Map for Mobile Robots

Qianhao Wang

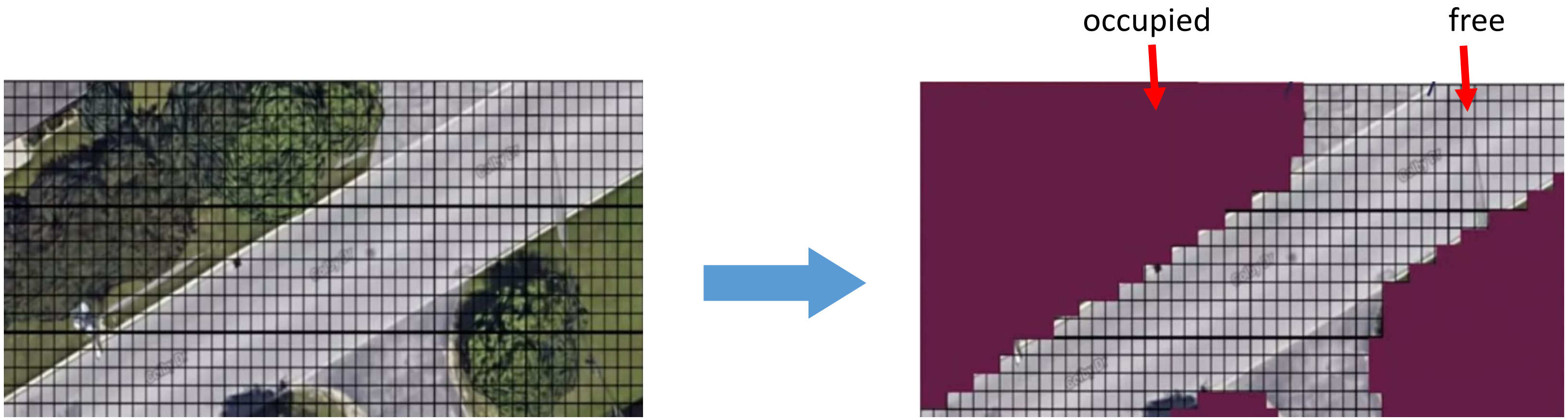
- Occupancy Grid Map
- Euclidean Signed Distance Field

Occupancy Grid Map[1]

[1] https://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture06_agiri_dmcconac_kumarsha_nbhakta.pdf

Gird Map

- In trajectory planning, we need to know which places are obstacles and which places are passable. Grid map is a widely used method.
- By dividing the map into grids, each grid is filled with binary values (0 or 1, 0 means free; 1 means occupied) according to whether it is an obstacle or not.



Occupancy Gird Map

- In practice , the sensor measurement data is noisy and uncertain.
- The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment.

- Based on existing observations of a grid , occupancy grid algorithms compute approximate posterior estimates for these random variables.

Occupancy Gird Map

$$p(m_i = 1) \longrightarrow p(m_i)$$
$$p(m_i = 0) \longrightarrow p(\overline{m_i})$$

◆ Notation Definition

For a grid m_i in occupancy grid map,

- the probability of the grid state being occupied: $p(m_i = 1)$, called $p(m_i)$,
- the probability of the grid state being free : $p(m_i = 0)$, called $p(\overline{m_i})$,

note that $p(m_i) = 1 - p(\overline{m_i})$.

For observations z ,

- the observation that we get at the t-th time is denoted as z_t.

Calculate the posterior probability of the grid state based on existing observations

Calculate $p(m_i|z_{1:t})$ **and** $p(\overline{m_i}|z_{1:t})$

Occupancy Gird Map

◆ Bayesian Filter

Bayes Formula： $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$

- The probability of the grid state being occupied

$$p(m_i|\boxed{z_{1:t}}) = \frac{p(z_t|z_{1:t-1}, m_i)p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

Expansion based on Bayes Formula

$z_1, z_2, z_3, \dots, z_{t-1}, z_t$, can also be written as: $z_{1:t-1}, z_t$

$$p(m_i|z_t) = \frac{p(z_t|m_i)p(m_i)}{p(z_t)}$$

Occupancy Gird Map

◆ Bayesian Filter

Bayes Formula： $p(A|B) = \frac{p(B|A)p(A)}{p(B)}$

- The probability of the grid state being occupied

$$p(m_i|z_{1:t}) = \frac{\boxed{p(z_t|z_{1:t-1}, m_i)}p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

$$= \frac{\boxed{p(z_t|m_i)}p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

$$= \frac{p(m_i|z_t)p(z_t)}{p(m_i)} \frac{p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

We then make the Markov assumption：
$p(z_t|z_{1:t-1}, m_i) = p(z_t|m_i)$

Expansion based on Bayes Formula

Occupancy Gird Map

◆ Bayesian Filter

update rule

- The probability of the grid state being occupied

$$\boxed{p(m_i|z_{1:t})} = \frac{p(m_i|z_t)p(z_t)}{p(m_i)} \frac{\boxed{p(m_i|z_{1:t-1})}}{p(z_t|z_{1:t-1})}$$

- Based on the above, we can obtain the probability of the grid state being free

$$p(\overline{m_i}|z_{1:t}) = \frac{p(\overline{m_i}|z_t)p(z_t)}{p(\overline{m_i})} \frac{p(\overline{m_i}|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

Occupancy Gird Map

◆ Divide Two Update Rules

$$\boxed{\frac{p(m_i|z_{1:t})}{p(\overline{m_i}|z_{1:t})}} = \frac{\dfrac{p(m_i|z_t)p(z_t)}{p(m_i)}\dfrac{p(m_i|z_{1:t-1})}{p(z_t|z_{1:t-1})}}{\dfrac{p(\overline{m_i}|z_t)p(z_t)}{p(\overline{m_i})}\dfrac{p(\overline{m_i}|z_{1:t-1})}{p(z_t|z_{1:t-1})}}$$

$$= \frac{p(m_i|z_t)p(m_i|z_{1:t-1})p(\overline{m_i})}{p(\overline{m_i}|z_t)p(\overline{m_i}|z_{1:t-1})p(m_i)}$$

recursive

$$= \frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)}\frac{p(\overline{m_i})}{p(m_i)}\boxed{\frac{p(m_i|z_{1:t-1})}{p(\overline{m_i}|z_{1:t-1})}}$$

Occupancy Gird Map

- **Use log Function**

$$\boxed{\log\frac{p(m_i|z_{1:t})}{p(\overline{m_i}|z_{1:t})}} = \log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)}\frac{p(\overline{m_i})}{p(m_i)}\frac{p(m_i|z_{1:t-1})}{p(\overline{m_i}|z_{1:t-1})}$$

For simplicity, we write it as $l_t(m_i)$

The grid state prior, we denote it as $l_0(m_i)$

$$l_t(m_i) = \log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)} - \boxed{\log\frac{p(m_i)}{p(\overline{m_i})}} + l_{t-1}(m_i)$$

Occupancy Gird Map

Recursive Update

- The equation the grid m_i 's state updated recursively from observations z are

$$l_t(m_i) = \log \frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)} - l_0(m_i) + l_{t-1}(m_i)$$

where

$p(z_t|m_i)$ is sensor model,

$p(m_i|z_t)$ is inverse sensor model.

Occupancy Gird Map

◆ Inverse Sensor Model

- Inverse sensor model specifies a distribution over the (binary) state variable as a function of the measurement z_t .
- Expansion based on Bayes Formula

$$\begin{cases} p(m_i|z_t) = \dfrac{p(z_t|m_i)p(m_i)}{p(z_t)} \\ p(\overline{m_i}|z_t) = \dfrac{p(z_t|\overline{m_i})p(\overline{m_i})}{p(z_t)} \end{cases}$$

Occupancy Gird Map

$$\begin{cases} p(m_i|z_t) = \dfrac{p(z_t|m_i)p(m_i)}{p(z_t)} \\ p(\overline{m_i}|z_t) = \dfrac{p(z_t|\overline{m_i})p(\overline{m_i})}{p(z_t)} \end{cases}$$

◆ Recursive Update

- Substitute the formula on the previous page into the incremental term of the recursive update formula,
- recursive update formula： $l_t(m_i) = \boxed{\log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)}} - l_0(m_i) + l_{t-1}(m_i)$

incremental term

- We can get

$$\log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)} = \log\frac{\frac{p(z_t|m_i)p(m_i)}{p(z_t)}}{\frac{p(z_t|\overline{m_i})p(\overline{m_i})}{p(z_t)}} = \log\frac{p(z_t|m_i)p(m_i)}{p(z_t|\overline{m_i})p(\overline{m_i})} = \log\frac{p(z_t|m_i)}{p(z_t|\overline{m_i})} + \boxed{\log\frac{p(m_i)}{p(\overline{m_i})}}$$

The grid state prior, we denote as $l_0(m_i)$

Occupancy Gird Map

◆ Recursive Update

- Substitute the formula on the previous page into the incremental term of the recursive update formula,
- recursive update formula： $l_t(m_i) = \log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)} - l_0(m_i) + l_{t-1}(m_i)$
- We can get

$$\log\frac{p(m_i|z_t)}{p(\overline{m_i}|z_t)} = \log\frac{p(z_t|m_i)}{p(z_t|\overline{m_i})} + l_0(m_i)$$

sensor model

- New recursive update formula：

$$l_t(m_i) = \log\frac{\boxed{p(z_t|m_i)}}{p(z_t|\overline{m_i})} + l_{t-1}(m_i)$$

Occupancy Gird Map

Recursive Update

Recall the previous notion definition:

- The probability of the grid state being occupied: $p(m_i = 1)$, called $p(m_i)$
- The probability of the grid state being free : $p(m_i = 0)$, called $p(\overline{m_i})$

- so

$$\frac{p(z_t|m_i)}{p(z_t|\overline{m_i})} = \frac{p(z_t|m_i = 1)}{p(z_t|m_i = 0)}$$

Occupancy Gird Map

Recursive Update

- $\frac{p(z_t|m_i=1)}{p(z_t|m_i=0\ )}$ only has two cases based on the observation.
- $$\begin{cases} \text{the grid is observed as free:} & \log\left(\frac{p(z_t=0|m_i=1)}{p(z_t=0|m_i=0)}\right) \\ \text{the grid is observed as occupancy:} & \log\left(\frac{p(z_t=1|m_i=1)}{p(z_t=1|m_i=0)}\right) \end{cases}$$
- We assume that there will be no change in the sensor model during mapping, so the above two terms are fixed values. After the above derivation, updating the state of a grid only requires simple addition and subtraction according to the following recursive formula

$$l_t(m_i) = \log\frac{p(z_t|m_i)}{p(z_t|\overline{m_i})} + l_{t-1}(m_i)$$

Occupancy Gird Map

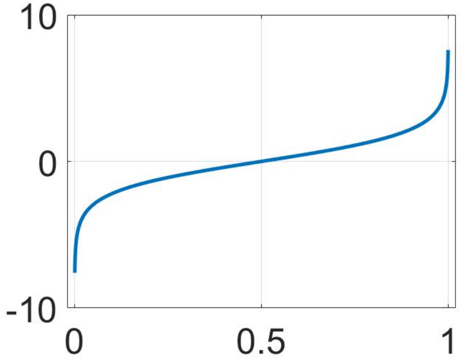
◆ Supplementary Knowledge

- Recall the definition of $l_t(m_i)$

$$l_t(m_i) = \log \frac{p(m_i|z_{1:t})}{p(\overline{m_i}|z_{1:t})} = \log \frac{p(m_i|z_{1:t})}{1 - p(m_i|z_{1:t})}$$

- We analysis the function

$$f(x) = \log \frac{x}{1 - x}$$



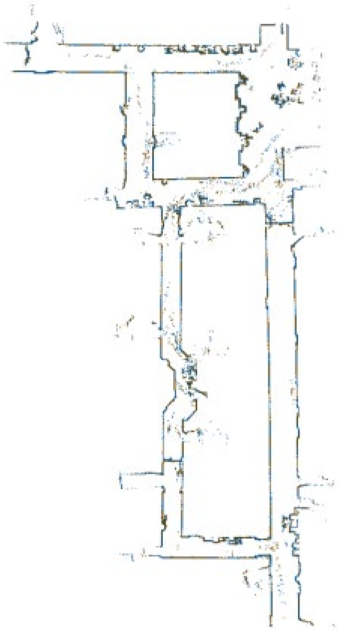
$f(x)$ can map the independent variable from $(0,1)$ monotonically to $(-\infty, +\infty)$ of the range.

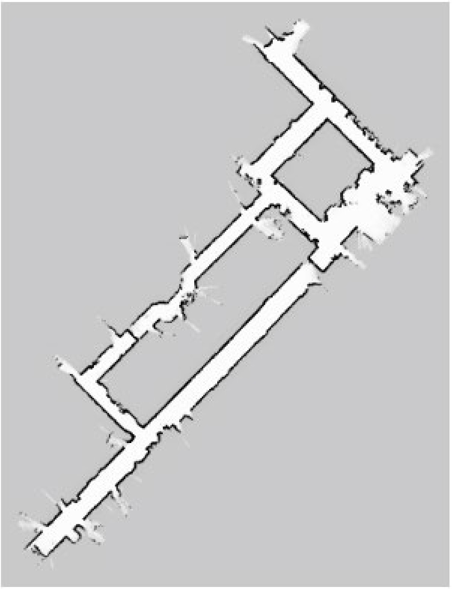
Occupancy Gird Map

- Occupancy grid map divides the map into separate grids.
- Inside each grid is a binary random variable to estimate whether the grid is occupied or free.
- Each grid is a binary Bayesian filter.
- This method can be easily used in the mapping of sensors with known sensor posture but noisy observation data.
- The log function is introduced to update the grid state, which only needs addition and subtraction calculation.

Occupancy Gird Map

- **2D-Lidar**





Map noise is reduced obviously

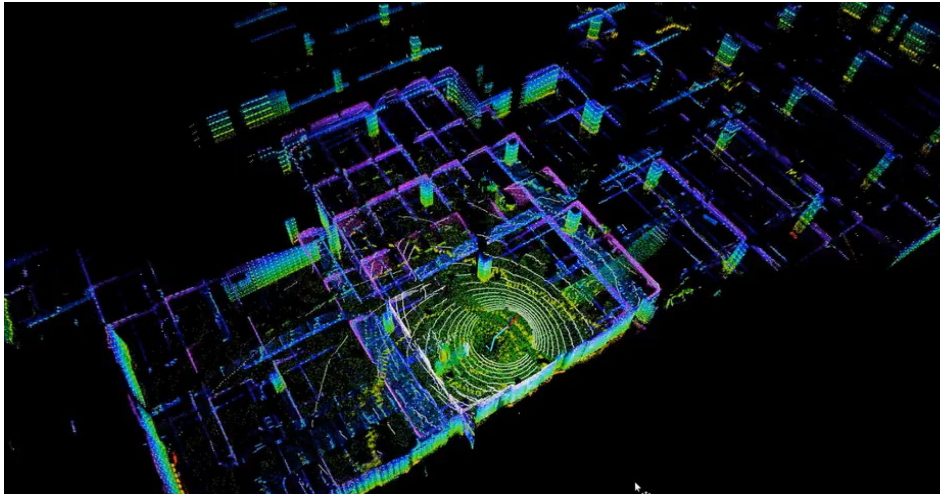
Occupancy Gird Map

The map, which was incorrect due to the noise of the original observation data, was updated to the correct state after multiple observations.

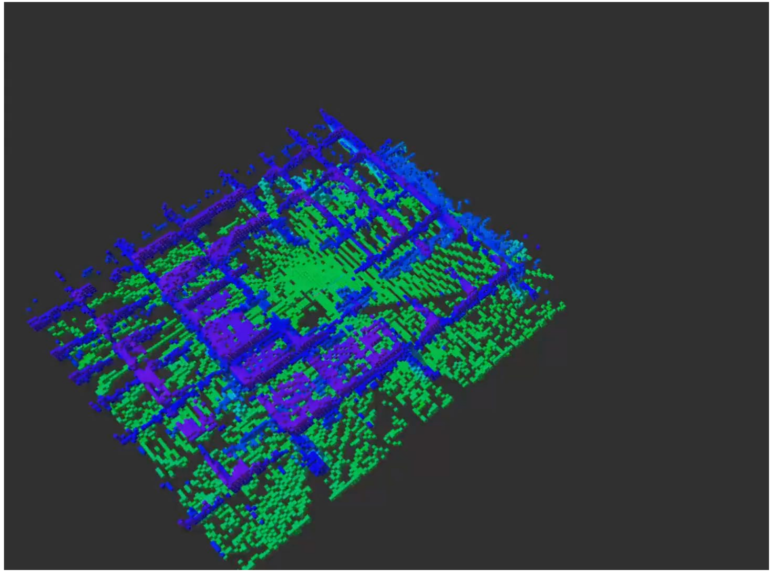
- **Depth Camera**

Occupancy Gird Map

- **3D-Lidar**



3D-Lidar SLAM



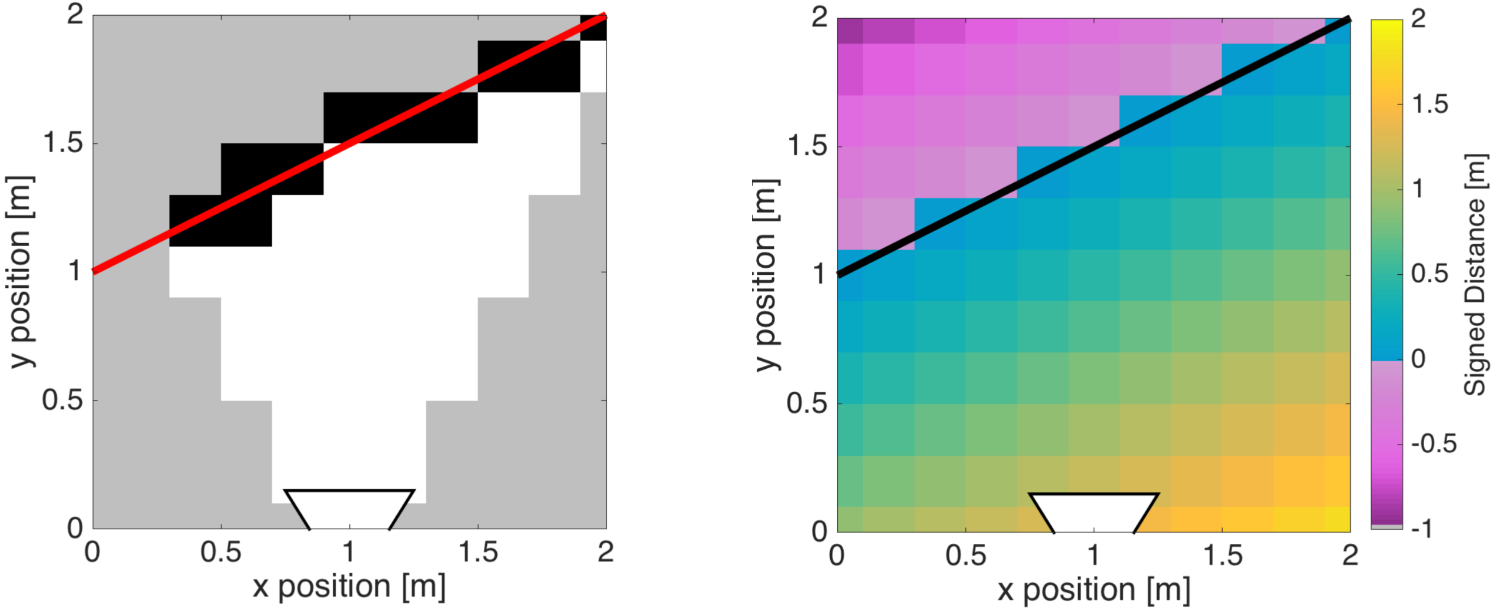
scorll occupancy grid map

Euclidean Signed Distance Field[1]

[1] *Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Distance transforms of sampled functions." *Theory of computing* 8.1 (2012): 415-428.

ESDF

◆ Map Comparison

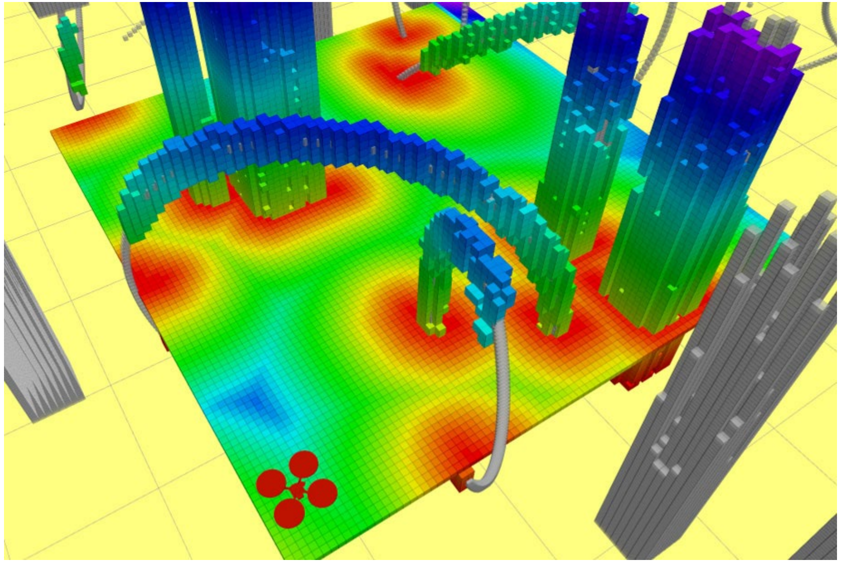


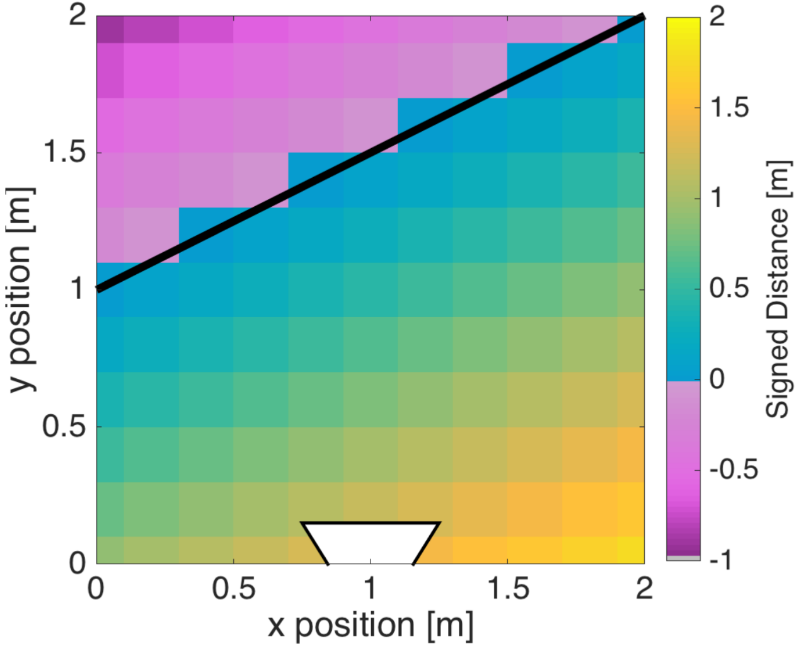
Left: Occupancy representation, where each cell is either labelled as occupied or free.

Right: ESDF, which represents the Euclidean distance to the surface at each cell.

ESDF

- **Euclidean Signed Distance Field is a grid map that stores the distance to the nearest obstacle in each grid.**





ESDF

Classic Frameworks

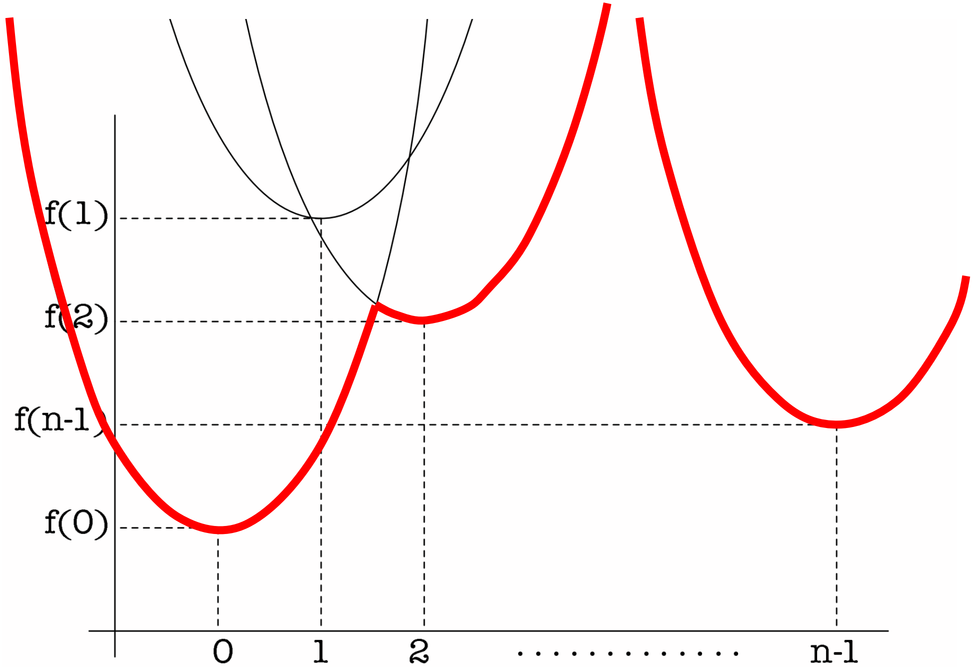
- **ESDF incremental update from grid map** → Improved Updating of Euclidean Distance Maps and Voronoi Diagrams[1]
- **ESDF incremental update from TSDF** → FIESTA[2], Voxblox[3]
- **ESDF generation in batch from grid map** → The Chapter 2 of ***Distance Transforms of Sampled Functions***[4]

[1] *Lau, Boris, Christoph Sprunk, and Wolfram Burgard. "Improved updating of Euclidean distance maps and Voronoi diagrams." 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010.*
[2] *Han, Luxin, et al. "Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots." 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2019.*
[3] *Oleynikova, Helen, et al. "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.*
[4] *Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Distance transforms of sampled functions." *Theory of computing* 8.1 (2012): 415-428.

ESDF

Generation in Batch

The basic idea:



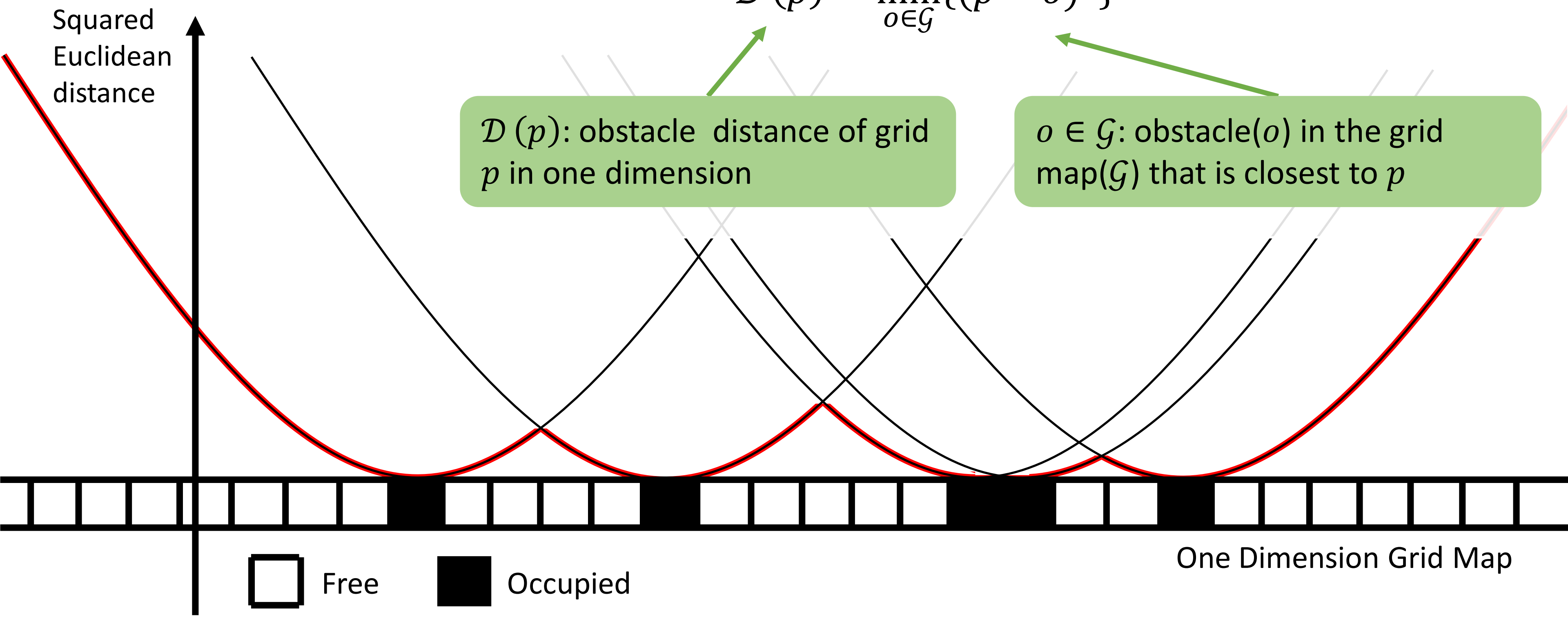
ESDF

- **One Dimension Case**

$$\mathcal{D}(p) = \min_{o \in \mathcal{G}}\{(p-o)^2\}$$

$\mathcal{D}(p)$: obstacle distance of grid p in one dimension

$o \in \mathcal{G}$: obstacle(o) in the grid map($\mathcal{G}$) that is closest to p



ESDF

◆ General Form

$q \in \mathcal{G}$: grid(q) in the grid map($\mathcal{G}$)

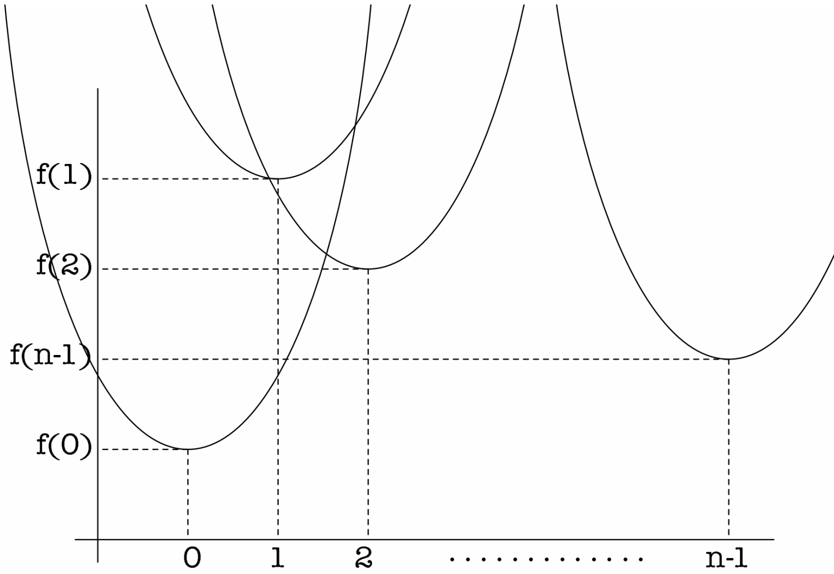
$$\mathcal{D}(p) = \min_{q \in \mathcal{G}} \{(p - q)^2 + f(q)\}$$

$\mathcal{D}(p)$: distance value of grid p

$f(q)$: sampled function of q

- Note that any two parabolas defining the distance transform intersect at exactly one point
- Simple algebra yields the horizontal position of the intersection between the parabola coming from grid position q and the one from p as

$$s = \frac{(p^2 + f(p)) - (q^2 + f(q))}{2(p - q)}$$



ESDF

- **General Form**

Algorithm $DT(f)$

1. $k \leftarrow 0$ (∗ Index of rightmost parabola in lower envelope ∗)
2. $v[0] \leftarrow 0$ (∗ Locations of parabolas in lower envelope ∗)
3. $z[0] \leftarrow -\infty$ (∗ Locations of boundaries between parabolas ∗)
4. $z[1] \leftarrow +\infty$
5. **for** $q = 1$ **to** $n - 1$ (∗ Compute lower envelope ∗)
6. $\quad s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
7. $\quad$ **if** $s \leq z[k]$
8. $\quad\quad$ **then** $k \leftarrow k - 1$
9. $\quad\quad\quad$ **goto** 6
10. $\quad\quad$ **else** $k \leftarrow k + 1$
11. $\quad\quad\quad v[k] \leftarrow q$
12. $\quad\quad\quad z[k] \leftarrow s$
13. $\quad\quad\quad z[k+1] \leftarrow +\infty$
14. $k \leftarrow 0$
15. **for** $q = 0$ **to** $n - 1$ (∗ Fill in values of distance transform ∗)
16. $\quad$ **while** $z[k+1] < q$
17. $\quad\quad k \leftarrow k + 1$
18. $\quad \mathcal{D}_f(q) \leftarrow (q - v[k])^2 + f(v[k])$

Algorithm 1: The distance transform algorithm for the the squared Euclidean distance in one-dimension.

ESDF

Arbitrary Dimensions

- The two dimensional distance transform under the squared Euclidean distance is given by

$$\mathcal{D}_f(x,y) = \min_{x',y'}\{(x-x')^2 + (y-y')^2 + f(x',y')\}$$

- The first term does not depend on y' so we can rewrite this equation as

$$\begin{aligned}\mathcal{D}_f(x,y) &= \min_{x'}\{(x-x')^2\} + \min_{x',y'}\{(y-y')^2 + f(x',y')\} \\ &= \min_{x'}\{\,(x-x')^2 + \mathcal{D}_{f|x'}(x',y)\}\end{aligned}$$

ESDF

$$\mathcal{D}_f(x,y) = \min_{x'}\{(x-x')^2\} + \boxed{\min_{x',y'}\{(y-y')^2 + \boxed{f(x',y')}\}}$$
$$= \min_{x'}\{(x-x')^2 + \boxed{\mathcal{D}_{f|x'}(x',y)}\}$$

0

One Dimension Case

- **Arbitrary Dimensions**

5^2	∞^2	∞^2	∞^2	3^2	3^2	∞^2	1^2	5^2	6^2
4^2	∞^2	∞^2	∞^2	2^2	2^2	∞^2	0^2	4^2	5^2
3^2	∞^2	∞^2	∞^2	1^2	1^2	∞^2	1^2	3^2	4^2
2^2	∞^2	∞^2	∞^2	0^2	0^2	∞^2	2^2	2^2	3^2
1^2	∞^2	∞^2	∞^2	1^2	1^2	∞^2	3^2	1^2	2^2
0^2	∞^2	∞^2	∞^2	2^2	2^2	∞^2	4^2	0^2	1^2
1^2	∞^2	∞^2	∞^2	3^2	3^2	∞^2	5^2	1^2	0^2

ESDF

$$\mathcal{D}_f(x,y) = \min_{x'}\{(x-x')^2\} + \min_{x',y'}\{(y-y')^2 + f(x',y')\}$$
$$= \boxed{\min_{x'}\{(x-x')^2 + \mathcal{D}_{f|x'}(x',y)}\}$$

- **Arbitrary Dimensions**

5^2	∞^2	∞^2	∞^2	3^2	3^2	∞^2	1^2	5^2	6^2
4^2	∞^2	∞^2	∞^2	2^2	2^2	∞^2	0^2	4^2	5^2
3^2	∞^2	∞^2	∞^2	1^2	1^2	∞^2	1^2	3^2	4^2
2^2	∞^2	∞^2	∞^2	0^2	0^2	∞^2	2^2	2^2	3^2
1^2	∞^2	∞^2	∞^2	1^2	1^2	∞^2	3^2	1^2	2^2
0^2	∞^2	∞^2	∞^2	2^2	2^2	∞^2	4^2	0^2	1^2
1^2	∞^2	∞^2	∞^2	3^2	3^2	∞^2	5^2	1^2	0^2

$$\mathcal{D}(x) = \min_{x'}\{(x - x')^2 + f(x')\}$$

One Dimension Case

ESDF

$$\mathcal{D}_f(x,y) = \min_{x'}\{(x-x')^2\} + \min_{x',y'}\{(y-y')^2+f(x',y')\}$$
$$= \boxed{\min_{x'}\{\,(x-x')^2+\mathcal{D}_{f|x'}(x',y)}\}$$

- **Arbitrary Dimensions**

1^2	∞^2	∞^2	∞^2	1^2	1^2	∞^2	3^2	1^2	2^2
0^2	∞^2	∞^2	∞^2	2^2	2^2	∞^2	4^2	0^2	1^2
1^2	∞^2	∞^2	∞^2	3^2	3^2	∞^2	5^2	1^2	0^2

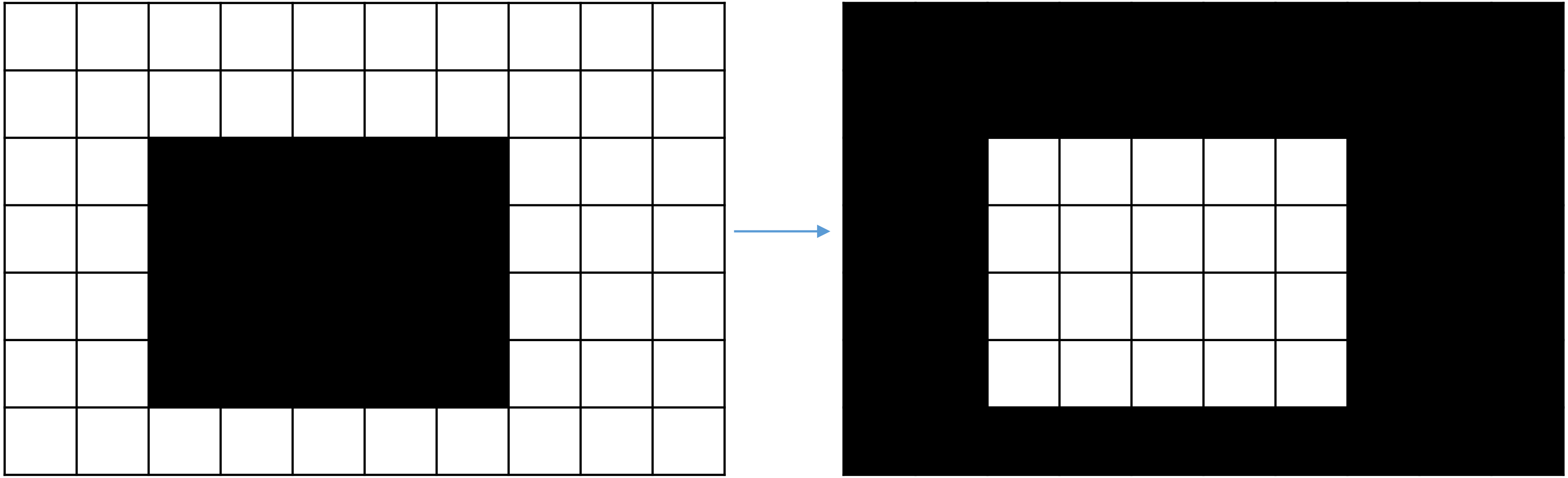
$1^2 + 0^2$	$1^2 + 1^2$	$0^2 + 2^2$	$1^2 + 1^2$	$1^2 + 0^2$	$1^2 + 1^2$	$1^2 + 1^2$	$1^2 + 1^2$	$1^2 + 0^2$	$1^2 + 1^2$
$0^2 + 0^2$	$0^2 + 1^2$	$0^2 + 2^2$	$2^2 + 1^2$	$2^2 + 0^2$	$2^2 + 0^2$	$0^2 + 2^2$	$0^2 + 1^2$	$0^2 + 0^2$	$1^2 + 0^2$
$1^2 + 0^2$	$1^2 + 1^2$	$1^2 + 2^2$	$1^2 + 3^2$	$3^2 + 0^2$	$3^2 + 0^2$	$1^2 + 2^2$	$1^2 + 1$	$1^2 + 0^2$	$0^2 + 0^2$

$$\mathcal{D}(x) = \min_{x'}\{(x-x')^2+f(x')\}$$

One Dimension Case

ESDF

- **Signed**



ESDF

◆ Signed

```cpp
void SDFMap::updateESDF3d() {
  Eigen::Vector3i min_esdf = md_.local_bound_min_;
  Eigen::Vector3i max_esdf = md_.local_bound_max_;

  /* ========== compute positive DT ========== */

  for (int x = min_esdf[0]; x <= max_esdf[0]; x++) {
    for (int y = min_esdf[1]; y <= max_esdf[1]; y++) {
      fillESDF(
          [&](int z) {
            return md_.occupancy_buffer_inflate_[toAddress(x, y, z)] == 1 ?
                0 :
                std::numeric_limits<double>::max();
          },
          [&](int z, double val) { md_.tmp_buffer1_[toAddress(x, y, z)] = val; }, min_esdf[2],
          max_esdf[2], 2);
    }
  }

  for (int x = min_esdf[0]; x <= max_esdf[0]; x++) {
    for (int z = min_esdf[2]; z <= max_esdf[2]; z++) {
      fillESDF([&](int y) { return md_.tmp_buffer1_[toAddress(x, y, z)]; },
               [&](int y, double val) { md_.tmp_buffer2_[toAddress(x, y, z)] = val; }, min_esdf[1],
               max_esdf[1], 1);
    }
  }

  for (int y = min_esdf[1]; y <= max_esdf[1]; y++) {
    for (int z = min_esdf[2]; z <= max_esdf[2]; z++) {
      fillESDF([&](int x) { return md_.tmp_buffer2_[toAddress(x, y, z)]; },
               [&](int x, double val) {
                 md_.distance_buffer_[toAddress(x, y, z)] = mp_.resolution_ * std::sqrt(val);
                 //  min(mp_.resolution_ * std::sqrt(val),
                 //      md_.distance_buffer_[toAddress(x, y, z)]);
               },
               min_esdf[0], max_esdf[0], 0);
    }
  }
```

```cpp
  /* ========== compute negative distance ========== */
  for (int x = min_esdf(0); x <= max_esdf(0); ++x)
    for (int y = min_esdf(1); y <= max_esdf(1); ++y)
      for (int z = min_esdf(2); z <= max_esdf(2); ++z) {

        int idx = toAddress(x, y, z);
        if (md_.occupancy_buffer_inflate_[idx] == 0) {
          md_.occupancy_buffer_neg[idx] = 1;

        } else if (md_.occupancy_buffer_inflate_[idx] == 1) {
          md_.occupancy_buffer_neg[idx] = 0;
        } else {
          ROS_ERROR("what?");
        }
      }

  ros::Time t1, t2;

  for (int x = min_esdf[0]; x <= max_esdf[0]; x++) {
    for (int y = min_esdf[1]; y <= max_esdf[1]; y++) {
      fillESDF(
          [&](int z) {
            return md_.occupancy_buffer_neg[x * mp_.map_voxel_num_(1) * mp_.map_voxel_num_(2) +
                                            y * mp_.map_voxel_num_(2) + z] == 1 ?
                0 :
                std::numeric_limits<double>::max();
          },
          [&](int z, double val) { md_.tmp_buffer1_[toAddress(x, y, z)] = val; }, min_esdf[2],
          max_esdf[2], 2);
    }
  }

  for (int x = min_esdf[0]; x <= max_esdf[0]; x++) {
    for (int z = min_esdf[2]; z <= max_esdf[2]; z++) {
      fillESDF([&](int y) { return md_.tmp_buffer1_[toAddress(x, y, z)]; },
               [&](int y, double val) { md_.tmp_buffer2_[toAddress(x, y, z)] = val; }, min_esdf[1],
               max_esdf[1], 1);
    }
  }

  for (int y = min_esdf[1]; y <= max_esdf[1]; y++) {
    for (int z = min_esdf[2]; z <= max_esdf[2]; z++) {
      fillESDF([&](int x) { return md_.tmp_buffer2_[toAddress(x, y, z)]; },
               [&](int x, double val) {
                 md_.distance_buffer_neg_[toAddress(x, y, z)] = mp_.resolution_ * std::sqrt(val);
               },
               min_esdf[0], max_esdf[0], 0);
    }
  }
```

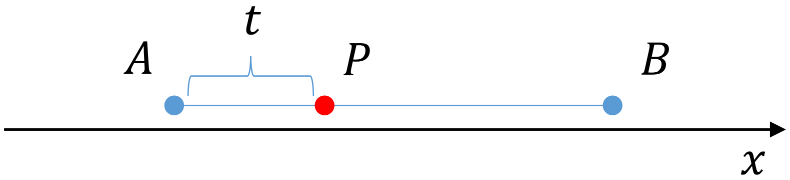
ESDF

- **Signed**

```cpp
/* ========== combine pos and neg DT ========== */
for (int x = min_esdf(0); x <= max_esdf(0); ++x)
  for (int y = min_esdf(1); y <= max_esdf(1); ++y)
    for (int z = min_esdf(2); z <= max_esdf(2); ++z) {

      int idx = toAddress(x, y, z);
      md_.distance_buffer_all_[idx] = md_.distance_buffer_[idx];

      if (md_.distance_buffer_neg_[idx] > 0.0)
        md_.distance_buffer_all_[idx] += (-md_.distance_buffer_neg_[idx] + mp_.resolution_);
    }
}

```

ESDF

Linear Interpolation

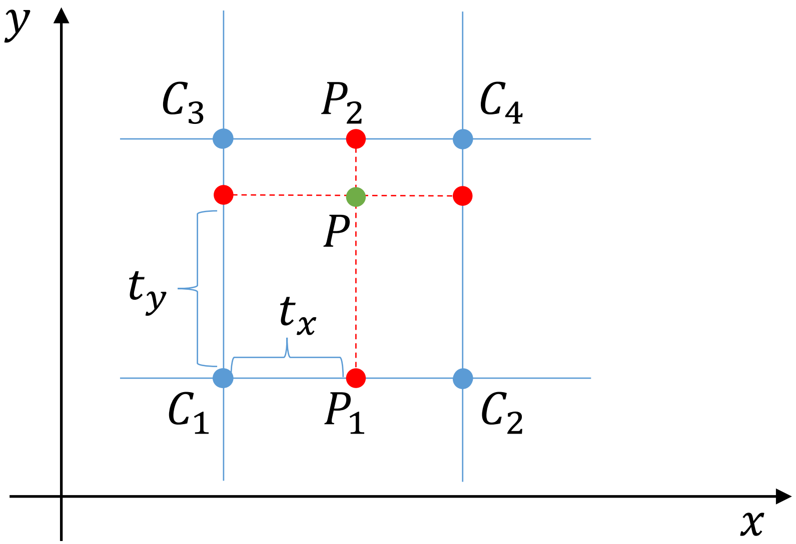


$$\|AB\|_2 = 1 \text{ and } t \in [0,1]$$

$$P = A + (B - A)t = A(1 - t) + B$$

Denote as $L(A, B, t)$

Bilinear Interpolation



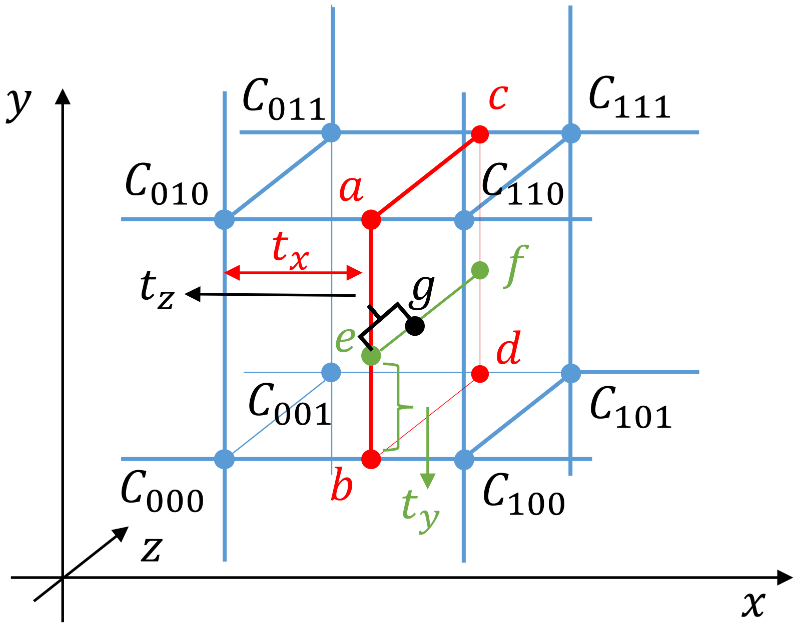
$$P_1 = L(C_1, C_2, t_x)$$

$$P_2 = L(C_3, C_4, t_x)$$

$$P \;\; = L(P_1, P_2, t_y)$$

ESDF

◆ **Trilinear Interpolation**



$$a = L(C_{010}, C_{110}, t_x)$$

$$b = L(C_{000}, C_{100}, t_x)$$

$$c = L(C_{011}, C_{111}, t_x)$$

$$d = L(C_{001}, C_{101}, t_x)$$

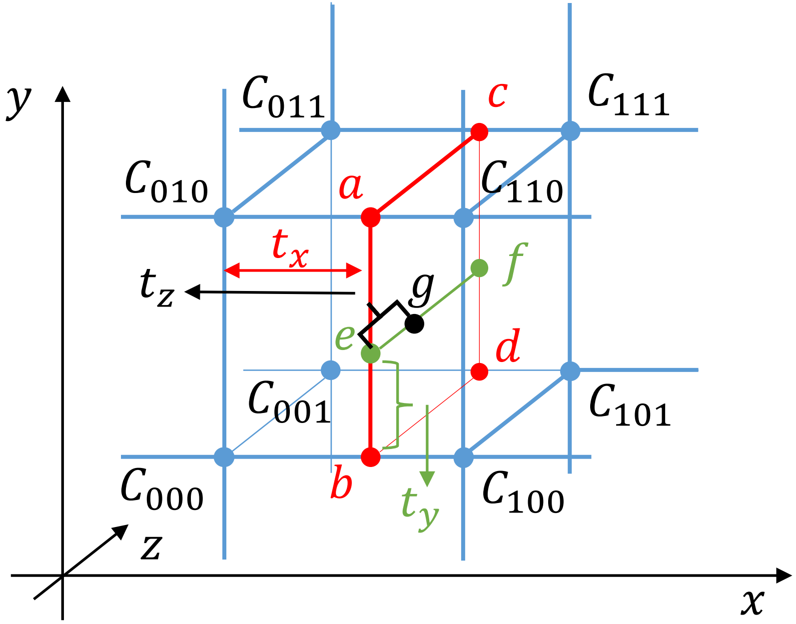
$$e = L(b, a, t_y)$$

$$f = L(d, c, t_y)$$

$$g = L(e, f, t_z)$$

ESDF

- **Gradient**



$$grad_z = \frac{f-e}{\Delta}$$

$$grad_y(e) = \frac{a-b}{\Delta}, grad_y(f) = \frac{c-d}{\Delta}$$

$$grad_y(g) = L\big(grad_y(e), grad_y(f), t_z\big)$$

$$grad_x(a) = \frac{C_{110}-C_{010}}{\Delta}, grad_x(b) = \frac{C_{100}-C_{000}}{\Delta}$$

$$grad_x(c) = \frac{C_{111}-C_{011}}{\Delta}, grad_x(d) = \frac{C_{101}-C_{001}}{\Delta}$$

$$grad_x(e) = L\big(grad_x(b), grad_x(a), t_y\big)$$

$$grad_x(f) = L\big(grad_x(d), grad_x(c), t_y\big)$$

$$grad_x(g) = L\big(grad_x(e), grad_x(f), t_z\big)$$

Thanks for Listening!