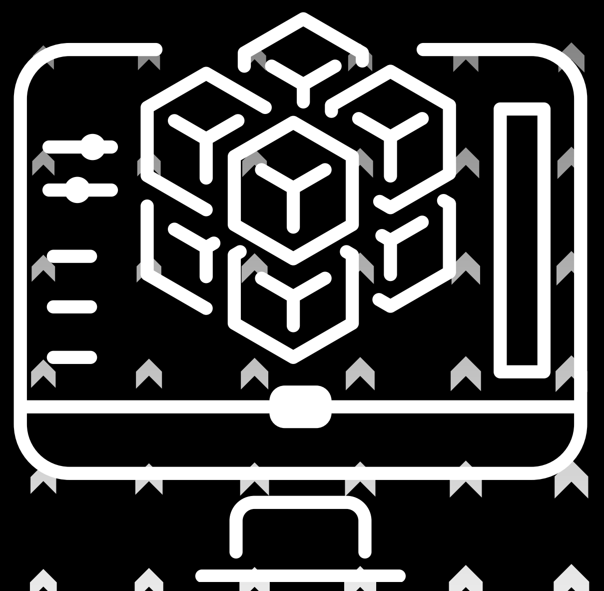
Visibility in 3D Terrain

Ingrid-Adriana Corobană

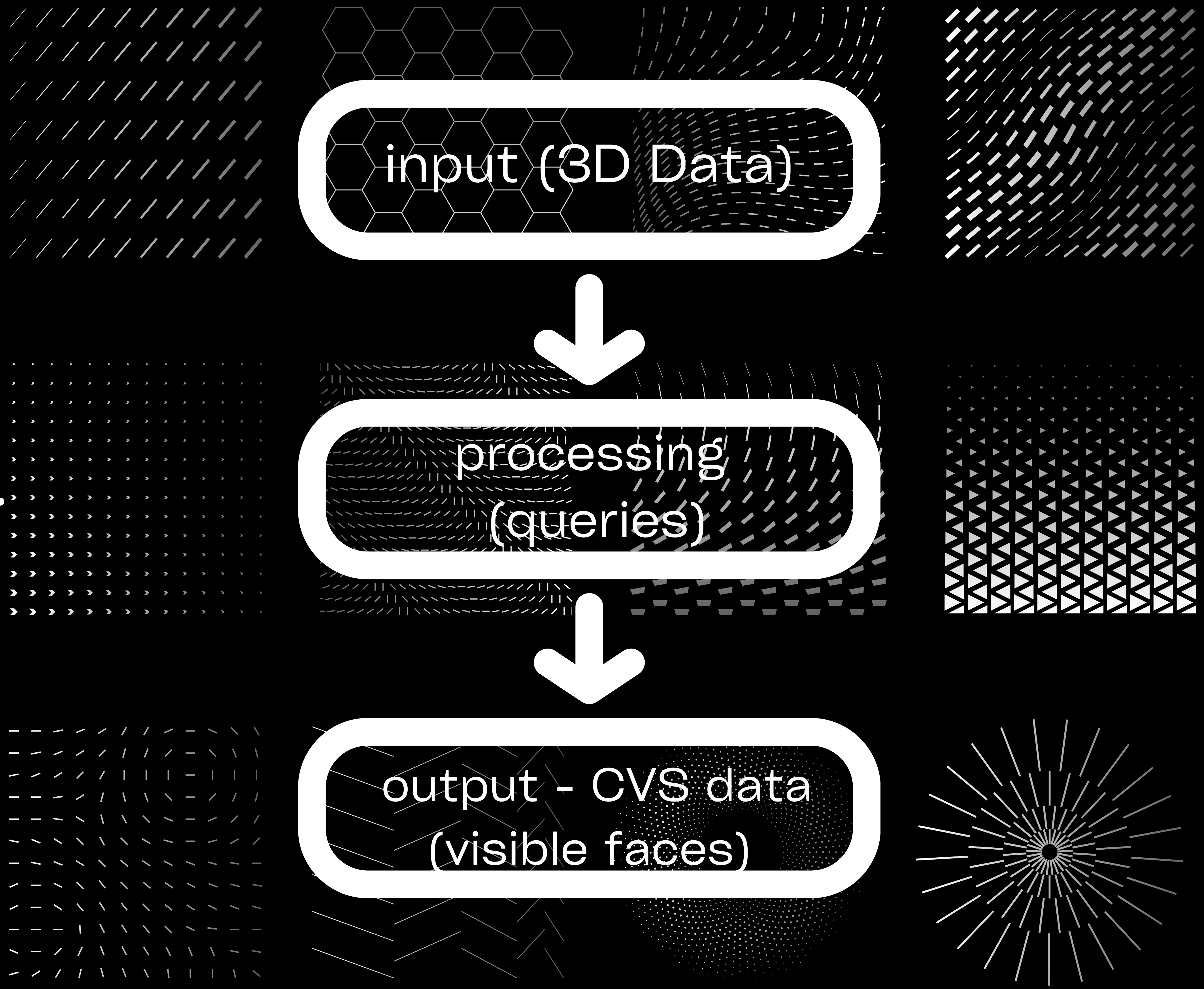
What Problem Are We Solving?



- Calculate visible surfaces of 3D boxes from a viewer's perspective.
- Address spatial data challenges:
  - Geometry calculations
  - Efficient queries
  - Scalability

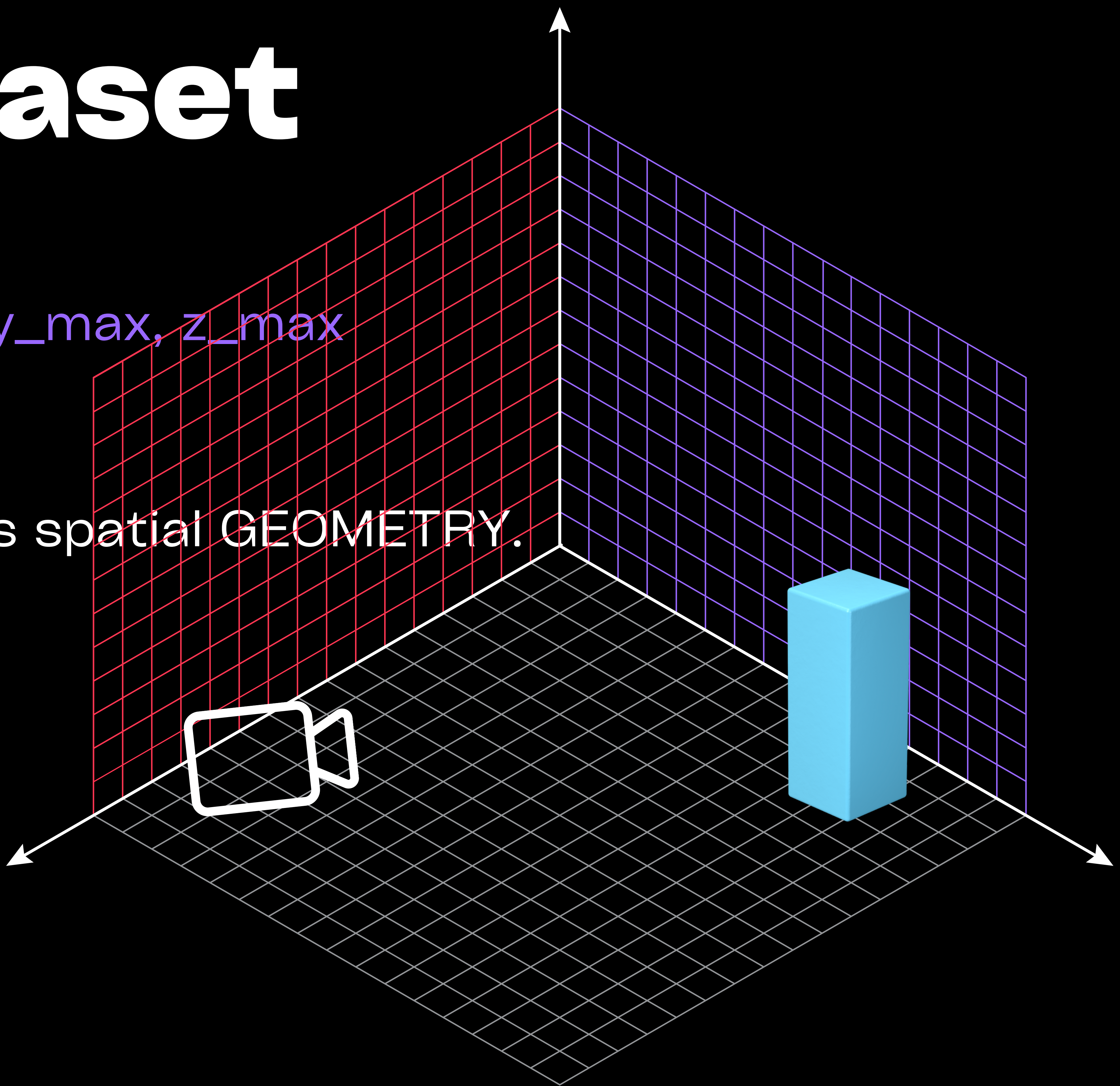
Project Goals

- Efficiently compute visibility of 3D box faces from one POV.
- Ensure results are visualizable and scalable.

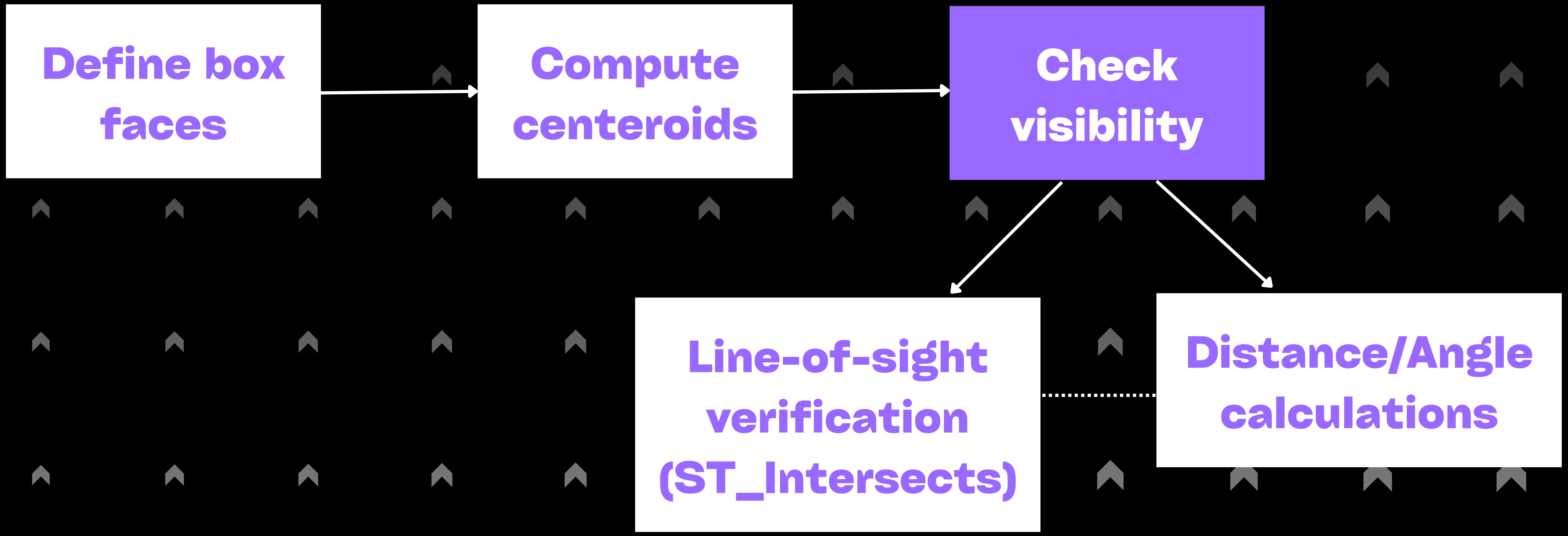


The Dataset

- Boxes with dimensions:
  - x_min, y_min, z_min, x_max, y_max, z_max
- Viewer Position:
  - Coordinates (x, y, z) stored as spatial GEOMETRY.



Algorithm Overview



SQL Techniques and Tools

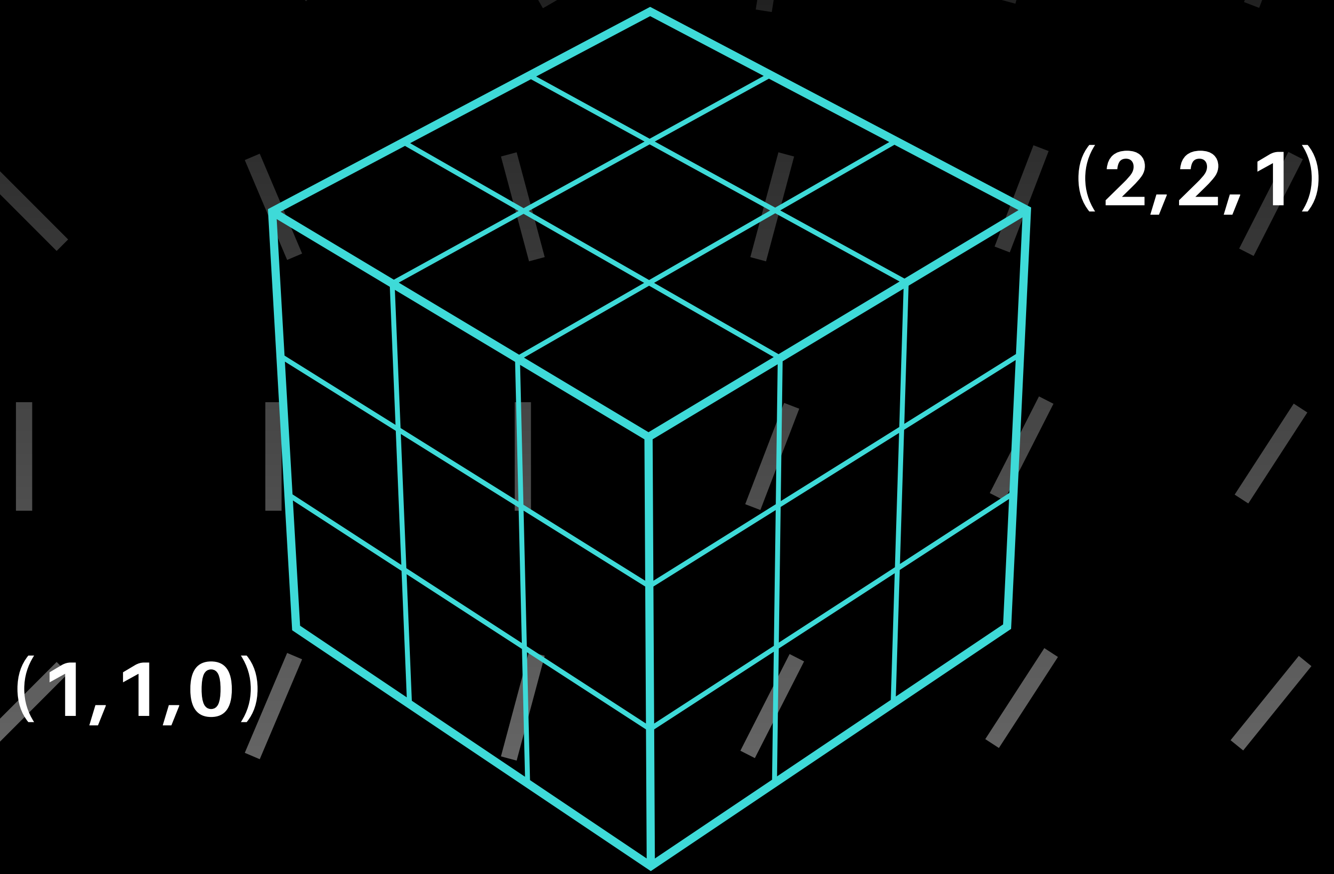
- Recursive CTEs for face generation.
- PostGIS Functions:
  - **ST_MakePolygon**
  - **ST_SetSRID**
  - **ST_Distance**
  - **ST_Intersects**
- Optimization through precomputations.

```sql
WITH RECURSIVE next_box AS (
    -- Base case: Start with the first box
    SELECT
        id AS box_id,
        x_min, y_min, z_min, x_max, y_max, z_max
    FROM boxes
    WHERE id = (SELECT MIN(id) FROM boxes)

    UNION ALL

    -- Recursive case: Move to the next box
    SELECT
        b.id AS box_id,
        b.x_min, b.y_min, b.z_min, b.x_max, b.y_max, b.z_max
    FROM next_box nb
    JOIN boxes b ON b.id > nb.box_id -- Move to the next box
),
```

```sql
SELECT 1
FROM face_angles fa3
WHERE
    fa3.angle > fa.angle -- face has a higher angle
  AND ST_Distance(
        ST_SetSRID(vp.point, 4326),
        ST_SetSRID(
            ST_MakePoint(
               ...
            ),
            4326
        )
    ) <=
    ST_Distance(
        ST_SetSRID(vp.point, 4326),
        ST_SetSRID(
            ST_MakePoint(
                (fa.vertices[1] + fa.vertices[4] +..... / 4
            ),
            4326
        )
    ) -- Obstructing face is closer
    AND ST_Intersects( -- Check line-of-sight intersection
        ST_MakeLine(
```

Visibility Calculation Example: Box 4

- Box Coordinates: **(x_min = 1, y_min = 1, z_min = 0) (x_max = 2, y_max = 2, z_max = 1)**
- Viewer Position: **(x = 3, y = 3, z = 5)**
- Objective: Determine visibility of each face (front, back, left, right, top, bottom).



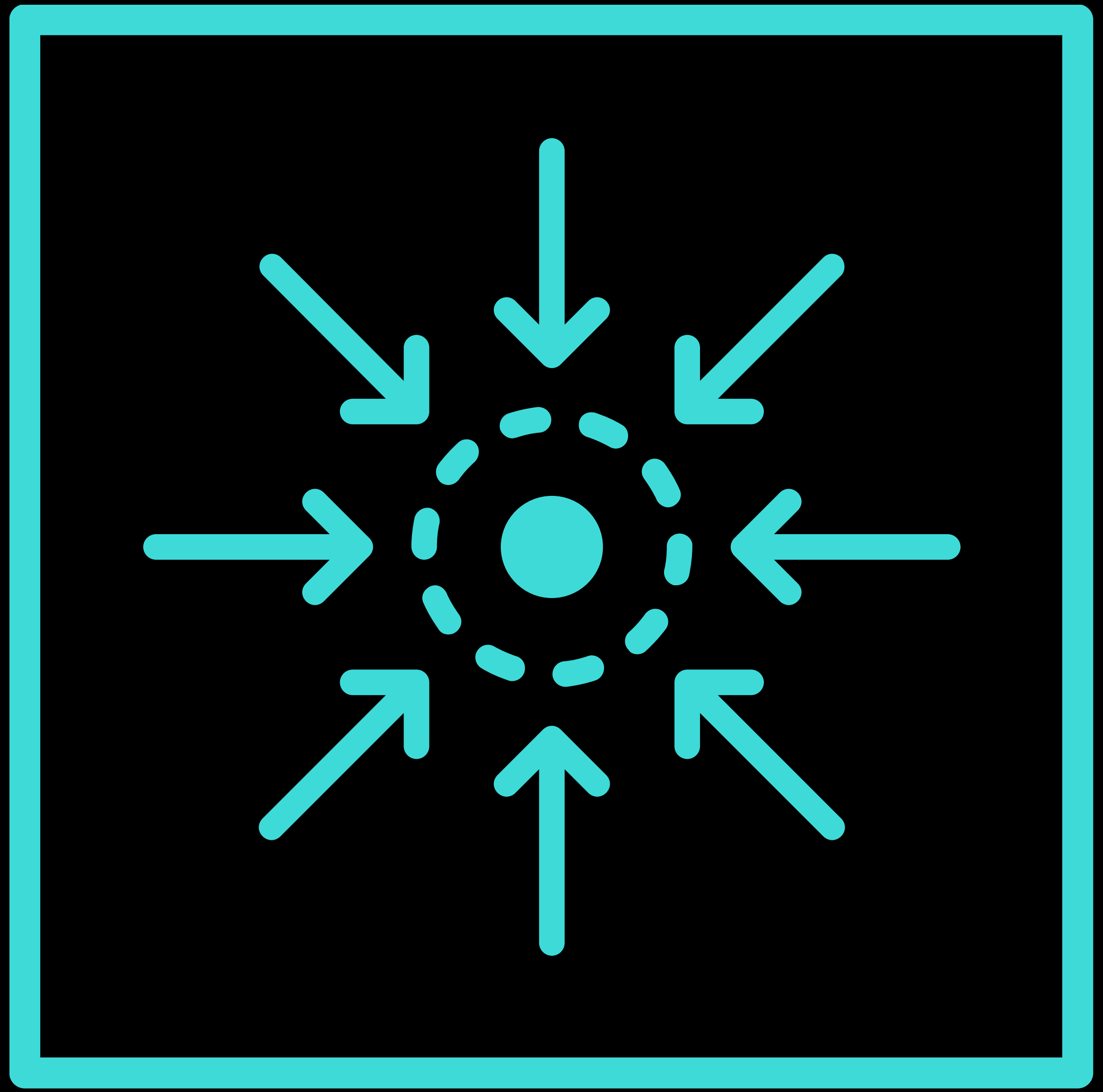
Step 1: Process Input Values for Box 4:

- Box Dimensions: Find coordinates for each face:
  - Front: (1, 2, 0) → (2, 2, 1)
  - Back: (1, 1, 0) → (2, 1, 1)
  - Left: (1, 1, 0) → (1, 2, 1)
  - Right: (2, 1, 0) → (2, 2, 1)
  - Top: (1, 1, 1) → (2, 2, 1)
  - Bottom: (1, 1, 0) → (2, 2, 0)
- Viewer Position: Stored as ST_MakePoint(3, 3, 5).

```sql
CREATE TABLE boxes (
    id SERIAL PRIMARY KEY,
    x_min FLOAT,
    y_min FLOAT,
    z_min FLOAT,
    x_max FLOAT,
    y_max FLOAT,
    z_max FLOAT,
    geom GEOMETRY(POLYGON, 4326)
);


CREATE TABLE viewer_position (
    id SERIAL PRIMARY KEY,
    x FLOAT,
    y FLOAT,
    z FLOAT,
    point GEOMETRY(POINTZ, 4326)
);

DROP TABLE viewer_position;
-- insert example boxes (grid-like terrain)
INSERT INTO boxes (x_min, y_min, z_min, x_max, y_max, z_max)
VALUES
    (0, 0, 0, 1, 1, 1),
    (1, 0, 0, 2, 1, 2),
    (0, 1, 0, 1, 2, 3),
    (1, 1, 0, 2, 2, 1);

-- insert viewer position
INSERT INTO viewer_position (x, y, z, point)
VALUES (3, 3, 5, ST_SetSRID(ST_MakePoint(3, 3, 5), 4326));
```

Step 2: Compute Face Centroids:

- Centroid Calculations:
- Front Face: average coordinates:
  - Centroid = ((1 + 2)/2, (2 + 2)/2, (0 + 1)/2) = (1.5, 2, 0.5)
- Repeat for other faces (e.g., back, left, right, top, bottom).



Visibility Verification Overview

- Goal: Determine which faces of a 3D box are visible from the viewer's position.

- Key Steps:
  a. Calculate distances and angles.
  b. Identify potential obstructions.
  c. Verify line-of-sight intersections.
  d. Eliminate duplicates and finalize visibility.

```
ST_Distance(
    ST_SetSRID(vp.point, 4326),
    ST_SetSRID(
        ST_MakePoint(
            (vertices[1] + vertices[4] + vertices[7] + vertices[10]) / 4,
            (vertices[2] + vertices[5] + vertices[8] + vertices[11]) / 4,
            (vertices[3] + vertices[6] + vertices[9] + vertices[12]) / 4
        ),
        4326
    )
)
```

```
ATAN2(
 ((bf.vertices[3] + bf.vertices[6] + bf.vertices[9] + bf.vertices[12]) / 4) - vp.z,
          ST_Distance(ST_SetSRID(vp.point, 4326), ST_SetSRID(ST_MakePoint(
                     (bf.vertices[1] + bf.vertices[4] + bf.vertices[7] + bf.vertices[10]) / 4,
                     (bf.vertices[2] + bf.vertices[5] + bf.vertices[8] + bf.vertices[11]) / 4,
                     (bf.vertices[3] + bf.vertices[6] + bf.vertices[9] + bf.vertices[12]) / 4
                 ),   4326
               )
            )
        ) AS angle
```

Step 3: Distance and Angle Calculations

Distance Calculation:

- Compute the distance from the viewer to each face centroid.

Angle Calculation:

- Determine the vertical angle relative to the viewer using ATAN2.

Step 4: Identifying Potential Obstructions

Logic:

- Compare angles and distances:
  - If a face has a higher angle and is closer to the viewer, it can obstruct other faces.

Outcome: Identify faces that may block visibility.

```
WHERE fa2.angle > fa.angle
  AND ST_Distance(vp.point, fa2.centroid) <= ST_Distance(vp.point, fa.centroid)
```

Step 5: Line-of-Sight Intersection

Logic:

- Draw a line from the viewer to each face centroid using ST_MakeLine.
- Check if the line intersects any other face polygon using ST_Intersects.

Condition:

- If the line intersects another face, the target face is obstructed.

```
ST_Intersects(
    ST_MakeLine(vp.point, fa.centroid),
    ST_MakePolygon(ST_MakeLine(ARRAY[...]))
)
```

Step 6: Final Visibility Determination

Logic:

- After filtering obstructed faces, ensure no duplicates remain.
- Use SELECT DISTINCT to clean the results.

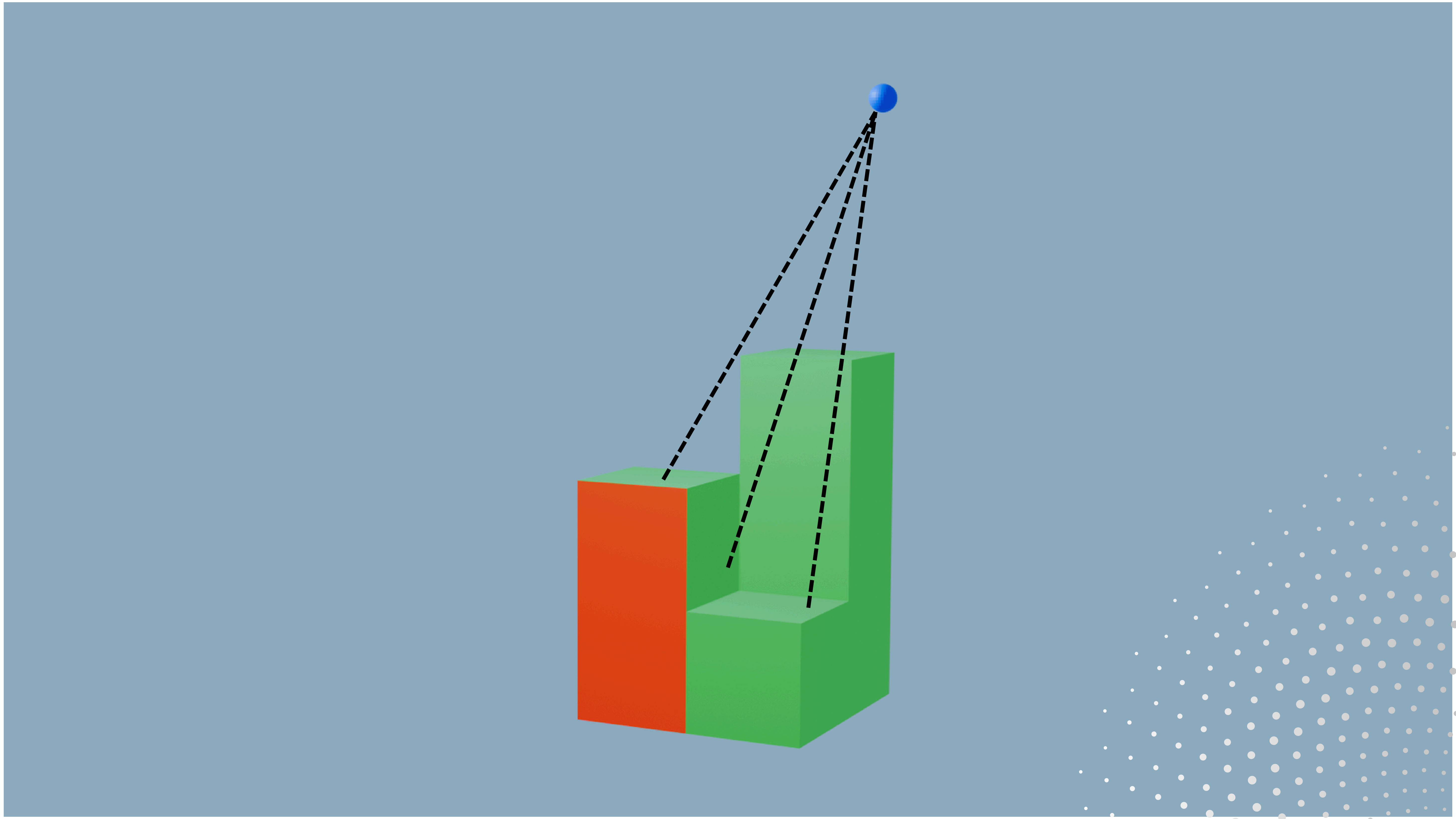
Output:

- Final list of visible faces for the box.

```
SELECT DISTINCT fa.box_id, fa.face, fa.visible
FROM visible_faces fa
```

Step 6: Visualizing the Results

Box	Face	Visible
Box 4	Front	TRUE
Box 4	Top	TRUE
Box 4	Right	TRUE

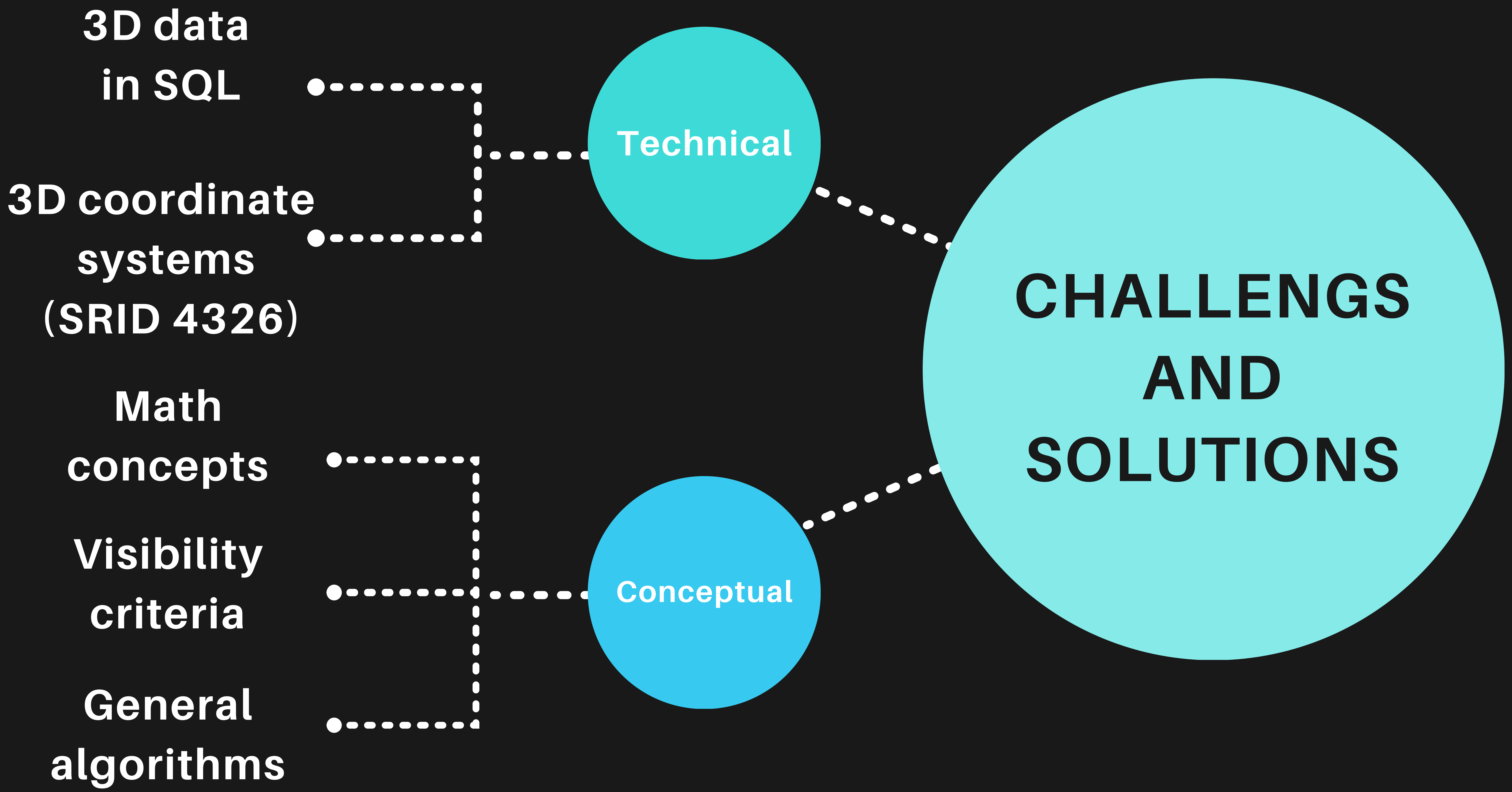


Visualizing the Solution

box_id	face	x1	y1	z1	x2	y2	z2	x3	y3	z3	x4	y4	z4	viewer_x	viewer_y	viewer_z	visible
1	back	0	0	0	1	0	0	1	0	1	0	0	1	3	3	5	FALSE
1	bottom	0	0	0	1	0	0	1	1	0	0	1	0	3	3	5	FALSE
	. . .																
2	right	2	0	0	2	1	0	2	1	2	2	0	2	3	3	5	FALSE
1	right	1	0	0	1	1	0	1	1	1	1	0	1	3	3	5	FALSE
1	top	0	0	1	1	0	1	1	1	1	0	1	1	3	3	5	FALSE
2	top	1	0	2	2	0	2	2	1	2	1	1	2	3	3	5	TRUE
3	back	0	1	0	1	1	0	1	1	3	0	1	3	3	3	5	FALSE
	. . .																
4	left	1	1	0	1	2	0	1	2	1	1	1	1	3	3	5	FALSE
4	right	2	1	0	2	2	0	2	2	1	2	1	1	3	3	5	TRUE
4	top	1	1	1	2	1	1	2	2	1	1	2	1	3	3	5	TRUE

Results

CVS data with face coordinates and visibility, also viewer (x, y, z).


3D data in SQL
3D coordinate systems (SRID 4326)
Math concepts
Visibility criteria
General algorithms
Technical
Conceptual
CHALLENGS AND SOLUTIONS

- Extend to dynamic viewer positions.
- Optimize for real-time 3D applications.
- Apply in GIS, gaming, and CAD.

Future Work

Wrapping Up

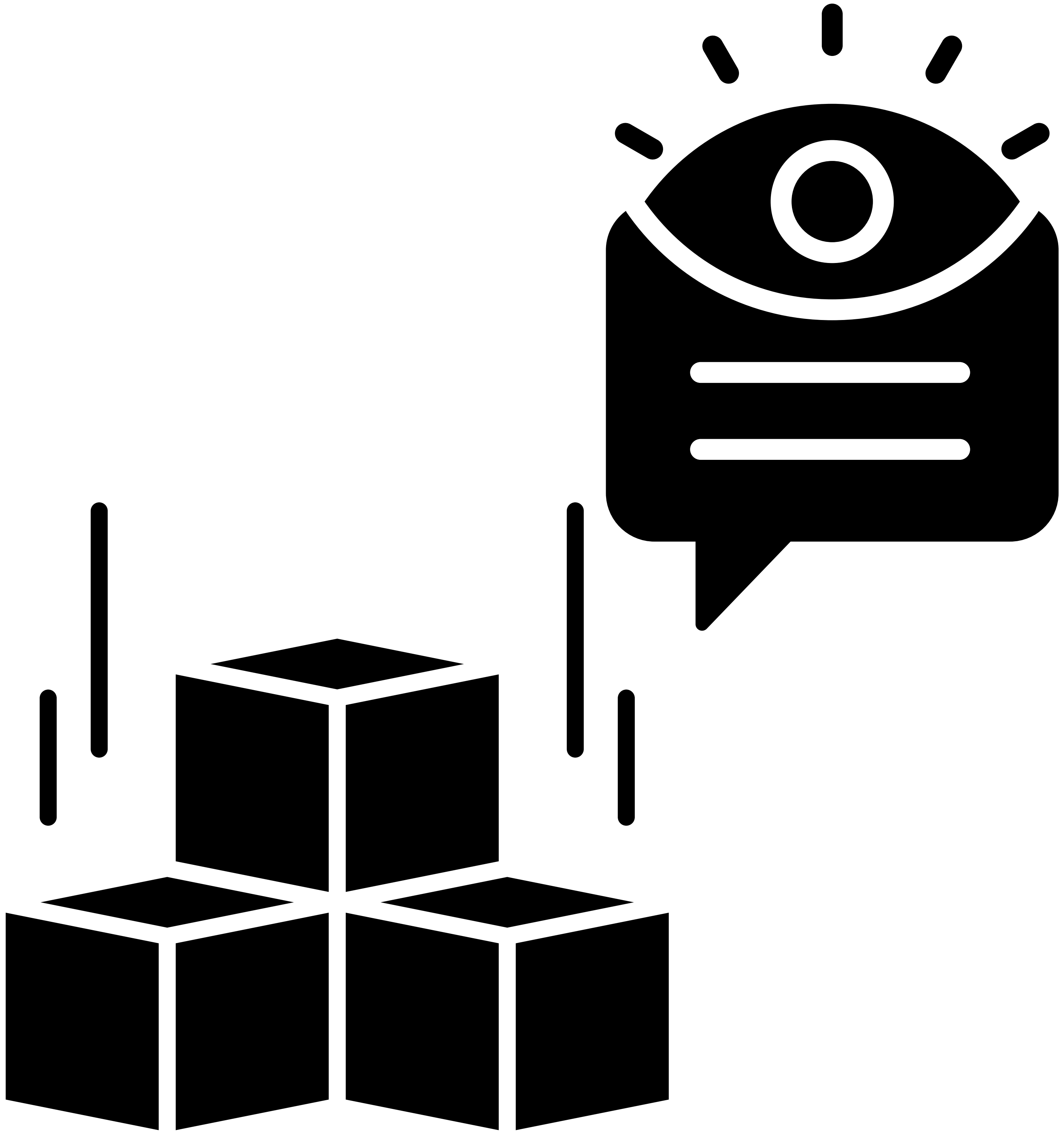
- **Achievements:**
  - Implemented visibility detection for 3D box faces using SQL and PostGIS.
  - Recursive CTEs and geospatial functions for complex spatial calculations.

- **Key Takeaways:**
  - SQL and PostGIS are powerful tools for handling 3D spatial data.
  - Precomputing centroids and using modular CTEs would reduce computational overhead.
  - Visual outputs make results more interpretable for practical applications.

Wrapping Up

- **Challenges Addressed:**
  - Ensuring line-of-sight accuracy.
  - Finding a general algorithm.
  - Using SQL with 3D datasets.

- **Real-World Relevance:**
  - This method can be applied in areas like GIS, gaming, architecture, and 3D modeling.



Questions?