# Comparative Analysis of Horticultural Lighting Systems: A Computational and Experimental Approach

Austin Rouse

March 3, 2025

**Abstract**

This paper presents a comprehensive comparative analysis of two horticultural lighting systems: a novel system that employs Samsung COB LEDs [3] arranged in a centered square sequence and a conventional system that utilizes a uniform matrix of Samsung Gen 2 Horticulture Modules [2]. Luminous flux simulations are conducted using a custom radiosity model that is integrated with a global optimization algorithm based on Differential Evolution (DE) - which generates initial luminous flux assignments that are subsequently validated using DIALux, an industry standard lighting simulation tool. The lux values output by DIALux were converted to PPFD using a methodology based on the LED's normalized spectral power distribution, yielding conversion factors of 0.0179 and 0.0138 µmol/m$^2$/s/lx for the Samsung Gen 2 module ($\lambda \in [380, 780]$ nm) and the Samsung COB, respectively (see Section 3.1 for details). The DE-based global optimization algorithm is applied to refine LED intensity assignments in a stratified arrangement (described by the centered square number sequence, $C(n) = (2n-1)^2$). The results indicate that the novel design achieves superior PPFD uniformity (Degree of Uniformity (DOU) of approximately 98%) and lower LED component costs (up to 3.7× lower) compared to conventional systems. These findings underscore the technical and economic advantages of the novel design for advanced horticultural applications.

## Contents

# 1 Introduction

Efficient and uniform illumination is critical for optimal plant growth and yield in controlled environment agriculture (CEA). This study investigates the performance of a novel horticultural lighting system, comparing it to a conventional approach. The primary focus is on analyzing the distribution of Photosynthetic Photon Flux Density (PPFD), a key metric for assessing lighting effectiveness. Luminous flux simulations were performed using a custom radiosity model integrated with a DE-based global optimization algorithm, in conjunction with DIALux, where the radiosity model is used only to provide initial luminous flux assignments and DIALux (a trusted standard) validates the results. The simulated lux values are then converted to PPFD using a conversion factor specific to the spectral characteristics of the LEDs.

# 2 System Descriptions

## 2.1 Novel Lighting System

The novel lighting system utilizes a plurality of chip-on-board (COB) LEDs (BXRE-30E6500-C-83). These COBs exhibit a near-Lambertian radiation pattern without secondary optics. The corresponding .ies ray file for the COBs was sourced from Samsung and used in the DIALux simulations.

Their arrangement follows a centered square number sequence pattern (OEIS: A001844) [1], rotated by 45 degrees to form a square or rectangular lighting array. Mathematically, the pattern is described by:

$$C(n) = (2n - 1)^2,$$

where $n$ indicates the layer (with 1 LED in the center, 4 in the first square layer, 8 in the second, etc.). For example, in some embodiments, a square grow space may use 61 COBs, whereas a rectangular space may incorporate 127 COBs by combining two 61-COB squares with a 5-COB connector module. This design is intended to optimize the uniformity of the PPFD in the illuminated area by enabling layer-wise luminous flux assignments informed by the proprietary DE-based global optimization algorithm.

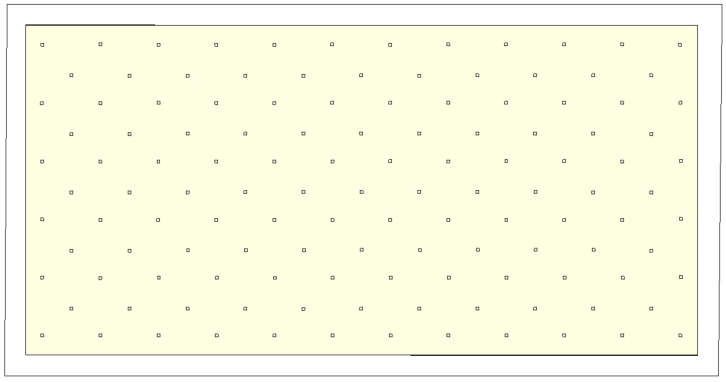

Figure 1: Centered Square Sequence Pattern Arrangement of COB LEDs.

## 2.2 Conventional Lighting System

The conventional system employs a plurality of Samsung Gen 2 Horticulture Modules (SI-B8T502560WW), arranged in a uniform rectangular matrix—a common configuration in horticultural lighting. The corresponding .ies ray file was sourced from Samsung and used in the DIALux simulations.

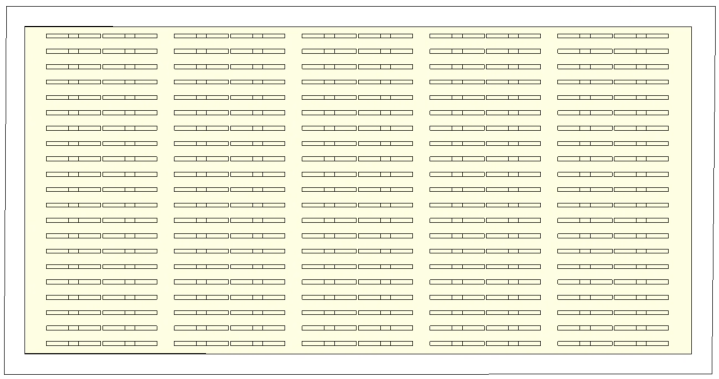

Figure 2: Uniform Matrix Arrangement of Samsung Gen 2 Horticulture Modules.

# 3 Simulation Methodology

The simulations were performed using our custom radiosity model in conjunction with DIALux evo lighting design software. The radiosity model accurately predicts interreflections within the enclosed environment and utilizes a DE-based global optimization algorithm to discover optimal layer-wise luminous flux assignments for the COBs. It is used solely to generate initial intensity assignments that are then manually entered into DIALux. All LED modules in the conventional arrangement were assigned the same luminous flux, following the standard appraoch.

The photometric data (from the .ies files) along with the geometric arrangement and surface reflectances are used by DIALux to produce validated lux distributions. These lux values are converted to PPFD (expressed in µmol/m$^2$/s) using a conversion factor. Section 3.1 details this conversion.

## 3.1 Lux to PPFD Conversion

The conversion from illuminance (lux) to photosynthetic photon flux density (PPFD, in µmol/m$^2$/s) is determined by digitizing the LED's normalized spectral power distribution (SPD), $I(\lambda)$, over a wavelength range $[\lambda_1, \lambda_2]$. In our approach, the PPFD is calculated as

$$\text{PPFD} = \frac{10^6}{N_A\, h\, c} \int_{\lambda_1}^{\lambda_2} I(\lambda)\, \left(\lambda \times 10^{-9}\right) d\lambda,$$

where $h$ is Planck's constant, $c$ is the speed of light, $N_A$ is Avogadro's number, and the factor $10^6$ converts moles to micromoles. The illuminance is determined by

$$E = 683 \int_{\lambda_1}^{\lambda_2} I(\lambda)\, V(\lambda)\, d\lambda,$$

with $V(\lambda)$ being the CIE photopic luminous efficiency function and 683 lx W$^{-1}$ the maximum luminous efficacy of radiation. Consequently, the lux-to-PPFD conversion factor, $C$ (in µmol/m$^2$/s/lx), is given by

$$C = \frac{\text{PPFD}}{E} = \frac{\frac{10^6}{N_A\, h\, c} \int_{\lambda_1}^{\lambda_2} I(\lambda)\, \left(\lambda \times 10^{-9}\right) d\lambda}{683 \int_{\lambda_1}^{\lambda_2} I(\lambda)\, V(\lambda)\, d\lambda}.$$

For the Samsung Gen 2 Horticulture Module ($\lambda \in [380, 780]$ nm) and the Samsung COBs ($\lambda \in [400, 800]$ nm), our analysis yields conversion factors of 0.0179 and 0.0138 µmol/m$^2$/s/lx, respectively. A more accurate determination would require absolute SPD measurements; however, these values provide a useful basis for comparison in our study.

# 4 Results and Discussion

## 4.1 PPFD Uniformity Analysis

Figures 3 and 4 show the PAR maps for the novel and conventional systems, respectively. Figures 5 and 6 display the corresponding heatmaps.

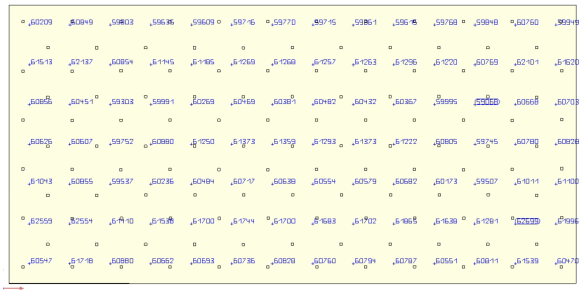

Figure 3: Novel Lighting System PAR Map

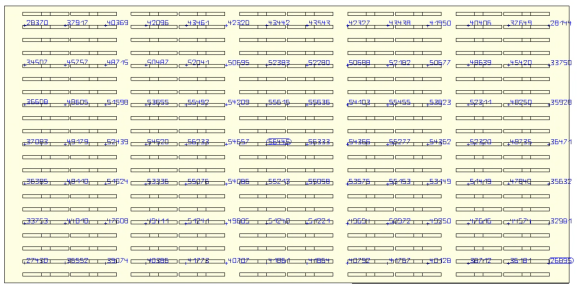

Figure 4: Conventional Lighting System PAR Map

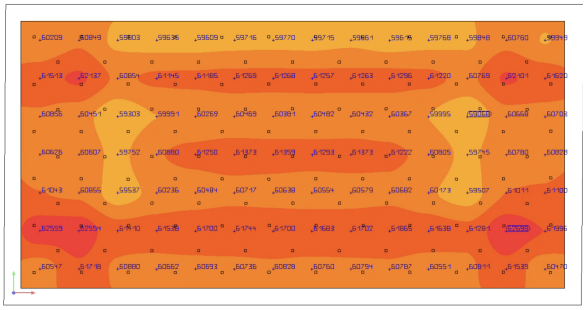

Figure 5: Novel Lighting System Heatmap

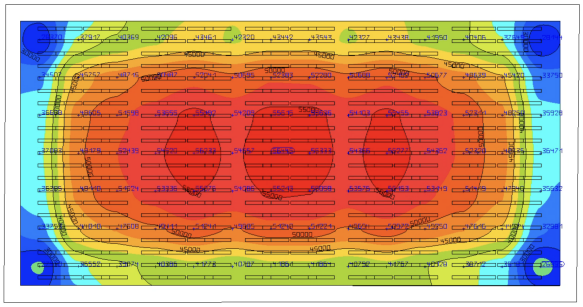

Figure 6: Conventional Lighting System Heatmap

Tables 1 and 2 summarize the key performance metrics: Average PPFD, Root Mean Squared Error (RMSE), Degree of Uniformity (DOU), Mean Absolute Deviation (MAD), Coefficient of Variation (CV), the Minimum PPFD, the Maximum PPFD, and the Min./Avg. PPFD and Min./Max. PPFD. The novel system achieves a DOU of 98.77% versus 83.37% for the conventional system.

Table 1: Simulation Results - Novel System

| **Metric** | **Value** |
|---|---|
| RMSE | 10.28 |
| DOU (%) | 98.77 |
| MAD | 7.99 |
| CV (%) | 1.23 |
| Average PPFD | 838.98 µmol/m$^2$/s |
| Min. PPFD | 815.14 µmol/m$^2$/s |
| Max PPFD | 865.25 µmol/m$^2$/s |
| Min/Max PPFD | 0.94 |
| Min/Avg. PPFD | 0.97 |

Table 2: Simulation Results - Conventional System

| Metric | Value |
|---|---|
| RMSE | 138.10 |
| DOU (%) | 83.37 |
| MAD | 119.40 |
| CV (%) | 16.63 |
| Average PPFD | 830.55 µmol/m$^2$/s |
| Min. PPFD | 481.42 µmol/m$^2$/s |
| Max PPFD | 1010.28 µmol/m$^2$/s |
| Min/Max PPFD | 0.48 |
| Min/Avg. PPFD | 0.58 |

# Mathematical Representation of Metrics

Here, we define the mathematical formulations used to calculate the metrics presented in Tables 1 and 2. Let $P_i$ represent the PPFD value at the $i$-th measurement point, and let $n$ be the total number of measurement points (in this case, $n = 98$).

1. **Average PPFD ($\mathbf{PPFD_{avg}}$):** The average PPFD is simply the arithmetic mean of all PPFD measurements:
$$\text{PPFD}_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n} P_i$$

2. **Root Mean Squared Error (RMSE):** RMSE quantifies the difference between the measured PPFD values and the average PPFD:
$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(P_i - \text{PPFD}_{\text{avg}})^2}{n}}$$

3. **Degree of Uniformity (DOU):** DOU is calculated based on the RMSE and the average PPFD:
$$\text{DOU} = 100 \times \left(1 - \frac{\text{RMSE}}{\text{PPFD}_{\text{avg}}}\right)$$

4. **Mean Absolute Deviation (MAD):** MAD measures the average absolute difference between each PPFD value and the average PPFD:
$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n} |P_i - \text{PPFD}_{\text{avg}}|$$

5. **Coefficient of Variation (CV):** CV is the ratio of the standard deviation ($\sigma$) to the average PPFD, expressed as a percentage:
$$\text{CV} = 100 \times \frac{\sigma}{\text{PPFD}_{\text{avg}}}$$
where the standard deviation, $\sigma$, is calculated as:
$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(P_i - \text{PPFD}_{\text{avg}})^2}{n}}$$
Note: In this specific case, since we are calculating sample statistics and using all data points, the sample standard deviation and population standard deviation are equivalent.

## 4.2 Cost Analysis

A preliminary cost analysis reveals that using 127 Samsung COBs at $11.81352 per COB results in a total LED cost of $1,500.32. In contrast, employing 210 Samsung Gen 2 modules at $26.56317 per module results in a cost of $5,578.27. This indicates that the conventional system's LED cost is approximately 3.7 times higher than that of the novel design.

# 5 Optimization Strategy

The enhanced performance and cost-effectiveness of the novel lighting system are achieved via a global optimization algorithm based on differential evolution. The design optimizes crop yield by adjusting the LED intensities while meeting PPFD targets and ensuring uniformity. Key factors in the optimization model include LED type, spacing, geometric arrangement, and luminous flux assignment.

The optimization adjusts a 15-element parameter vector:

- **COB Layer Intensities (elements 0-5):** Luminous flux for each of the six COB layers.
- **LED Strip Group Intensities (elements 6-10):** Luminous flux for the LED strips.
- **Corner COB Intensities (elements 11-14):** Luminous flux for the four corner COBs.

The objective function to be minimized is given by:

$$\text{Objective} = w_{rmse} \cdot \text{RMSE} + w_{penalty} \cdot \text{Penalty}$$

where RMSE is computed over the PPFD distribution and the penalty enforces a minimum DOU of 96% and restricts average PPFD to within the target and target+250 µmol/m$^2$/s range.

Figure 7 illustrates a potential enhancement of the novel system in which LED strips are integrated into the gaps between COBs to further improve uniformity.

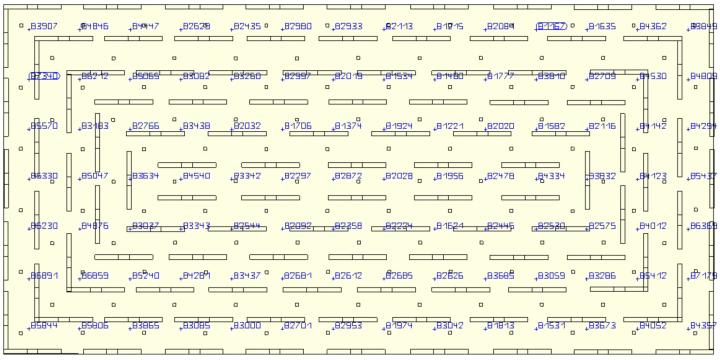

Figure 7: Novel Horticultural Lighting System Arrangement with LED Strips Disposed in the Gaps Between COBs.

# 6 Global Optimization Algorithm for Enhanced Uniformity

To refine the intensity settings, we employ a global optimization algorithm using differential evolution. This algorithm iteratively updates a 15-element parameter vector to minimize the objective function described above.

## 6.1 Parameterization

The parameter vector corresponds to the luminous flux (in lumens) assigned to each LED group as detailed in the previous section.

## 6.2 Objective Function

The objective function is defined as:

$$\text{Objective} = w_{rmse} \cdot \sqrt{\frac{1}{N}\sum_{i=1}^{N}(PPFD_i - PPFD_{target})^2} + w_{penalty} \cdot \text{Penalty}$$

with penalty terms enforcing a DOU threshold (96%) and a PPFD constraint (target to target+250 µmol/m$^2$/s). The full calculation requires running the complete radiosity simulation (via the `simulate_lighting` function).

### 6.3 The `simulate_lighting` Function: A Radiosity-Based Model

The `simulate_lighting` function implements the radiosity method to simulate light distribution. Its key steps include:

- **Lambertian Emission Model:** Each COB is assumed to follow $I(\theta) = I_0 \cos(\theta)$.
- **Surface Subdivision:** All surfaces (walls, ceiling, floor) are divided into patches.
- **Form Factor Calculation:** The fraction of energy exchanged between patches is computed.
- **Iterative Radiosity Solution:** The radiosity values are updated over a fixed number of bounces.
- **PPFD Calculation:** Direct and reflected irradiances are summed and converted to PPFD.

The detailed Python code for these steps is provided in Listings 1–6. (For clarity, a pseudocode summary is presented in Appendix B.)

```python
import numpy as np
import math

def calculate_direct_irradiance(light_positions, light_fluxes, X, Y, H, luminous_efficacy):
    """Calculates the direct irradiance on the floor from a set of LEDs."""
    direct_irr = np.zeros_like(X, dtype=np.float64)
    for (pos, lum) in zip(light_positions, light_fluxes):
        P = lum / luminous_efficacy  # Radiant flux (W)
        x0, y0, z0 = pos
        dx = X - x0
        dy = Y - y0
        dz = -z0  # Floor is at z=0
        dist2 = dx*dx + dy*dy + dz*dz
        dist = np.sqrt(dist2)
        with np.errstate(divide='ignore', invalid='ignore'):
            cos_th = -dz / dist  # Cosine of angle
        cos_th[cos_th < 0] = 0  # Only downward light
        with np.errstate(divide='ignore'):
            E = (P / math.pi) * (cos_th / dist2)
        direct_irr += np.nan_to_num(E)  # Accumulate irradiance
    return direct_irr
```

Listing 1: Direct Irradiance Calculation (Python)

```python
def subdivide_wall(x0, y0, z0, width, height, nx, ny, normal):
    """Subdivides a wall into rectangular patches."""
    patch_centers = []
    patch_areas = []
    patch_normals = []
    dx = width / nx
    dy = height / ny
    area = dx * dy
    for i in range(nx):
        for j in range(ny):
            cx = x0 + (i + 0.5) * dx
            cy = y0 + (j + 0.5) * dy
            cz = z0
            patch_centers.append((cx, cy, cz))
            patch_areas.append(area)
            patch_normals.append(normal)
    return patch_centers, patch_areas, patch_normals
```

Listing 2: Surface Subdivision (Python - Wall Example)

```python
def calculate_form_factor(patch_center_i, normal_i, patch_center_j, normal_j, area_j):
    """Calculates the form factor between two patches."""
    dd = np.array(patch_center_j) - np.array(patch_center_i)
    dist2 = np.dot(dd, dd)
    if dist2 < 1e-12:
        return 0.0
    dist = np.sqrt(dist2)
    cos_i = np.dot(normal_i, dd) / (dist * np.linalg.norm(normal_i))
    cos_j = np.dot(normal_j, -dd) / (dist * np.linalg.norm(normal_j))
```

```
    if cos_i < 0 or cos_j < 0:
        return 0.0
    ff = (cos_i * cos_j * area_j) / (math.pi * dist2)
    return ff
```

Listing 3: Form Factor Calculation (Python)

```
def calculate_radiosity(patch_centers, patch_areas, patch_normals, patch_refl,
                        light_positions, light_fluxes, luminous_efficacy, num_bounces):
    """Calculates the radiosity of each patch using an iterative method."""
    Np = len(patch_centers)
    patch_rad = np.zeros(Np)
    # 1. Initialize radiosity (direct irradiance on patches)
    patch_direct = np.zeros(Np)
    for ip in range(Np):
      pc = patch_centers[ip]
      n = patch_normals[ip]
      accum = 0.0
      for (lp, lum) in zip(light_positions, light_fluxes):
        P = lum/luminous_efficacy
        dx = pc[0]-lp[0]
        dy = pc[1]-lp[1]
        dz = pc[2]-lp[2]
        dist2 = dx*dx + dy*dy + dz*dz
        if dist2 < 1e-12:
          continue
        dist = math.sqrt(dist2)
        dd = np.array([dx, dy, dz])
        dist = np.linalg.norm(dd)
        cos_th_led = -dz/dist
        if cos_th_led < 0:
          continue
        E_led = (P/math.pi)*(cos_th_led/(dist2))
        cos_in_patch = np.dot(-dd, n)/(dist*np.linalg.norm(n))
        if cos_in_patch < 0:
          cos_in_patch = 0
        accum += E_led * cos_in_patch
      patch_direct[ip] = accum
    patch_rad = patch_direct
    # 2. Iterative radiosity calculation
    for bounce in range(num_bounces):
        new_flux = np.zeros(Np)
        for j in range(Np):
            if patch_refl[j] <= 0:
                continue
            outF = patch_rad[j] * patch_areas[j] * patch_refl[j]
            for i in range(Np):
                if i == j:
                    continue
                ff = calculate_form_factor(patch_centers[j], patch_normals[j],
                                           patch_centers[i], patch_normals[i],
                                           patch_areas[i])
                new_flux[i] += outF * ff
        patch_rad = patch_direct + new_flux / patch_areas
    return patch_rad
```

Listing 4: Iterative Radiosity Solution (Python)

```
def calculate_floor_ppfd(X, Y, patch_centers, patch_areas, patch_normals,
                         patch_rad, patch_refl, light_positions, light_fluxes,
                         luminous_efficacy, conversion_factor):
    """Calculates the total PPFD (direct + reflected) on the floor."""
    direct_irr = calculate_direct_irradiance(light_positions, light_fluxes,
                                             X, Y, H, luminous_efficacy)
    reflect_irr = np.zeros_like(X, dtype=np.float64)
    floor_pts = np.stack([X.ravel(), Y.ravel(), np.zeros_like(X.ravel())], axis=1)
    floor_n = np.array([0, 0, 1], dtype=float)
    for p in range(len(patch_centers)):
        outF = patch_rad[p] * patch_areas[p] * patch_refl[p]
        if outF < 1e-15:
            continue
        pc = patch_centers[p]
        n = patch_normals[p]
```

```
        dv = floor_pts - pc
        dist2 = np.einsum('ij,ij->i', dv, dv)
        dist = np.sqrt(dist2)
        cos_p = np.einsum('ij,j->i', dv, n) / (dist * np.linalg.norm(n) + 1e-15)
        cos_f = np.einsum('ij,j->i', -dv, floor_n) / (dist + 1e-15)
        cos_p[cos_p < 0] = 0
        cos_f[cos_f < 0] = 0
        ff = (cos_p * cos_f) / (math.pi * dist2 + 1e-15)
        cell_flux = outF * ff
        cell_area = FLOOR_GRID_RES * FLOOR_GRID_RES
        cell_irr = cell_flux / cell_area
        reflect_irr.ravel()[:] += np.nan_to_num(cell_irr)
    floor_irr = direct_irr + reflect_irr
    floor_ppfd = floor_irr * conversion_factor
    return floor_ppfd
```

Listing 5: PPFD Calculation on Floor (Python)

```
def simulate_lighting(x, fixed_base_height, plot_result=False):
    """
    Simulates the lighting in the grow room using radiosity.
    Args:
        x: 15-element parameter vector (LED intensities)
           (First 6: COB layers, next 5: LED strips, last 4: COB corners)
        fixed_base_height: Height of the base LEDs above the floor.
        plot_result: (Optional) Whether to create visualization plots
    Returns: average_ppfd, rmse, dou
    """
    # 1. Define Room Geometry and Parameters (Example Values)
    L = \SI{2.43}{m}  % Room Length
    W = \SI{1.21}{m}  % Room Width
    H = \SI{2.13}{m}  % Room Height
    FLOOR_GRID_RES = \SI{0.01}{m}
    WALL_SUBDIVS_X = 20
    WALL_SUBDIVS_Y = 10
    # 2. Define LED positions (using a function setup_lights)
    light_positions, light_fluxes = setup_lights(x, fixed_base_height)
    # 3. Create the floor grid:
    x_coords = np.arange(0, L, FLOOR_GRID_RES)
    y_coords = np.arange(0, W, FLOOR_GRID_RES)
    X, Y = np.meshgrid(x_coords, y_coords)
    % 4. Define surface reflectances:
    refl_wall = 0.8
    refl_ceil = 0.8
    refl_floor = 0.2
    % 5. Subdivide surfaces into patches:
    wall_x0_centers, wall_x0_areas, wall_x0_normals = subdivide_wall(
        0, 0, 0, W, H, WALL_SUBDIVS_Y, WALL_SUBDIVS_X, (-1, 0, 0)
    )
    wall_x1_centers, wall_x1_areas, wall_x1_normals = subdivide_wall(
        L, 0, 0, W, H, WALL_SUBDIVS_Y, WALL_SUBDIVS_X, (1, 0, 0)
    )
    wall_y0_centers, wall_y0_areas, wall_y0_normals = subdivide_wall(
        0, 0, 0, L, H, WALL_SUBDIVS_X, WALL_SUBDIVS_Y, (0, -1, 0)
    )
    wall_y1_centers, wall_y1_areas, wall_y1_normals = subdivide_wall(
        0, W, 0, L, H, WALL_SUBDIVS_X, WALL_SUBDIVS_Y, (0, 1, 0)
    )
    ceil_centers, ceil_areas, ceil_normals = subdivide_wall(
        0, 0, H, L, W, WALL_SUBDIVS_X, WALL_SUBDIVS_Y, (0, 0, -1)
    )
    floor_centers, floor_areas, floor_normals = subdivide_wall(
        0, 0, 0, L, W, WALL_SUBDIVS_X, WALL_SUBDIVS_Y, (0, 0, 1)
    )
    % 6. Combine all patches:
    patch_centers = (wall_x0_centers + wall_x1_centers + wall_y0_centers +
    wall_y1_centers + ceil_centers + floor_centers)
    patch_areas = (wall_x0_areas + wall_x1_areas + wall_y0_areas + wall_y1_areas +
    ceil_areas + floor_areas)
    patch_normals = (wall_x0_normals + wall_x1_normals + wall_y0_normals +
    wall_y1_normals + ceil_normals + floor_normals)
    patch_refl = ([refl_wall] * (len(wall_x0_centers) + len(wall_x1_centers) + len(
    wall_y0_centers) + len(wall_y1_centers)) +
                  [refl_ceil] * len(ceil_centers) +
```

```
                    [refl_floor] * len(floor_centers))
    % 7. Calculate radiosity:
    num_radiosity_bounces = 5
    luminous_efficacy = \SI{150}{\lumen\per\watt}
    patch_radiosity = calculate_radiosity(patch_centers, patch_areas, patch_normals,
    patch_refl,
                                          light_positions, light_fluxes, luminous_efficacy
    , num_radiosity_bounces)
    % 8. Calculate PPFD on the floor:
    conversion_factor = 0.0171
    floor_ppfd = calculate_floor_ppfd(X, Y, patch_centers, patch_areas, patch_normals,
                                      patch_radiosity, patch_refl, light_positions,
    light_fluxes,
                                      luminous_efficacy, conversion_factor)
    % 9. Calculate metrics:
    average_ppfd = np.mean(floor_ppfd)
    rmse = np.sqrt(np.mean((floor_ppfd - average_ppfd)**2))
    dou = 100 * (1 - (rmse / average_ppfd))
    % 10. (Optional) Visualization:
    if plot_result:
        import matplotlib.pyplot as plt
        fig, ax = plt.subplots(figsize=(8,6))
        im = ax.imshow(floor_ppfd, extent=[0, L, 0, W], origin='lower', cmap='viridis')
        ax.set_xlabel("X (m)")
        ax.set_ylabel("Y (m)")
        ax.set_title(f"PPFD Distribution (\SI{}{\micro\mol\per\m^2\per\second})")
        cbar = fig.colorbar(im)
        plt.show()
    return average_ppfd, rmse, dou
```

Listing 6: Overall simulate lighting Function (Python - Simplified)

```
import numpy as np
def objective_function(x, target_ppfd, fixed_base_height, rmse_weight=1.0,
    dou_penalty_weight=50.0):
    """
    Calculates the objective function value.
    Args:
        x (np.ndarray): The 15-element parameter vector (LED intensities).
        target_ppfd (float): The target PPFD value.
        fixed_base_height (float): Height of the LEDs above the floor.
        rmse_weight (float): Weight for the RMSE term.
        dou_penalty_weight (float): Weight for the DOU penalty.
    Returns:
        float: The objective function value.
    """
    avg_ppfd, rmse, dou = simulate_lighting(x, fixed_base_height, plot_result=False)
    penalty = 0.0
    if dou < 96:
        penalty = dou_penalty_weight * (96 - dou)**2
    obj_val = rmse_weight * rmse + penalty
    print(f"Obj: {obj_val:.3f} | RMSE: {rmse:.3f}, DOU: {dou:.3f}, Penalty: {penalty:.3f}
    ")
    return obj_val
```

Listing 7: Objective Function (Python - Simplified)

```
from scipy.optimize import differential_evolution, NonlinearConstraint
import numpy as np

def constraint_lower(x, target_ppfd, fixed_base_height):
    avg_ppfd, _, _ = simulate_lighting(x, fixed_base_height, plot_result=False)
    return avg_ppfd - target_ppfd  % Must be >= 0

def constraint_upper(x, target_ppfd, fixed_base_height):
    avg_ppfd, _, _ = simulate_lighting(x, fixed_base_height, plot_result=False)
    return target_ppfd + 250 - avg_ppfd  % Must be >= 0

def optimize_lighting_global(target_ppfd, fixed_base_height):
    """
    Use differential evolution for global optimization.
    """
    bounds = [(1000.0, 15000.0)] * 15  % Bounds for each parameter
```

```
    nc_lower = NonlinearConstraint(lambda x: constraint_lower(x, target_ppfd,
    fixed_base_height), 0, np.inf)
    nc_upper = NonlinearConstraint(lambda x: constraint_upper(x, target_ppfd,
    fixed_base_height), 0, np.inf)
    constraints = [nc_lower, nc_upper]
    result = differential_evolution(
        lambda x: objective_function(x, target_ppfd, fixed_base_height),
        bounds,
        constraints=constraints,
        maxiter=1000,
        popsize=15,
        recombination=0.7,
        disp=True
    )
    return result
```

Listing 8: Optimization with Differential Evolution (Python)

### 6.4 Integration with Radiosity

The optimization algorithm iteratively interacts with the radiosity simulation via the `simulate_lighting` function:

1. **Initial Guess:** Start with a random initial guess for LED intensities.
2. **Radiosity Simulation:** Compute average PPFD, RMSE, and DOU by running `simulate_lighting`.
3. **Objective Evaluation:** Calculate the objective function value.
4. **Parameter Update:** Differential Evolution updates the parameter vector.
5. **Repeat:** Continue until convergence.

## 7 Conclusion

This study demonstrates that the novel horticultural lighting system utilizing COB LEDs arranged in a centered square sequence achieves superior PPFD uniformity and lower material costs compared to conventional designs. The approach leverages a custom radiosity model—used for initial intensity assignment and validated by DIALux—and a global optimization algorithm based on differential evolution. These findings confirm the technical and economic advantages of the novel design. Future work will involve experimental validation and further algorithmic refinement.

# A Supplementary Material: Full Code Listings

All detailed Python code used in the simulations and optimization is provided above in Listings 1–8.

# B Pseudocode Summary of the Simulation and Optimization Process

Below is a high-level pseudocode summary for clarity:

```
Define room geometry and surface reflectances.
For each LED in the array:
    Compute direct irradiance using Lambertian emission:
        I(theta) = I0 * cos(theta)
Divide all surfaces (walls, ceiling, floor) into patches.
For each patch:
    Initialize radiosity using direct irradiance.
For a set number of bounces:
    For each patch j:
        Compute outgoing flux = radiosity * patch area * reflectance.
        For each other patch i:
            Compute form factor F_ij between patch j and patch i.
            Accumulate received flux: new_flux[i] += outgoing flux * F_ij.
    Update radiosity: B_i = direct irradiance + (new_flux[i] / patch_area[i]).
Create floor grid and compute:
    Direct irradiance from LEDs.
    Reflected irradiance from patches using form factors.
Sum both contributions and convert total irradiance to PPFD using conversion factor.
Compute metrics: Average PPFD, RMSE, and DOU.
Define objective function:
    Objective = (rmse_weight * RMSE) + (dou_penalty_weight * penalty)
    where penalty enforces DOU >= 96% and PPFD constraints.
Apply Differential Evolution:
    Set bounds for LED intensities.
    Use nonlinear constraints to ensure PPFD within target range.
    Iteratively update intensity vector until convergence.
Output optimized LED settings and corresponding PPFD metrics.
```

# References

[1] Sloane, N. J. A. (editor). The On-Line Encyclopedia of Integer Sequences, Sequence A001844. Published electronically at https://oeis.org/A001844.

[2] Samsung Gen 2 Horticulture Module Datasheet. Retrieved from https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/5337/Standard_Horticulture_Gen2_Rev1.0_11-18-21.pdf.

[3] Samsung COB Datasheet. Retrieved from https://www.bridgelux.com/sites/default/files/resource_media/Bridgelux%20DS416%20V22%20Gen%208%20Array%20Data%20Sheet%2020200622%20Rev%20A.pdf.