Floor Plan Shape Layout Logic

Overview

This document explains the algorithmic approach for generating L-shaped, H-shaped, M-shaped, and rectangular floor plans with required rooms: Living Room, Maid’s Room, Kitchen, and Garage.

Supported Shapes

1. Rectangular (Standard)

- Single continuous rectangular boundary
- Rooms arranged in a grid pattern
- Most space-efficient for small plots

2. L-Shaped

- Two perpendicular rectangular sections
- Ideal for corner plots
- Creates natural separation between public and private zones

3. H-Shaped

- Two parallel wings connected by a central bridge
- Excellent for larger plots
- Provides maximum natural light and ventilation
- Natural courtyard formation

4. M-Shaped (U-Shaped Variant)

- Central section with two projecting wings
- Creates a semi-enclosed courtyard
- Ideal for privacy-focused designs

Room Requirements

All shapes must accommodate these rooms (each rectangular or square):

1. **Living Room** - Main gathering space (min 12m²)
2. **Maid’s Room** - Service quarters (min 8m²)
3. **Kitchen** - Cooking area (min 8m²)
4. **Garage** - Vehicle parking (min 15m²)

Layout Algorithm

Phase 1: Shape Definition

Rectangular Layout

```
interface RectangularShape {
  sections: [{
    x: 0, y: 0,
    width: totalWidth,
    height: totalHeight
  }]
}
```

L-Shaped Layout

```
interface LShapedLayout {
  sections: [
    // Horizontal section (longer)
    { x: 0, y: 0, width: W1, height: H1 },
    // Vertical section (perpendicular)
    { x: 0, y: H1, width: W2, height: H2 }
  ]
  // Where: W1 > W2 and (H1 + H2) = totalHeight
}
```

Constraints:
- `W1 >= 1.5 * W2` (horizontal dominance)
- `H2 >= 6m` (minimum vertical wing length)
- Sections overlap at corner to form continuous structure

H-Shaped Layout

```
interface HShapedLayout {
  sections: [
    // Left wing
    { x: 0, y: 0, width: W_wing, height: H_total },
    // Central bridge
    { x: W_wing, y: H_bridge_offset, width: W_bridge, height: H_bridge },
    // Right wing
    { x: W_wing + W_bridge, y: 0, width: W_wing, height: H_total }
  ]
}
```

Constraints:
- `W_wing >= 5m` (minimum wing width)
- `W_bridge >= 3m` (corridor/connecting space)
- `H_bridge >= 4m` (functional bridge height)
- Bridge positioned 30-50% from top for optimal space usage

M-Shaped Layout

```
interface MShapedLayout {
  sections: [
    // Central base
    { x: W_wing, y: 0, width: W_center, height: H_total },
    // Left wing (projecting forward)
    { x: 0, y: 0, width: W_wing, height: H_wing },
    // Right wing (projecting forward)
    { x: W_wing + W_center, y: 0, width: W_wing, height: H_wing }
  ]
}
```

Constraints:

- `W_center >= 6m` (functional central space)
- `W_wing >= 4m` (minimum wing projection)
- `H_wing <= 0.7 * H_total` (wings shorter than central section)
- Wings project 2-4m forward to create courtyard effect

Phase 2: Room Placement Logic

Priority-Based Placement Algorithm

```
function placeRooms(shape: Shape, requiredRooms: Room[]): Layout {
  // Step 1: Calculate total available area
  const totalArea = calculateShapeArea(shape);

  // Step 2: Assign rooms to sections based on shape
  const placement = assignRoomsToSections(shape, requiredRooms);

  // Step 3: Optimize room positions within sections
  return optimizeLayout(placement);
}
```

Room Assignment Rules

For L-Shaped Plans:

```
Horizontal Section (Public Zone):
  - Living Room (primary position)
  - Kitchen (near living for easy access)

Vertical Section (Service Zone):
  - Garage (ground level, easy access)
  - Maid's Room (adjacent to garage/back entrance)
```

For H-Shaped Plans:

```
Left Wing:
  - Living Room (primary social space)
  - Kitchen (adjacent to living)

Central Bridge:
  - Dining area or circulation space

Right Wing:
  - Garage (separate from living areas)
  - Maid's Room (near garage for service access)
```

For M-Shaped Plans:

```
Central Section:
  - Living Room (focal point)
  - Kitchen (central location)

Left Wing:
  - Garage (separate entrance)

Right Wing:
  - Maid's Room (separate quarters with privacy)
```

For Rectangular Plans:

```
Linear Arrangement (left to right):
  1. Garage (entry side)
  2. Kitchen (adjacent to garage for supplies)
  3. Living Room (central position)
  4. Maid's Room (rear position)
```

Phase 3: Space Optimization

Room Sizing Algorithm

```
function calculateRoomDimensions(availableArea: number, roomType: string) {
  const minSizes = {
    'living': { min: 12, ideal: 20, ratio: 1.5 }, // 1.5:1 aspect ratio
    'kitchen': { min: 8, ideal: 12, ratio: 1.2 },
    'maid': { min: 8, ideal: 10, ratio: 1.0 },
    'garage': { min: 15, ideal: 18, ratio: 1.8 } // Car: 5m × 2.5m + buffer
  };

  const config = minSizes[roomType];

  // Calculate dimensions based on aspect ratio
  if (availableArea >= config.ideal) {
    const width = Math.sqrt(config.ideal * config.ratio);
    const height = config.ideal / width;
    return { width, height };
  } else {
    // Use minimum dimensions
    const width = Math.sqrt(config.min * config.ratio);
    const height = config.min / width;
    return { width, height };
  }
}
```

Constraints

1. **Minimum Room Dimensions:**
   - Living Room: 3m × 4m (12m²)
   - Kitchen: 2.5m × 3.2m (8m²)
   - Maid’s Room: 2.8m × 2.8m (7.84m²)
   - Garage: 5m × 3m (15m²)

2. **Circulation Space:**
   - Minimum corridor width: 1.2m
   - Door swing clearance: 1m
   - Between-room buffer: 0.3m (wall thickness)

3. **Aspect Ratios:**
   - Living rooms: 1.3:1 to 1.8:1 (avoid square)
   - Kitchens: 1.0:1 to 1.5:1 (functional work triangle)
   - Maid’s rooms: 1.0:1 (compact efficiency)
   - Garages: 1.5:1 to 2.0:1 (vehicle proportions)

Phase 4: Validation

Layout Validation Checklist

```
function validateLayout(layout: Layout): boolean {
  return (
    checkMinimumRoomSizes(layout) &&
    checkRoomOverlaps(layout) &&
    checkAccessibility(layout) &&
    checkBuildingCode(layout) &&
    checkCirculation(layout)
  );
}
```

1. **Room Size Validation:**
   - Each room meets minimum area requirements
   - Aspect ratios are within acceptable range

2. **Overlap Prevention:**
   - No rooms overlap
   - Wall thickness accounted for

3. **Accessibility:**
   - All rooms accessible from main circulation
   - Kitchen accessible from living room
   - Garage has direct exterior access
   - Maid’s room has separate/service entrance

4. **Building Code Compliance:**
   - Minimum window area (10% of floor area)
   - Emergency exits where required
   - Ventilation requirements met

Implementation Example

L-Shaped Layout Generation

```
function generateLShapedLayout(totalWidth: 12, totalHeight: 10) {
  // Define sections
  const horizontal = { x: 0, y: 0, width: 12, height: 6 };
  const vertical = { x: 0, y: 6, width: 7, height: 4 };

  // Place rooms
  const rooms = [
    {
      name: 'Living Room',
      section: horizontal,
      x: 1, y: 1,
      width: 5, height: 4,
      area: 20
    },
    {
      name: 'Kitchen',
      section: horizontal,
      x: 7, y: 1,
      width: 4, height: 3.5,
      area: 14
    },
    {
      name: 'Garage',
      section: vertical,
      x: 1, y: 6.5,
      width: 5, height: 3,
      area: 15
    },
    {
      name: 'Maid\'s Room',
      section: vertical,
      x: 1, y: 7,
      width: 3, height: 2.5,
      area: 7.5
    }
  ];

  return { sections: [horizontal, vertical], rooms };
}
```

Advantages of Each Shape

Rectangular

✅ Most efficient use of space
✅ Simplest to construct
✅ Lower construction cost
❌ Less privacy between zones
❌ Limited natural light penetration

L-Shaped

✅ Good for corner plots
✅ Natural zone separation
✅ Increased perimeter for windows
❌ Slightly more complex construction
❌ Potential corner drainage issues

H-Shaped

✅ Maximum natural light and ventilation
✅ Creates courtyards for outdoor space
✅ Clear zone separation
❌ Requires larger plot
❌ Higher construction cost
❌ More complex roofing

M-Shaped

✅ Semi-private courtyard
✅ Impressive street facade
✅ Good zone separation
❌ Requires wide plot frontage
❌ More expensive to build
❌ Complex roof drainage

Conclusion

This algorithmic approach ensures:

1. **Functional layouts** - All required rooms fit efficiently
2. **Architectural standards** - Proper room sizes and proportions
3. **Flexibility** - Adapts to different plot shapes and sizes
4. **Code compliance** - Meets minimum building requirements
5. **Scalability** - Can be extended to more complex shapes

The system prioritizes practical usability while maintaining architectural integrity across all shape variants.