Samuel Levin
2 May 2018
GISC6384
Dr. Qiu

GISC6384 Final Project: Ideal Observer Locations Algorithm for sUAS Mission Planning

INTRODUCTION

This project was conducted in support of the UT Dallas *Smart Campus Project*, in association with the GAIA Lab. As part of this ongoing project, drones will be used to survey the entirety of the UT Dallas campus. Multispectral imagery acquired by these drones will be used to construct three-dimensional models, which form the foundations of the project's geospatial data infrastructure. However, flying drones in a large, populous area such as the UT Dallas campus requires careful flight planning to ensure a safe and legal mission. For one, the Federal Aviation Authority (FAA) requires that all visual line-of-sight (VLOS) between the drone and collective ground crew is maintained at all times during the flight. The drone must be easily visible without aid such as binoculars and cannot at any move into an obscured position where VLOS is not maintained by at least one ground observer (FAA 2016). Thus, an algorithm was developed to identify ideal observer locations that will maximize VLOS with the minimum number of observers.

PROBLEM OBJECTIVE

Due to the limitations of human visual acuity and many visual obstructions such as buildings and trees, VLOS cannot be maintained using only one ground observer on the UTD campus. Thus, an algorithm was designed to locate the ideal observer locations for any defined survey area. This algorithm makes use of a LiDAR derived surface model and Python scripting to identify the minimum number of observers and exact coordinate locations that will fulfill the VLOS mission requirements. Furthermore, the product of this algorithm can be used as a supplement to FAA Part 107 Authorizations and Waivers, demonstrating that the project has taken necessary steps to ensure a safe mission.

LITERATURE REVIEW

Viewshed analysis in geospatial science has applications ranging from telecommunications infrastructure to archaeological site analysis. Viewshed processes work by assessing the line of sight from the observer location to the target location and all elevations in between. If an elevation value along this line of sight is greater than that of the observer or target, the line of sight is broken, and the target is obscured. This process is extremely computationally intensive, and alternative algorithms have harnessed the power of GPU shading to produce similar outputs with lower processing costs (Feng et al. 2015). Nevertheless, the viewshed multiple observer points can be merged to create a cumulative viewshed raster, identifying which surface locations are most visible to the set of observers (Kang-Tsung 2010). However, the inverse of this problem is substantially more challenging. Locating the set of observers that maintain a particular viewshed condition is more difficult due to the number of potential combination of locations possible on a surface. The problem defined in this project can be broadly recognized as part of the *multiple-observer sitting problem*, which has been the subject of multiple GIS and computer science publications. One such example assembles list of potential observer locations by identifying peaks in a DEM and iteratively making new combinations of observers until a comprehensive viewshed is achieved (Kim, Rana, and Wise 2004). Another iteratively identifies a best observer from a DEM's total viewshed, masks out the area viewable by this observer, and continues the process on the remaining DEM until all locations have been collectively viewed (Cervilla, Tabik, and Romero 2015). This problem frequently arises due to the inability of most commercial GIS packages to perform viewshed optimization functions where multiple observers are desired in the output. While ArcGIS functions such as Viewshed 2, enable cumulative viewsheds to be produced from an array of observer features, the combination of individuals observers contributing to the viewshed is obscured in the output. Moreover, this *Best Observers* processing method is limited to only 16 observer features – hardly sufficient for the scale of the intended project (ESRI 2012). Furthermore, existing studies have worked with observer location situated directly on or uniformly above the elevation surface. In the problem explored here, the observer locations and terrain surface (UAV flight points with AGL altitude and DTM) are based on different elevation surfaces. Thus, a novel solution to the multiple-observer sitting problem needed to be developed, allowing observers on the ground surface to be optimized for airspace visibility.

DATA SOURCES

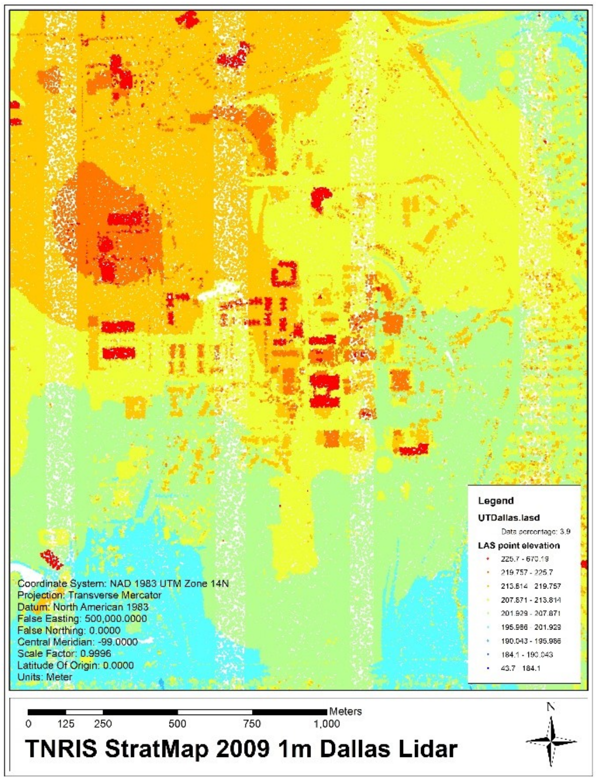
The most significant data requirement of this project is an accurate Digital Surface Model (DSM) of the study area, complete with all above-ground-level visual obstructions. While DEMs that cover the UTD study area are readily available, surface models are not. However, raw LiDAR LAS files are available from Texas Natural Resources Information System (TNRIS). These LiDAR point clouds with 0.5 meter nominal point spacing are well suited to constructing a DSM. However, this data was collected in 2009 and therefore lacks certain recent construction features, a data gap that is addressed in the following section. In additional, a standard Digital Terrain Model (without above-surface obstructions) is also needed so that sUAS above-ground-level flight altitudes can be calculated. Finally, basic parameters of the drone flight plan are required to generate an array of locations the drone will occupy during flight. For this particular flight, the drone will fly at an altitude of 121.92 meters (400 feet) above-ground-level, with a transect spacing of 19 meters. These simple data requirements should be fulfillable by any other mission where comparable LiDAR data exist, including much of Texas where TNRIS can provide this resource. This underscores the algorithm's relevance to future missions needing to mitigate VLOS challenges.

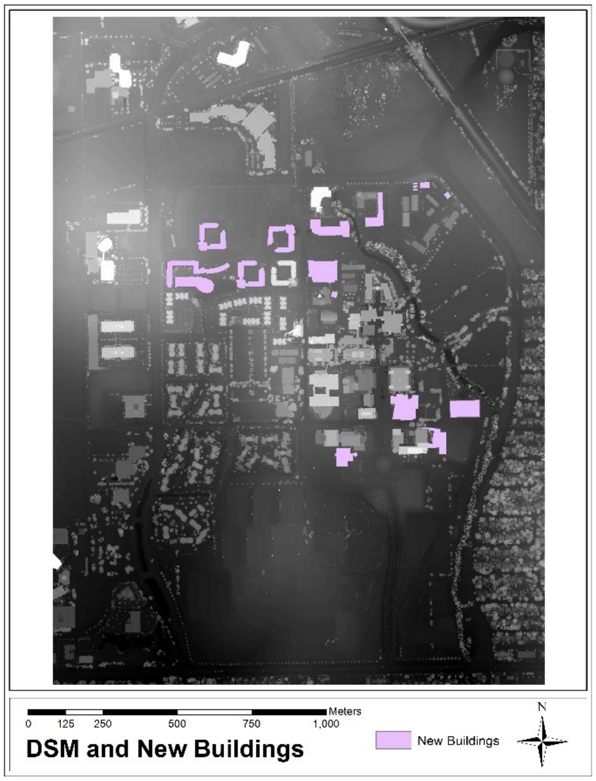
ANALYSIS AND METHODOLOGY

Part One: Preparing Data Sources

The TNRIS provided LiDAR data were used to create a DSM raster that includes all above ground surface obstructions. This was accomplished using a two LAS tiles in an LAS Dataset that collectively covered the entire campus. The highest elevation surfaces were defined by filtering the LAS dataset by first return points. From these points, the *LAS Dataset to Raster* tool was used to produce a DSM. As an additional parameter for this tool, the interpolation method was set to binning, with a cell assignment type of MAXIMUM. Cell size was set to 1 meter. Thus, the first return LiDAR point with the maximum (greatest Z) value in a given 1x1 meter area was used as the final cell value in the DSM.

```
arcpy.LasDatasetToRaster_conversion(UTDallas_lasd,surface_1m,
"ELEVATION", "BINNING MAXIMUM LINEAR", "FLOAT", "CELLSIZE", "1", "1")
```

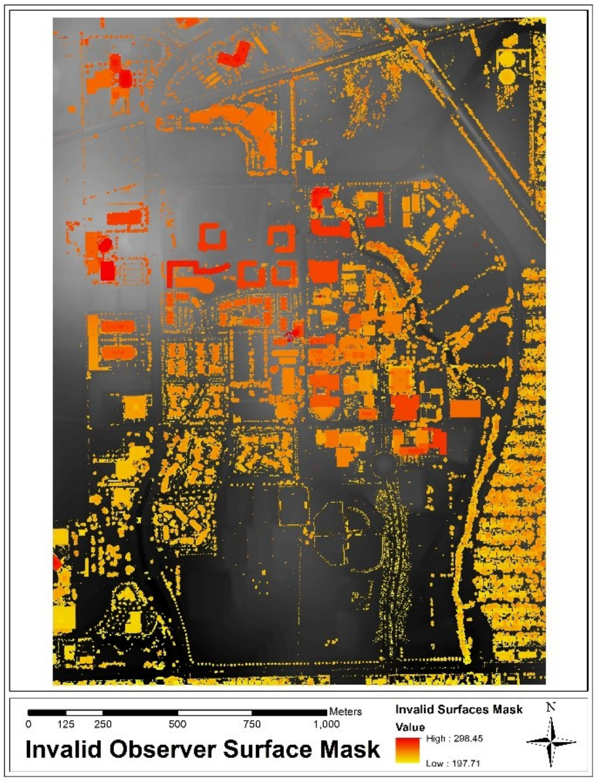


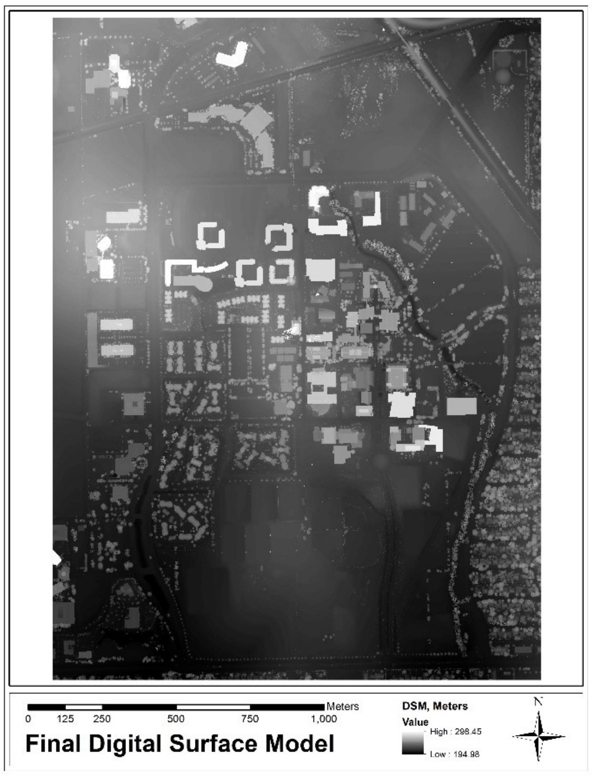


While this effectively produced a DSM, recent buildings constructed since the LiDAR acquisition in 2009 were not represented. These buildings were digitized using ESRI basemap imagery. The polygons were attributed with two elevations. First, the building base mean-sea-level (MSL) elevation was interpolated from the TNRIS DEM. Second, each building's above-ground-level (AGL) elevation was attributed based on measured heights in Google Earth 3D viewer. By adding the AGL elevation to the base MSL elevation, the final MSL height of each building was calculated for new buildings. These polygons were then rasterized and final MSL height incorporated into the DSM using a conditional statement.

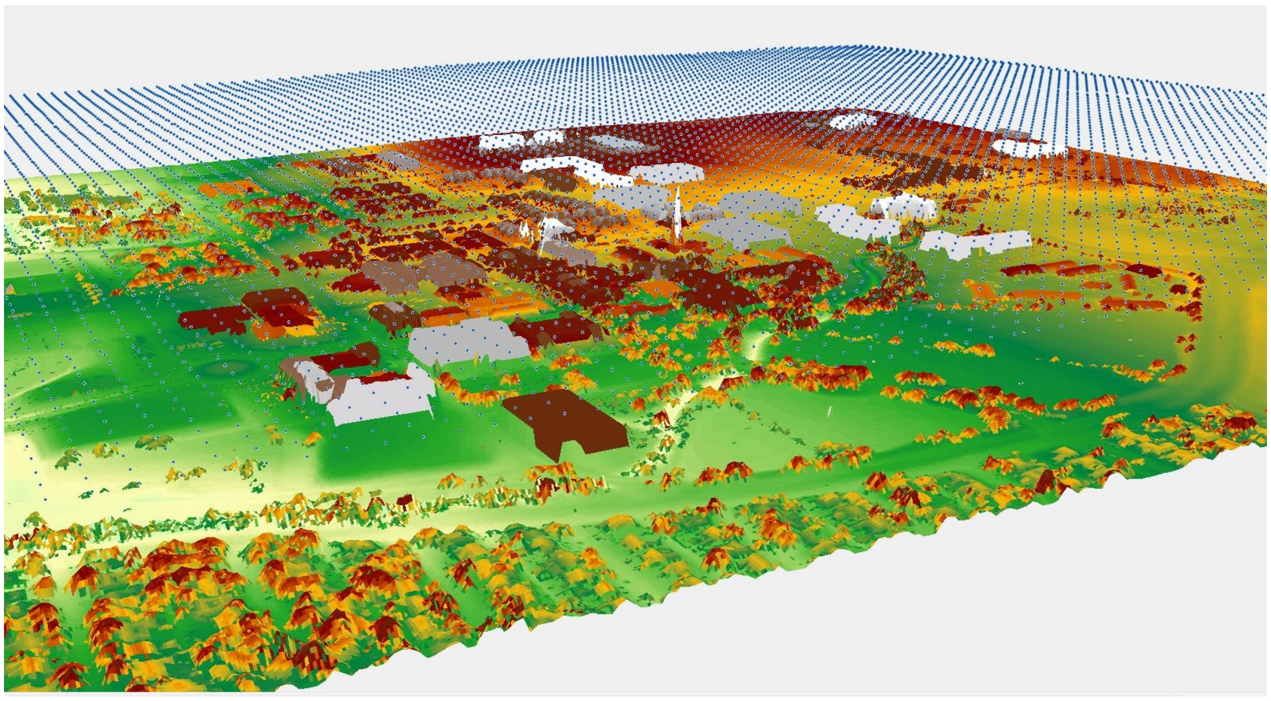
```
Con(IsNull("Buildings"),"DSM","Buildings").
```

With the DSM complete, a binary mask of invalid observer locations needed to be generated. These invalid observer locations include the rooftops of buildings, and any other above ground level obstruction that would prevent an observer from standing at the given location. This was again created using the *LAS Dataset to Raster* tool, with the LAS dataset filtered by non-ground points. Again, the new buildings were added to this mask using a conditional statement. Finally, the mask was reclassified to binary format for use in *Raster Calculator* operations in the best observers algorithm.





The next preparation requirement was creation of drone flight points, an array of 3D point features representative of the drone's location at any time during the flight. The *Create Fishnet* tool was used to generate linear transects across campus with 19 meter spacing. Each line was then converted to points at 19 meter distance intervals. Thus, an array of points representing the drone location at 19 x 19 meter intervals was generated. These 2D flight points were then converted to 3D flight points using the *Interpolate Shape* tool, adding a Z-Value each point based on the TNRIS DTM. Each point's final flight elevation was then defined by adding 121.92 meters (400 ft) to the this Z-Value and storing the final MSL altitude in the attribute table.



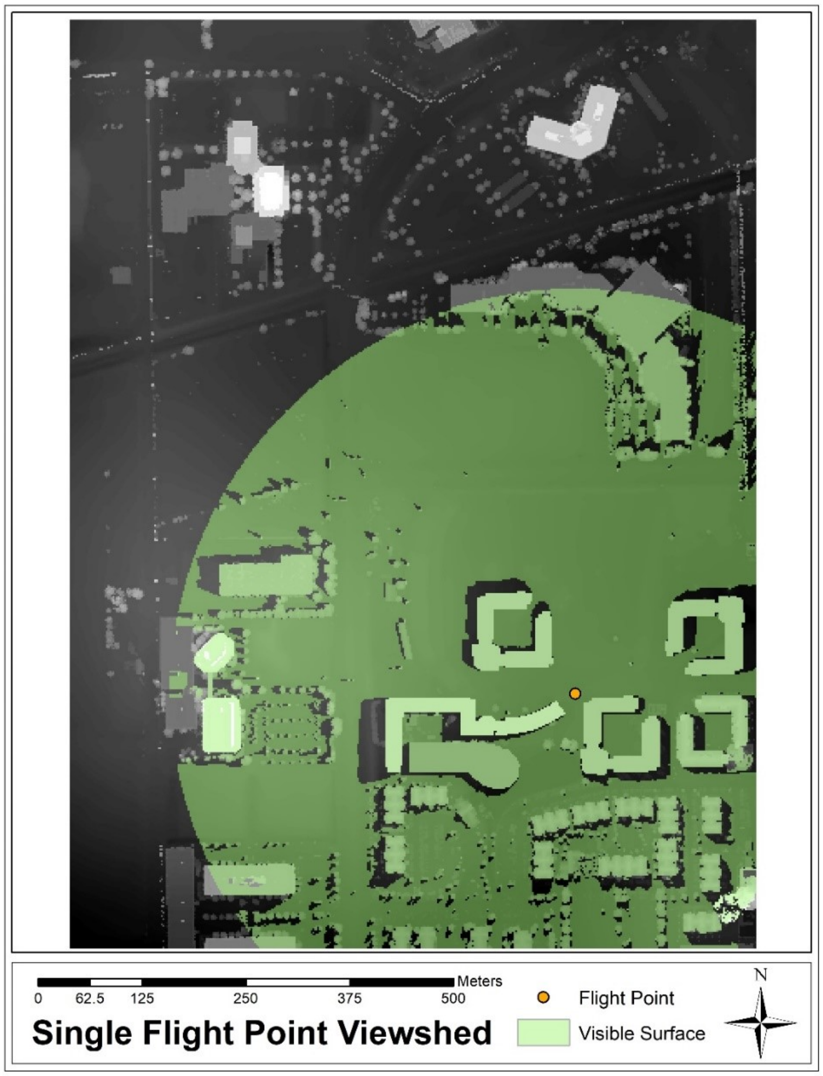
Finally, the UTD Campus needed to be divided into quadrants. This was achieved using the *Create Fishnet* tool to divide a bounding rectangle of the study area into 4 equally sized quadrants. Each quadrant was then exported as a discrete feature class (i.e. StudyArea_NW). Similarly, flight points for each quadrant were clipped to their respective study area (i.e. FlightPts_NW).

__Part Two: Designing and Running the Algorithm__

****All Python code written for this algorithm is included in the SOURCE CODE section of this report. .py files are also submitted for testing and experimentation****

With all data prepared, the algorithm works as follows:

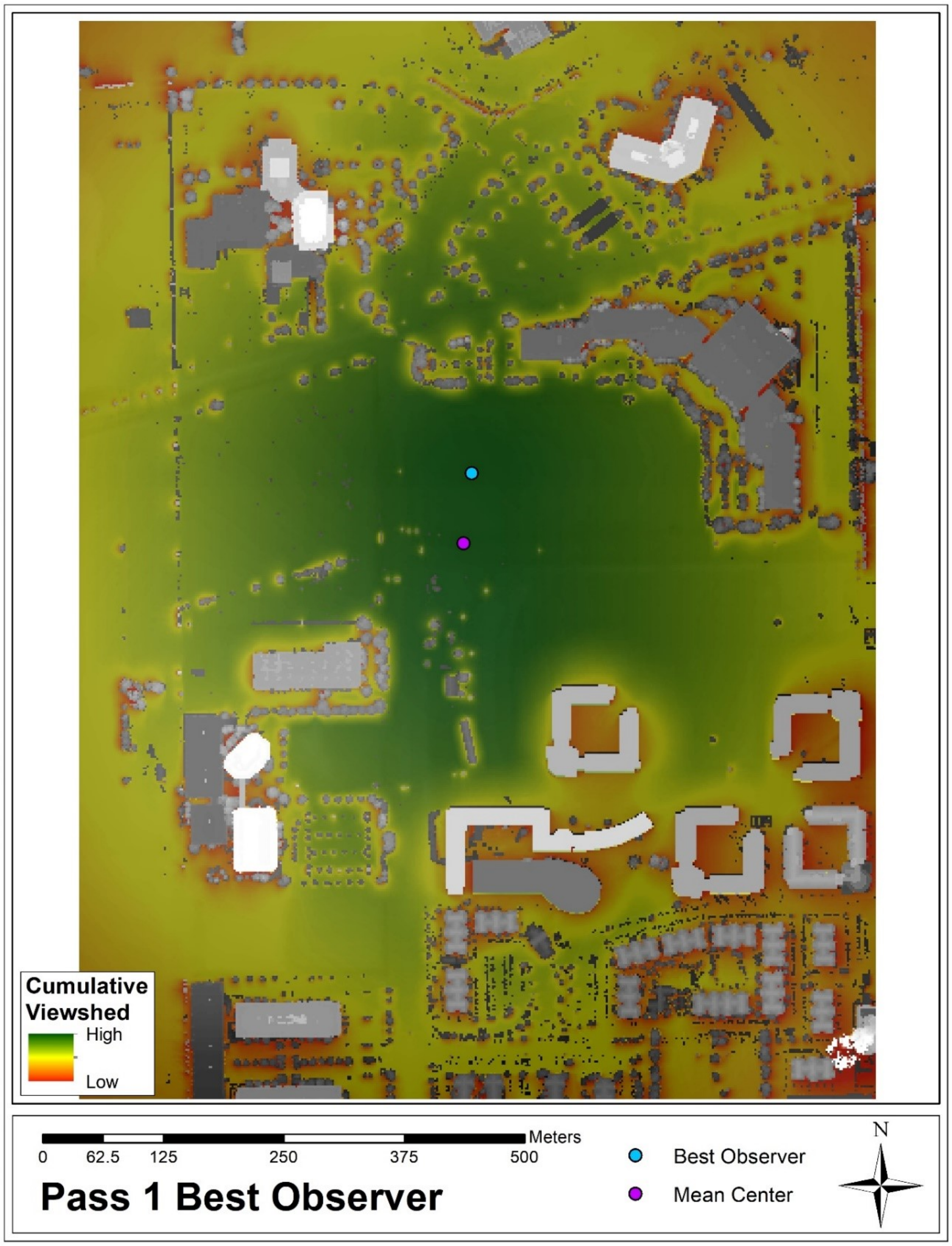
The array of all flight points is split into individual flight point feature classes, maintaining the 3D Z-value. From each individual flight point, the Viewshed 2 is used to create a binary viewshed raster of visible and not-visible locations on the DSM raster (clipped to the study area quadrant). Parameters are set to a surface offset of 1.63 meters (average eyelevel of an adult-human) and outer radius of 500 meters (a conservative estimate of the limits of human visual acuity). Thus, the surface locations where a human would be able to maintain VLOS with each flight point are generated. Note that at this point invalid surface locations have not been masked out – this is addressed in a later step.

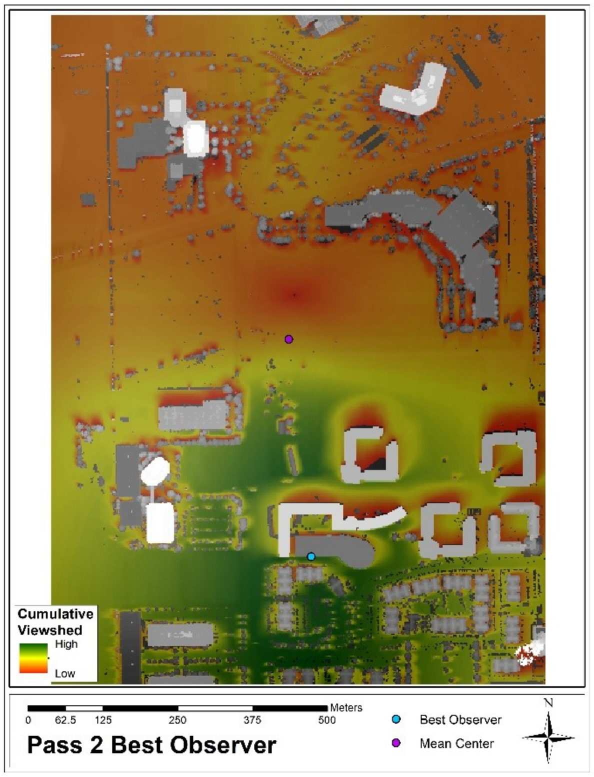


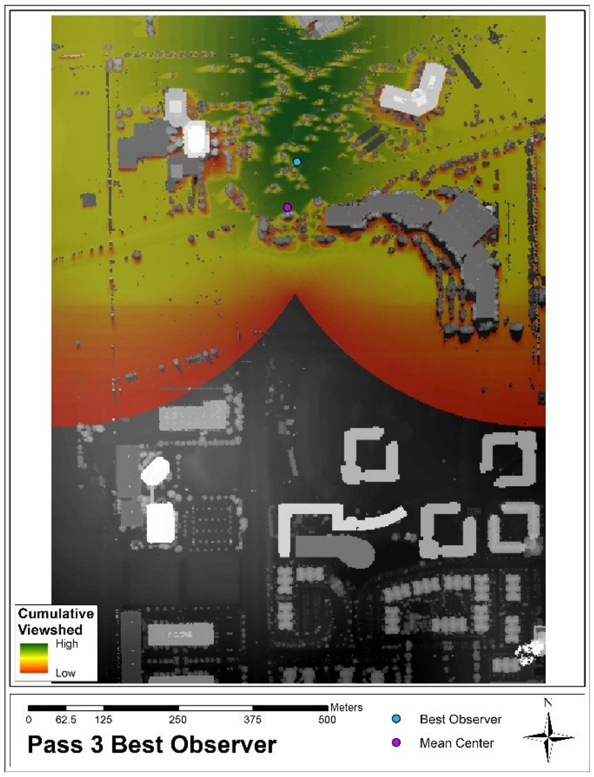
The algorithm begins by generating a list of all flight points, identified by their ObjectID. This serves as a list of unaccounted flight points where VLOS has not yet been achieved. The basic mechanism of the algorithm works as follows: 1) identify an ideal observer location 2) find which flight points are visible from that location 3) remove these flight points from the list of unaccounted flight points. This iterative process repeats until all flight points have been accounted for.

For each pass of the algorithm, the remaining list of unaccounted flight points is used to create a cumulative viewshed raster of potential observer locations. This sums the individual viewshed rasters from each flight point in the list, masks out invalid observer surfaces, and sets locations that are not visible or invalid in that cumulative viewshed to NoData. From this cumulative raster, the centroid is generated and saved as a point feature. Next, the cumulative raster is converted to surface points. The attribute table of surface points is populated with the number of visible flight points and XY coordinates at each location. The Euclidean from each surface point to the centroid is then generated. Using the Pandas python library, the surface points and distance table are merged. This merged table is sorted by two values: first, by the gridcode in descending order; second, by the distance to centroid in ascending order. Thus, the first entry in sorted table has both the *highest gridcode* (visible flight points) and *lowest distance to the centroid*. This location is selected as the best observer for the first iteration of the algorithm. The XY coordinates of this location, also in the sorted table, are saved in a new Pandas dataframe as a best observer.

With the XY location of the best observer selected, flight points visible from this location must be removed from list of unaccounted flight points. For each remaining point, it viewshed raster is queried to determine if the value is 1 (visible) or NoData (not-visible) at the best observer point. Points that are determined to be visible are removed from the list of unaccounted flight points. Points that are not visible remain in the list of unaccounted flight points If the length of unaccounted flight points is greater than zero, the algorithm repeats. This continues until all flight point have been accounted for by a best observer, ensuring that full VLOS has been collectively achieved. Finally, the coordinates of best observers are used to create a point feature class of selected best observer locations.

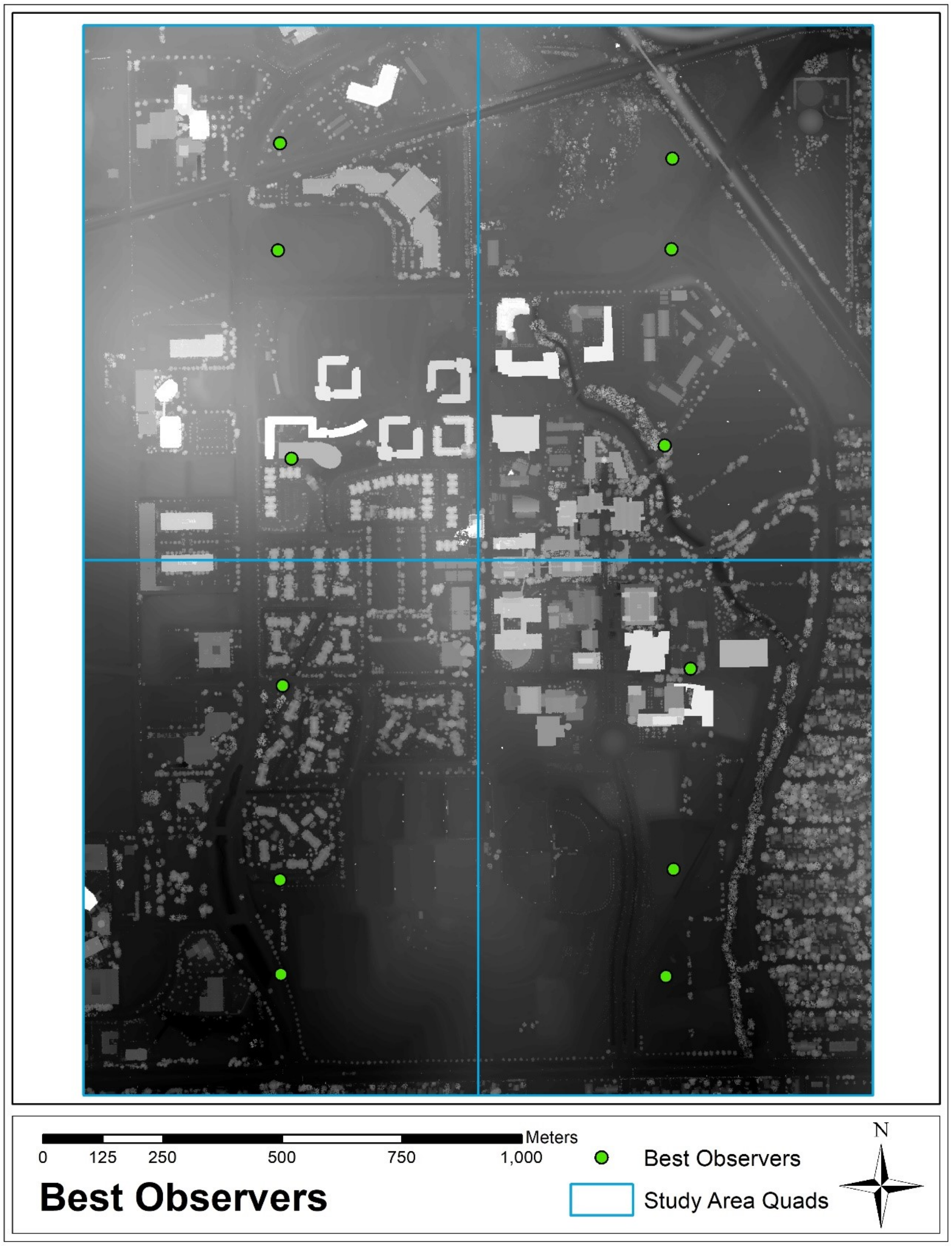






RESULTS AND DISCUSSION

The best observer points clearly identify ideal locations to position viewers for each sUAS flight. This is accomplished using the minimum number of observers, where each one has the greatest possible contribution in accounting for the remaining flight points. However, though the point file of best observers is the final output of the algorithm, it also produces a number of intermediate results such as the cumulative viewshed raster. While a single point is identified in each pass of the algorithm, viewing this point in relation to the cumulative viewshed raster helps to understand general areas of high and low visibility. As unseen flight points are removed, these areas of peak visibility shift to reflect only the remaining points. In this example, ideal observer locations are distributed in a north-south linear arrangement, centered on the study area. This is expected, as these locations maximize east-west coverage, introducing additional observers as the maximum north-south limits are reached. This processing algorithm allowed observer locations from all four quadrants to be generated quickly and accurately, with certainty that these locations fulfill necessary FAA requirements.



CONCLUSIONS

This algorithm provides a simple solution to VLOS compliance for all potential drone operations. In particular, it is useful when a Part 107 waiver is being requested. These waivers require additional safety precautions that must be thoroughly support and documented. By providing an idea observer analysis using the algorithm presented here, the Remote Pilot in Command can demonstrate with certainty that VLOS will be maintained throughout the flight. This is especially relevant for flights over populous areas, flights with pronounced above-ground-level obstructions, or flights in close proximity to controlled airspace – all of which are encountered on the Smart Campus project.

The relatively simple data requirements of publicly available LiDAR and standard Digital Terrain Models can be easily achieved in other projects. Once an acceptable digital surface model, invalid surfaces mask, and flight points have been generated, the algorithm can be run by simply directing the Python script to the correct data sources. In the future, the user friendliness of this algorithm can be enhanced by implementing it as a Python script tool that can be run from within ArcGIS with the GUI to select parameters.

Additional functionality can also be added in the future by adding a field to the attribute table of all flight points that denotes which best observer monitors each point. This will allow the flight points to be symbolized in an intuitive fashion that easily denote areas of responsibility each observer has on the drone's flight path.

REFERENCES

Cervilla, A.R., S. Tabik, and L.F. Romero. 2015. "Siting Multiple Observers for Maximum Coverage: An Accurate Approach." *Procedia Computer Science* 51 (1). Elsevier Masson SAS: 356–65. https://doi.org/10.1016/j.procs.2015.05.255.

ESRI. 2012. "Using Viewshed and Observer Points for Visibility Analysis." ArcGIS Resources. 2012. http://resources.arcgis.com/en/help/main/10.1/index.html#//009z000000v8000000.

FAA. 2016. "Advisory Circular: Small Unmanned Aircraft Systems (sUAS)." *AC No. 107-2*. U.S. Department of Transportation. https://doi.org/AFS-800 AC 91-97.

Feng, Wang, Wang Gang, Pan Deji, Liu Yuan, Yang Liuzhong, and Wang Hongbo. 2015. “A Parallel Algorithm for Viewshed Analysis in Three-Dimensional Digital Earth.” *Computers & Geosciences* 75 (February). Elsevier: 57–65. https://doi.org/10.1016/j.cageo.2014.10.012.

Kang-Tsung, Chang. 2010. *Introduction to Geographic Information Systems*. 5th ed. New York, NY: McGraw-Hill.

Kim, Young-Hoon, Sanjay Rana, and Steve Wise. 2004. “Exploring Multiple Viewshed Analysis Using Terrain Features and Optimisation Techniques.” *Computers & Geosciences* 30 (9–10): 1019–32. https://doi.org/10.1016/j.cageo.2004.07.008.

TNRIS. 2009. *StratMap 2009 1m Dallas Lidar*, Texas Natural Resources Information System (TNRIS). https://tnris.org/data-catalog/entry/stratmap-2009-1m-dallas/

SOURCE CODE

```
# -------------------------------------------------------------
# SplitFlightPts.py
# Split all flight points into invididual feature classes by OBJECTID
# Revised 28 April 2018
# Written by Samuel Levin
# -------------------------------------------------------------

import arcpy

# set flightpts equal to quadrant flight points feature class
flightpts =
'C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Fli
ghtPts_SE'
# get number of flight point features
n = int(arcpy.GetCount_management(flightpts).getOutput(0))
print('Splitting {} flight points...'.format(n))


# add OID_orig field with original OBJECTID. Uses so that original flight
point location can be retained
arcpy.AddField_management(flightpts,'OID_orig', 'SHORT')
arcpy.CalculateField_management(flightpts,'OID_orig','!OBJECTID!','PYTHON')
print('OID_orig field populated.')

# split flightpts into individual feature classes of 1 point each
for i in range(1,n+1):
    out_path =
'C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb'
    out_name = 'fltpt_' + str(i)
    #where = 'Split = ' + "'{}'".format(str(b[0]))
    where = 'OID_orig = {}'.format(i)
    print('Exporting flight point where {}'.format(where))
    arcpy.FeatureClassToFeatureClass_conversion(flightpts, out_path,
out_name, where)
    print('Feature class {} exported'.format(out_name))

print('Process complete. All flight points split.')
```

```
# --------------------------------------------------------
# RunViewshed.py
# Run Viewshed analysis on each individual flight point
# Revised 28 April 2018
# Written by Samuel Levin
# --------------------------------------------------------

import arcpy
arcpy.CheckOutExtension('Spatial')
arcpy.CheckOutExtension('3D')

# get number of flight point features
flightpts =
'C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Fli
ghtPts_SE'
n = int(arcpy.GetCount_management(flightpts).getOutput(0))
print('Running viewshed analysis on {} flight points...'.format(n))

for p in range(1,n+1):

    # variables:
    surface =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Sur
face_SE_2m"
    fltpt_ =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\flt
pt_" + str(p)
    vs_ =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\vs_
" + str(p)
    Output_above_ground_level_raster = ""
    Output_observer_region_relationship_table = ""

    print('Running viewshed analysis on fltpt_{}'.format(p))
    # Refractivity coeff = 0.13 (default), Surface Offet = 1.63, Outer radius
500 meters(3D distance), -90 to 0 vertical
    arcpy.gp.Viewshed2_sa(surface, fltpt_, vs_,
Output_above_ground_level_raster, "FREQUENCY", "0 Meters",
                          Output_observer_region_relationship_table, "0.13",
"1.63 Meters", "Altitude", "0 Meters", "",
                          "GROUND", "500 Meters", "3D", "0", "360", "0", "-
90", "ALL_SIGHTLINES")
    print('fltpt_{} viewshed complete.'.format(p))

print('\n')
print('All viewshed processes executed.')
```

```
# ----------------------------------------------------------
# CumulativeVS.py
# Calculate cumulative viewshed and generate potential observer points
# Revised 28 April 2018
# Written by Samuel Levin
# ----------------------------------------------------------
import arcpy
from arcpy.sa import *
arcpy.CheckOutExtension('Spatial')
arcpy.CheckOutExtension('3D')


def makechunks(lst, n):
    """
    Divides unseen flight points into chunks to make raster summation more
efficient.
    :param lst: List of remaining unseen flight points
    :param n: Size of chunks
    :return: Generator object, iterating chunks of unseen flight points
    """
    for i in range(0,len(lst),n):
        yield lst[i:i+n]

def calcVS(unsnfltpts, bldg_veg_mask, ct, datapath):
    """
    Calculates the cumulative viewshed from remaining unseen flight points.
Generates Euclidean distance
    to mean center table.
    :param unsnfltpts: List of remaining unseen flight points
    :param bldg_veg_mask: Binary mask to remove invalid observer surfaces
from cumulative VS raster
    :param ct: Pass number
    :param datapath:  Path to input/output directory
    """

    print('Calculating cumulative viewshed for {} unseen flight
points...'.format(len(unsnfltpts)))
    # Make chunks of flight points
    usfp_chunks = makechunks(unsnfltpts,500)
    print('Flight point chunks generated')
    print(len(chunk) for chunk in usfp_chunks)
    chunk_sums = []
    chunkpass = 1
    # Sum each chunk of single viewshed rasters
    for chunk in usfp_chunks:
        print('Chunksum operation {} on {} flight
points...'.format(chunkpass, len(chunk)))
        # Set null values equal to 0 to avoid NoData holes
        chunkgen = (Con(IsNull(arcpy.Raster(datapath + "vs_" + str(usfp))),
0, 1) for usfp in chunk)
        chunkstats = arcpy.sa.CellStatistics(chunkgen,'SUM','NODATA')
        chunk_sums.append((chunkstats))
        print('...Done.')
        chunkpass += 1
    # Sum chunks
    sumrast = arcpy.sa.CellStatistics(chunk_sums,'SUM','NODATA')
    sumrast.save(datapath + "vs_pass_" + str(ct) + "_unmasked")
    print('Unmasked cumulative viewshed saved.')

    # mask out buildings and vegetation
    # set Bldg_Veg_Mask cells to 0
    unmasked = arcpy.Raster(datapath + "vs_pass_" + str(ct)+"_unmasked")
    cumulative_masked = unmasked * bldg_veg_mask
    print('Invalid observer surfaces masked.')
    # set 0 value cells to Null
    cumulative_masked = SetNull(cumulative_masked == 0,cumulative_masked)
    print('Setting null values.')
    # save to .GDB as cumulative raster
    cumulative_masked.save(datapath + "vs_pass_" + str(ct))
    print('Masked cumulative viewshed saved.')

    # Convert raster to points with number views for VS pass and X Y location
    vs_total_pts_ = datapath + "vs_pass_" + str(ct) + "_pts"
    arcpy.RasterToPoint_conversion(cumulative_masked, vs_total_pts_)
    arcpy.AddGeometryAttributes_management(vs_total_pts_,['POINT_X_Y_Z_M'])
    print('Viewshed points for pass {} generated'.format(ct))
    # Find mean center of cumulative viewshed for pass, save as feature class
    vs_center_ = datapath + "vs_pass_" + str(ct) + "_cntr"
    arcpy.MeanCenter_stats(vs_total_pts_,vs_center_)
    print('Mean center calculated.')
    # Calculate distance of each observation from centroid of observer
masspoints
    vs_dist_ = datapath + "vs_pass_" + str(ct) + "_dist"
    arcpy.PointDistance_analysis(vs_total_pts_,vs_center_,vs_dist_)
    print('Observer distances table calculated.')
```

```
# ---------------------------------------------------------
# BestObserver.py
# Find best observation based on potential observation points views and
distances from mean center
# Adjust list of unseen flight points
# Revised 28 April 2018
# Written by Samuel Levin
# ---------------------------------------------------------

import arcpy
import pandas as pd

def feature_class_to_pandas_data_frame(feature_class, field_list):
    """
    Load data into a Pandas Data Frame for subsequent analysis.
    :param feature_class: Feature class.
    :param field_list: Desired fields.
    :return: Pandas DataFrame object.
    """
    return
pd.DataFrame(arcpy.da.FeatureClassToNumPyArray(in_table=feature_class,field_n
ames=field_list,

skip_nulls=False,null_value=-99999))


# Set return of this function to bestloc variable in findvisible()
def findbestobs(ct,datapath,best_df):
    """
    Find best observer by merging raster points to distance table, sorting by
grid_code and distance
    :param ct: Pass number
    :param datapath: Path to input/output directory
    :param best_df: Pandas DataFrame of best observers
    :return: Pandas DataFrame of best observers
    """
    vspts = datapath + "vs_pass_" + str(ct) + "_pts"
    vsfields =  ['OBJECTID', 'POINT_X','POINT_Y','grid_code']
    vs_df = feature_class_to_pandas_data_frame(vspts, vsfields)
    dst = datapath + "vs_pass_" + str(ct) + "_dist"
    dstfields = ['OBJECTID','DISTANCE']
    dst_df = feature_class_to_pandas_data_frame(dst,dstfields)
    vsdist = vs_df.merge(dst_df, on='OBJECTID')
    vsdist.sort_values(['grid_code', 'DISTANCE'], ascending=[False,
True],inplace=True)
    x = vsdist.iloc[0]['POINT_X']
    y = vsdist.iloc[0]['POINT_Y']
    # Append X Y cooridnates of best observation to DataFrame of all best
    best_df = best_df.append(vsdist.iloc[0],ignore_index=True)
    print('Best observer for pass {} located'.format(ct))
    print(best_df)
    # Return X Y coordinates of best observation for the current pass
    print('\n')
    return best_df
```

```
def findvisible(datapath,unseenfltpts,x,y):
    """

    :param datapath: Path to input/output directory
    :param unseenfltpts: List of remaining unseen flight points
    :param x: X coordinate location
    :param y: Y coordinate location
    :return: List of observed flight points
    """
    coord = str(x) + " " + str(y)
    observed = []
    for usfp in unseenfltpts:
        print('Determining visibility of flight point {}'.format(usfp))
        cell = arcpy.GetCellValue_management(datapath + "vs_" +
str(usfp),coord)
        try:
            if int(cell.getOutput(0)) == 1:
                observed.append(usfp)
                print('Visible.')
        except:
            print('Not visible.')
    print('Number of visible flight points: {}'.format(len(observed)))
    print(observed)
    return observed
```

```
# --------------------------------------------------------
# FindBest.py
# Final script for running viewshed best observation analysis
# Revised 28 April 2018
# Written by Samuel Levin
# --------------------------------------------------------

import arcpy
import pandas as pd
import CumulativeVS
import BestObservers


surface =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Sur
face_SE_2m"
mask =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Bld
gVegMask_V2_bin"
path =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\"

flightpts =
'C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\UTD_Viewshed_SEV1.gdb\\Fli
ghtPts_SE'
n = int(arcpy.GetCount_management(flightpts).getOutput(0))
unseen_fltpts = list(range(1,n+1))

count = 0
best = pd.DataFrame()

passvis = {}
while len(unseen_fltpts) > 0 and count < 10:
    count += 1
    print('\n')
    print('PASS {}'.format(count))
    CumulativeVS.calcVS(unseen_fltpts,mask,count,path)
    best = BestObservers.findbestobs(count,path,best)
    best_x= best.iloc[count-1]['POINT_X']
    best_y= best.iloc[count-1]['POINT_Y']
    passviewed = BestObservers.findvisible(path,unseen_fltpts,best_x,best_y)
    passvis[count] = passviewed
    unseen_fltpts = [us for us in unseen_fltpts if us not in passviewed]
    print('REMAINING UNSEEN: {}'.format(len(unseen_fltpts)))

savebest =
"C:\\Users\\sjl170230\\Documents\\UTD_Viewshed_V3\\OutTables\\bestobservers.c
sv"
best.to_csv(savebest)
print('Best observers CSV exported to: {}'.format(savebest))
print('\n')
print('PROCESS COMPLETE.')
print('BEST OBSERVERS: ')
print(best)
for pv in passvis:
    print(pv)
    print('\n')
```