

Chapter 03 - Deep Dive

Testing find_intersection()

A Visual Guide to Multi-Dimensional Test Design

Software Engineering | SoSe 2026

Hochschule Aalen

Why This Deserves Its Own Deck

This is your first encounter with:

-  Multi-dimensional equivalence class testing
-  Combinatorial complexity (600 potential tests!)
-  Strategic test reduction using domain knowledge
-  Visual understanding of geometric test cases

Goal: By the end, you'll understand how to systematically design tests for complex functions!

 **Important:** This builds on concepts from the main "Equivalence Classes" slide deck. If you haven't seen that yet, go back and review it first!

 **Full details:** See the [lecture notes section on `find\_intersection\(\)`](#)

The Function We're Testing

From your Road Profile Viewer project:

```
def find_intersection(
    x_road: np.ndarray,          # Road profile x-coordinates
    y_road: np.ndarray,          # Road profile y-coordinates
    angle_degrees: float,        # Ray direction (-90° to 90°)
    camera_x: float = 0,         # Camera x position
    camera_y: float = 1.5        # Camera y position
) -> tuple[float, float] | None:
    """
    Find where a ray from camera intersects the road profile.
    Returns (x, y) of intersection, or None if no intersection.
    """
    # ... 70 lines of geometric logic ...
```

The Challenge: Five Dimensions of Complexity

We identified 5 independent dimensions:

1. **Array Structure** (5 classes): Empty, single point, two points, three points, many points
2. **Angle** (5 classes): Downward, horizontal, upward, vertical down, vertical up
3. **Camera Position** (6 classes): Before road, on road, after road, above, at level, below
4. **Outcome** (4 classes): No intersection, first segment, middle segment, last segment
5. **Road Shape** (3 classes): Flat, sloped, curved

Math: $5 \times 5 \times 6 \times 4 \times 3 = 1800$ potential combinations! 🤖

Our Strategy: 1800 → 22 Tests

How we reduced the test count:

1. **Cover each dimension** with 2-4 representatives
2. **Use domain knowledge** to eliminate impossible/redundant combinations
3. **Focus on distinct behaviors** rather than all parameter combinations
4. **Add critical edge cases** that don't fit neat categories

Result: 22 strategic tests that cover the essential equivalence classes!

Let's see each test visually... 🎨

Reading the Visualizations

Legend for all diagrams:

-  **Blue line with dots** = Road profile
-  **Red circle** = Camera position
-  **Red dashed line** = Ray from camera (defined by angle)
-  **Green circle** = Intersection point (if found)
-  **Green dotted line** = Distance from camera to intersection

Note: All intersections are **computed geometrically**, not approximated!

Category 1: Structural Tests

Dimension: How many points define the road?

Why this matters:

- Empty arrays → error handling
- Single point → no line segment (invalid)
- Two points → minimum valid road (1 segment)
- Many points → typical case

Tests 1-4 cover this dimension...

Test 1: Empty Arrays

Input: `x_road = []`, `y_road = []`

Expected: `None` (no road to intersect)

What it tests: Error handling for invalid input structure

Test 2: Single Point

Input: `x_road = [5.0]` , `y_road = [2.0]`

Expected: `None` (a point doesn't define a line segment)

What it tests: Minimum structural validation (need at least 2 points)

Test 3: Two Points - Minimum Valid Road

Input: `x_road = [0, 10]` , `y_road = [0, 2]` , `angle = -10°`

Expected: `None` (ray misses the short road)

What it tests: Minimum valid structure (1 segment), ray geometry

Test 4: Many Points (50 Points)

Input: 50 points creating a long sloped road

Expected: Intersection found at ($x \approx 13.9$, $y \approx 1.4$)

What it tests: Complex road with many segments, typical use case

Category 2: Angle Equivalence Classes

Dimension: Direction of the ray

Why this matters:

- Different angles exercise different trigonometry
- Vertical rays ($\pm 90^\circ$) have special handling (no slope)
- Horizontal rays (0°) are edge cases
- Direction affects which segments can be hit

Tests 5-9 cover this dimension...

Test 5: Downward Angle (-10°)

Input: angle = -10° (looking down), camera above road

Expected: Intersection found at (x≈33.1, y≈4.1)

What it tests: Typical downward ray, positive slope math

Test 6: Horizontal Ray (0°)

Input: angle = 0° (horizontal), camera at road level

Expected: Intersection at (x=0, y=2)

What it tests: Zero slope case, tangent to flat road

Test 7: Upward Angle (45°)

Input: angle = 45° (looking up), camera at road level

Expected: Intersection at (x=1, y=1)

What it tests: Positive angle, negative slope math

Test 8: Vertical Downward Ray (-90°)

Input: angle = -90° (straight down), camera above road

Expected: None (vertical ray, no intersection detected)

What it tests: Special case - undefined slope, vertical ray handling

Test 9: Vertical Upward Ray (90°)

Input: angle = 90° (straight up), camera at ground

Expected: None (vertical ray, no intersection)

What it tests: Another vertical edge case, symmetry with -90°

Category 3: Position Equivalence Classes

Dimension: Where is the camera relative to the road?

Why this matters:

- Position affects which segments are "in front" of camera
- Edge positions test boundary logic
- Height affects intersection angles
- Tests camera_x and camera_y parameters

Tests 10-15 cover this dimension...

Test 10: Camera Before Road Start

Input: camera_x = 0 (before road starts at x=10)

Expected: Intersection at (x≈41.5, y≈3.8)

What it tests: Camera sees full road ahead, hits middle segment

Test 11: Camera On Road (x=15)

Input: camera_x = 15 (between road points)

Expected: Intersection at (x≈41.8, y≈4.2)

What it tests: Camera positioned within road bounds

Test 12: Camera After Road End

Input: camera_x = 30 (after road ends at x=20)

Expected: None (ray looks forward, road is behind)

What it tests: Camera past all segments, no forward intersection

Test 13: Camera Above Road - Looking Down

Input: camera_y = 20 (high above road), angle = -10°

Expected: None (ray misses low road)

What it tests: Vertical position affects reachability

Test 14: Camera At Road Level - Tangent

Input: camera_y = 2.0 (same height as flat road), angle = 0°

Expected: Intersection at (x=0, y=2) - ray touches road immediately

What it tests: Tangent case, ray along surface

Test 15: Camera Below Road - Looking Up

Input: camera_y = 0 (below road), angle = 10° (upward)

Expected: Intersection at (x≈28.3, y=5)

What it tests: Camera below surface looking up to hit road

Category 4: Outcome-Based Tests

Dimension: What does the function return?

Why this matters:

- Different outcomes exercise different code paths
- "Where" intersection occurs affects return value computation
- No intersection is a valid outcome to test

Tests 16-19 cover this dimension...

Test 16: Intersection on First Segment

Input: camera_x = -5 (before road), steep angle = -30°

Expected: Intersection at $(x \approx 8.2, y \approx 3.6)$ - hits first segment

What it tests: Ray hits the earliest possible segment

Test 17: Intersection on Middle Segment

Input: camera at $x=7$, gentle angle = -5°

Expected: Intersection at $(x \approx 13, y \approx 2.5)$ - middle of road

What it tests: Ray passes first segments, hits middle one

Test 18: Intersection on Last Segment

Input: camera at $x=15$, shallow angle = -2°

Expected: Intersection at $(x \approx 27.5, y \approx 0.55)$ - last segment

What it tests: Ray reaches far end of road

Test 19: No Intersection - Ray Misses Road

Input: camera at origin, upward angle = 45° , road far ahead

Expected: None (ray shoots upward, never hits road)

What it tests: Ray trajectory doesn't intersect road geometry

Category 5: Road Shape Variations

Dimension: Geometric properties of the road

Why this matters:

- Flat roads simplify geometry (all y-values same)
- Sloped roads test consistent gradient
- Curved roads test varying slopes

Tests 20-22 cover this dimension...

Test 20: Flat Horizontal Road - Tangent Case

Input: All y-values = 3.0 (perfectly flat), angle = 0°

Expected: Intersection at (x=0, y=3) - ray along surface

What it tests: Degenerate case, horizontal ray on horizontal road

Test 21: Consistently Sloped Road

Input: $y = 0.2x$ (linear slope), angle = -15°

Expected: Intersection at $(x \approx 22.7, y \approx 4.5)$

What it tests: Uniform gradient, all segments have same slope

Test 22: Curved/Wavy Road (Sinusoidal)

Input: $y = 2 \cdot \sin(x/5) + 3$ (wavy road)

Expected: Intersection at $(x \approx 27.8, y \approx 4.9)$

What it tests: Non-linear road, varying slopes, realistic complexity

Key Insight: Same Output, Different Dimensions

Notice that Test 3 and Test 13 both return `None` (no intersection).

But they test completely different things!

Test 3: Two-point road
(*structural dimension*)

- Camera at (0, 5), ray at -10°
- Road: (0, 0) $\rightarrow$ (10, 2)
- Ray would intersect at $x \approx 13.3$, but **road ends at x=10**

Test 13: Camera above road
(*position dimension*)

- Camera at (0, 20), ray at -10°
- Road: (0, 0) $\rightarrow$ (10, 1) $\rightarrow$ (20, 2)
- Ray would intersect at $x \approx 72.5$, but **road ends at x=20**

Why Both Tests Are Necessary

Test 3 validates: "Can the function handle a minimal road structure (just 1 segment)?"

- Tests structural robustness
- Ensures algorithm works with minimal valid input

Test 13 validates: "Can the function handle a camera positioned far above the road?"

- Tests positional logic
- Ensures vertical displacement doesn't break intersection math

Critical principle:

*You need to cover different input **dimensions** even if they produce the same*

The Complete Test Suite Structure

```
class TestFindIntersection:
    # Structural (4 tests)
    def test_empty_arrays(self): ...
    def test_single_point(self): ...
    def test_two_point_road(self): ...
    def test_many_point_road(self): ...

    # Angle (5 tests)
    def test_downward_angle(self): ...
    def test_horizontal_angle(self): ...
    def test_upward_angle(self): ...
    def test_vertical_downward(self): ...
    def test_vertical_upward(self): ...
```

The Complete Test Suite Structure (continued)

```
# Position (6 tests)
def test_camera_before_road(self): ...
def test_camera_on_road(self): ...
def test_camera_after_road(self): ...
def test_camera_above_road(self): ...
def test_camera_at_road_level(self): ...
def test_camera_below_road(self): ...

# Outcome (4 tests)
def test_first_segment(self): ...
def test_middle_segment(self): ...
def test_last_segment(self): ...
def test_no_intersection(self): ...
```

The Complete Test Suite Structure (continued)

```
# Shape (3 tests)
def test_flat_road(self): ...
def test_sloped_road(self): ...
def test_curved_road(self): ...

# Total: 22 strategic tests covering 5 dimensions
# Instead of: 1800 possible combinations!
```

What You've Learned

1. Multi-dimensional equivalence class testing

- Real functions have multiple independent input dimensions
- Each dimension can be tested somewhat independently

2. Strategic test reduction

- 1800 combinations → 22 tests using domain knowledge
- Cover each dimension with 2-4

3. Visual understanding

- Geometric problems benefit from visual test design
- Illustrations help verify test correctness
- Makes abstract concepts concrete

What You've Learned (continued)

4. Same output $\neq$ same test

- Tests can return identical values but cover different dimensions
- Test 3 (structural) vs Test 13 (positional) both return `None`
- Complete coverage requires testing all dimensions

5. Domain knowledge is irreplaceable

- Human understanding of geometry enables smart reduction
- LLMs can't visualize or prioritize like domain experts
- You provide insight, LLMs provide implementation help

Summary: The Testing Process

For complex multi-dimensional functions:

1. **Identify dimensions** - What are the independent input factors?
2. **Partition each dimension** - What are the equivalence classes?
3. **Calculate combinations** - How many total tests theoretically?
4. **Apply domain knowledge** - Which combinations are impossible/redundant?
5. **Choose representatives** - Pick specific values for each class
6. **Visualize if possible** - Diagrams help verify correctness
7. **Implement strategically** - Focus on distinct behaviors

Result: Manageable test suite with strong coverage! 

Questions?

What we covered:

- ✓ All 22 test cases for `find_intersection()` with visualizations
- ✓ Five dimensions: Structural, Angle, Position, Outcome, Shape
 - ✓ Strategic reduction: 1800 → 22 tests
- ✓ Key insight: Same output, different dimensions
 - ✓ Complete test suite structure

Back to: Main Equivalence Classes slides for more theory

Next: Boundary value analysis (testing at edges)

Questions? Discussion?