🌿 Canopy Architecture Model - Crop Physiologist Expert Guide

🔬 Physiological Foundation of Canopy Light Interception

Understanding Canopy Architecture: The Art of Light Capture

As a crop physiologist, canopy architecture represents the three-dimensional organization of leaves that determines how effectively plants capture and utilize light energy. It's the interface between environmental light supply and photosynthetic demand, making it crucial for maximizing productivity in controlled environments.

Fundamental Concepts

1. **Beer's Law in Plant Canopies**
   - **Principle**: Light extinction follows exponential decay
   - **Formula**: $I = I_0 \times e^{(-k \times LAI)}$
   - **Light extinction coefficient (k)**: Depends on leaf angle, arrangement
   - **Applications**: Predicting light distribution through canopy layers
2. **Leaf Area Index (LAI)**
   - **Definition**: Total leaf area per unit ground area
   - **Optimal LAI**: 3-5 for most crops (diminishing returns beyond)
   - **Critical LAI**: Minimum for 95% light interception
   - **Dynamic changes**: Growth, senescence, defoliation
3. **Leaf Angle Distribution**
   - **Planophile**: Horizontal leaves (high light interception)
   - **Erectophile**: Vertical leaves (deeper penetration)
   - **Spherical**: Random angles (intermediate characteristics)
   - **Adaptive significance**: Balance between interception and penetration
4. **Sunlit vs. Shaded Leaves**
   - **Sunlit**: Direct beam radiation, higher photosynthesis
   - **Shaded**: Diffuse radiation only, lower light compensation
   - **Acclimation**: Different photosynthetic capacity and morphology
   - **Canopy efficiency**: Optimizing both leaf types

📊 Mathematical Framework

Beer's Law and Light Extinction

```python
```

```python
import numpy as np
import math
from typing import List, Tuple, Dict

def calculate_light_extinction(incident_ppfd: float, lai_cumulative: float,
                              extinction_coefficient: float) -> float:
    """
    Calculate light transmission through canopy using Beer's Law

    I(LAI) = I₀ × exp(-k × LAI)

    Where:
    - I(LAI): Light intensity at cumulative LAI depth
    - I₀: Incident light intensity (μmol m⁻² s⁻¹)
    - k: Light extinction coefficient
    - LAI: Cumulative leaf area index from top
    """
    return incident_ppfd * math.exp(-extinction_coefficient * lai_cumulative)

def calculate_extinction_coefficient(leaf_angle_distribution: str, solar_zenith: float) -> float:
    """
    Calculate light extinction coefficient based on canopy architecture

    k = G(θ) / cos(β)

    Where:
    - G(θ): Projection function for leaf angle distribution
    - β: Solar zenith angle
    """

    # Projection functions for different leaf angle distributions
    zenith_rad = math.radians(solar_zenith)
    cos_zenith = math.cos(zenith_rad)

    if leaf_angle_distribution == "spherical":
        # Random leaf angles
        g_function = 0.5
    elif leaf_angle_distribution == "planophile":
        # Horizontal leaves
        g_function = cos_zenith
    elif leaf_angle_distribution == "erectophile":
        # Vertical leaves
        g_function = 2.0 / math.pi * (1.0 - cos_zenith)
```

```python
    elif leaf_angle_distribution == "plagiophile":
        # 45-degree leaves
        g_function = 0.5 * (1.0 + cos_zenith)
    else:
        # Default to spherical
        g_function = 0.5

    # Extinction coefficient
    k = g_function / cos_zenith if cos_zenith > 0.1 else g_function / 0.1

    return min(k, 2.0)  # Cap at reasonable maximum

def calculate_multi_layer_light_profile(incident_ppfd: float, total_lai: float,
                                        num_layers: int, extinction_coeff: float) -> List[Dict]:
    """
    Calculate light profile through multiple canopy layers

    Returns PPFD and LAI for each layer from top to bottom
    """

    lai_per_layer = total_lai / num_layers
    layers = []

    for layer in range(num_layers):
        # Cumulative LAI from top to current layer
        lai_cumulative = (layer + 0.5) * lai_per_layer

        # Light intensity at layer center
        layer_ppfd = calculate_light_extinction(incident_ppfd, lai_cumulative, extinction_coeff)

        # Layer characteristics
        layer_info = {
            'layer_number': layer + 1,
            'lai_cumulative': lai_cumulative,
            'lai_in_layer': lai_per_layer,
            'ppfd': layer_ppfd,
            'light_fraction': layer_ppfd / incident_ppfd if incident_ppfd > 0 else 0,
            'position': 'top' if layer == 0 else 'bottom' if layer == num_layers - 1 else 'middle'
        }

        layers.append(layer_info)

    return layers
```

Sunlit and Shaded Leaf Separation

python

```python
def calculate_sunlit_shaded_fractions(total_lai: float, extinction_coeff: float,
                                      solar_zenith: float) -> Dict[str, float]:
    """
    Calculate sunlit and shaded leaf area fractions

    Based on Norman & Campbell (1989) approach:
    - Sunlit LAI: Leaves receiving direct beam radiation
    - Shaded LAI: Leaves receiving only diffuse radiation
    """

    zenith_rad = math.radians(solar_zenith)
    cos_zenith = math.cos(zenith_rad)

    # Extinction coefficient for direct beam
    k_beam = extinction_coeff / cos_zenith if cos_zenith > 0.1 else extinction_coeff / 0.1

    # Sunlit leaf area index
    if total_lai > 0:
        lai_sunlit = (1.0 - math.exp(-k_beam * total_lai)) / k_beam
    else:
        lai_sunlit = 0.0

    # Shaded leaf area index
    lai_shaded = total_lai - lai_sunlit

    # Fractions
    sunlit_fraction = lai_sunlit / total_lai if total_lai > 0 else 0
    shaded_fraction = lai_shaded / total_lai if total_lai > 0 else 0

    return {
        'lai_sunlit': lai_sunlit,
        'lai_shaded': lai_shaded,
        'sunlit_fraction': sunlit_fraction,
        'shaded_fraction': shaded_fraction,
        'k_beam': k_beam
    }

def calculate_light_components(incident_ppfd: float, diffuse_fraction: float,
                               lai_cumulative: float, extinction_coeff: float,
                               solar_zenith: float) -> Dict[str, float]:
    """
    Separate direct beam and diffuse radiation components
```

```
    Important for accurate photosynthesis calculation:
    - Direct beam: Higher intensity, directional
    - Diffuse: Lower intensity, multidirectional
    """

    # Separate incident radiation
    direct_beam = incident_ppfd * (1.0 - diffuse_fraction)
    diffuse_radiation = incident_ppfd * diffuse_fraction

    # Extinction of direct beam
    zenith_rad = math.radians(solar_zenith)
    cos_zenith = math.cos(zenith_rad)
    k_beam = extinction_coeff / cos_zenith if cos_zenith > 0.1 else extinction_coeff / 0.1

    direct_transmitted = direct_beam * math.exp(-k_beam * lai_cumulative)

    # Extinction of diffuse radiation (uses standard extinction coefficient)
    diffuse_transmitted = diffuse_radiation * math.exp(-extinction_coeff * lai_cumulative)

    # Total transmitted radiation
    total_transmitted = direct_transmitted + diffuse_transmitted

    return {
        'direct_beam_incident': direct_beam,
        'diffuse_incident': diffuse_radiation,
        'direct_beam_transmitted': direct_transmitted,
        'diffuse_transmitted': diffuse_transmitted,
        'total_transmitted': total_transmitted,
        'k_beam': k_beam,
        'k_diffuse': extinction_coeff
    }
```

🌱 Lettuce-Specific Canopy Characteristics

Morphological Parameters

```python
```

```python
@dataclass
class LettuceCanopyParameters:
    """Canopy architecture parameters specific to lettuce"""

    # Leaf angle distribution
    leaf_angle_distribution: str = "plagiophile"  # 45-degree angles typical
    mean_leaf_angle: float = 45.0  # degrees from horizontal
    leaf_angle_variance: float = 15.0  # degrees

    # Light extinction
    extinction_coefficient: float = 0.65  # Typical for lettuce
    k_diffuse: float = 0.75  # Higher for diffuse radiation

    # Canopy structure
    max_lai: float = 4.5  # Maximum LAI for lettuce
    critical_lai: float = 2.8  # LAI for 95% light interception
    optimal_lai: float = 3.5  # LAI for maximum productivity

    # Plant spacing effects
    plant_spacing: float = 0.20  # m between plants
    row_spacing: float = 0.25  # m between rows
    plant_density: float = 20.0  # plants m⁻²

    # Leaf characteristics
    specific_leaf_area: float = 25.0  # m² kg⁻¹ (varies with age)
    leaf_thickness: float = 0.25  # mm
    leaf_reflectance: float = 0.15  # Fraction of incident light reflected
    leaf_transmittance: float = 0.05  # Fraction transmitted through leaf

def calculate_lettuce_canopy_development(days_after_emergence: int,
                                         growing_conditions: Dict) -> Dict:
    """
    Model lettuce canopy development over time

    Lettuce canopy characteristics change dramatically during growth:
    - Early: Small rosette, planophile leaves
    - Mid: Expanding rosette, more erect leaves
    - Late: Large rosette or head formation
    """

    # Growth stage classification
    if days_after_emergence < 14:
        stage = "early_rosette"
```

```python
elif days_after_emergence < 35:
    stage = "expanding_rosette"
elif days_after_emergence < 50:
    stage = "mature_rosette"
else:
    stage = "heading_or_bolting"

# Stage-specific parameters
stage_params = {
    "early_rosette": {
        "lai": min(0.5 + 0.08 * days_after_emergence, 1.2),
        "leaf_angle": 25.0,  # More horizontal
        "extinction_coeff": 0.8,  # Higher k
        "canopy_height": 0.05 + 0.002 * days_after_emergence,  # m
        "leaf_area_per_plant": 0.025 + 0.004 * days_after_emergence  # m²
    },

    "expanding_rosette": {
        "lai": 1.2 + 0.12 * (days_after_emergence - 14),
        "leaf_angle": 35.0,  # Intermediate angle
        "extinction_coeff": 0.70,
        "canopy_height": 0.08 + 0.004 * (days_after_emergence - 14),
        "leaf_area_per_plant": 0.08 + 0.015 * (days_after_emergence - 14)
    },

    "mature_rosette": {
        "lai": min(3.0 + 0.08 * (days_after_emergence - 35), 4.0),
        "leaf_angle": 45.0,  # More erect
        "extinction_coeff": 0.65,
        "canopy_height": 0.15 + 0.003 * (days_after_emergence - 35),
        "leaf_area_per_plant": 0.25 + 0.01 * (days_after_emergence - 35)
    },

    "heading_or_bolting": {
        "lai": max(3.5 - 0.05 * (days_after_emergence - 50), 2.0),  # May decline
        "leaf_angle": 50.0,  # More vertical
        "extinction_coeff": 0.60,
        "canopy_height": 0.20 + 0.005 * (days_after_emergence - 50),
        "leaf_area_per_plant": max(0.35 - 0.005 * (days_after_emergence - 50), 0.20)
    }
}

current_params = stage_params[stage]
```

```python
    # Environmental modifications
    light_level = growing_conditions.get('daily_light_integral', 15)
    temperature = growing_conditions.get('temperature', 22)

    # Light acclimation
    if light_level > 20:  # High light
        current_params['leaf_angle'] += 10  # More erect leaves
        current_params['extinction_coeff'] *= 0.9  # Better light penetration
    elif light_level < 12:  # Low light
        current_params['leaf_angle'] -= 10  # More horizontal leaves
        current_params['extinction_coeff'] *= 1.1  # Greater interception

    # Temperature effects on morphology
    if temperature > 25:  # High temperature
        current_params['leaf_angle'] += 5  # Slightly more erect
        current_params['canopy_height'] *= 1.1  # Taller growth

    return {
        'growth_stage': stage,
        'days_after_emergence': days_after_emergence,
        'canopy_parameters': current_params,
        'light_interception_efficiency': calculate_light_interception_efficiency(
            current_params['lai'], current_params['extinction_coeff']
        )
    }

def calculate_light_interception_efficiency(lai: float, k: float) -> float:
    """Calculate fraction of incident light intercepted by canopy"""
    return 1.0 - math.exp(-k * lai)
```

💡 Photosynthesis Integration

Layer-by-Layer Photosynthesis

```python
```

```python
def calculate_canopy_photosynthesis(light_layers: List[Dict], leaf_photosynthesis_params: Dict,
                                    temperature: float, co2_concentration: float) -> Dict:
    """
    Calculate total canopy photosynthesis by integrating over all layers

    Each layer has different light environment and potentially different
    photosynthetic capacity (acclimation to light level)
    """

    total_photosynthesis = 0.0
    layer_details = []

    for layer in light_layers:
        # Light level for this layer
        layer_ppfd = layer['ppfd']
        layer_lai = layer['lai_in_layer']

        # Light acclimation factor
        acclimation_factor = calculate_light_acclimation_factor(layer_ppfd)

        # Adjusted photosynthetic parameters for this layer
        adjusted_vcmax = leaf_photosynthesis_params['vcmax_25'] * acclimation_factor
        adjusted_jmax = leaf_photosynthesis_params['jmax_25'] * acclimation_factor

        # Leaf-level photosynthesis rate
        leaf_photosynthesis = calculate_leaf_photosynthesis(
            layer_ppfd, temperature, co2_concentration, adjusted_vcmax, adjusted_jmax
        )

        # Scale by LAI in this layer
        layer_photosynthesis = leaf_photosynthesis * layer_lai
        total_photosynthesis += layer_photosynthesis

        layer_details.append({
            'layer': layer['layer_number'],
            'ppfd': layer_ppfd,
            'lai': layer_lai,
            'acclimation_factor': acclimation_factor,
            'leaf_photosynthesis': leaf_photosynthesis,
            'layer_photosynthesis': layer_photosynthesis,
            'contribution_percent': (layer_photosynthesis / total_photosynthesis * 100) if total_photosynthesis > 0 else
        })
```

```python
    # Calculate efficiency metrics
    average_leaf_photosynthesis = total_photosynthesis / sum(l['lai'] for l in light_layers) if light_layers else 0

    return {
        'total_canopy_photosynthesis': total_photosynthesis,  # μmol CO₂ m⁻² ground s⁻¹
        'average_leaf_photosynthesis': average_leaf_photosynthesis,  # μmol CO₂ m⁻² leaf s⁻¹
        'layer_details': layer_details,
        'photosynthetic_efficiency': total_photosynthesis / sum(l['ppfd'] * l['lai'] for l in light_layers) if light_layers els
    }

def calculate_light_acclimation_factor(ppfd: float) -> float:
    """
    Calculate light acclimation factor for photosynthetic capacity

    Leaves acclimate to their average light environment:
    - High light: Higher Vcmax, Jmax (sun leaves)
    - Low light: Lower Vcmax, Jmax but higher efficiency (shade leaves)
    """

    # Reference PPFD for full acclimation
    reference_ppfd = 800.0  # μmol m⁻² s⁻¹

    # Acclimation response (saturating function)
    if ppfd <= 0:
        return 0.3  # Minimum capacity for very low light
    elif ppfd < 50:
        # Very low light acclimation
        return 0.3 + 0.2 * (ppfd / 50)
    elif ppfd < 200:
        # Low light acclimation
        return 0.5 + 0.3 * ((ppfd - 50) / 150)
    elif ppfd < reference_ppfd:
        # Moderate to high light acclimation
        return 0.8 + 0.2 * ((ppfd - 200) / (reference_ppfd - 200))
    else:
        # Full sun acclimation
        return 1.0

def calculate_leaf_photosynthesis(ppfd: float, temperature: float, co2: float,
                                 vcmax: float, jmax: float) -> float:
    """
    Calculate leaf-level photosynthesis using simplified FvCB model

    This is a simplified version - the full model would include all
```

```
    temperature dependencies and environmental factors
    """

    # Light response
    alpha = 0.24  # Quantum efficiency
    theta = 0.70  # Curvature factor

    # Electron transport rate
    j = (alpha * ppfd + jmax - math.sqrt((alpha * ppfd + jmax)**2 - 4 * theta * alpha * ppfd * jmax)) / (2 * theta)

    # CO₂ response parameters (simplified)
    gamma_star = 42.75  # CO₂ compensation point
    kc = 460.0  # Michaelis constant for CO₂
    ko = 330.0  # Michaelis constant for O₂
    o2 = 210000.0  # O₂ concentration (μmol mol⁻¹)

    # Intercellular CO₂ (assuming 70% of ambient)
    ci = co2 * 0.7

    # RuBisCO-limited rate
    wc = vcmax * (ci - gamma_star) / (ci + kc * (1 + o2 / ko))

    # RuBP regeneration-limited rate
    wj = j * (ci - gamma_star) / (4 * (ci + 2 * gamma_star))

    # Net photosynthesis (simplified, no TPU limitation)
    rd = 0.015 * vcmax  # Dark respiration
    an = min(wc, wj) - rd

    return max(0, an)  # μmol CO₂ m⁻² s⁻¹
```

Canopy Light Use Efficiency

```
python
```

```
def calculate_canopy_light_use_efficiency(canopy_photosynthesis: float,
                                          intercepted_light: float,
                                          biomass_growth_rate: float) -> Dict:
    """
    Calculate various measures of canopy light use efficiency

    Multiple metrics provide different insights:
    - Quantum efficiency: mol CO₂ per mol photons
    - Radiation use efficiency: g biomass per MJ intercepted
    - Photosynthetic light use efficiency: μmol CO₂ per μmol photons
    """

    # Convert units for calculations
    # 1 μmol photons m⁻² s⁻¹ = 0.2174 J m⁻² s⁻¹ (for PAR)
    intercepted_energy = intercepted_light * 0.2174  # J m⁻² s⁻¹

    # Photosynthetic light use efficiency
    if intercepted_light > 0:
        photosynthetic_lue = canopy_photosynthesis / intercepted_light  # mol CO₂ mol⁻¹ photons
    else:
        photosynthetic_lue = 0

    # Radiation use efficiency (biomass per unit energy)
    if intercepted_energy > 0:
        radiation_use_efficiency = biomass_growth_rate / (intercepted_energy * 86400)  # g biomass MJ⁻¹
    else:
        radiation_use_efficiency = 0

    # Theoretical maximum efficiency
    theoretical_max_quantum_efficiency = 0.125  # mol CO₂ mol⁻¹ photons (C3 plants)
    efficiency_ratio = photosynthetic_lue / theoretical_max_quantum_efficiency

    return {
        'photosynthetic_lue': photosynthetic_lue,
        'radiation_use_efficiency': radiation_use_efficiency,
        'efficiency_ratio': efficiency_ratio,
        'theoretical_maximum': theoretical_max_quantum_efficiency,
        'intercepted_light': intercepted_light,
        'intercepted_energy': intercepted_energy
    }

def optimize_canopy_architecture(target_lai_range: Tuple[float, float],
                                 light_conditions: Dict,
```

```
                        spacing_constraints: Dict) -> Dict:
    """
    Optimize canopy architecture for given conditions

    Balances:
    - Light interception vs. penetration
    - Plant density vs. individual plant size
    - Early vs. late canopy closure
    """

    optimization_results = []

    # Test different LAI values within range
    lai_values = np.linspace(target_lai_range[0], target_lai_range[1], 10)

    for lai in lai_values:
        # Test different extinction coefficients
        k_values = [0.5, 0.6, 0.7, 0.8]

        for k in k_values:
            # Calculate light interception
            light_interception = 1 - math.exp(-k * lai)

            # Calculate light penetration to bottom layers
            bottom_light = light_conditions['incident_ppfd'] * math.exp(-k * lai)

            # Simple productivity estimate
            avg_light = light_conditions['incident_ppfd'] * (1 - math.exp(-k * lai)) / (k * lai) if lai > 0 else 0
            estimated_productivity = avg_light * lai * 0.05  # Simplified conversion

            # Penalties for poor light distribution
            if bottom_light < 50:  # Too dark at bottom
                estimated_productivity *= 0.8
            if light_interception < 0.90:  # Poor light capture
                estimated_productivity *= 0.9

            optimization_results.append({
                'lai': lai,
                'extinction_coefficient': k,
                'light_interception': light_interception,
                'bottom_light': bottom_light,
                'average_light': avg_light,
                'estimated_productivity': estimated_productivity
            })
```

```
    # Find optimal configuration
    best_config = max(optimization_results, key=lambda x: x['estimated_productivity'])

    return {
        'optimal_configuration': best_config,
        'all_configurations': optimization_results,
        'optimization_criteria': {
            'light_interception_target': "> 90%",
            'bottom_light_minimum': "> 50 µmol m⁻² s⁻¹",
            'productivity_maximization': "Primary objective"
        }
    }
```

🎯 Practical Applications

LED Lighting Optimization

```python
```

```
def optimize_led_lighting_for_canopy(canopy_height: float, lai: float,
                                    target_ppfd_bottom: float,
                                    led_specifications: Dict) -> Dict:
    """
    Optimize LED lighting configuration for lettuce canopy

    Considerations:
    - Uniform light distribution
    - Energy efficiency
    - Avoid photoinhibition at top
    - Maintain minimum light at bottom
    """

    # LED parameters
    led_power = led_specifications.get('power_per_unit', 100)  # W
    led_efficiency = led_specifications.get('efficiency', 2.5)  # μmol J⁻¹
    led_height_above_canopy = led_specifications.get('height', 0.3)  # m

    # Calculate required top-of-canopy PPFD
    extinction_coeff = 0.65  # Typical for lettuce
    top_ppfd_required = target_ppfd_bottom / math.exp(-extinction_coeff * lai)

    # Account for distance from LED to canopy top
    # Inverse square law approximation
    distance_factor = (led_height_above_canopy + 0.1) / 0.1  # Reference distance
    led_output_required = top_ppfd_required * (distance_factor ** 2)

    # Calculate LED requirements
    photon_flux_required = led_output_required  # μmol m⁻² s⁻¹
    power_required = photon_flux_required / led_efficiency  # W m⁻²

    # Daily light integral calculation
    photoperiod = led_specifications.get('photoperiod', 16)  # hours
    daily_light_integral = photon_flux_required * photoperiod * 3.6 / 1000  # mol m⁻² d⁻¹

    # Energy consumption
    daily_energy = power_required * photoperiod  # Wh m⁻² d⁻¹

    # Check for photoinhibition risk
    photoinhibition_risk = "Low"
    if top_ppfd_required > 1000:
        photoinhibition_risk = "Medium"
    if top_ppfd_required > 1500:
```

```python
        photoinhibition_risk = "High"

    return {
        'lighting_requirements': {
            'top_canopy_ppfd': top_ppfd_required,
            'bottom_canopy_ppfd': target_ppfd_bottom,
            'led_output_required': led_output_required,
            'power_density': power_required,
            'daily_light_integral': daily_light_integral,
            'daily_energy_consumption': daily_energy
        },
        'system_specifications': {
            'led_height': led_height_above_canopy,
            'photoperiod': photoperiod,
            'extinction_coefficient': extinction_coeff,
            'photoinhibition_risk': photoinhibition_risk
        },
        'efficiency_metrics': {
            'photons_per_watt': led_efficiency,
            'energy_per_mol_photons': 1 / led_efficiency * 1000,  # kJ mol⁻¹
            'cost_effectiveness': daily_energy / daily_light_integral  # Wh mol⁻¹
        }
    }

def calculate_plant_spacing_optimization(target_lai: float, individual_plant_characteristics: Dict,
                                       growth_duration: int) -> Dict:
    """
    Optimize plant spacing for target LAI and uniform canopy development

    Balances:
    - Individual plant development
    - Canopy closure timing
    - Light competition effects
    - Harvest efficiency
    """

    # Plant characteristics
    max_plant_diameter = individual_plant_characteristics.get('max_diameter', 0.25)  # m
    leaf_area_per_plant = individual_plant_characteristics.get('leaf_area', 0.3)  # m²
    growth_rate = individual_plant_characteristics.get('growth_rate', 0.015)  # m² d⁻¹

    # Calculate required plant density
    plants_per_m2 = target_lai / leaf_area_per_plant
```

```python
# Calculate spacing
area_per_plant = 1.0 / plants_per_m2  # m²
square_spacing = math.sqrt(area_per_plant)  # m (for square arrangement)

# Check spacing constraints
minimum_spacing = max_plant_diameter * 0.8  # 80% of diameter for slight overlap
maximum_spacing = max_plant_diameter * 1.5  # 150% for good light penetration

# Spacing feasibility
if square_spacing < minimum_spacing:
    feasibility = "Too dense - plants will compete"
    recommended_spacing = minimum_spacing
    achieved_lai = (1.0 / recommended_spacing**2) * leaf_area_per_plant
elif square_spacing > maximum_spacing:
    feasibility = "Too sparse - poor light utilization"
    recommended_spacing = maximum_spacing
    achieved_lai = (1.0 / recommended_spacing**2) * leaf_area_per_plant
else:
    feasibility = "Optimal"
    recommended_spacing = square_spacing
    achieved_lai = target_lai

# Canopy closure timing
days_to_closure = calculate_canopy_closure_time(
    recommended_spacing, individual_plant_characteristics, growth_rate
)

return {
    'spacing_optimization': {
        'target_lai': target_lai,
        'achieved_lai': achieved_lai,
        'recommended_spacing': recommended_spacing,
        'plants_per_m2': 1.0 / recommended_spacing**2,
        'feasibility': feasibility
    },
    'timing_analysis': {
        'days_to_canopy_closure': days_to_closure,
        'closure_percentage_of_cycle': (days_to_closure / growth_duration) * 100,
        'optimal_closure_timing': "60-70% of growth cycle"
    },
    'production_metrics': {
        'leaf_area_efficiency': achieved_lai / (1.0 / recommended_spacing**2),  # m² plant⁻¹
        'space_utilization': (max_plant_diameter / recommended_spacing)**2,
        'harvest_accessibility': "Good" if recommended_spacing > 0.15 else "Limited"
```

```python
        }
    }

def calculate_canopy_closure_time(spacing: float, plant_characteristics: Dict,
                                  growth_rate: float) -> int:
    """Calculate days until canopy closure at given spacing"""

    initial_diameter = plant_characteristics.get('initial_diameter', 0.05)  # m
    area_to_fill = spacing**2  # m²

    # Simple exponential growth model
    # A(t) = A₀ × e^(r×t)
    # Solve for t when A(t) = area_to_fill

    if growth_rate > 0:
        days_to_closure = math.log(area_to_fill / (math.pi * (initial_diameter/2)**2)) / growth_rate
    else:
        days_to_closure = float('inf')

    return max(1, int(days_to_closure))
```

This canopy architecture model enables precise optimization of light distribution and plant spacing in hydroponic systems, maximizing photosynthetic efficiency and crop productivity through science-based canopy management.