

Rainshadow Simulation:

Using the SIMPLE Algorithm to Model Orographic Precipitation with CFD

Ted Yee

ME7310: Graduate Computational Fluid Dynamics with Heat Transfer

Term Project Phase 3 Report

Wednesday, April 23, 2025

Overview

Code modification or studies: For this portion of your exam, you need to make **two types** of modifications to your code. You have six options on how you can change your code. For each option, you need to submit the lines of code you need to change before and after modification and an explanation for this change (Use the Compare feature). Then you will need to demonstrate that the change was effective, and what the impact was on the rest of the domain. The necessary demonstration will be detailed in each of the options below.

When you modify your codes, copy all of your original codes into a new folder named `Option_N` where N is the option number you work on. Submit to blackboard your code modification description (pdf), figures, and each of the folders with the entire code for each modification.

In the event that you were not able to get a functioning code, you can make modifications to the sample code contained in `step_down.zip`.

This is the report of the 2 modification and some closing remarks about the model's validity and a bug fix since Phase 2. Options 1 and 2 were chosen for the modifications because the others would take too long to run all parameter variations. Changing the inlet velocity profile was simply a weather pattern not capable of producing a rainshadow, and changing surfaces to no-slip had negligible effect.

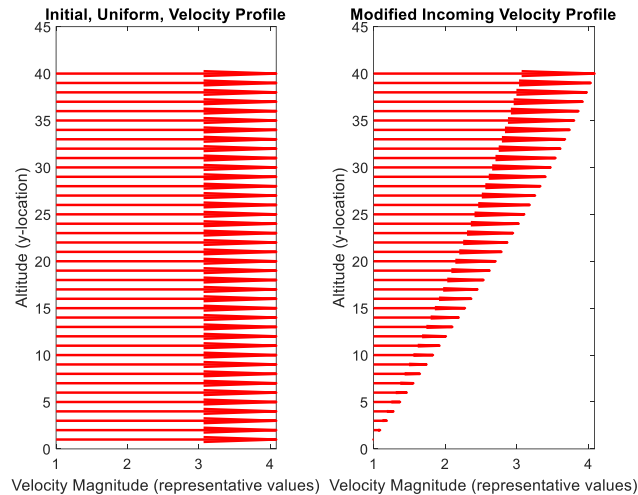
Option 1: Inlet BC

Option 1: Change the inlet boundary condition. Change one of your inlet boundary condition from a constant velocity inlet to the spatially varying inlet

$$u_{inlet}(y) = \sin\left(\frac{y - y_{bottom}}{y_{top} - y_{bottom}}\pi\right)$$

where y_{bottom} is the position of the bottom of the inlet and y_{top} is the position of the top of the inlet (or equivalent in case you have a horizontal inlet, with $u_{inlet}(x)$). To demonstrate the change, in one figure, plot the horizontal velocity of the u -cell centers one column in from the inlet before and after the change (see red dashed line/arrows for example placement and velocity component in figure). This will show the contrast between the constant and variable inlet condition. Also plot the full domain result before and after the change and comment on what you find.

The most direct result of changing the velocity profile is that the conditions to create a rainshadow are not met. In the base simulation, the inlet is a uniform velocity value of 10 at every altitude. Below is a graphical representation (not that the arrow lengths are representative and not the actual velocity values) of the new input velocity profile.



Changing the code to implement this was really as simple as changing one line, but I included the plotting functions as well. In order to display both the uniform and varying inlet profile, I plotted and *then* updated within the main script after the values are set by solver_gui.m. The comparison was done in VSCode because the MATLAB comparison was hard to read.

```

1 clear; close all; clc;
2 tic
3
4 %%
5 cell_visualizations_on_off =1;
6 blocked = 1;
7 rainshadow = 0;
8 %%%Ideally a geometry gui would set the values currently in
9 %%%system_parameters, then once submitted would generate, then gui would
10 %%%Setup methods then begin once the initialize and run button is clicked
11 system_parameters %%% Set system, geometry, and solver parameters
12 solver_gui
13
14
15
16
17
18
19
20
21
22
23
24
25
26 scalefactor = 10000; %%%default = 10000, Vertical scale factor in simulation length

```

```

1 clear; close all; clc;
2 tic
3
4 %%
5 cell_visualizations_on_off =1;
6 blocked = 1;
7 rainshadow = 0;
8 %%%Ideally a geometry gui would set the values currently in
9 %%%system_parameters, then once submitted would generate, then gui would
10 %%%Setup methods then begin once the initialize and run button is clicked
11 system_parameters %%% Set system, geometry, and solver parameters
12 solver_gui
13
14 [X,Y] = meshgrid(1:N.y_u,1:N.x_u);
15 left_boundary_x = ones(N.y_u, 1);
16 left_boundary_y = (1:N.y_u)';
17 figure(19)
18 tiledlayout(1,2)
19 nexttile
20 quiver(left_boundary_x, left_boundary_y, BC.u_lef_value, zeros(N.y_u,1), 0.5, 'r', 'LineWidth', 1.5);
21 title('Initial, Uniform, Velocity Profile')
22 xlabel('Velocity Magnitude (representative values)')
23 ylabel('Altitude (y-location)')
24 nexttile
25 BC.u_lef_value = sin(grids.y_u'-grids.y_u(1)/(grids.y_u(end)-grids.y_u(1)*pi));
26 quiver(left_boundary_x, left_boundary_y, BC.u_lef_value, zeros(N.y_u,1), 0.5, 'r', 'LineWidth', 1.5);
27 title('Modified Incoming Velocity Profile')
28 xlabel('Velocity Magnitude (representative values)')
29 ylabel('Altitude (y-location)')
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 scalefactor = 10000; %%%default = 10000, Vertical scale factor in simulation length

```

```

13 if blocked ==1
14     set_cell_type_blocked
15 else
16     set_cell_type
17 end
18
19 %%% Mountain Parameters
20 rise_slope = .5;
21 fall_slope = -1;
22 x_mountain_start = floor(N.x_p*50/100);
23 x_mountain_peak = floor(N.x_p*60/100);
24 x_mountain_end = floor(N.x_p*120/100);
25 ambienthumidity = 0;
26

```

```

32 if blocked ==1
33     set_cell_type_blocked
34 else
35     set_cell_type
36 end
37
38 %%% Mountain Parameters
39 rise_slope = .5;
40 fall_slope = -1;
41 x_mountain_start = floor(N.x_p*50/100);
42 x_mountain_peak = floor(N.x_p*60/100);
43 x_mountain_end = floor(N.x_p*120/100);
44 ambienthumidity = 0;
45

```

Since ANTSSSS is already set up to handle varying (text string, function, scalar, and vector) inputs, there were no further changes needed. The input data is read from the GUI textboxes and passed through a local function (inside solver_gui.m) that converts all boundary conditions into 1D vectors of the length of the edge they are applied to.

The boundary condition vectors are then stored in a structure called BC and when they are applied in the simple algorithm, the nonuniform inlet profile is (and was in the base code) handled by a set of indexes.

```
function BC = update_plot(fig, grids, N,base)
try
    % Retrieve the stored value from guidata
    data = guidata(fig);
    disp(fieldnames(data)); % This will help in debugging if the fields e

    % Retrieve and evaluate inputs for boundary conditions
    BC.u_lef_value = evaluate_input(data.u_lef_value_box.Value, grids, N);
    BC.u_rig_value = evaluate_input(data.u_rig_value_box.Value, grids, N);
    BC.u_bot_value = evaluate_input(data.u_bot_value_box.Value, grids, N);
    BC.u_top_value = evaluate_input(data.u_top_value_box.Value, grids, N);
    BC.u_lef_type = data.u_lef_type_dropdown.Value;
    BC.u_rig_type = data.u_rig_type_dropdown.Value;
    BC.u_bot_type = data.u_bot_type_dropdown.Value;
    BC.u_top_type = data.u_top_type_dropdown.Value;

    BC.v_lef_value = evaluate_input(data.v_lef_value_box.Value, grids, N);
    BC.v_rig_value = evaluate_input(data.v_rig_value_box.Value, grids, N);
    BC.v_bot_value = evaluate_input(data.v_bot_value_box.Value, grids, N);
    BC.v_top_value = evaluate_input(data.v_top_value_box.Value, grids, N);
    BC.v_lef_type = data.v_lef_type_dropdown.Value;
    BC.v_rig_type = data.v_rig_type_dropdown.Value;
    BC.v_bot_type = data.v_bot_type_dropdown.Value;
    BC.v_top_type = data.v_top_type_dropdown.Value;

    assignin('base', 'BC', BC);
    disp('✓ BC stored in base workspace');
    uiresume(base);
end
```

```
function value = evaluate_input(input_str, grids, N)
try
    % Replace 'x' and 'y' with grid variables
    input_str = strrep(input_str, 'x', 'grids.x_p');
    input_str = strrep(input_str, 'y', 'grids.y_p');

    func = str2func(['@(grids) ' input_str]);
    value = func(grids);

    if isscalar(value)
        if isvector(value)
            value = value * ones(N.y_p, 1);
        else
            value = value * ones(N.x_p, 1);
        end
    end
catch ME
    error('Error evaluating input "%s": %s', input_str, ME.message);
end
```

A vector of 4 indexes keeps track of which point in the varying boundary condition is being applied, and since MATLAB reads the cell types in order, the indexes can be incremented and applied one data point at a time

Results

It is important to note that the inlet humidity profile (which is handled in the same way as the velocity inlet conditions) is by default set to a sigmoid function. In the solver_gui.m, where the rainshadow preset values are applied, line 211 (which existed in the base simulation, though labels were added here) can be uncommented to show the humidity profile. The default setting is to have high humidity at low altitudes (modeling evaporation low over a body of water), which drops off as altitude increases:

```
index=ones(1,4); %%% indices for scrolling through nonuniform BC

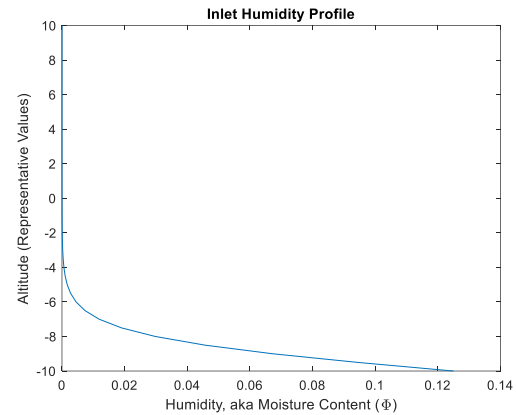
%% Update Matrices by cell type
for i = 1:length(u_type(:))
    %%%Right/East Edge
    if u_type(i) == 2
        A_u(i,i) = 1;
        A_u(i,i-1) = -1;
        b_u(i) = BC.u_rig_value(index(1));

    %%%Bottom/South Edge
    elseif u_type(i) == 4
        if BC.u_bot_type == "Dirichlet"
            A_u(i,i) = 1;
            b_u(i) = BC.u_bot_value(index(2));
            index(2)=index(2)+1;
        elseif BC.u_bot_type == "Neumann"
            A_u(i,i) = 1;
            A_u(i,i-1) = -1;
            b_u(i) = BC.u_bot_value(index(2));
            index(2)=index(2)+1;
        else %%% for free-slip
            A_u(i,i) = Fe_u(i)-Fw_u(i)+Fn_u(i)-Fs_u(i)+De+Dw+Dn+2*Ds + 1/dt;
            A_u(i,i-Ny_u) = -Fw_u(i)-Dw;
            A_u(i,i+Ny_u) = Fe_u(i)-De;
        end
    end
end
```

```

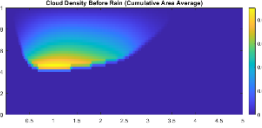
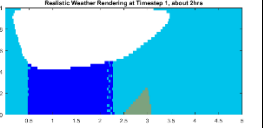
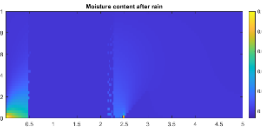
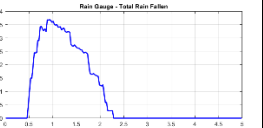
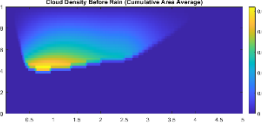
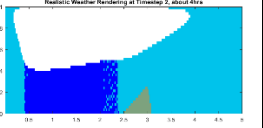
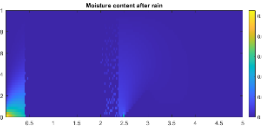
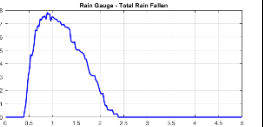
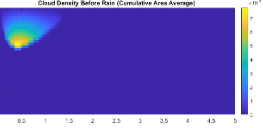
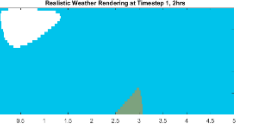
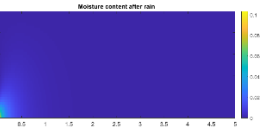
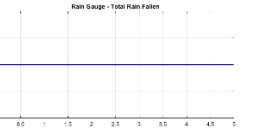
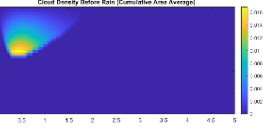
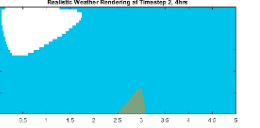
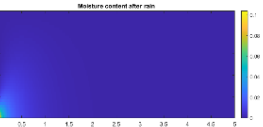
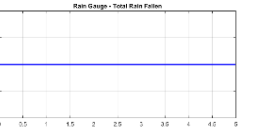
206         BC.v_bot_value = zeros(N.x_v,1);
207         BC.v_top_type = "Dirichlet"; % edge of atmosphere
208         BC.v_top_value = zeros(N.x_v,1);
209         x = linspace(-10, 10, N.y_p+1); % Adjust range for sharp
210         phi_in = .25*(1- (1 ./ (1 + exp(-10-x)))); % Sigmoid fu
211         % plot(phi_in,x)
212         % title('Inlet Humidity Profile')
213         % ylabel('Altitude (Representative Values)')
214         % xlabel('Humidity, aka Moisture Content (\Phi)')
215
216         uiresume(base);
217     end
218
219

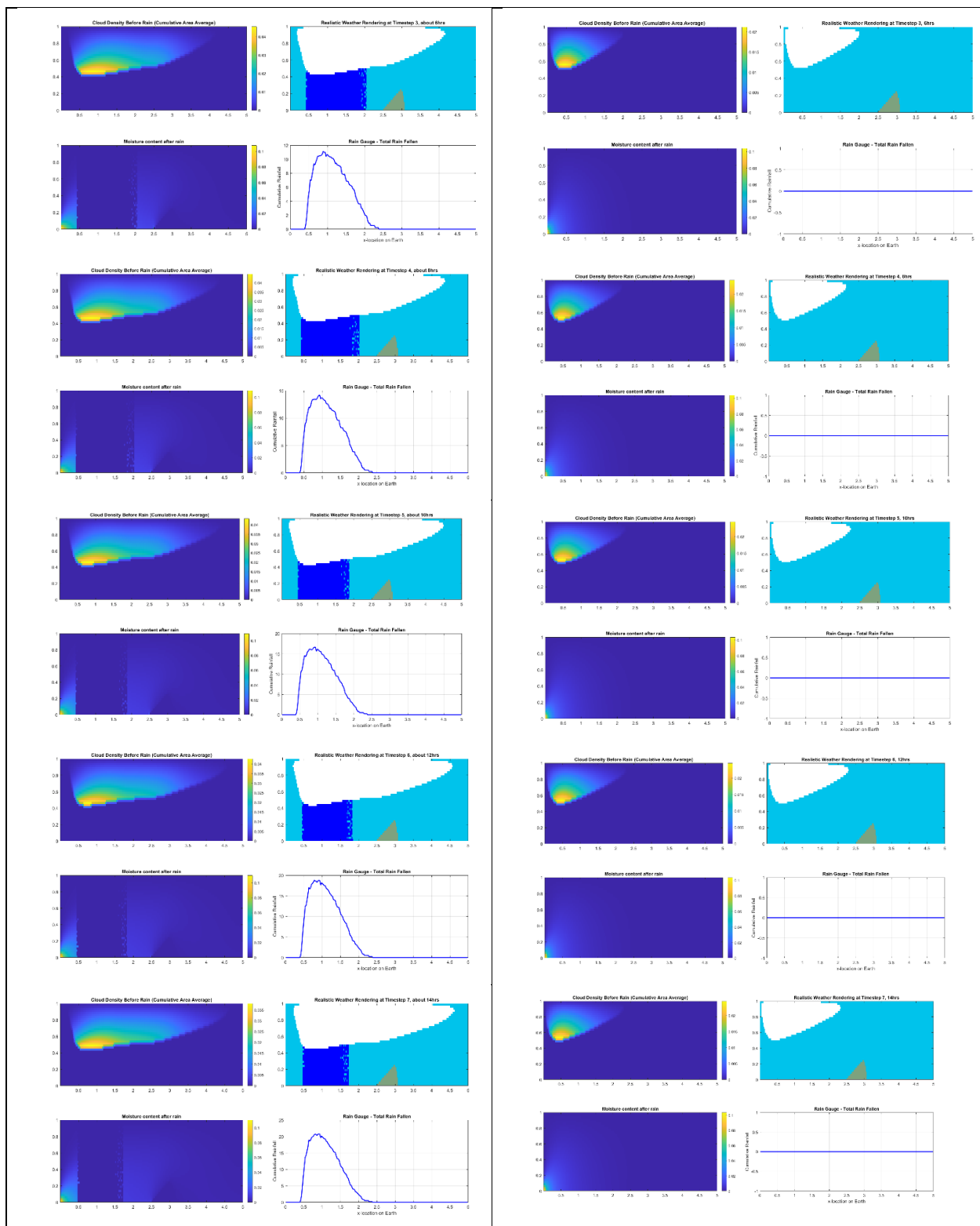
```

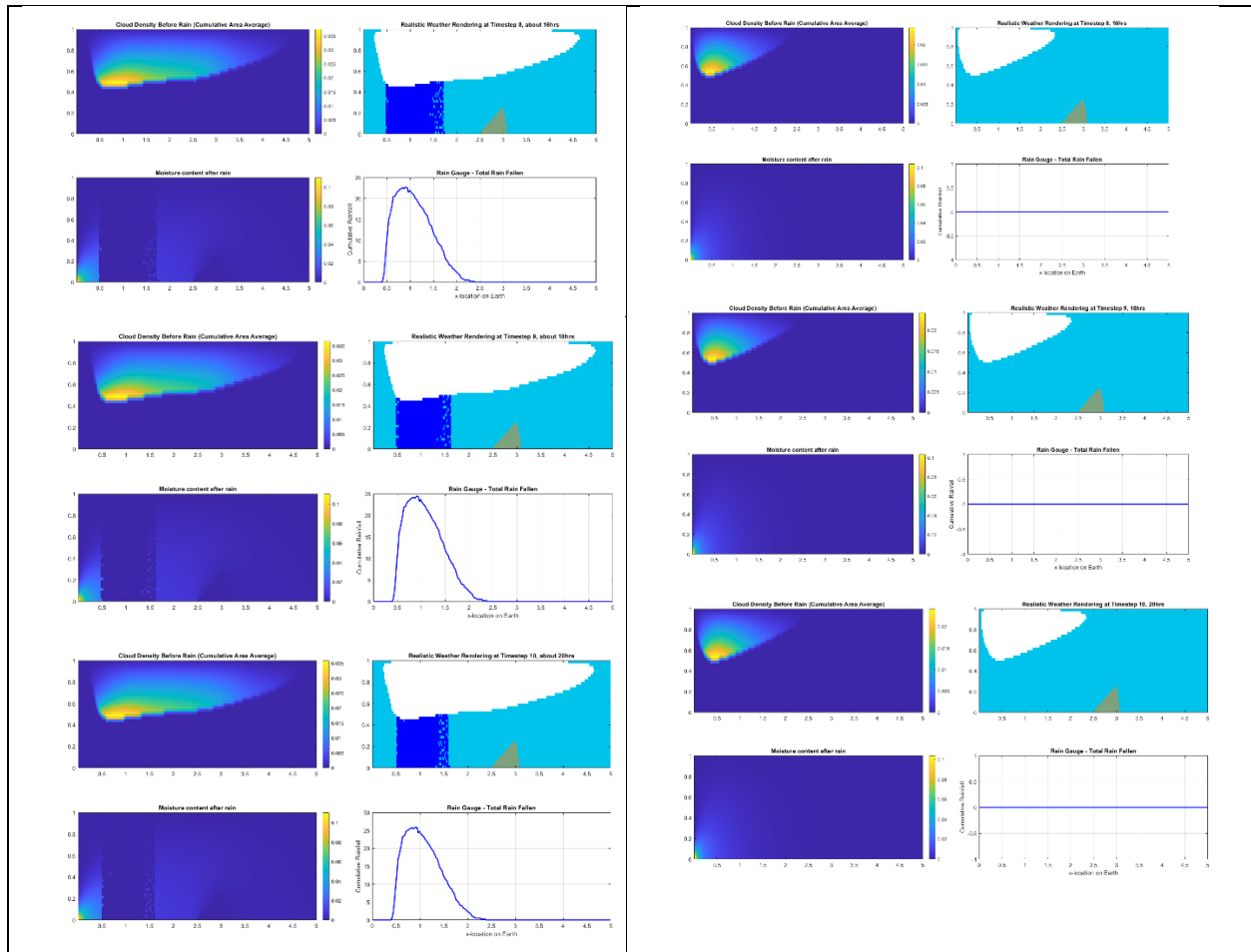


By changing the inlet velocity to be very low at the bottom (where most of the moisture is) and high at the top (where there is little moisture), there is not enough movement to model a rainshadow within the first 10 timesteps. In the below table, each timestep is compared to the default preset values where velocity is uniform.

The simple conclusions are that the nonuniform boundary condition-handling works as expected and that this wind profile does not create a rainshadow. As the vector-handling was already tested and the rainshadow conditions were derived through testing, neither were surprising.

Uniform Inlet Velocity Profile	Varying Inlet Velocity Profile
       	       

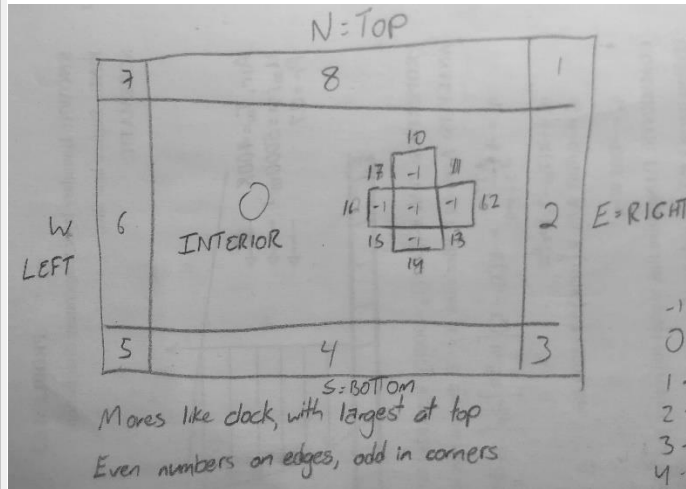
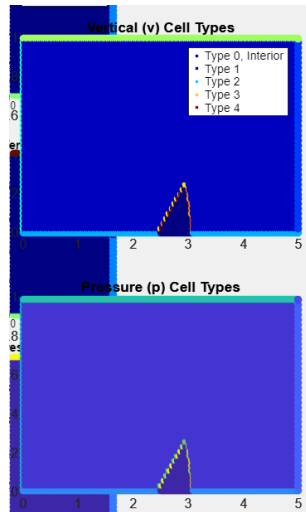




Option 2: No-Slip/Free-Slip

Option 2: Change a no-slip or a free-slip conditions. For this option, pick a no-slip or free-slip boundary condition in your domain and change it to the opposite (no-slip becomes free-slip or free-slip becomes no-slip). State clearly which boundary you are going to change and what it is changed to. Plot a before and after of a vertical profile of the horizontal velocity (see green in figure). Also plot the full domain result before and after the change and comment on what you find.

This change of free-slip to no-slip on the mountain surface actually had negligible effect. The boundary condition that I changed was actually one that should've been set in the base simulation to better simulate real-life. In the base simulation, the surfaces of the mountain (which contain 7 different cell types, per the cell-type setting and key below) were all free-slip. To model the wind-resistance caused by surface roughness, trees, etc., both sides of the mountain should be no-slip.



Once again, the base model already had a lot of the structure to handle cases like these. First, by initializing a value called `noslipground` in the `main.m` script, ANTSSSSS can handle cases both with and without noslip mountains.

```

3
4
5
6
7
8
9
10
11
12
13
14

```

```

%%
cell_visualizations_on_off = 1;
blocked = 1;
rainshadow = 0;
noslipground = 0;
%%%ideally a geometry gui would set t
%%%system parameters, then once subm
%%%setup methods then begin once the
system_parameters
solver_gui
if blocked == 1

```

To ensure that this parameter is 1 when the 'Run Preset Rainshadow Simulation Instead' button is pressed, the value just needs to be set in `solver_gui`, and passed out of the GUI.

```

176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194

```

```

uiwait(base);
function rainshadow_callback(N, base)
[rainshadow, blocked, BC, phi_in, noslipground] = begin_rainshadow(N, base);
assignin('base', 'rainshadow', rainshadow);
assignin('base', 'blocked', blocked);
assignin('base', 'BC', BC);
assignin('base', 'phi_in', phi_in);
assignin('base', 'noslipground', noslipground);
end
function [rainshadow, blocked, BC, phi_in, noslipground] = begin_rainshadow(N, base)
rainshadow = 1;
blocked = 0;
noslipground = 0;
BC.u_lef_type = "Dirichlet"; % angled incoming wind
BC.u_lef_value = 10* ones(N.y_u, 1);
BC.u_rig_type = "FreeSlip"; % outlet

```

By passing the variable into `A_u` and `A_v` creation scripts, the variable can be used in an if statement (so it can be changed easily later) and the `A_matrices` are set in the exact same way as Dirichlet conditions in the other cell types.

[illegible]

```

1 A_p_creation * +
2 function (A_p, b_p, Ap_u, Ap_v) = A_p_creation(u_star, v_star, p_guess, A_u,
3           dx, dy, p_type, A_p)
4
5 %S I have no clue why, but the if statements should be
6 % p_type(1) == 5 || p_type(1) == 11
7 % p_type(1) == 7 || p_type(1) == 13
8 % p_type(1) == 6 || p_type(1) == 12
9 % p_type(1) == 2 || p_type(1) == 7
10 % p_type(1) == 1 || p_type(1) == 15
11 % p_type(1) == 2 || p_type(1) == 16
12 % p_type(1) == 4 || p_type(1) == 10
13 % but when these are used, the matrix becomes singular. It seems to work
14 % without though.
15
16 [hy_u, hx_u] = size(u_star);
17 [hy_v, hx_v] = size(v_star);
18 [hy_p, hx_p] = size(p_guess);
19
20

```

Results

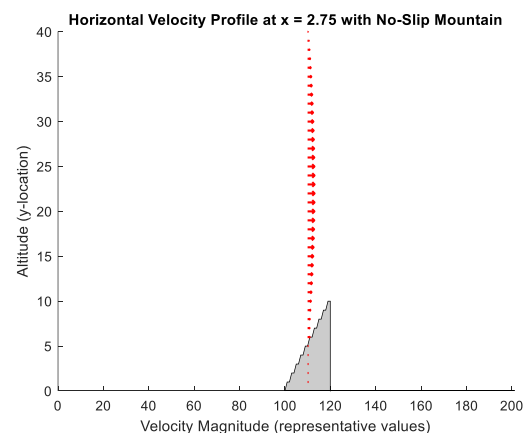
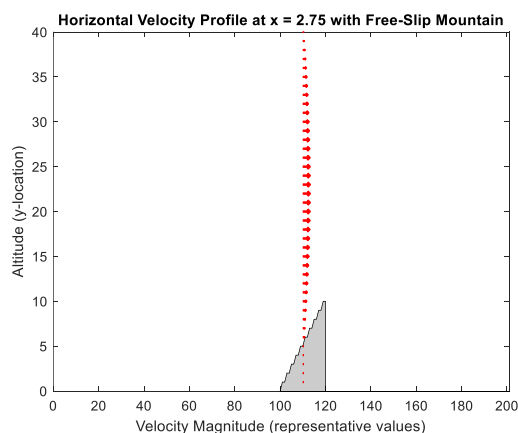
To better visualize the changes, the following code can be added to the end of the main.m script.

```

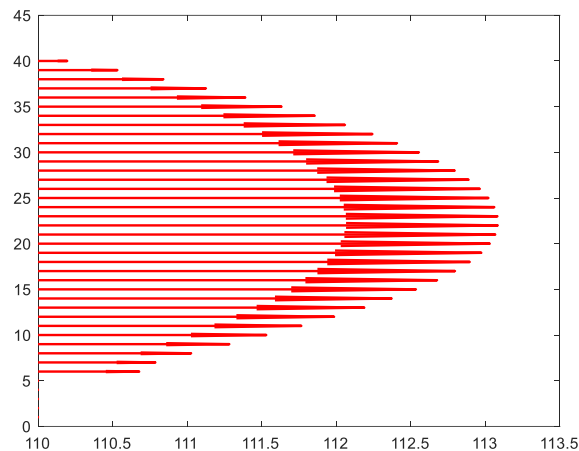
41 end
42 plot_cell_type
43 guess_initialization           %% Generate the initial guess for
44
45
46 for n = 1:Nt-1                 %%%%%%%%%%% TIME MARCHING
47     disp(['TIME STEP NUMBER: ', num2str(n)])
48     disp('time_step_initialization')
49     time_step_initialization    %% Set the variables to begin a ne
50     disp('simple_algorithm')
51     simple_algorithm            %% Begin recursive calculation of
52     disp('heat_transfer_step')
53     heat_transfer_step          %% Perform a heat transfer step
54     disp('plot_store_results')
55     plot_store_results          %% Plot results and store the u'n
56     disp('clouds')
57     clouds                     %% Clouds and Rain
58 end
59
60 figure(19)
61 hold on
62 x_mountain = [x_mountain_start, x_mountain_peak, x_mountain_end];
63 % Number of points on each side of mountain
64 n_left = x_mountain_peak - x_mountain_start + 1;
65 n_right = x_mountain_end - x_mountain_peak;
66
67 left_x = linspace(x_mountain_start, x_mountain_peak, n_left);
68 left_y = round(rise_slope * (left_x - x_mountain_start));
69 right_x = linspace(x_mountain_peak, x_mountain_end, n_right);
70 right_y = round(fall_slope * (right_x - x_mountain_peak));
71 x_values = [left_x, right_x];
72 y_mountain = [left_y, right_y];
73
74 x_values = [x_values, x_mountain_end]; % Add the end point for x
75 y_mountain = [y_mountain, 0]; % Add the bottom point (y = 0) for closing the shape
76 fill(x_values, y_mountain, 'k', 'FaceAlpha', 0.2);
77 %Quiver
78 xlocation = floor(2.75/dx);
79 x_quiver = ones(N.y_u, 1) * xlocation;
80 left_boundary_y = (1:N.y_u)';
81 quiver(x_quiver, left_boundary_y, u_guess(1:end, xlocation), v_guess(1:end-1, xlocation))
82 title(['Horizontal Velocity Profile at x = ', num2str(xlocation*dx), ' with No-Slip Moun
83 xlabel('Velocity Magnitude (representative values)');
84 ylabel('Altitude (y-location)');
85 axis([0 N.x_u 0 N.y_u]);
86 hold off;
87
88 %%
89 toc

```

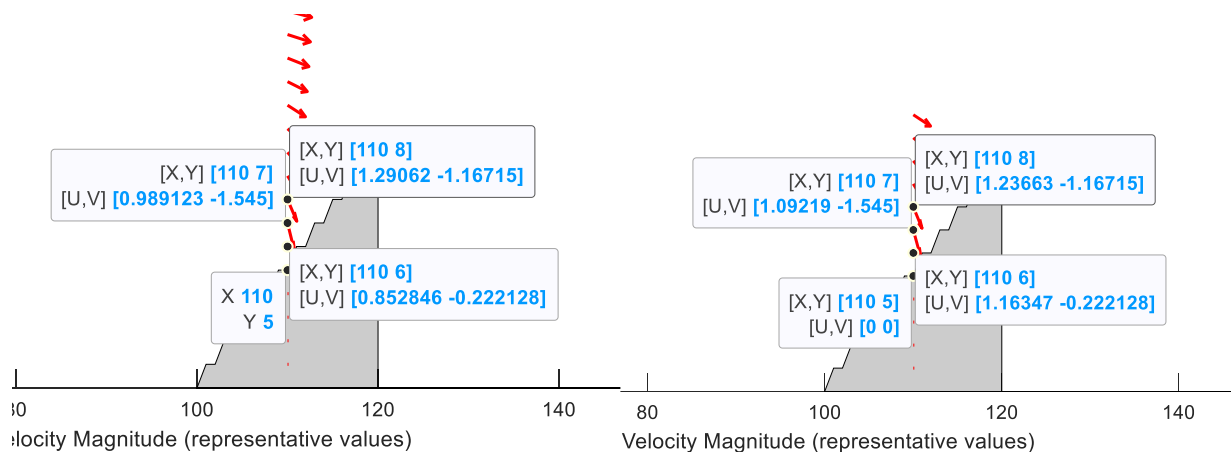
It turns out that changing the free-slip/no-slip conditions of the mountain surface has negligible (but measurable) effect. This is probably due to the fact that ample space is given both upwind and downwind of the mountain and that the atmospheric ceiling (10,000m, where the temperature lapse rate deviates from linear) is well above the 3000m mountain. Plotting the velocity profiles in each appear almost identical:



By rescaling the profile, confidence is added that the velocity profile is the typical fluid parabola:



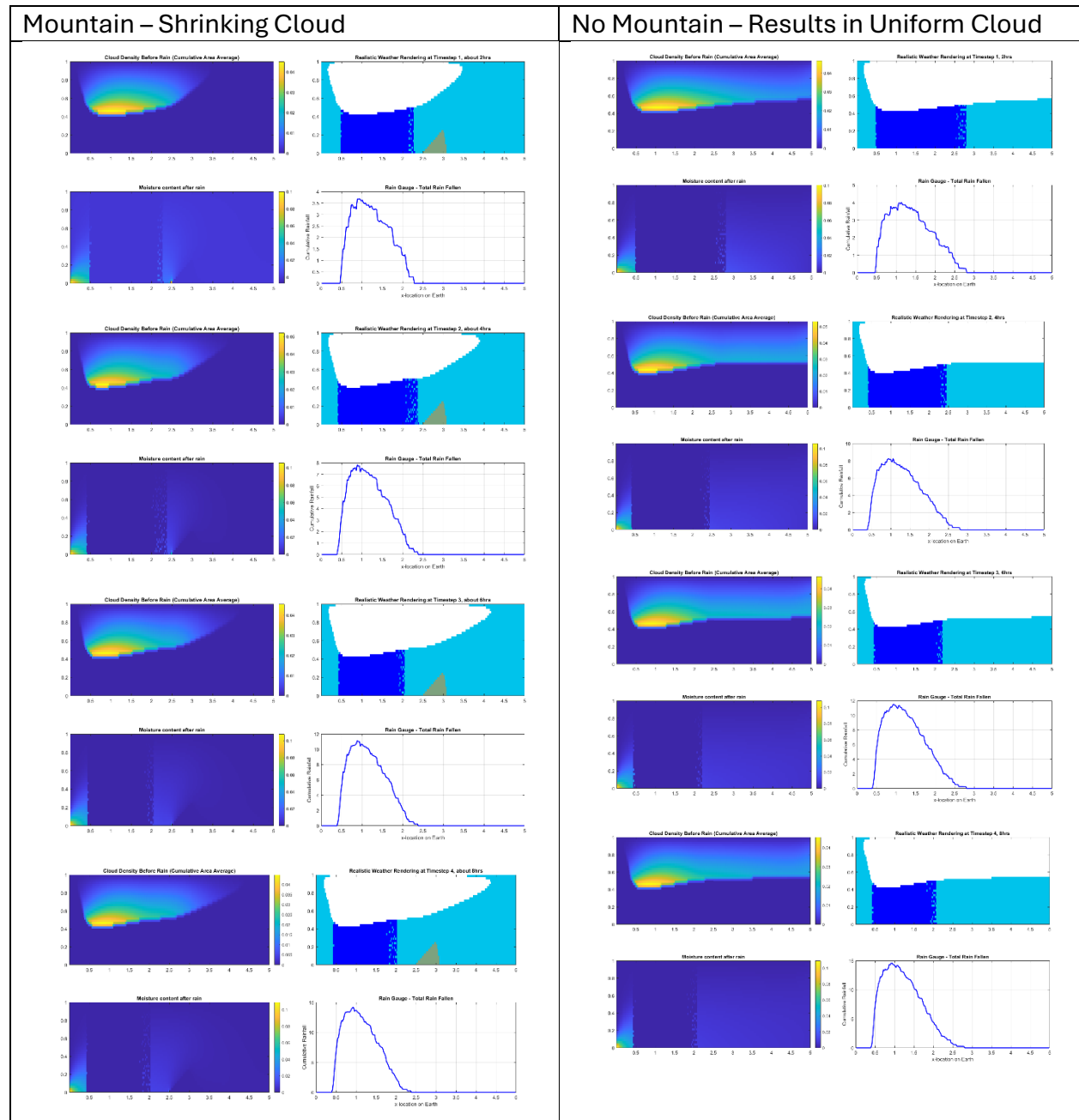
And by adding data labels, the values are definitely different, with a value able to be set on the boundary in the right (no-slip) setup.

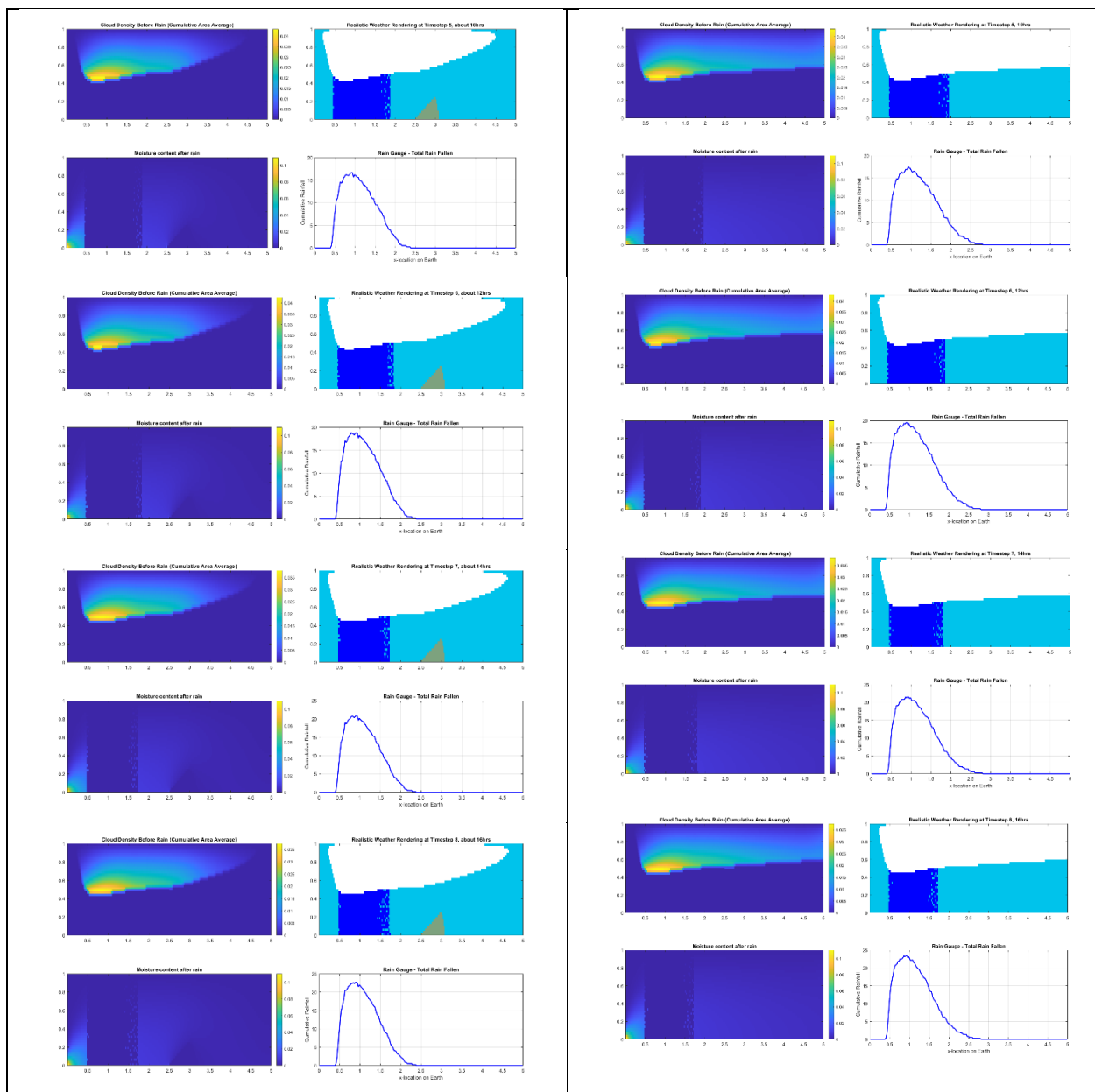


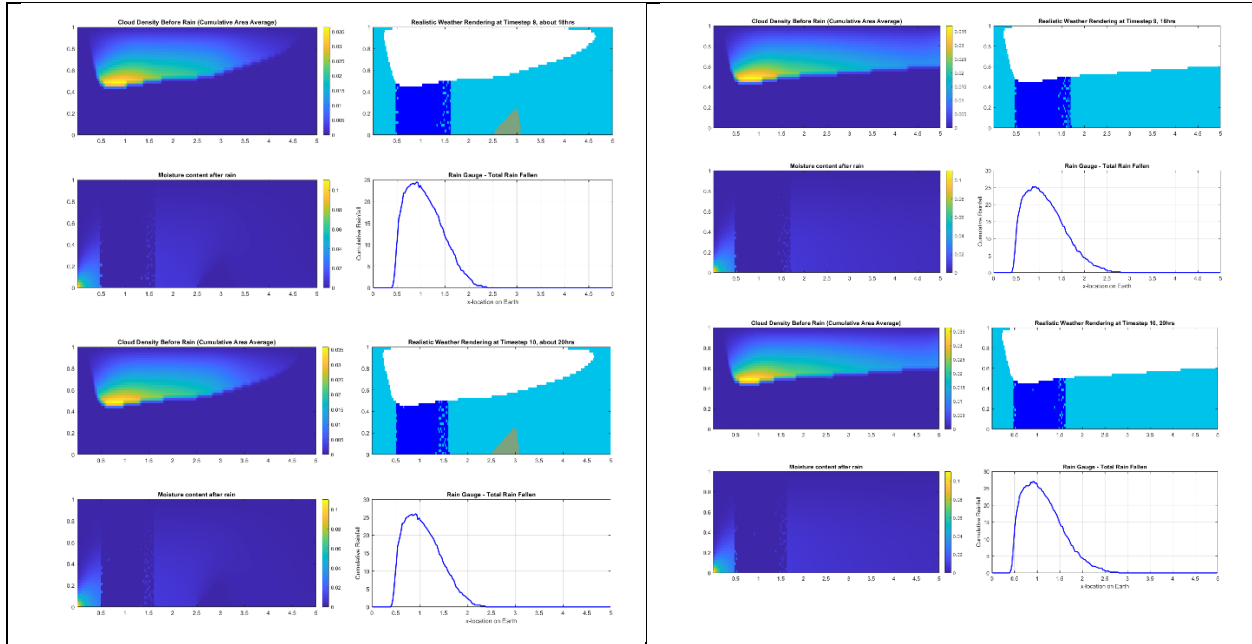
Closing Remarks

While it would have been more interesting to vary parameters of the rainshadow, resolution, time, geometry etc., the inordinate amount of time to complete even 10 timesteps is not worth the cost. These changes were chosen because they required only one different run. It is good practice in ensuring that code is complete before submitting a job, as is paramount in using high-powered computers.

To test the validity of the model as a rainshadow, one more simulation was run with no mountain. The upwind rain accumulation is not the most reassuring, but the mountain does prove to condense the clouds and moisture together. Otherwise, the clouds do not reach the critical mass required to rain and just pass onto land uniformly. This would be the equivalent of a rainy shore, and then downwind, other patterns would disperse the moisture.







Bug Fix For Non-preset Option

In the base simulation, there were some bugs that prevented the user from successfully running a non-rainshadow simulation. Fixes were made as below and incorporated into the option_2 codebase.

First, the main.m code needed to initialize the variables that were only meant for the clouds because they are referenced in shared functions:

```

rainshadow = 0;

% Ideally a geometry gui would set the values currently in
% WMSysParameters, then once submitted would generate, then gui would
% WMSysParameters then begin once the initialize and run button is clicked
system_parameters
solver_gui

% If blocked = 1
if blocked == 1
    set_cell_type_blocked
else
    set_cell_type
end

if rainshadow == 1
    % Mountain Parameters
    rise_slope = .5;
    fall_slope = .5;
    x_mountain_start = floor(N_x*p/100);
    x_mountain_peak = floor(N_x*p/40/100);
    x_mountain_end = floor(N_x*p/10/100);
    ambient_humidity = 0;
    scalefactor = 10000; % Default = 10000. Vertical scale factor in simulation length units to m, J

% Atmospheric Parameters
T_surface = 300; % Default = 300 Kelvin. Temperature at sea level
lapse_rate = 0.8/1000*scalefactor; % Default = 0.8/1000 Kelvin/meter, lapse rate
criticalness = .025; % Default = 3.2 for phi_will_philow. More realistic parameter controls
rainremovalrate = .7; % Default = 0.6 under 0.5 is 1000 removal, up to 1 is 500. It steps over 1

% Set the heat transfer properties
kappa = 26.3e-3*scalefactor; % 26.3e-3 for air at 300K
phi_will = 0; % Amount of moisture coming from above atmosphere, not needed for rainshadow

% Initialize rain gauge to track moisture cleared in each column
rain_gauge = zeros(1, N_x,p);
create_mountain

end

plot_cell_type
guess_initialization % Generate the initial guess for the system

for n = 1:N-1
    % ===== TIME MARCHING =====
    disp(['TIME STEP NUMBER: ', num2str(n)])
    disp('time_step_initialization')
    time_step_initialization % Set the variables to begin a new time step
    disp('simple_algorithm')
    simple_algorithm % Begin recursive calculation of v^m and v^m+1 using A_n, A_n, v^m
    disp('heat_transfer_step')
    heat_transfer_step % Perform a heat transfer step
    disp('plot_store_results')
    plot_store_results % Plot results and store the v^m+1, v^m+1, and p^m+1 fields
    disp('clouds')
    clouds % Clouds and Rain
end

rainshadow = 0;
background = 0;

% Ideally a geometry gui would set the values currently in
% WMSysParameters, then once submitted would generate, then gui would
% WMSysParameters then begin once the initialize and run button is clicked
system_parameters
solver_gui

% If blocked = 1
if blocked == 1
    set_cell_type_blocked
else
    set_cell_type
end

if rainshadow == 1
    % Mountain Parameters
    rise_slope = .5;
    fall_slope = .5;
    x_mountain_start = floor(N_x*p/100);
    x_mountain_peak = floor(N_x*p/40/100);
    x_mountain_end = floor(N_x*p/10/100);
    ambient_humidity = 0;
    scalefactor = 10000; % Default = 10000. Vertical scale

% Atmospheric Parameters
T_surface = 300; % Default = 300 Kelvin. Temperature at sea level
lapse_rate = 0.8/1000*scalefactor; % Default = 0.8/1000 Kelvin/meter, lapse rate
criticalness = .025; % Default = 3.2 for phi_will_philow. More realistic parameter controls
rainremovalrate = .7; % Default = 0.6 under 0.5 is 1000 removal, up to 1 is 500. It steps over 1

% Set the heat transfer properties
kappa = 26.3e-3*scalefactor; % 26.3e-3 for air at 300K
phi_will = 0; % Amount of moisture coming from above atmosphere, not needed for rainshadow

% Initialize rain gauge to track moisture cleared in each column
rain_gauge = zeros(1, N_x,p);
create_mountain

end

plot_cell_type
guess_initialization % Generate the initial guess for the system

for n = 1:N-1
    % ===== TIME MARCHING =====
    disp(['TIME STEP NUMBER: ', num2str(n)])
    disp('time_step_initialization')
    time_step_initialization % Set the variables to begin a new time step
    disp('simple_algorithm')
    simple_algorithm % Begin recursive calculation of v^m and v^m+1 using A_n, A_n, v^m
    disp('heat_transfer_step')
    heat_transfer_step % Perform a heat transfer step
    disp('plot_store_results')
    plot_store_results % Plot results and store the v^m+1, v^m+1, and p^m+1 fields
    if rainshadow == 1
        disp('clouds')
        clouds % Clouds and Rain
    end
end

```

Next, the obstruction edges of the block were hard coded from the sample code and didn't even fit within the domain, so they were made dynamic:

<pre> 2 dx = 1/40; 3 dy = 1/40; 4 x_min = 0; x_max = 5; 5 y_min = 0; y_max = 1; 6 7 %% Set obstruction geometry data 8 y_obs_top = -0.1; 9 x_obs_front = 2; 10 11 % Temporal Domain 12 dt = 0.2; 13 t = 0:dt:2; 14 Nt = length(t); 15 16 % Set the Reynolds number </pre>	<pre> 2 dx = 1/40; 3 dy = 1/40; 4 x_min = 0; x_max = 5; 5 y_min = 0; y_max = 1; 6 7 %% Set obstruction geometry data 8 x_obs_front = 1.5*(x_max-x_min); 9 y_obs_top = 0.5*(y_max-y_min); 10 11 % Temporal Domain 12 dt = 0.2; 13 t = 0:dt:2; 14 Nt = length(t); 15 16 % Set the Reynolds number </pre>
---	--

In solver_gui.m, this section was removed because rainshadow will never be set before the button is set.

```

136 end
137 if rainshadow == 1
138     create_mountain
139 end
140 plot_cell_type
141

```

As noted above, the no slip ground variable needed to be included:

<pre> 176 uiwait(base); 177 178 function rainshadow_callback(N, base) 179 [rainshadow, blocked, BC, phi_in] = begin_rainshadow(N, base); 180 assignin('base', 'rainshadow', rainshadow); 181 assignin('base', 'blocked', blocked); 182 assignin('base', 'BC', BC); 183 assignin('base', 'phi_in', phi_in); 184 185 end 186 187 function [rainshadow,blocked, BC, phi_in] = begin_rainshadow(N, base) 188 rainshadow = 1; 189 blocked = 0; 190 191 BC.u_lef_type = "Dirichlet"; % angled incoming wind 192 BC.u_lef_value = 10* ones(N.y_u, 1); 193 BC.u_rig_type = "FreeSlip"; % outlet 194 BC.u_rig_value = zeros(N.y_u,1); 195 BC.u_bot_type = "Dirichlet"; % no slip at ground 196 BC.u_bot_value = zeros(N.x_u,1); </pre>	<pre> 173 uiwait(base); 174 175 function rainshadow_callback(N, base) 176 [rainshadow, blocked, BC, phi_in, noslipground] = be 177 assignin('base', 'rainshadow', rainshadow); 178 assignin('base', 'blocked', blocked); 179 assignin('base', 'BC', BC); 180 assignin('base', 'phi_in', phi_in); 181 assignin('base', 'noslipground', noslipground); 182 183 end 184 185 function [rainshadow,blocked, BC, phi_in, noslipground] 186 rainshadow = 1; 187 blocked = 0; 188 noslipground = 0; 189 190 BC.u_lef_type = "Dirichlet"; % ar 191 BC.u_lef_value = 10* ones(N.y_u, 1); 192 BC.u_rig_type = "FreeSlip"; % outlet 193 BC.u_rig_value = zeros(N.y_u,1); 194 BC.u_bot_type = "Dirichlet"; % no 195 BC.u_bot_value = zeros(N.x_u,1); </pre>
---	--

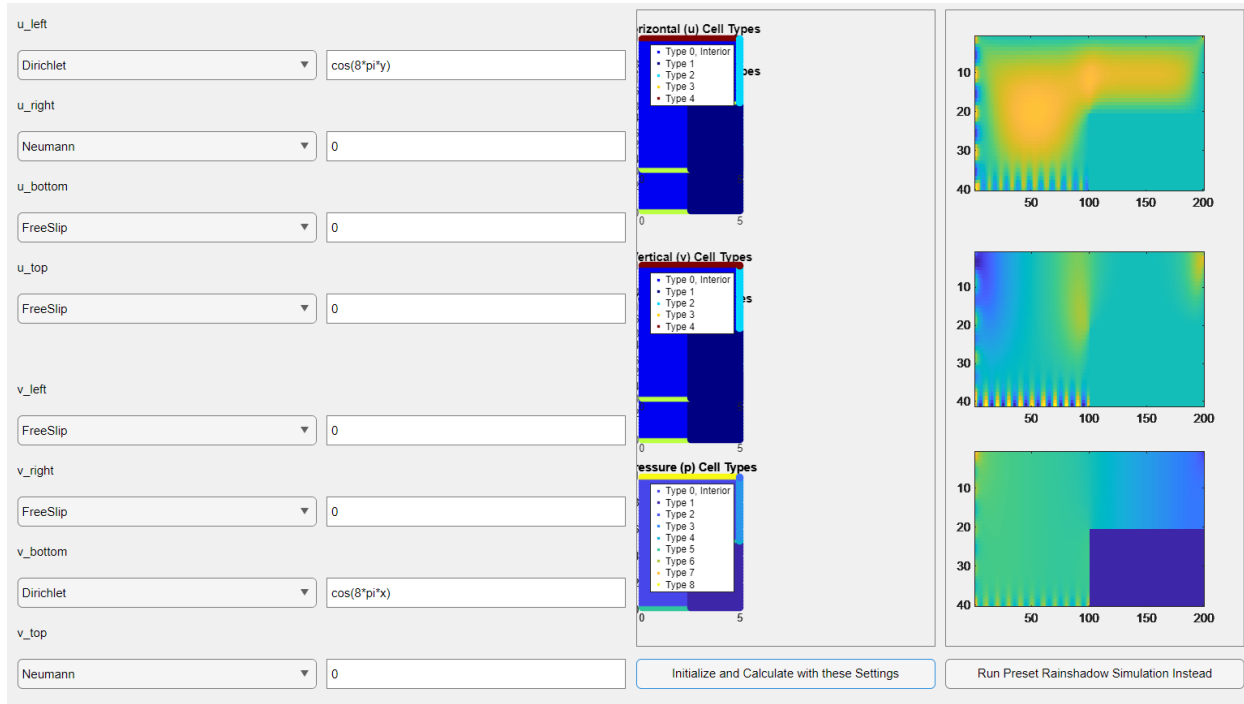
215	215
216	216
217	217
218	218
219	219
220	220
221	221
222	222
223	223
224	224
225	225
226	226
227	227
228	228
229	229
230	230
231	231
232	232
233	233
234	234
235	235
236	236
237	237
238	238
239	239
240	240
241	241
242	242
243	243
244	244
245	245
246	246
247	247
248	248
249	249
250	250
251	251
252	252
253	253
254	254
255	255
256	256
257	257
258	258
259	259
260	260
261	261
262	262
263	263
264	264
265	265
266	266
267	267
268	268
269	269
270	270
271	271
272	272
273	273
274	274
275	275
276	276
277	277
278	278
279	279
280	280
281	281
282	282
283	283
284	284
285	285
286	286
287	287
288	288
289	289
290	290
291	291
292	292
293	293
294	294
295	295
296	296
297	297
298	298
299	299
300	300
301	301
302	302
303	303
304	304
305	305
306	306
307	307
308	308
309	309
310	310
311	311
312	312
313	313
314	314
315	315
316	316
317	317
318	318
319	319
320	320
321	321
322	322
323	323
324	324
325	325
326	326
327	327
328	328
329	329
330	330
331	331
332	332
333	333
334	334
335	335
336	336
337	337
338	338
339	339
340	340
341	341
342	342
343	343
344	344
345	345
346	346
347	347
348	348
349	349
350	350
351	351
352	352
353	353
354	354
355	355
356	356
357	357
358	358
359	359
360	360
361	361
362	362
363	363
364	364
365	365
366	366
367	367
368	368
369	369
370	370
371	371
372	372
373	373
374	374
375	375
376	376
377	377
378	378
379	379
380	380
381	381
382	382
383	383
384	384
385	385
386	386
387	387
388	388
389	389
390	390
391	391
392	392
393	393
394	394
395	395
396	396
397	397
398	398
399	399
400	400

```

11
12 % Cells with a south face on a no slip boundary
13 f_x = find( grids.x_v <= x_obs_front & grids.x_u > 0 ); % Points after the obstruction
14 u_type(1,f_x) = 4; % Points ON the top of the obstruction
15 f_x = find( grids.x_u >= x_obs_front & grids.x_u < grids.x_u(end));
16 f_y = find( grids.y_u > y_obs_top, 1, 'first' );
17 u_type(f_y,f_x) = 4; % Points in the middle that NEIGHBOR the bottom boundary
18
19 u_type(:,1) = 6; % NW/East/West Edge
20
21 % Cells with the east face above obstruction
22 f = find( grids.y_u > y_obs_top );
23 u_type(f,end) = 2;
24
25 u_type(end,2:end-1) = 8; % NW/Top/North Edge
26
27 % Cells with a cell center in or on the obstruction
28 f_y = find( grids.y_s < y_obs_top );
29 f_x = find( grids.x_u >= x_obs_front );
30 u_type(f_y,f_x) = -1; % Points in the middle that NEIGHBOR the top boundary
31
32
33 % Set the v-cell type
34 % NW Set the v-cell type
35
36 % Cells ON the no slip boundary
37 f_x = find( grids.x_v < x_obs_front ); % Points before the obstruction
38 v_type(1,f_x) = 4;
39 f_x = find( grids.x_v >= x_obs_front );
40 f_y = find( abs( grids.y_v - y_obs_top ) < 10^-4 );
41 v_type(f_y,f_x) = 4; % Points ON the top of the obstruction
42
43 v_type(2:end-1,1) = 6; % NW/East/West Edge
44 v_type(end,1) = 8; % NW/Top/North Edge
45
46 % Cells with the East face ON the outlet
47 f_y = find( grids.y_v > y_obs_topndy/2 & grids.y_v < 1/2 ); % Above the obstruction and below the top
48 v_type(f_y,end) = 2;
49
50 % Cells with the East face on the no-slip obstruction
51 f_x = find( grids.x_v >= x_obs_front, 1, 'last' );
52 f_y = find( abs( grids.y_v - y_obs_top ) < 10^-4 );

```

The result is that the code can successfully simulate GUI input conditions:



Congratulations to Professor Michael Allshouse for his new position in Ohio! It was a pleasure taking classes with you and seeing you on the marathon course.