Module 3A - Geometric Raytracing

Omar Betancourt, Payton Goodrich, Emre Mengi

June 12, 2023

Contents

Objectives: Understanding basics of geometric raytracing, simulating the decomposed beams as rays.
Prerequisite Knowledge: N/A
Prerequisite Modules: 1A - Calculus, 1B - Linear Algebra, 2D - Optics, 3C - Generic Time Stepping
Difficulty: Intermediate
Summary: Student is introduced to light-based methods by using geometric raytracing to track rays in space and time and its interaction with various surfaces.

1 Theory

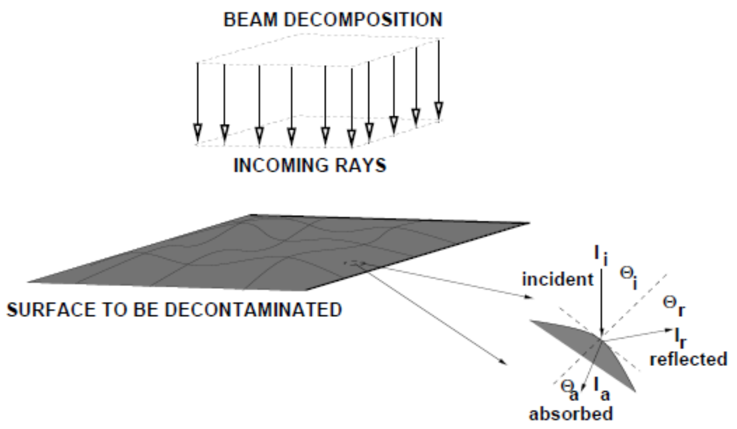

Figure 1.1: An electromagnetic pulse applied to a surface.

This work develops an efficient and rapid computational method to simulate a light pulse in order to evaluate the absorption characteristics of a surface. It is based on decomposition of a pulse into a groups of rays, which are then tracked as they progress towards the target contact surface. The algorithm computes the absorption at the point of contact and color codes it relative to the incoming irradiation. This allows one to rapidly quantify the decontamination uncertainty by identifying regions where the absorption is inadequate and serves as a guide for practitioners to ascertain where problems may occur a priori to experiments. Additionally, the reflections are calculated, and can be used in ascertaining safety to a bystander. The interest here is on the absorption of an initially coherent pulse (Figure 1.1), represented by multiple collimated (parallel) rays (initially forming a planar wave front), where each ray is a vector in the direction of the flow of energy (the rays are parallel to the initial wave's propagation vector). We make the following observations:

- It is assumed that the features of the surface to be irradiated are at least an order of magnitude larger than the wavelength of the incident radiation (essentially specular surfaces), therefore "geometrical" ray tracing theory is applicable, and is well-suited for the systems of interest. It is important to emphasize the regimes of validity of such a model are where the surface features are larger than the light wavelength. For example, if we were to use UV-rays ($10^{-8}m \leq \lambda \leq 4 \times 10^{7}m$), the features in this analysis would be assumed to possess scales larger than approximately $4 \times 10^{-6}m$. For systems containing features smaller than this, one can simply use the model as a qualitative guide.

- Ray-tracing is a method that is employed to produce rapid approximate solutions to wave equations for high-frequency/small-wavelength applications where the primary interest is in the overall propagation of energy.[1]

- Ray-tracing methods proceed by initially representing wave fronts by an array of discrete rays. *Thereafter, the problem becomes one of a primarily geometric character*, where one tracks the changing

[1]Resolving diffraction (which ray theory is incapable of describing) is unimportant for the applications of interest.

trajectories and magnitudes of individual rays which are dictated by the reflectivity and the Fresnel conditions (if a ray encounters a material interface).

- Ray-tracing methods are well-suited for computation of scattering in complex systems that are difficult to mesh/discretize, relative to procedures such as the Finite Difference Time Domain Method or the Finite Element Method.
- Other high frequency irradiation regimes can also be considered in the same manner, such as X-rays and gamma rays, provided that the scattering target has the appropriate (larger) lengthscale. Even in the case where this clear separation of length scales is not present, this model still provides valuable information on the propagation of the beam and the reflected response of the dispersed system.

1.1 Electromagnetic energy propagation

1.1.1 Beam-ray decomposition

In order to connect the concept of a ray with a pulse/beam, since $\bar{I}$ is the energy per unit area per unit time, we obtain the energy associated with an entire pulse/beam by multiplying the irradiance by the cross-sectional area of an initially coherent beam, $\bar{I}A^b$, where A^b is the cross-sectional area of the beam (comprising all of the rays). The energy per unit time (power) for a ray in the pulse/beam is then given by

$$I = \bar{I}A^r = \bar{I}A^b/N_r, \tag{1.1}$$

where N_r is the number of rays in the beam (Figure 1.1) and A_r can be considered the area associated with a ray. Essentially, rays are a mathematical construction/discretization of a pulse/beam We refer the reader to Gross [1], Zohdi [3-9] for details.

1.1.2 Reflection and absorption of rays

Following a framework found in Zohdi [3-9] for details, we consider a ray of light incident upon a material interface which produces a reflected ray and a transmitted/absorbed (refracted) ray (Figure 1.1), the amount of incident electromagnetic energy per unit time, power (I_i), that is reflected (I_r) is given by the total reflectance $\boldsymbol{I\!R} \overset{\text{def}}{=} \frac{I_r}{I_i}$, where $0 \leq \boldsymbol{I\!R} \leq 1$. $\boldsymbol{I\!R}$ is given by Equation 1.3, for unpolarized electromagnetic radiation. We have the following observations:

- The angle between the point of contact of a ray (Figure 1.1) and the outward normal to the surface at that point is the angle of incidence, θ_i. The classical reflection law states that the angle at which a ray is reflected is the same as the angle of incidence and that the incoming (incident, θ_i) and outgoing (reflected, θ_r) ray lays in the same plane, and $\theta_i = \theta_r$
- The classical refraction law states that, if the ray passes from one medium into a second one (with a different index of refraction), and, if the index of refraction of the second medium is less than that of the first, the angle the ray makes with the normal to the interface is always less than the angle of incidence, where $\hat{n} \overset{\text{def}}{=} \frac{v_i}{v_a} = \sqrt{\frac{\epsilon_a \mu_a}{\epsilon_1 \mu_i}} = \frac{\sin\theta_i}{\sin\theta_a}, \theta_a$ being the angle of the absorbed ray (Figure 1.1), v_a is the propagation speed in the absorbing medium, v_i is the propagation speed in the incident medium, ϵ_a is the electric permittivity of the absorbing medium, μ_a magnetic permeability of the absorbing medium, ϵ_i is the electric permittivity in the incident medium and μ_i magnetic permeability in the incident medium.
- Recall, all electromagnetic radiation travels, in a vacuum, at the speed $c \approx 2.99792458 \times 10^8 \pm 1.1 m/s$. In any another medium $v = \frac{1}{\sqrt{\epsilon\mu}}$ for electromagnetic waves.[2]

[2] The free space electric permittivity is $\epsilon_o = \frac{1}{c^2\mu_o} = 8.8542\times10^{-12} CN^{-1}m^{-1}$and the free space magnetic permeability is $\mu_o = 4\pi \times 10^{-7} WbA^{-1}m^{-1} = 1.2566 \times 10^{-6} WbA^{-1}m^{-1}$.

- We define $\hat{n}$ as the ratio of the refractive indices of the ambient (incident) medium (n_i) and absorbing medium (n_a), $\hat{n} = n_a/n_i$, where $\hat{\mu}$ is the ratio of the magnetic permeabilities of the surrounding incident medium (μ_i) and scattering/absorbing medium (μ_a), $\hat{\mu} = \mu_a/\mu_i$. Thus, we have

$$\hat{n} = \frac{n_a}{n_i} = \sqrt{\frac{\epsilon_a \mu_a}{\epsilon_i \mu_i}} \Rightarrow \epsilon_a \mu_a = (\hat{n})^2 \epsilon_i \mu_i. \tag{1.2}$$

- For a pulse of light, the reflectivity $\boldsymbol{I\!R}$ can be shown to be (see [1] for example)

$$\boldsymbol{I\!R} = \frac{I_r}{I_i} = \frac{1}{2}\left(\left(\frac{\hat{n}^2\cos\theta_i - \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}{\frac{\hat{n}^2}{\mu}\cos\theta_i + \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}\right)^2 + \left(\frac{\cos\theta_i - \frac{1}{\hat{\mu}}\left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}{\cos\theta_i + \frac{1}{\hat{\mu}}\left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}\right)^2\right) \tag{1.3}$$

  where I_i is the incoming irradiance, I_r the reflected irradiance, $\hat{n}$ is the ratio of the refractive indices of the of absorbing (n_a) and incident media (n_i), where the refractive index is defined as the ratio of the speed of light in a vacuum (c) to that of the medium (v), where the speed of electromagnetic waves is $c = \frac{1}{\sqrt{\epsilon_o \mu_o}}$, where ϵ is the electric permittivity and μ is the magnetic permeability.

- We consider applications with non-magnetic media and frequencies where the magnetic permeability is virtually the same for both the incident medium (usually the atmosphere) and the scattering/absorbing medium. Thus, for the remainder of the work, we shall take $\hat{\mu} = 1 (\mu_o = \mu_i = \mu_a)$, thus

$$\hat{n} = \frac{n_a}{n_i} = \sqrt{\frac{\epsilon_a \mu_a}{\epsilon_i \mu_i}} \Rightarrow \epsilon_a \mu_a = (\hat{n})^2 \epsilon_i \mu_i \Rightarrow \epsilon_a = (\hat{n})^2 \epsilon_i \tag{1.4}$$

  This yields

$$\boldsymbol{I\!R} = \frac{I_r}{I_i} = \frac{1}{2}\left(\left(\frac{\hat{n}^2\cos\theta_i - \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}{\hat{n}^2\cos\theta_i + \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}\right)^2 + \left(\frac{\cos\theta_i - \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}{\cos\theta_i + \left(\hat{n}^2 - \sin^2\theta_i\right)^{\frac{1}{2}}}\right)^2\right) \tag{1.5}$$

- Notice that as $\hat{n} \rightarrow 1$ we have complete absorption, while as $\hat{n} \rightarrow \infty$ we have complete reflection. The total amount of absorbed power by the material is $(1 - \boldsymbol{I\!R}) I_i$.

The 'Optics' module supplies the theory underpinning electromagnetic wave propagation and rays, therefore, refer to that module for details.

Remark: We now recall Equation 1.1 connects the concept of a ray with a pulse/beam and observe:

- Since $\bar{I}$ is the energy per unit area per unit time, we obtain the energy associated with an entire pulse/beam by multiplying the irradiance by the cross-sectional area of an initially coherent beam, $\bar{I}A^b$, where A^b is the cross-sectional area of the beam (comprising all of the rays).
- The energy per unit time (power) for a ray in the pulse/beam is then given by $I = \bar{I}A^r = \bar{I}A^b/N_r$, where N_r is the number of rays in the beam (Figure 1.1) and A^r can be considered the area associated with a ray.
- The reflection relation, Equation 1.5 can then be used to compute changes in the magnitude of the reflected rays (and the amount absorbed), with directional changes given by the laws of reflection.

We refer the reader to Gross [1] and Zohdi [3-9] for details.

2 Example

From this point forth, we assume that the ambient medium behaves as a vacuum. Accordingly, there are no energetic losses as the rays move through the surrounding medium.

2.1 Tracking of beam-decomposed rays

Starting at $t = 0$ and ending at $t = T$, the simple overall algorithm to track rays is as follows, at each time increment:

1. Check for intersections of rays with surfaces (hence a reflection), and compute the ray magnitudes and orientation if there are reflections (for all rays that are experiencing a reflection, $I_j^{ref}, j = 1, 2, \ldots Rays$)
2. Increment all ray positions ($\boldsymbol{r}_j(t + \Delta t) = \boldsymbol{r}_j(t) + \Delta t \boldsymbol{v}_j(t), j = 1, 2, \ldots,$ Rays)
3. Increment time forward ($t = t + \Delta t$) and repeat the process for the next time interval.

In order to capture all of the ray reflections that occur:

- The time step size Δt is dictated by the offset height of the source. A somewhat ad-hoc approach is to scale the time step size by the speed of ray propagation according to $\Delta t = \xi \frac{\mathcal{H}}{\|\boldsymbol{v}\|}$, where $\mathcal{H}$ is the height of the source and $0.0001 \leq \xi \leq 0.01$. Typically, the results are insensitive to ξ that are smaller than this range.
- Although outside the scope of this work, one can also use this algorithm to compute the thermal response by combining it with heat transfer equations via staggering schemes (Zohdi [3,6]).

2.2 Test surface

The discrete-ray approach is flexible enough to simulate a wide variety of systems. As a test surface, we consider a topology to be irradiated described by $F(x_1, x_2, x_3) = 0$. The outward surface normals, $\boldsymbol{n}$, needed during the scattering calculations, are easy to characterize by writing

$$\boldsymbol{n} = \frac{\nabla F}{\|\nabla F\|} \tag{2.1}$$

The components of the gradient are

$$\nabla F = \frac{\partial F}{\partial x_1}\boldsymbol{e}_1 + \frac{\partial F}{\partial x_2}\boldsymbol{e}_2 + \frac{\partial F}{\partial x_3}\boldsymbol{e}_3 \tag{2.2}$$

It is advantageous to write the surface in parametric form:

$$F(x_1, x_2, x_3) = G(x_1, x_2) - x_3 = 0 \tag{2.3}$$

which leads to

$$x_3 = G(x_1, x_2) \tag{2.4}$$

The gradient becomes

$$\nabla F = \frac{\partial G}{\partial x_1}\boldsymbol{e}_1 + \frac{\partial G}{\partial x_2}\boldsymbol{e}_2 - \boldsymbol{e}_3 \tag{2.5}$$

In order to determine whether a ray has made contact with a surface domain, one checks if the x_3 component of a ray ($\boldsymbol{r}_j$) is less that x_3 of the surface.

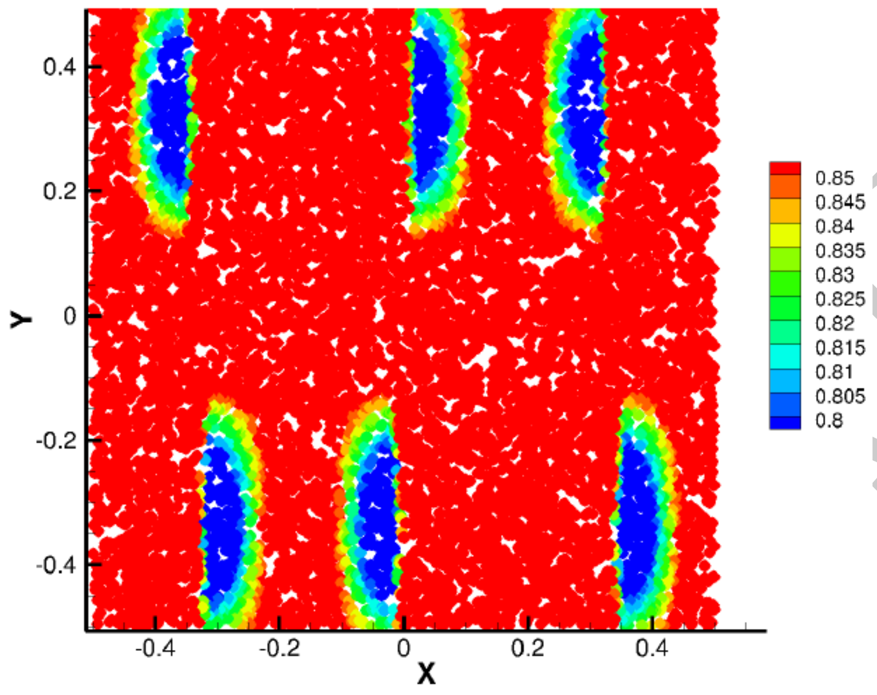

Figure 2.1: Top view for a surface with amplitude A = 0.3. Colors indicate the absorption, normalized by the incoming radiation level.

Table 2.1: The loss of absorption efficacy with contact surface amplitude oscillation (waviness).

Surface Amplitude	**Average Surface Absorption**
0.0	0.8888
0.1	0.8879
0.2	0.8808
0.3	0.8675
0.4	0.8507
0.5	0.8327
0.6	0.8126
0.7	0.7878
0.8	0.7679
0.9	0.7485
1.0	0.7315

2.3 Numerical/Quantitative examples

We have the following set up for a series of tests:

- The initial velocity vector for all initially collimated (parallel) rays comprising the beam was $v = (c, 0, 0)$, where $c = 3 \times 10^8 m/s$ is the speed of light in a vacuum.
- We used a parametrized test surface given by

$$x_3 = 2 + A\left(\sin\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right) \tag{2.6}$$

with $L_1 = L_2 = 1, \omega_1 = 1.5$ and $\omega_2 = 0.75$, where we vary A. We also added a flat cut-off so that the surface had a half-sine wave character (Figure 2.2).

- The number of rays in the beam were steadily increased from $N_r = 100, 200$, etc, until the results were insensitive to further refinements. This approach indicated that between approximately $9500 \leq N_r \leq 10000$ parallel rays in rectangular cross-sectional plane of the beam. The rays were randomly placed within the beam (Figures 1.1 and 2.2), to correspond to unpolarized incoming energy, and yielded stable results across the parameter study range.
- Figure 2.2 shows a sequence of frames of the detailed response of a surface to 10000 rays. Figure 2.1 shows a top view. Table 2.1 shows the steady loss of absorption efficacy with contact surface amplitude oscillation (waviness). The algorithm computes the absorption at the point of contact and color codes it relative to the incoming irradiation. This allows one to quickly quantify the decontamination across the topology of the structure.
- This approach also allows an analyst to explore nonuniform beam profiles, for example exponential central irradiance decay: $I(d) = I(d = 0)e^{-ad}$, where d is the distance from the center of the initial beam, where in the case of $a = 0$, one recaptures a flat beam, $I(d) = I(d = 0)$.[1]

[1]Note that algorithmically, we can the set total initial irradiance via $\sum_{i=1}^{N_r} I_i^{inc}(t = 0)\mathcal{A}_r = P$ Watts. To achieve this distribution, one would first place rays randomly in the plane, and then scale the individual I^{inc} by e^{-ad} and the normalized the average so that the total was P watts.

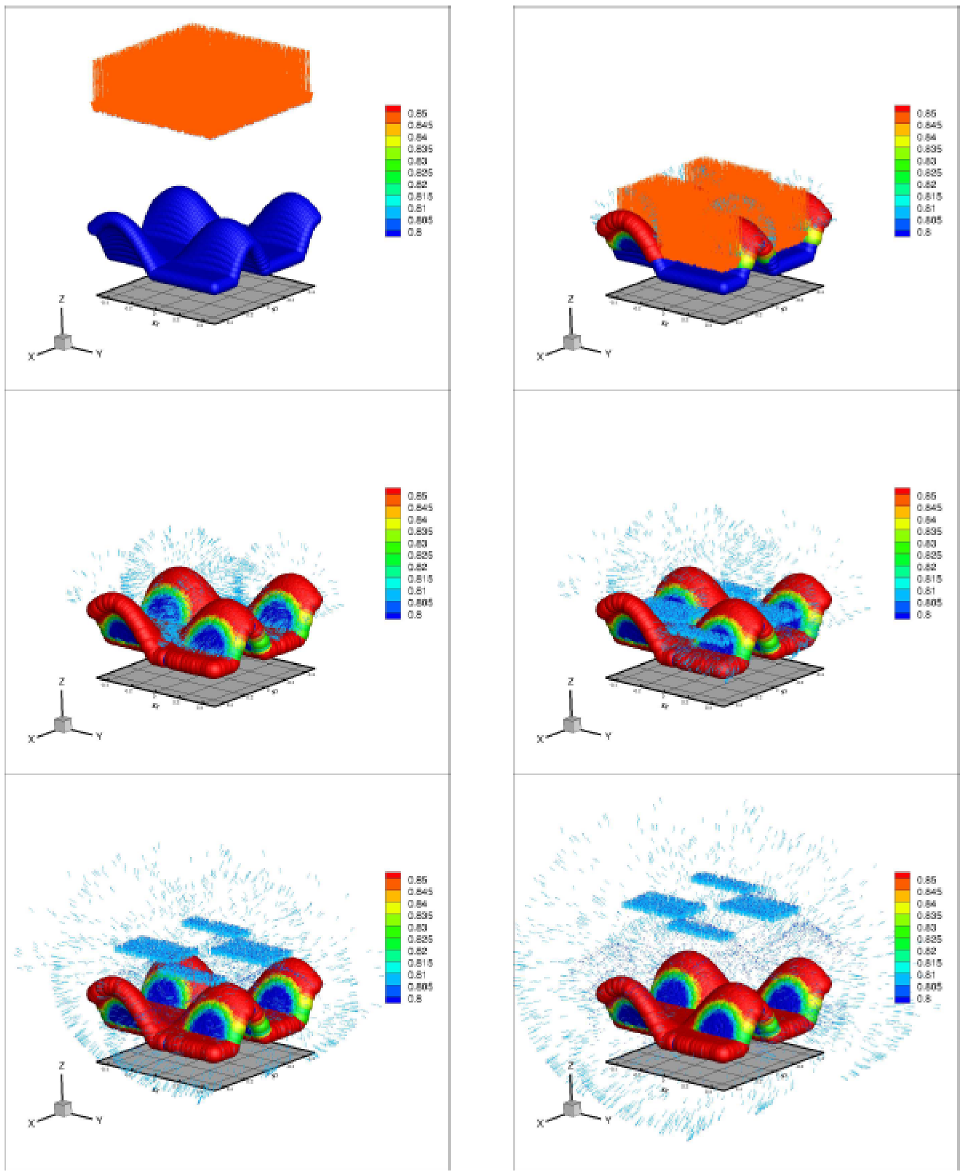

Figure 2.2: Sequence of frames for a surface with amplitude A = 0.3 (the amplitude was enhanced by a factor of 20 in graphics to more easily see the effects of the topology on absorption). Colors indicate the absorption, normalized by the incoming radiation level.

3 Assignment

In this assignment, you will be introduced to the basics of light-based simulation methods. If you use Python or another language for simplifying answers or evaluating specific numbers, clearly set up the equations before showing your numerical results.

Problem 1: Theory-Based Exercises

Answer the following questions *prior* to coding the assignment to better understand the background physics and mathematics that govern the given models. You **may** solve these problems by hand **and/or** using computational tools such as `Python` etc. Please include all handwritten work and code used to solve each problem.

The template code to complete the project is given here on `GitHub`.

Problem 1.1

For a parametrized test surface given by

$$x_3 = 2 + A\left(\sin\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right) \tag{3.1}$$

Analytically solve for the gradient of the surface.

Problem 1.2

Write down the Forward Euler equation for time discretization. Explain all the terms.

Problem 1.3

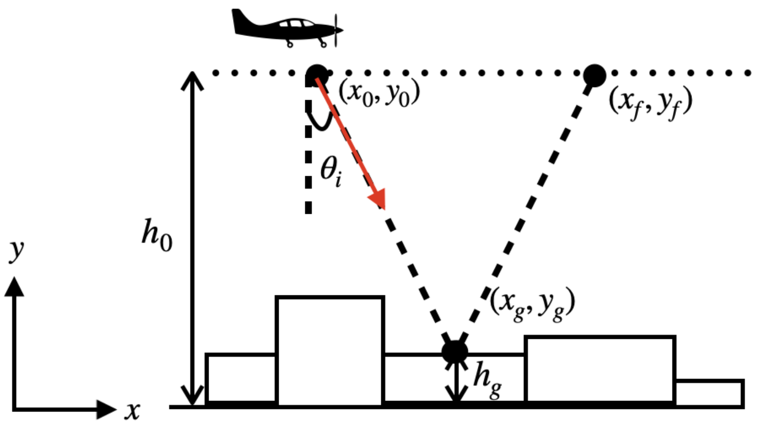

Figure 3.1: Drone mapping a surface.

Consider a drone scanning a surface, seen in Figure 3.1. Denote the position of the drone as (x_0, y_0), ground as (x_g, y_g), and the final position of the returned ray as (x_f, y_f). If the lidar system on the drone emits a ray with an angle θ_i, analytically solve for the final position of the ray, (x_f, y_f), in terms of h_0, h_g, θ_i, drone (x_0, y_0) and ground positions (x_g, y_g). Also, calculate the time it takes for the light ray to reach (x_f, y_f) if $||v_{ray}|| = c$. Assume the ground is where the ray intersects with the surface is flat.

Problem 2: Coding Exercises

Use the given python notebook template to complete the following coding exercises. For the given test surfaces , you are going to simulate a light beam positioned directly above the surface, with v = [0,0,-c].

Problem 2.1

Define the constants used in the simulation. Use the variable glossary at the end of the assignment.

Problem 2.2

Set up the first surface (surface A). The test surface A is described in mathematical equation:

$$x_3 = 1.5 + (0.3)\left(\sin\frac{2(1.5)\pi x_1}{1}\right) + \cos\left(\frac{2(0.75)\pi x_2}{1}\right) \tag{3.2}$$

Problem 2.3

Use Forward Euler time-stepping scheme to simulate a beam hitting the given test surfaces. (surface A).

Problem 2.4

Set up the second surface (surface B). The test surface B is described in mathematical equation:

$$x_3 = \sqrt{\left(5 - \sqrt{x_1^2 + x_2^2}\right)^2 - 4} - 1.75 \tag{3.3}$$

Problem 2.5

Use Forward Euler time-stepping scheme to simulate a beam hitting the given test surfaces. (surface B).

Problem 3: Analyzing Your Results

Answer the following questions about the code you created.

Problem 3.1

Discuss your observations from the raytracing animations. *Hint: When do the rays impact the surface? Does the time step capture the reflections properly? etc.)*

Problem 3.2

Try using different time-steps to observe the effect on ray reflections. Report any findings.

Table 3.1: Model Parameters.

Symbol	Type	Units	Value	Description
μ_o	Scalar	Wb/Am	$4\pi \times 10^{-7}$	free-space magnetic permeability
ϵ_o	Scalar	C/Nm	8.8542×10^{-7}	free-space electric permittivity
c	Scalar	m/sr	$\frac{1}{\mu_o \dot{\epsilon}_o}$	speed of light
$\hat{\mu}$	Scalar	none	$\frac{\mu_a}{\mu_i} = 1$	magnetic permeability ratio (incident/absorbing)
$\hat{\epsilon}$	Scalar	none	$\frac{\epsilon_a}{\epsilon_i} = 25$	electric permittivity ratio (incident/absorbing)
$\hat{n}$	Scalar	none	$\frac{n_a}{n_i} = 1.5$	refractive index ratio (incident/absorbing)
v_{beam}	Vector	m/s	$[0 \quad 0 \quad -c]$	beam initial velocity
h_{beam}	Scalar	m	3	beam initial height
x_1	Scalar	m	$-0.5 \leq x_1 \leq 0.5$	surface length limits
x_2	Scalar	m	$-0.5 \leq x_1 \leq 0.5$	surface width limits
t_f	Scalar	s	7.5×10^{-9}	final time for simulation
Δt	Scalar	s	8.66×10^{-13}	time step size

4 Solution

Problem 1: Theory-Based Exercises

Problem 1.1

Components of the gradient can be computed as:

$$\nabla F = \frac{\partial F}{\partial x_1}\boldsymbol{e}_1 + \frac{\partial F}{\partial x_2}\boldsymbol{e}_2 + \frac{\partial F}{\partial x_3}\boldsymbol{e}_3 \tag{4.1}$$

$G(x_1, x_2)$ in parametric can be computed by:

$$F(x_1, x_2, x_3) = G(x_1, x_2) - x_3 = 0 \tag{4.2}$$

The gradient can be rewritten as:

$$\nabla F = \frac{\partial G}{\partial x_1}\boldsymbol{e}_1 + \frac{\partial G}{\partial x_2}\boldsymbol{e}_2 - \boldsymbol{e}_3 \tag{4.3}$$

For the parametrized test surface given:

$$\begin{aligned} x_3 &= 2 + A\sin\left(\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right) \\ \nabla F &= \frac{\partial\left(2 + A\sin\left(\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right)\right)}{\partial x_1}\boldsymbol{e}_1 + \frac{\partial\left(2 + A\sin\left(\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right)\right)}{\partial x_2}\boldsymbol{e}_2 - \boldsymbol{e}_3 \\ &= \left(\frac{2A\omega_1\pi}{L_1}\right)\cos\left(\frac{2\omega_1\pi x_1}{L_1}\right)\sin\left(\frac{2\omega_2\pi x_2}{L_2}\right)\boldsymbol{e}_1 + \left(\frac{2A\omega_2\pi}{L_2}\right)\sin\left(\frac{2\omega_1\pi x_1}{L_1}\right)\cos\left(\frac{2\omega_2\pi x_2}{L_2}\right)\boldsymbol{e}_2 - \boldsymbol{e}_3 \end{aligned} \tag{4.4}$$

Problem 1.2

The forward Euler discretization for position and velocity terms are as follows:

$$r_i(t + \Delta t) \doteq r_i(t) + v_i(t)\Delta t \tag{4.5}$$

Since the speed of light is constant, the only PDE we are solving is to update our ray position.

Problem 1.3

The distance between the surface and the drone is $d = h_0 - h_g$. Given the ray is emitted at an angle θ_i when the drone is at (x_0, y_0). The drone collects the ray at (x_f, y_f). It can be seen that $y_f = y_0$.

The lateral distance traveled by the light is:

$$L = 2 * (h_0 - h_g)\tan\theta_i \tag{4.6}$$

Then, the final position of the drone is:

$$(x_f, y_f) = (x_0 + 2 * (h_0 - h_g)\tan\theta_i, y_0) \tag{4.7}$$

The time of flight for the ray can be calculated using the total distance traveled by the ray:

$$\begin{aligned} d_{tot} &= 2\sqrt{((h_0 - h_g)\tan\theta_i)^2 + (h_0 - h_g)^2} = 2\sqrt{(h_0 - h_g)^2\left(\tan^2\theta_i + 1\right)} \\ &= 2\sqrt{(h_0 - h_g)^2\sec^2\theta_i} = 2(h_0 - h_g)\sec\theta_i \end{aligned} \tag{4.8}$$

Then, the total time can be calculated using the speed of light:

$$t_{flight} = \frac{d_{tot}}{c} = \frac{2(h_0 - h_g)\sec\theta_i}{c} \tag{4.9}$$

Problem 2: Coding Exercises

Problem 2.1

Define the constants used in the simulation. Use the variable glossary at the end of the assignment.
Solution Cell Block:

```

################################ Define Constants ##########################################

#free-space magn. permeability [Wb/(Am)]
mu_0 = 4*np.pi*1e-7  #FILL IN HERE

#free-space elec. permittivity [C/(Nm)]
eps_0 = 8.8542*1e-12  #FILL IN HERE

#speed of light [m/s]
c = 1/np.sqrt(mu_0*eps_0) #FILL IN HERE

#mu_hat: ratio of magn. permeability scattering/absorbing (mu_a)...
mu_hat = 1 #FILL IN HERE

# electric permitivity (chosen)
eps_hat = 25 #FILL IN HERE

# refractive index
n_hat = 10 #FILL IN HERE
```

Problem 2.2

Set up the first surface (surface A).
Solution Cell Block:

```

######################### Set up the surface (SURFACE A)
    ##########################################

# set up surface parameters
L1 = 1
L2 = 1
w1 = 1.5
w2 = 0.75
A = 0.3

# set up limits of test surface
test_lim = 0.5;

# create grid of points using linspace
x1 = np.linspace(-test_lim,test_lim,10)
x2 = np.linspace(-test_lim,test_lim,10)

# create grid of points using meshgrid
[X1, X2] = np.meshgrid(x1,x2)

# reshape grid into a column vector
X1 = np.reshape(X1,[np.size(X1),1])
X2 = np.reshape(X2,[np.size(X2),1])

# write surface equation for surface a
X3 = 1.5 + A * np.sin(2 * w1 * np.pi * X1 / L1) * np.sin(2 * w2 * np.pi * X2 / L2) #FILL IN
    HERE

# Calculate normals

#surface A
#delF
```

```python
delGx1 = A*(2*w1*np.pi/L1)*np.cos(2*w1*np.pi*X1/L1)*np.sin(2*w2*np.pi*X2/L2) #FILL IN HERE
delGx2 = A*(2*w2*np.pi/L2)*np.sin(2*w1*np.pi*X1/L1)*np.cos(2*w2*np.pi*X2/L2) #FILL IN HERE

delFx1 = delGx1
delFx2 = delGx2
delFx3 = np.ones([np.size(delGx1),1])*-1

gradF = np.hstack([delFx1, delFx2, delFx3])
magdelF = np.reshape(np.linalg.norm(gradF,2,1),(np.size(X1),1))

normal = - gradF / magdelF

for i in range(X3.size): #uncomment for surface a NORMAL + SURFACE POINT FIX
    if X3[i]-1.5 < 0:
        X3[i] = 1.5
        normal[i,0] = 0
        normal[i,1] = 0
        normal[i,2] = 1


# plot surface
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(X1[:,0], X2[:,0], X3[:,0], linewidth=0.2, antialiased=True)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
# label plot
plt.title('Surface')
plt.show()
```

Problem 2.3

Use Forward Euler time-stepping scheme to simulate a beam hitting the given test surfaces. (surface A).
Solution Cell Block:

```python

######################### Create rays #########################################

beam_initvel = np.array([0,0,-c]) #initial velocity of beam
beam_N = 100 #number of rays
beam_initH = 2 #initial height
beam_N = round(np.sqrt(beam_N))**2 # updated number of rays to have it square
print('Updated number of rays is:',beam_N)

#full beam grid
[beam_posx1,  beam_posx2] = np.meshgrid(np.linspace(-0.5,0.5,round(np.sqrt(beam_N))),np.linspace(-0.5,0.5,round(np.sqrt(beam_N))))

# reshape grid into a column vector
beam_posx1 = np.reshape(beam_posx1,[np.size(beam_posx1),1])
beam_posx2 = np.reshape(beam_posx2,[np.size(beam_posx2),1])
beam_posx3 = np.ones([beam_N,1]) * beam_initH

# set up initial velocity of beam
beam_vel1 = np.ones([beam_N,1])  * beam_initvel[0]
beam_vel2 = np.ones([beam_N,1])  * beam_initvel[1]
beam_vel3 = np.ones([beam_N,1])  * beam_initvel[2]

# Time-stepping
totaltime = 0.00000001 # simulation time
xi = 0.00000005 # time-stepping constant
dt = xi * beam_initH / np.sqrt(c) # time-step
Ntimestep = round(totaltime / dt) # number of time-steps

# initialize variables
IR = np.zeros([np.size(beam_posx1),1])
```

```python
theta_i = np.zeros([np.size(beam_posx1),1])
X1_final = np.empty([np.size(beam_posx1),1])
X2_final = np.empty([np.size(beam_posx1),1])
X3_final = np.empty([np.size(beam_posx1),1])

# initialize animation variables
rayAnim = list() # list of ray objects
rayHit = list() # list of ray hit objects


for i in range(Ntimestep): # time-stepping loop

    #time-step announcer (every 100 time-steps)
    if i % 100 == 0:
        print('Time-step:',i,'/',Ntimestep)


    #update position
    beam_posx1 = beam_posx1 + beam_vel1 * dt
    beam_posx2 = beam_posx2 + beam_vel2 * dt
    beam_posx3 = beam_posx3 + beam_vel3 * dt

    for k in range(np.size(beam_posx1)): # loop over rays
        #check if below surface
        X3_calc = 1.5 + A*np.sin(2*w1*np.pi*beam_posx1[k]/L1)*np.sin(2*w2*np.pi*beam_posx2[k
    ]/L2); #FILL IN HERE
        for z in range(np.size(X3_calc)):
            if X3_calc-1.5 < 0: # if below surface, set to surface
                X3_calc = 1.5

        if beam_posx1[k] <= test_lim and beam_posx1[k] >= -test_lim and beam_posx2[k] <=
    test_lim and beam_posx2[k] >= -test_lim: # if within test limits
           if X3_calc>beam_posx3[k]: # if below surface
               if IR[k] == 0: # if not yet reflected

                    # find normal and ray velocity vector
                    # Calculate normals
                    # delF
                    # surface a
                    delGx1 = A*(2*w1*np.pi/L1)*np.cos(2*w1*np.pi*beam_posx1[k]/L1)*np.sin(2*
    w2*np.pi*beam_posx2[k]/L2) #FILL IN HERE
                    delGx2 = A*(2*w2*np.pi/L2)*np.sin(2*w1*np.pi*beam_posx1[k]/L1)*np.cos(2*
    w2*np.pi*beam_posx2[k]/L2) #FILL IN HERE

                    delFx1 = delGx1
                    delFx2 = delGx2
                    delFx3 = -1

                    gradF = np.hstack([delFx1, delFx2, delFx3])
                    magdelF = np.linalg.norm(gradF,2)

                    normal = - gradF / magdelF

                    # surface a
                    if X3_calc == 1.5:
                        normal[0] = 0
                        normal[1] = 0
                        normal[2] = 1

                    norm_k = normal # normal vector
                    beam_vel = np.hstack([beam_vel1[k], beam_vel2[k], beam_vel3[k]]) # ray
    velocity vector
                    theta_i = np.arccos(np.dot(norm_k,beam_vel)/c) # angle of incidence
                    if beam_vel3[k] <= 0: # if ray is going down
                        #record point of contact with surface
                        X1_final[k] = beam_posx1[k]
                        X2_final[k] = beam_posx2[k]
                        X3_final[k] = beam_posx3[k]

                        beam_vel = beam_vel - 2 * c * np.cos(theta_i) * norm_k # reflect ray

                        #update velocity
                        beam_vel1[k] = beam_vel[0]
                        beam_vel2[k] = beam_vel[1]
                        beam_vel3[k] = beam_vel[2]
                        #theta_i = np.pi - theta_i

                        #update IR
                        IR[k] = (np.abs(theta_i/n_hat) <= 1.)* \
                            (0.5*(((n_hat**2 * np.cos(theta_i) - (n_hat**2 - np.sin(theta_i)**2)**(0.5))/ \
                                    (n_hat**2 * np.cos(theta_i) + (n_hat**2 - np.sin(theta_i)**2)**(0.5)))**2 +\
                                   (((np.cos(theta_i) - (n_hat**2 - np.sin(theta_i)**2)**(0.5))/ \
                                     (np.cos(theta_i) + (n_hat**2 - np.sin(theta_i)**2)**(0.5)))**2)))+ \
                                (np.abs(theta_i/n_hat) > 1.)**1.



    if i % 50  == 0:

        #assign current loc, vel, and energy left in rays in an array

        rayInfo = np.hstack([beam_posx1, beam_posx2, beam_posx3, beam_vel1, beam_vel2, beam_vel3])

        # write into rayAnim list the current x,y,z positions of each ray and velocity vector
        rayAnim.append(rayInfo)
```

Problem 2.4

Set up the second surface (surface B).
Solution Cell Block:

```

######################## Set up the surface (SURFACE B) #########################################

# set up limits of test surface
test_lim = 0.5;

# create grid of points using linspace
x1 = np.linspace(-test_lim,test_lim,10)
x2 = np.linspace(-test_lim,test_lim,10)

# create grid of points using meshgrid
[X1, X2] = np.meshgrid(x1,x2)

# reshape grid into a column vector
X1 = np.reshape(X1,[np.size(X1),1])
X2 = np.reshape(X2,[np.size(X2),1])

# write surface equation for surface a
X3 = np.sqrt((5 - np.sqrt(X1**2 + X2**2))**2 - 4)-1.75 # surface b #FILL IN HERE

# Calculate normals

#surface A
#delF
delGx1 = 1/(2*np.sqrt((5 - np.sqrt(X1**2 + X2**2))**2 - 4))*2*(5 - np.sqrt(X1**2 + X2**2))*-1/(2*np.sqrt(X1**2 + X2**2))*2*X1 #FILL IN HERE
```

```
delGx2 = 1/(2*np.sqrt((5 - np.sqrt(X1**2 + X2**2))**2 - 4))*2*(5 - np.sqrt(X1**2 + X2**2))
    *-1/(2*np.sqrt(X1**2 + X2**2))*2*X2 #FILL IN HERE

delFx1 = delGx1
delFx2 = delGx2
delFx3 = np.ones([np.size(delGx1),1])*-1

gradF = np.hstack([delFx1, delFx2, delFx3])
magdelF = np.reshape(np.linalg.norm(gradF,2,1),(np.size(X1),1))

normal = - gradF / magdelF

# plot surface
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(X1[:,0], X2[:,0], X3[:,0], linewidth=0.2, antialiased=True)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3')
# label plot
plt.title('Surface')
plt.show()
```

Problem 2.5

Use Forward Euler time-stepping scheme to simulate a beam hitting the given test surfaces. (surface B).
Solution Cell Block:

```

######################### Create rays ##########################################

beam_initvel = np.array([0,0,-c]) #initial velocity of beam
beam_N = 100 #number of rays
beam_initH = 4 #initial height
beam_N = round(np.sqrt(beam_N))**2 # updated number of rays to have it square
print('Updated number of rays is:',beam_N)

#full beam grid
[beam_posx1,  beam_posx2] = np.meshgrid(np.linspace(-0.5,0.5,round(np.sqrt(beam_N))),np.
    linspace(-0.5,0.5,round(np.sqrt(beam_N))))

# reshape grid into a column vector
beam_posx1 = np.reshape(beam_posx1,[np.size(beam_posx1),1])
beam_posx2 = np.reshape(beam_posx2,[np.size(beam_posx2),1])
beam_posx3 = np.ones([beam_N,1]) * beam_initH

# set up initial velocity of beam
beam_vel1 = np.ones([beam_N,1])  * beam_initvel[0]
beam_vel2 = np.ones([beam_N,1])  * beam_initvel[1]
beam_vel3 = np.ones([beam_N,1])  * beam_initvel[2]

# Time-stepping
totaltime = 0.000000015 # simulation time
xi = 0.00000005 # time-stepping constant
dt = xi * beam_initH / np.sqrt(c) # time-step
Ntimestep = round(totaltime / dt) # number of time-steps

# initialize variables
IR = np.zeros([np.size(beam_posx1),1])
theta_i = np.zeros([np.size(beam_posx1),1])
X1_final = np.empty([np.size(beam_posx1),1])
X2_final = np.empty([np.size(beam_posx1),1])
X3_final = np.empty([np.size(beam_posx1),1])

# initialize animation variables
rayAnim = list() # list of ray objects
rayHit = list() # list of ray hit objects
```

```python


for i in range(Ntimestep): # time-stepping loop

    #time-step announcer (every 100 time-steps)
    if i % 100 == 0:
        print('Time-step:',i,'/',Ntimestep)


    #update position
    beam_posx1 = beam_posx1 + beam_vel1 * dt
    beam_posx2 = beam_posx2 + beam_vel2 * dt
    beam_posx3 = beam_posx3 + beam_vel3 * dt

    for k in range(np.size(beam_posx1)): # loop over rays
        #check if below surface
        X3_calc = np.sqrt((5 - np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))**2 - 4) - 1.75; #FILL IN HERE

        if beam_posx1[k] <= test_lim and beam_posx1[k] >= -test_lim and beam_posx2[k] <= test_lim and beam_posx2[k] >= -test_lim: # if within test limits
           if X3_calc>beam_posx3[k]: # if below surface
               if IR[k] == 0: # if not yet reflected

                    # find normal and ray velocity vector
                    # Calculate normals
                    # delF
                    # surface a
                    delGx1 = 1/(2*np.sqrt((5 - np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))**2 - 4))*2*(5 - np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))*-1/(2*np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))*2*beam_posx1[k] #FILL IN HERE
                    delGx2 = 1/(2*np.sqrt((5 - np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))**2 - 4))*2*(5 - np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))*-1/(2*np.sqrt(beam_posx1[k]**2 + beam_posx2[k]**2))*2*beam_posx2[k] #FILL IN HERE


                    delFx1 = delGx1
                    delFx2 = delGx2
                    delFx3 = -1

                    gradF = np.hstack([delFx1, delFx2, delFx3])
                    magdelF = np.linalg.norm(gradF,2)

                    normal = - gradF / magdelF


                    norm_k = normal # normal vector
                    beam_vel = np.hstack([beam_vel1[k], beam_vel2[k], beam_vel3[k]]) # ray velocity vector
                    theta_i = np.arccos(np.dot(norm_k,beam_vel)/c) # angle of incidence
                    if beam_vel3[k] <= 0: # if ray is going down
                        #record point of contact with surface
                        X1_final[k] = beam_posx1[k]
                        X2_final[k] = beam_posx2[k]
                        X3_final[k] = beam_posx3[k]

                        beam_vel = beam_vel - 2 * c * np.cos(theta_i) * norm_k # reflect ray

                        #update velocity
                        beam_vel1[k] = beam_vel[0]
                        beam_vel2[k] = beam_vel[1]
                        beam_vel3[k] = beam_vel[2]
                        #theta_i = np.pi - theta_i

                        #update IR
                        IR[k] = (np.abs(theta_i/n_hat) <= 1.)* \
                            (0.5*(((n_hat**2 * np.cos(theta_i) - (n_hat**2 - np.sin(theta_i)**2)**(0.5))/ \
                                     (n_hat**2 * np.cos(theta_i) + (n_hat**2 - np.sin(theta_i)**2)**(0.5)))**2 +\
                                    (((np.cos(theta_i) - (n_hat**2 - np.sin(theta_i)**2)**(0.5))/ \
                                      (np.cos(theta_i) + (n_hat**2 - np.sin(theta_i)**2)**(0.5)))**2)))+ \
                                (np.abs(theta_i/n_hat) > 1.)**1.



    if i % 50  == 0:

        #assign current loc, vel, and energy left in rays in an array

        rayInfo = np.hstack([beam_posx1, beam_posx2, beam_posx3, beam_vel1, beam_vel2, beam_vel3])

        # write into rayAnim list the current x,y,z positions of each ray and velocity vector
        rayAnim.append(rayInfo)
```

Problem 3: Analyzing Your Results

Answer the following questions about the code you created.

Problem 3.1

Discuss your observations from the raytracing animations. *Hint: When do the rays impact the surface? Does the time step capture the reflections properly? etc.*

We see that the rays hit the surface around 1.75 z height for surface A and around 2 z height for surface B. The time step captures the reflections properly as we see the rays bouncing off the surface. The time step is small enough to capture the reflections properly. (Other student answers may vary)

Problem 3.2

Try using different time-steps to observe the effect on ray reflections. Report any findings.

We see that the time step is small enough to capture the reflections properly. But if we increase the time step, the reflections may be detected at a smaller height and be inaccurate, if we were to plot the detected height vs. the actual height. When we make the time step smaller, the reflections are more accurate, but the simulation takes longer to run. (Other student answers may vary)

5 Ethical Considerations for this Project

A goal of this project is to enable advancements in science and engineering through to address critical national challenges associated with next generation food systems. There are deep ethical considerations associated with any technology, in particular for food systems. While technology has tremendous potential to identify greater efficiencies, when it is created without appropriate consideration for who will have access to and control over new resources, or how the new technologies will impact those who work in the system, the efficiencies identified may come at the cost of greater societal inequity. It is important to pursue harnessing technology to disrupt existing inequities, rather than further entrench existing power structures. The following areas should be considered:

- Labor: 1) occupational health, 2) food manufacturing, and 3) outdoor agriculture labor;
- Producers: 1) Small- to mid-size farms, 2) urban agriculture, and 3) research in farm transitions; Technology: 1) research in technology and democracy;
- Health Human Rights: 1) land rights, 2) social justice, and 3) decolonization in agriculture;

Please consider the following questions:

- What are the societal implications of the technology that you are developing?
- Can this technology be distributed fairly and equitably to a wide variety of entities in agricultural industry?
- Are there any potential unintended consequences of this technology becoming available?
- Are there any harmful "spinoffs" of this technology?
- Are there any useful "spinoffs" of this technology?

6 References

1. Anderson, J. G., Rowan, N. J., MacGregor, S. J., Fouracre, R. A., Farish, O. (2000). Inactivation of food-borne enteropathogenic bacteria and spoilage fungi using pulsed-light. IEEE Transactions on Plasma Science , 28 (1), 83-88.

2. Battelle (2020). Instructions for Healthcare Personnel: Preparation of Compatible N95 Respirators for Decontamination by the Battelle Memorial Institute Using the Battelle Decontamination System. https://www.fda.gov/media/137032/download

3. Bolton, J. and Colton, C. (2008). The Ultraviolet Disinfection Handbook, American Water Works Association. ISBN 978 1 58321 584 5, pp. 3-4.

4. Boyce, JM (2016). Modern technologies for improving cleaning and disinfection of environmental surfaces in hospitals. Antimicrobial Resistance and Infection Control. 5: 10. doi:10.1186/s13756-016-0111-x. PMC 4827199. PMID 27069623.

5. Card, K. J., Crozier, D., Dhawan, A., Dinh, M., Dolson, E., Farrokhian, N., Gopalakrishnan, V., Ho, E., King, E. S., Krishnan, N., Kuzmin, G., Maltas, J., Pelesko, J., Scarborough, J. A., Scott, J. G., Sedor, G., Weaver, D. T. (2020). UV Sterilization of Personal Protective Equipment with Idle Laboratory Biosafety Cabinets During the Covid-19 Pandemic [Preprint]. Occupational and Environmental Health. https://doi.org/10.1101/2020.03.25.20043489

6. Downes, A. and Blunt, T. P. (1878). On the Influence of Light upon Protoplasm. Proceedings of the Royal Society of London. 28 (190-195): 199-212. Bibcode:1878RSPS...28..199D. doi:10.1098/rspl.1878.0109

7. Gross, H. (2005). Handbook of optical systems. Fundamental of technical optics. H. Gross, Editor. Wiley-VCH.

8. Heimbuch, B. and Harnish, D. (2019). Research to Mitigate a Shortage of Respiratory Protection Devices During Public Health Emergencies (Report to the FDA No. HHSF223201400158C). Applied Research Associate, Inc.

9. Heimbuch, B. K., Wallace, W. H., Kinney, K., Lumley, A. E., Wu, C.-Y., Woo, M.-H., Wander, J. D. (2011). A pandemic influenza preparedness study: Use of energetic methods to decontaminate filtering facepiece respirators contaminated with H1N1 aerosols and droplets. American Journal of Infection Control , 39 (1), e1-e9.

10. Ito, A., Ito, T. (1986). Absorption spectra of deoxyribose, ribosephosphate, ATP and DNA by direct transmission measurements in the vacuum-UV (150-190 nm) and far-UV (190-260 nm) regions using synchrotron radiation as a light source. Photochemistry and Photobiology , 44 (3), 355-358.

11. Jackson, J. D. (1998). Classical Electrodynamics.

12. Kanemitsu, K. et al, (2005) Does incineration turn infectious waste aseptic?, Journal of Hospital Infection, 60(4):304-306.

13. Lin, T.-H., Tang, F.-C., Hung, P.-C., Hua, Z.-C., Lai, C.-Y. (2018). Relative survival of Bacillus subtilis spores loaded on filtering facepiece respirators after five decontamination methods. Indoor Air , 28 (5), 754-762.

14. Lindsley,W. G., Martin, S. B., Thewlis, R. E., Sarkisian, K., Nwoko, J. O., Mead, K. R., Noti, J. D. (2015). Effects of Ultraviolet Germicidal Irradiation (UVGI) on N95 Respirator Filtration Performance and Structural Integrity. Journal of Occupational and Environmental Hygiene , 12 (8), 509-517. https://doi.org/10.1080/15459624.2015.1018518

15. Lore, M. B., Heimbuch, B. K., Brown, T. L., Wander, J. D., Hinrichs, S. H. (2011). Effectiveness of Three Decontamination Treatments against Influenza Virus Applied to Filtering Facepiece Respirators. The Annals of Occupational Hygiene , 56 (1), 92-101.

16. Marra, A. R., Schweizer, M. L., Edmond, M. B. (2018). No-Touch Disinfection Methods to Decrease Multidrug-Resistant Organism Infections: A Systematic Review and Meta-analysis. Infection Control Hospital Epidemiology , 39 (1), 20-31.

17. Mills, D., Harnish, D. A., Lawrence, C., Sandoval-Powers, M., Heimbuch, B. K. (2018). Ultraviolet germicidal irradiation of influenza-contaminated N95 filtering facepiece respirators. American Journal of Infection Control , 46 (7), e49-e55.

18. Nerandzic, M. M., Cadnum, J. L., Pultz, M. J., Donskey, C. J. (2010). Evaluation of an automated ultraviolet radiation device for decontamination of Clostridium difficile and other healthcare-associated pathogens in hospital rooms. BMC Infectious Diseases , 10 (1), 197.

19. Tseng, C.-C., Li, C.-S. (2007). Inactivation of Viruses on Surfaces by Ultraviolet Germicidal Irradiation. Journal of Occupational and Environmental Hygiene , 4 (6), 400-405.

20. Viscusi, D. J., Bergman, M. S., Eimer, B. C., Shaffer, R. E. (2009). Evaluation of Five Decontamination Methods for Filtering Facepiece Respirators. The Annals of Occupational Hygiene , 53 (8), 815-827.

21. Zohdi, T. I. (2006a). Computation of the coupled thermo-optical scattering properties of random particulate systems. Computer Methods in Applied Mechanics and Engineering. Volume 195, 5813-5830.

22. Zohdi, T. I. (2006b). On the optical thickness of disordered particulate media. Mechanics of Materials. Volume 38, 969-981.

23. Zohdi, T. I and Kuypers, F. A. (2006c). Modeling and rapid simulation of multiple red blood cell light scattering. Proceedings of the Royal Society Interface. Volume 3, Number 11 Pages 823-831.

24. Zohdi, T. I. (2012). Electromagnetic properties of multiphase dielectrics. A primer on modeling, theory and computation. Springer-Verlag.

25. Zohdi, T. I. (2015). A computational modeling framework for high-frequency particulate obscurant cloud performance. The International Journal of Engineering Science. 89, 75-85.

26. Zohdi, T. I. (2016). On high-frequency radiation scattering sensitivity to surface roughness in particulate media. Computational Particle Mechanics. http://dx.doi.org/10.1007/s40571-016- 0118-3

27. Zohdi, T. I. (2019). Rapid simulation-based uncertainty quantification of flash-type time-offlight and Lidar-based body-scanning processes. Computer Methods in Applied Mechanics and Engineering. https://doi.org/10.1016/j.cma.2019.03.056

28. Zohdi, T.I. (2020) Rapid simulation of viral decontamination efficacy with UV irradiation. Computer Methods Appl. Mech. Eng. https://doi.org/10.1016/j.cma.2020.113216