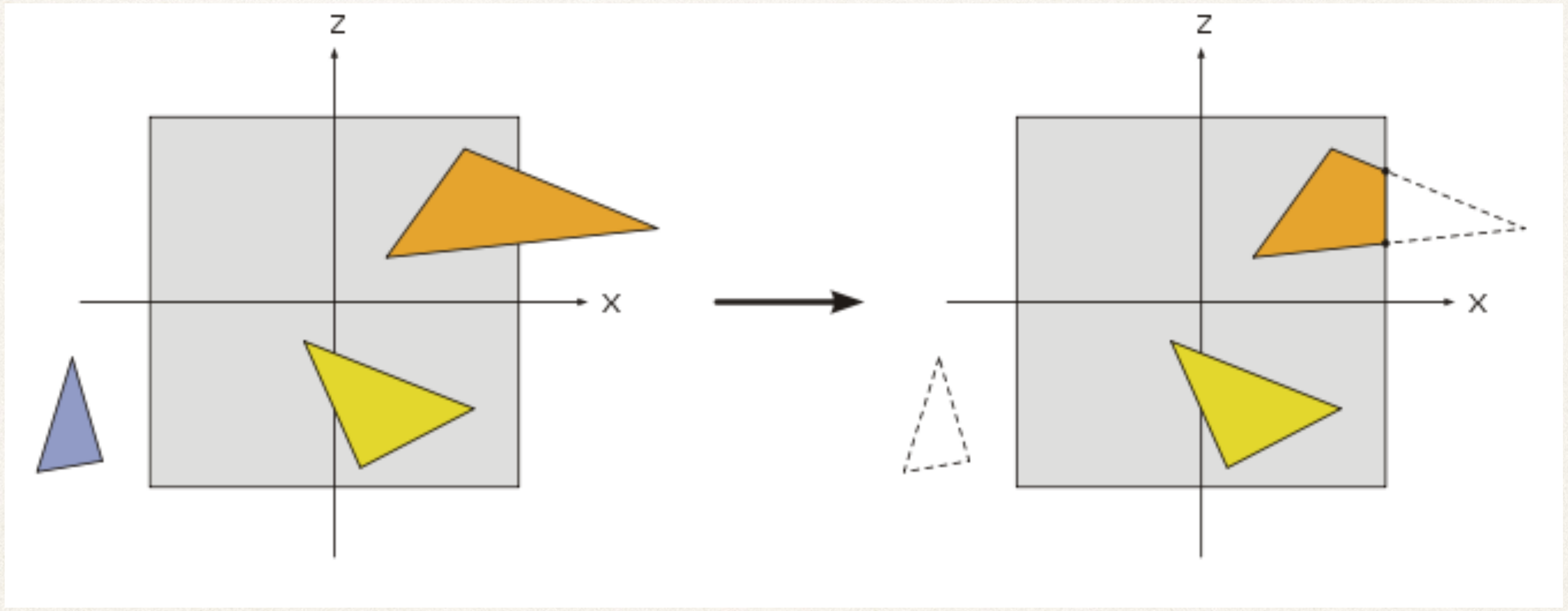
- The coordinates of all the points or vertices (of the polygons) that compose the scene are transformed into a common world coordinate system, and then projected onto the screen space.

Graphics Pipeline

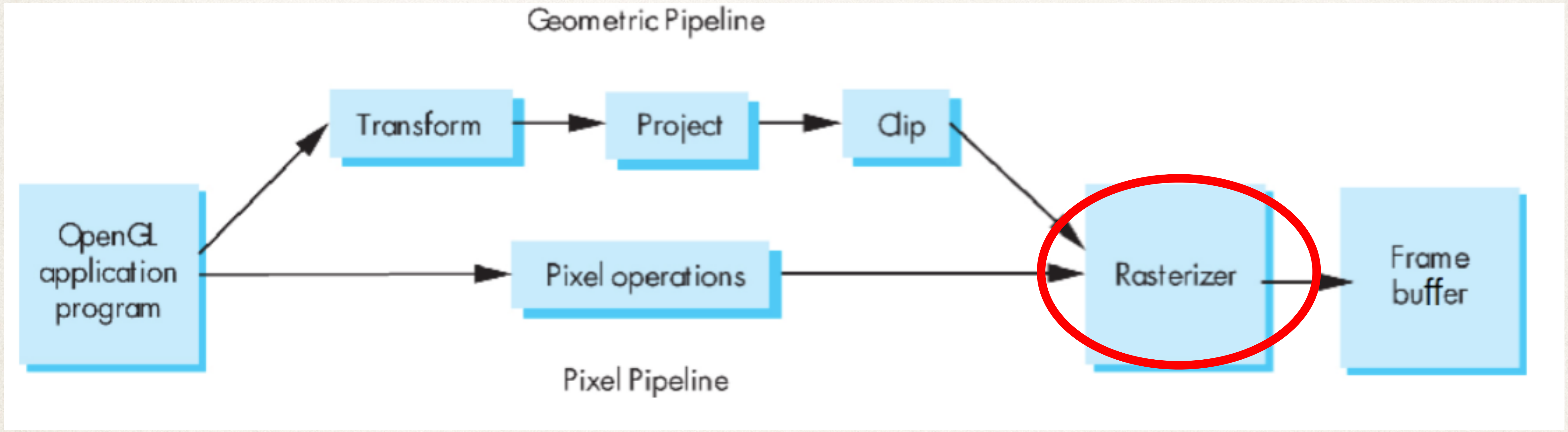


- Next, polygons or points falling outside the screen (also called viewport) are discarded, while those partially outside are clipped.

Graphics Pipeline



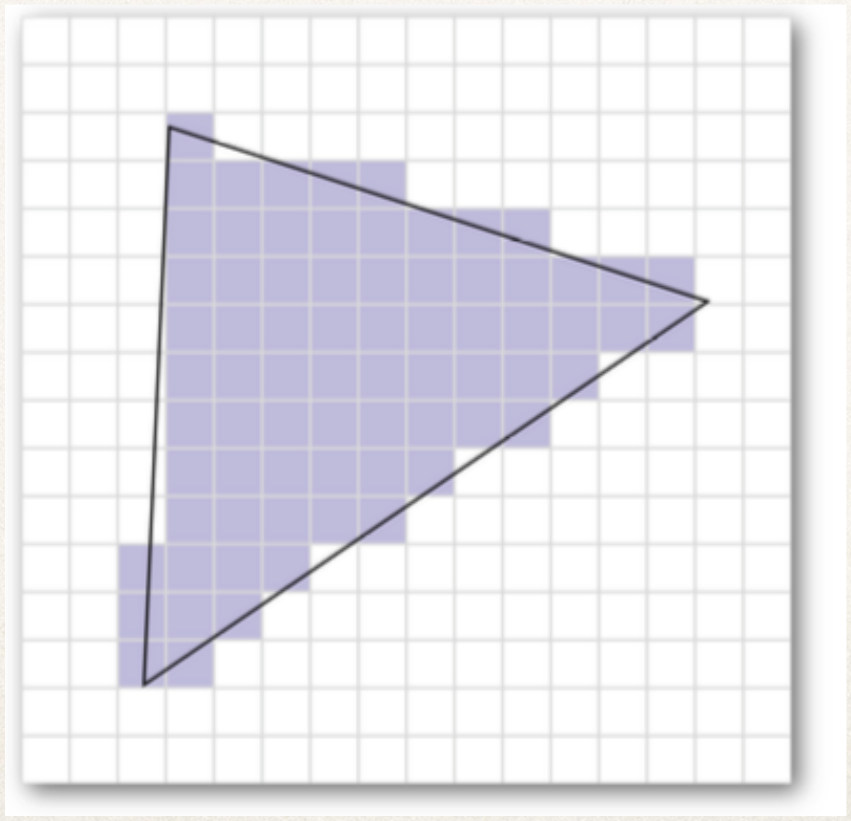
Graphics Pipeline



Graphics Pipeline

- *Rasterization* converts each triangle in the screen space into a collection of *fragments*. A *fragment* can be considered as the data corresponding to a pixel.

- Each fragment is appropriately colored and displayed onto the screen.



Raster Join

- Step 1. Draw points
  - Use the color channels of a pixel for storing the count of points falling in that pixel. (e.g., the red channel of the pixel is incremented by 1).

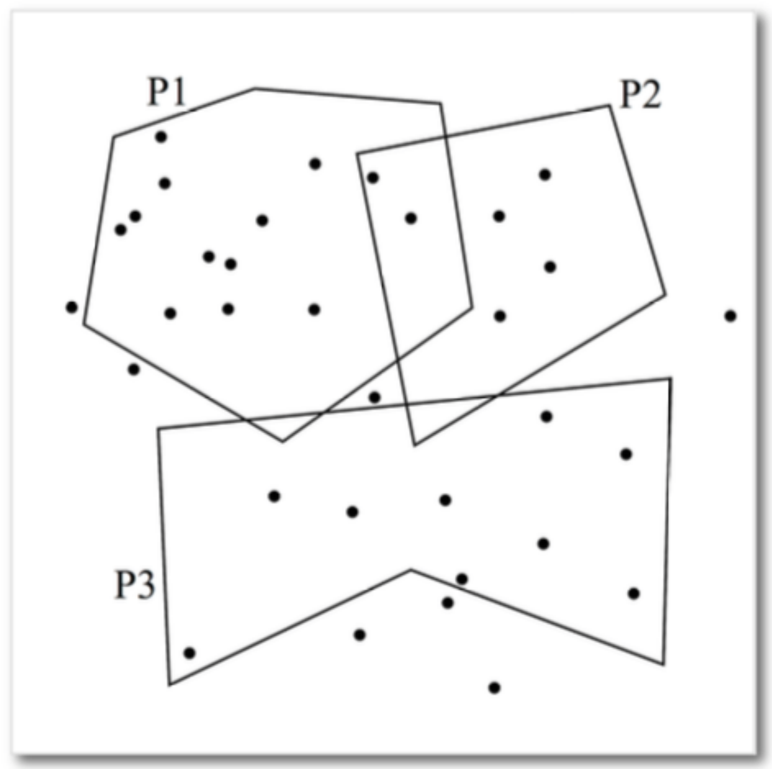
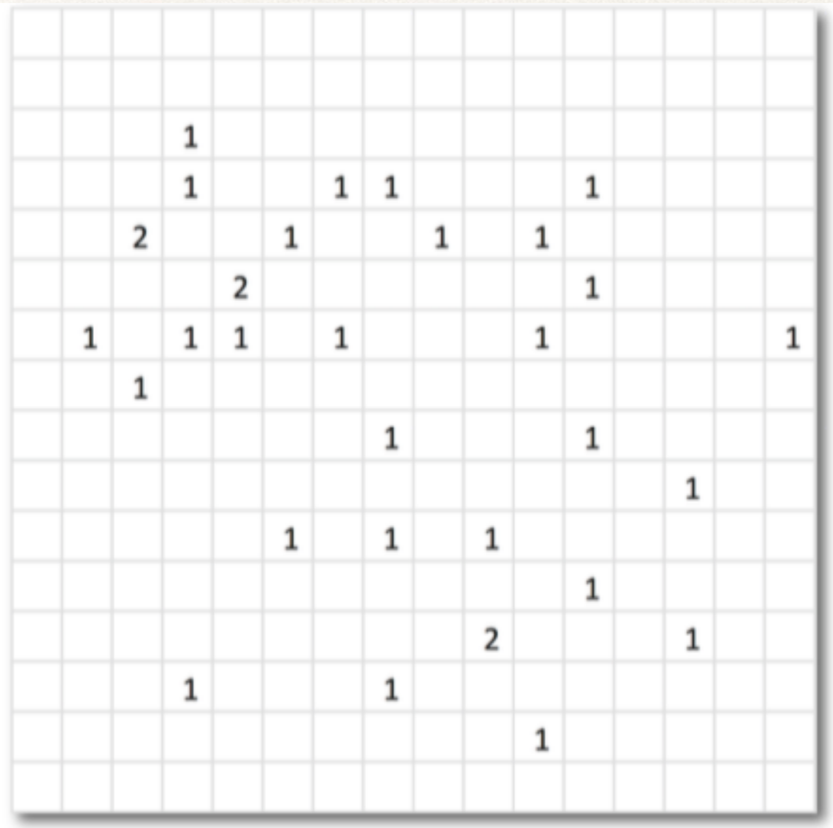


Figure 3: Example input.

A
0
0
0



(a)

Raster Join

Step 2. Draw polygons

- Triangulated -> Transformed -> Rasterization.
- When processing a fragment corresponding to a polygon with ID *i*, the count of points corresponding to this pixel is added to the result *A*[*i*] corresponding to polygon *i*.

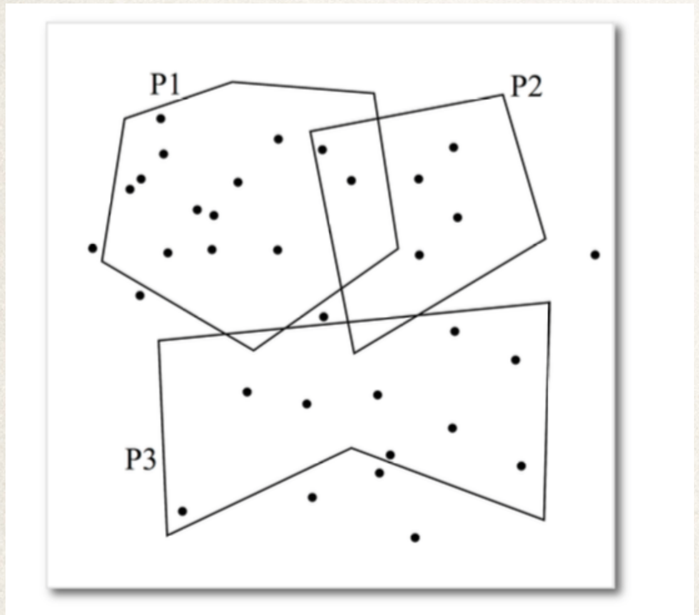
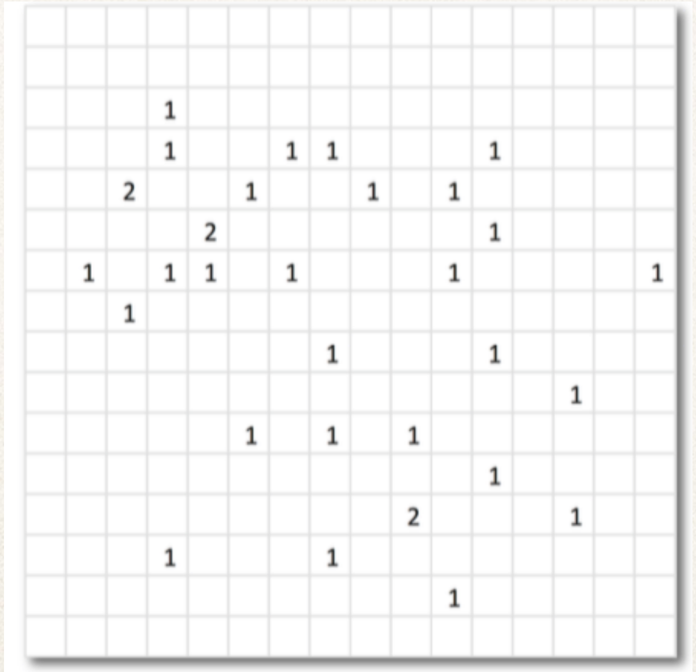
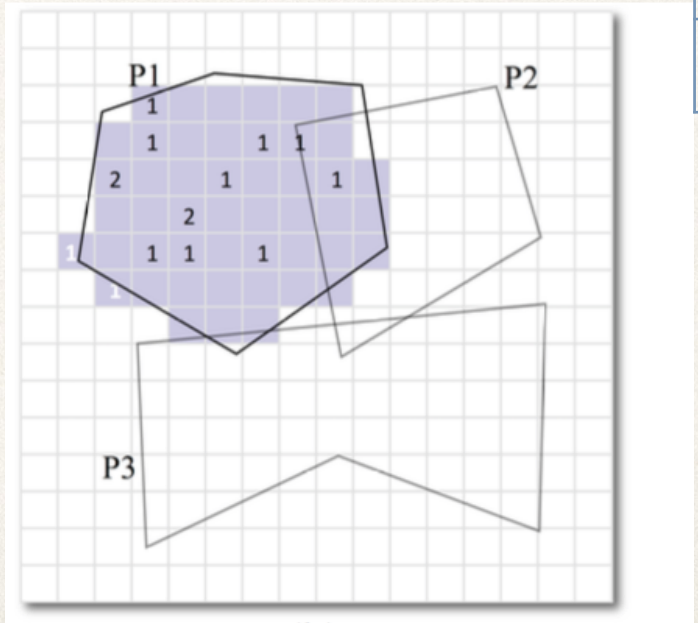


Figure 3: Example input.


(a)

A
0
0
0



(b)

A
15
6
11

Bounded Raster Join

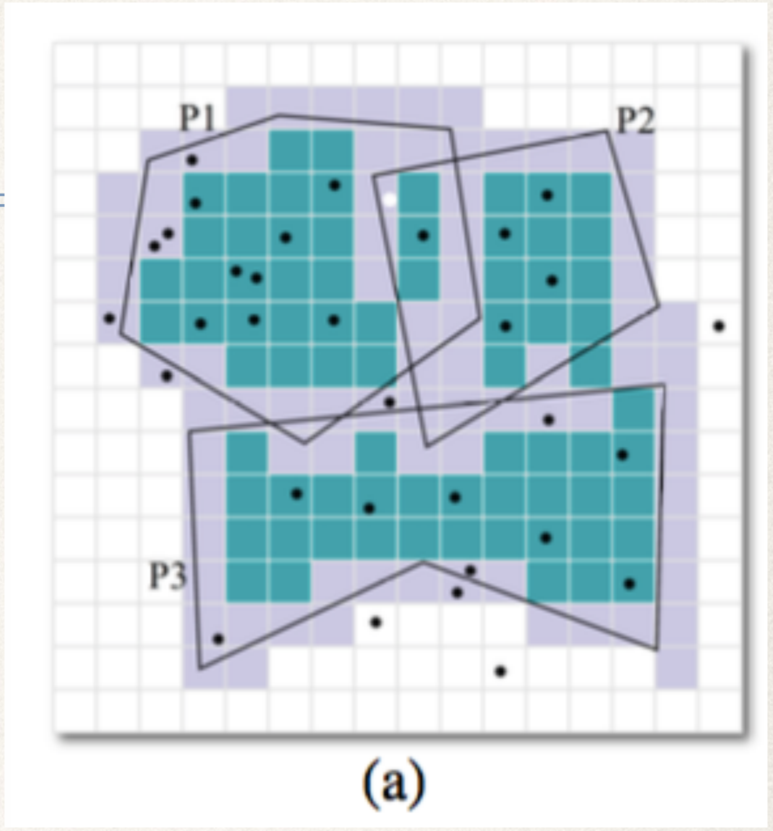
- Raster join introduces some false positive and false negatives points.

- With an appropriately high resolution, we can converge to the actual aggregate result.

- Give the ϵ -bound (error bound, the error is measured using Hausdorff distance
-): the required resolution is $w' \times h' = \frac{w}{\epsilon'} \times \frac{h}{\epsilon'}$, w X h are the dimensions of the bounding box of the polygon data set, $\epsilon' = \frac{\epsilon}{\sqrt{2}}$.

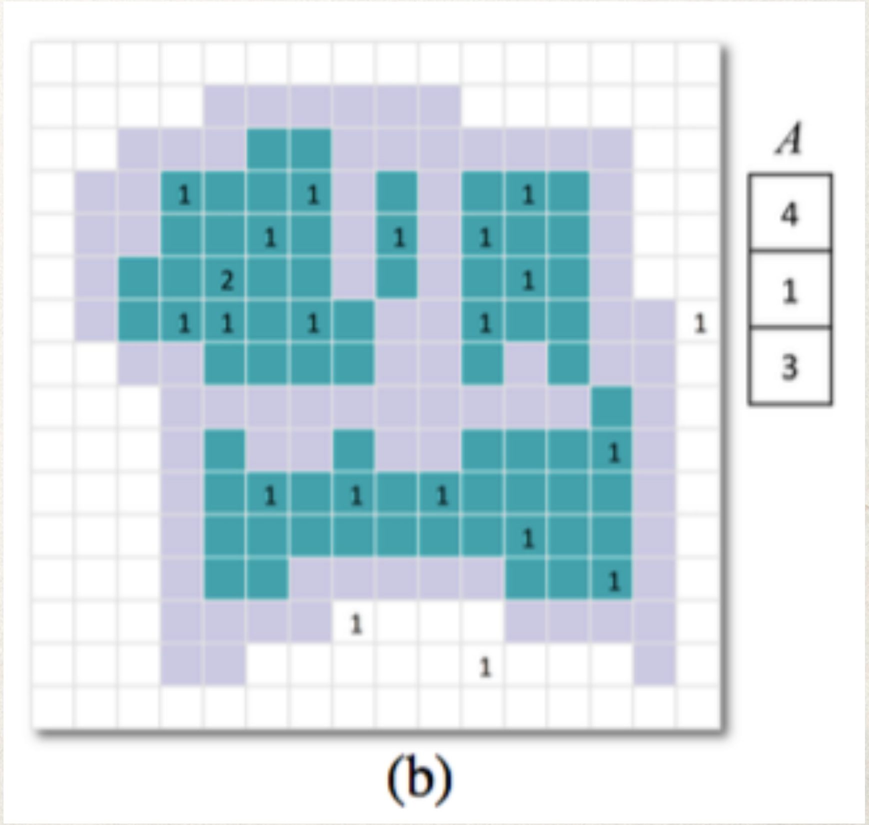
Accurate Raster Join

- Step1. Draw the outline of all the polygons
  - Assign a predetermined color to the fragments corresponding to the boundaries of the polygons. At the end of this step, only pixels on the boundary will have a color.



(a)

- Step2. Draw points.
  - Points falls into a boundary pixel: perform PIP test; A: stores the partial query result corresponding to data points that fall on the boundary of the polygons;
  - Points inside the polygons: stores the count of points fall into each of its pixels.

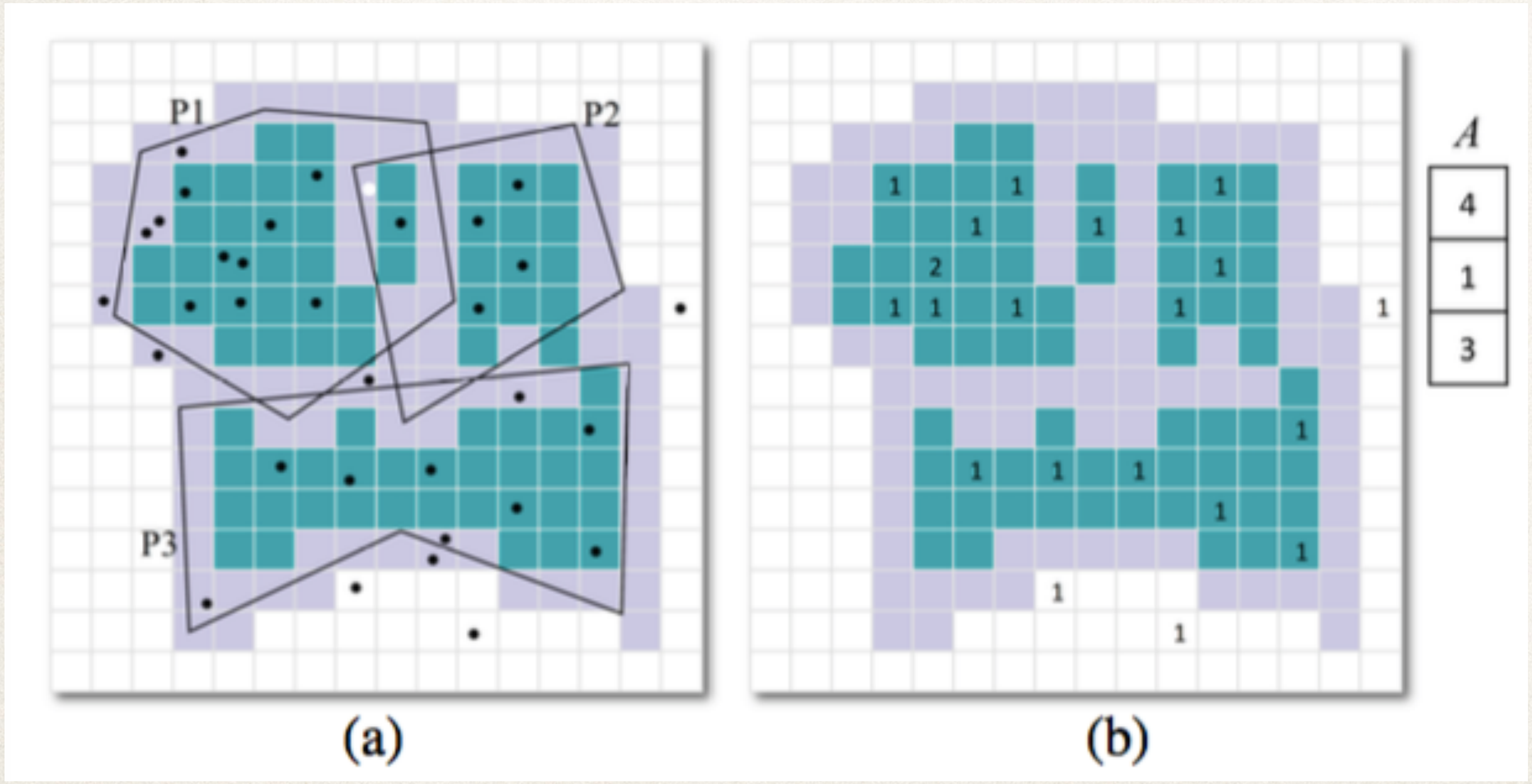


(b)

Accurate Raster Join

Step3. Render polygons

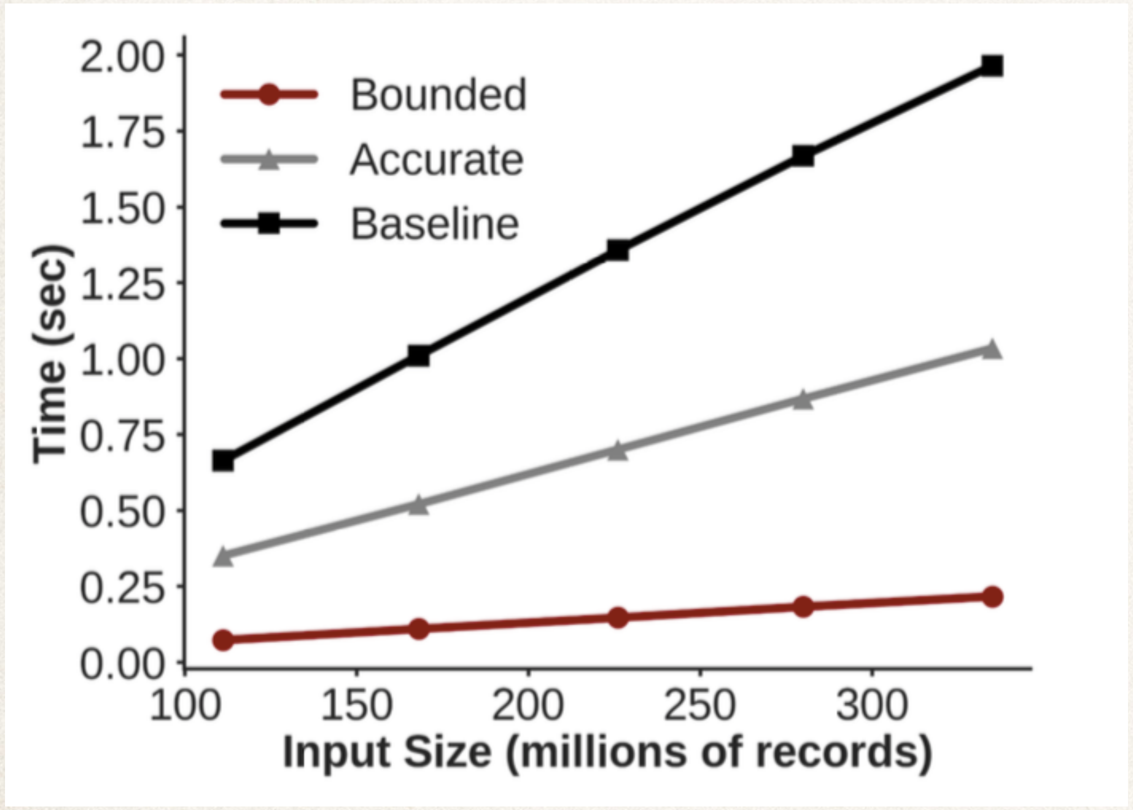
- Fragment on a boundary pixel: discard;
- Fragment inside polygon: A[i] = A[i] + point number of pixel(x, y), pixel (x, y) belong to polygon i.



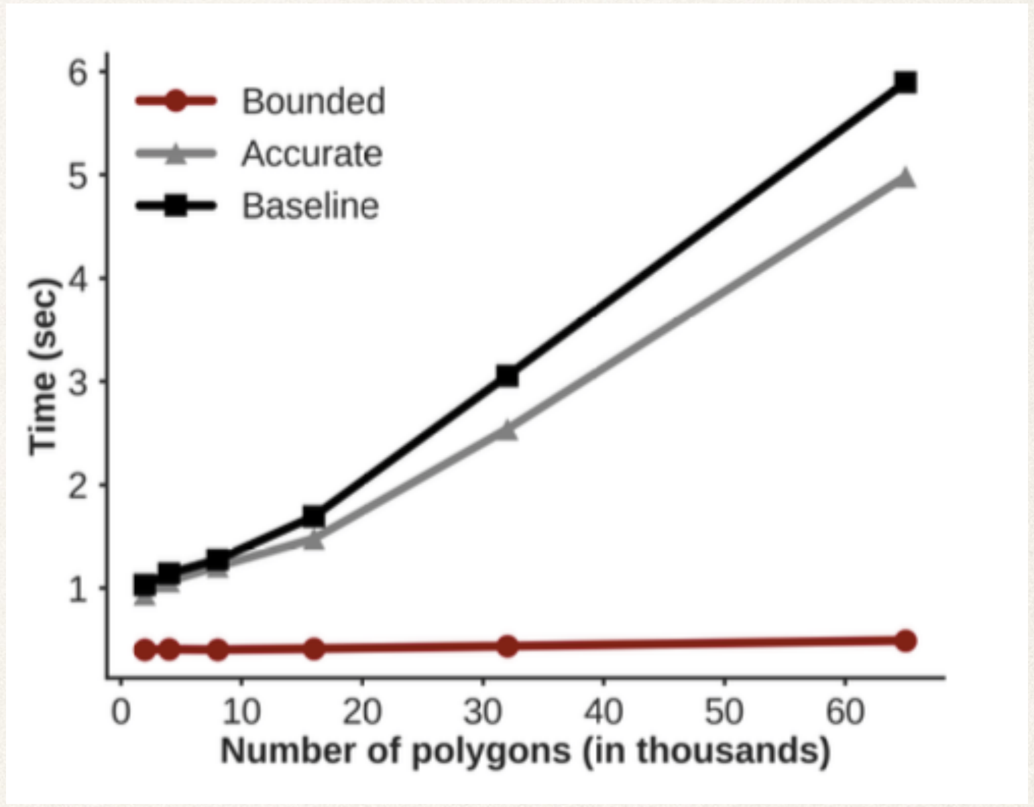
A
13
6
9

Experimental Evaluation

- **Query time analysis**



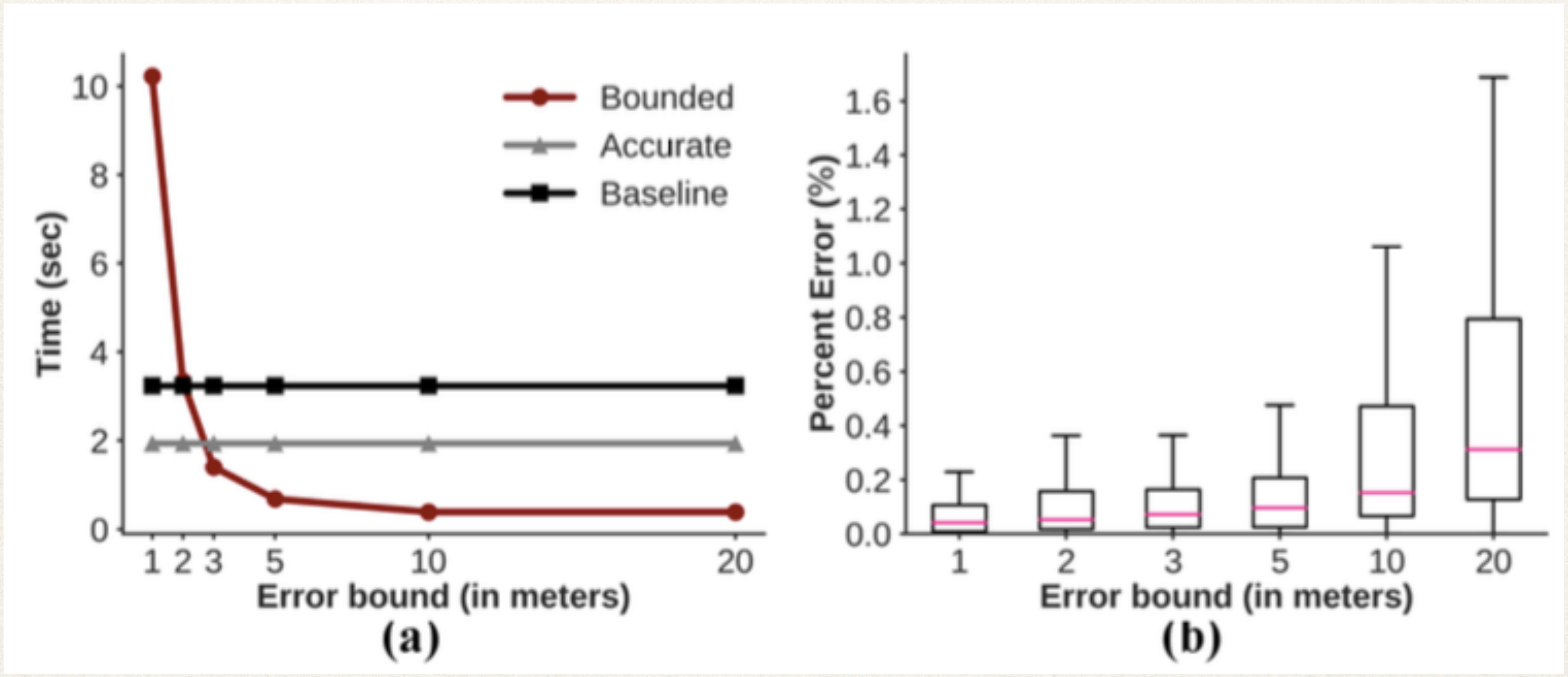
Scaling with points for Taxi▷Neighborhood. Bounded Raster Join has the best scalability as it eliminates all PIP tests. Accurate Raster Join performs fewer PIP tests than the Baseline.



Scaling with polygons. Note that increasing the number of polygons has almost no effect on Bounded Raster Join.

Experimental Evaluation

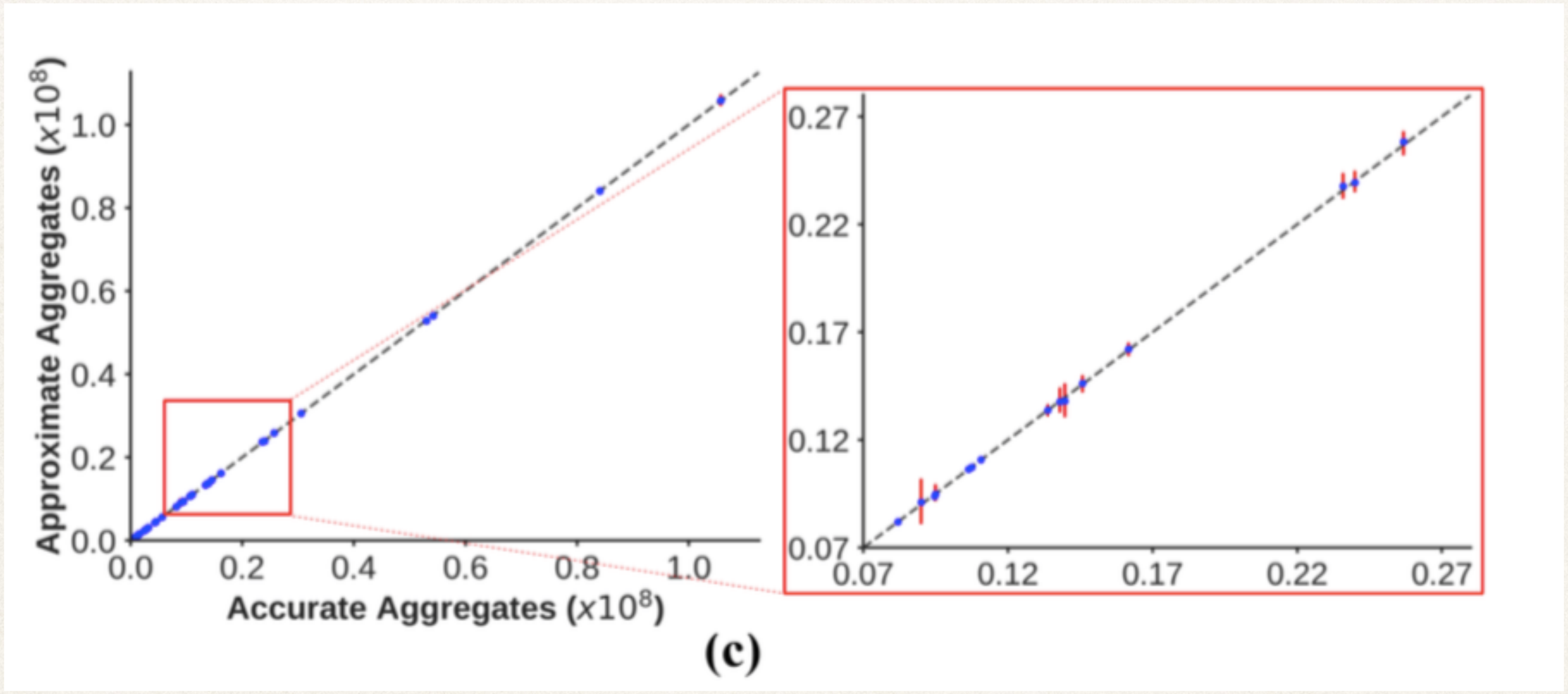
- **Accuracy analysis for bounded raster join**



(a) Accuracy-time trade-off. (b) Accuracy-ε-bound trade-off. The box plot shows the distribution of the percent error over the polygons for different ε-bounds.

Experimental Evaluation

- **Accuracy analysis for bounded raster join**



The scatter plot shows, for each polygon, the accurate value against the approximate value for ε = 20 m. The red error bars indicate the expected result intervals (see the enlarged highlighted region).

✤ Thank you!