

RT Systems Lab

**Weekly Report number (4)**



Contact Us

Julian Ewaied 215543497 Namir Ballan 326165156

Contents

[This Week's Tasks 3](#_Toc142511047)

[Building more interactive depth-map 3](#_Toc142511048)

[Presenting 3D results 4](#_Toc142511049)

[Making the code more elegant 5](#_Toc142511050)

[Making the code more efficient 6](#_Toc142511051)

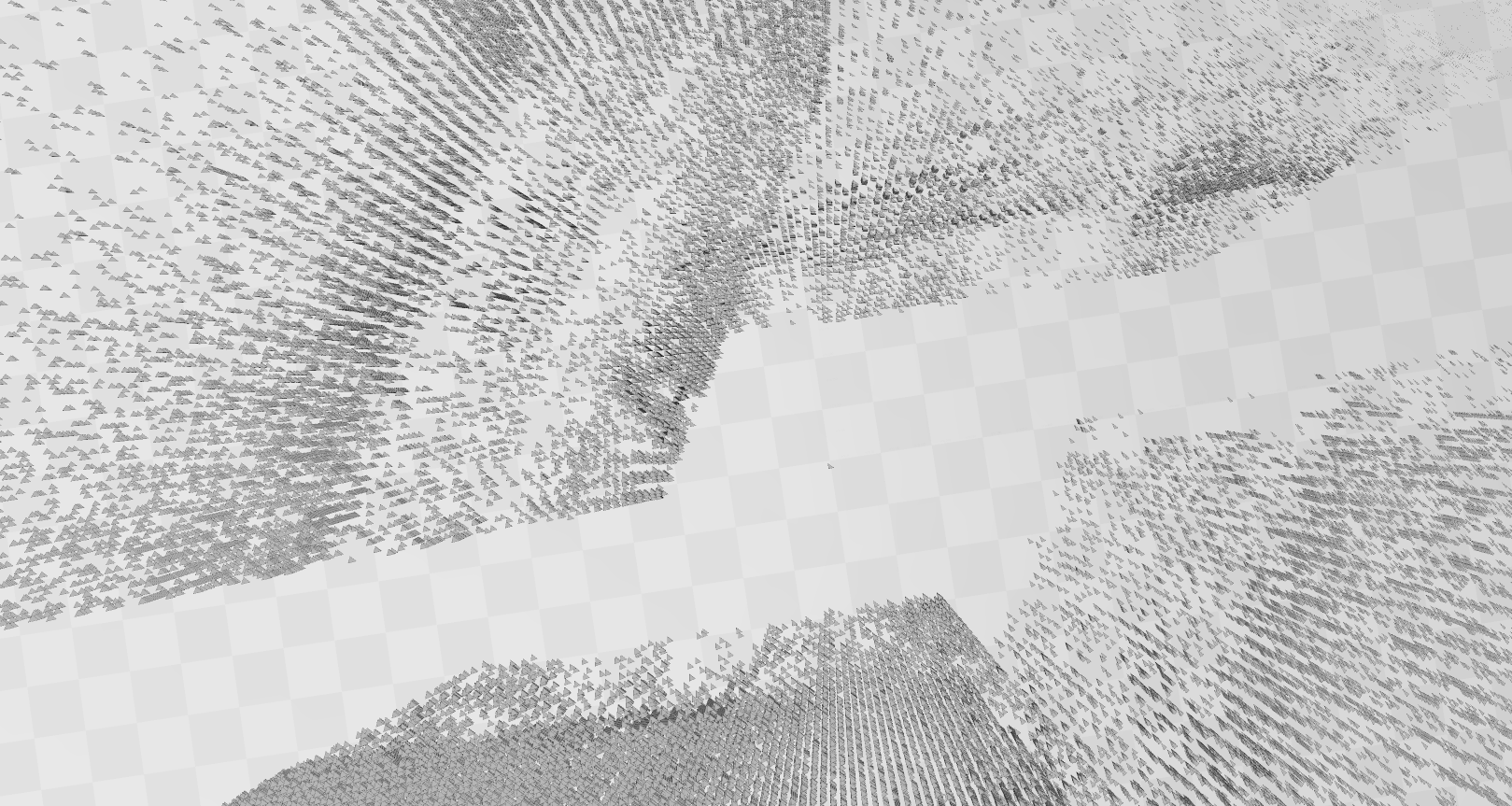
[Next Week's Tasks 7](#_Toc142511052)

# This Week's Tasks

* Checking the rotation is done well
* Matching between two representations of the same point in two different frames
* Making the code more efficient
* Automating the filtering policy

## Checking Rotations

We started by looking at the 3D representation as it is more flexible, and noticed something odd in the output when we zoomed in:



The shape of the rooms was actually well calculated, just covered with way too many outliers! This made our next mission very clear – finding the best filtering policy. Before moving to this problem, we must mention that the map has 2 gaps (exits), which is very problematic.

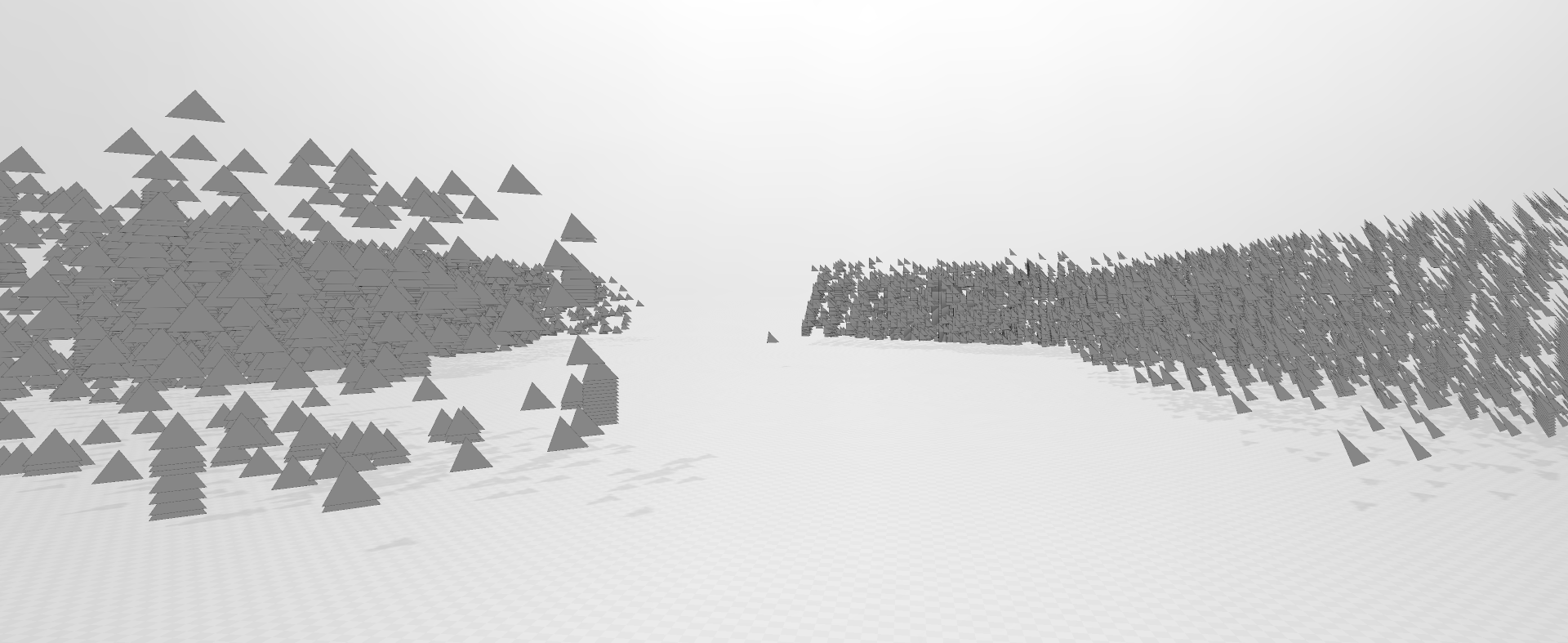
## Automating Filtering Policy

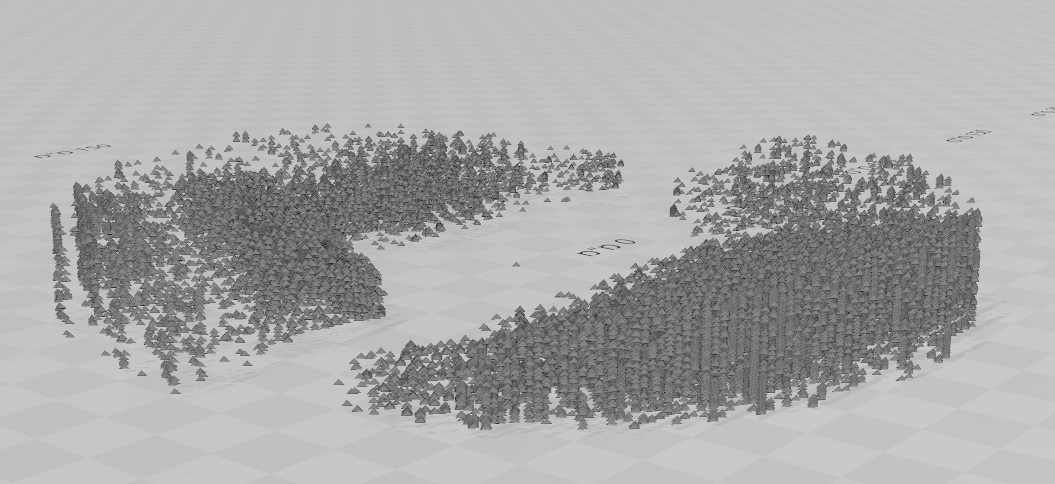
We considered multiple options, of which we mention the following:

1. Manually decided bounded interval of SAD values, which we found ineffective last week.
2. ML the boundaries, which is currently unsupervised meaning not possible.
3. Using computational geometry algorithms, of which we needed to do some more research.

## Matching Points

We can notice that points that we get are points with a shifted height (y coordinates). This is problematic in 3D, so we had to add the points height to y-coordinate of them. In addition, we filtered the points into a circle of radius ; here are the results (we can see the walls of some height).





Here's the code of the matching (we simply added the v(0)+=heights[k]):

// writes points to obj file to be projected in 3D Builder

void writeOBJ(const vector<string>& mvFiles, const vector<string>& heightFiles, const string& outputPath)

{

std::ofstream out;

out.open(outputPath);

if (!out.is\_open())

exit(1);

vector<Eigen::Vector3d> points;

Analyzer a(fx, fy, cx, cy);

for (int i = 0;i < mvFiles.size();i++)

{

// import all data for file

auto motionVectors = a.importMV(mvFiles[i]);

CSVFile height\_file(heightFiles[i], NUM\_FRM);

height\_file.openFile();

auto heights = height\_file.readColumn();

auto centers = a.getCenters();

// dH is a backup for calculating depths

vector<double> dh = heights;

vector<Eigen::Vector3d> tmp;

// continuize the heights function.

a.continuize(dh);

a.differences(dh);

// for each frame in the video

for (int k = 0;k < motionVectors.size();k++)

{

// get the mapped points then add the heights, and add it to the cloud

vector<Eigen::Vector3d> tmp = a.mapPoints(centers, motionVectors[k], dh[k]);

for (auto v : tmp)

{

v(1) += heights[k];

}

for(int i =0;i<tmp.size();i++)

{

if(NORM(tmp[i](0),tmp[i](2))<5000)

points.push\_back(tmp[i]);

}

}

Analyzer::rotatePoints(points, 60\*i);

std::cout << "Processing Angle : " << 60 \* i << std::endl;

}

// write to obj file

int line = 1;

vector<string> faces;

for (auto point : points)

{

string s = "f ";

for (int k = 0;k < 3;k++) {

int A = 1;

out << "v " << point(0) + ((k%3)/2)\*A << " " << point(2) + (((k+1)%3)/2)\*A << " " << point(1) / 10 + A\*((k+2)%3)/2 << std::endl;

s = s + std::to\_string(line++) + " ";

}

faces.push\_back(s+"\n");

}

for (auto f : faces)

{

out << f;

}

}

## Making the code more efficient

We decided that the code is sufficiently efficient in terms of running time (average running time is 0.2 seconds in my computer, and 10 times slower in raspberry-pi, so approximately 2 seconds), and is the most efficient in terms of kernel time (we read each file only once, so we do the minimal number of necessary queries). Therefore, I will be taking this task off the list with no further improvements, for the current time.