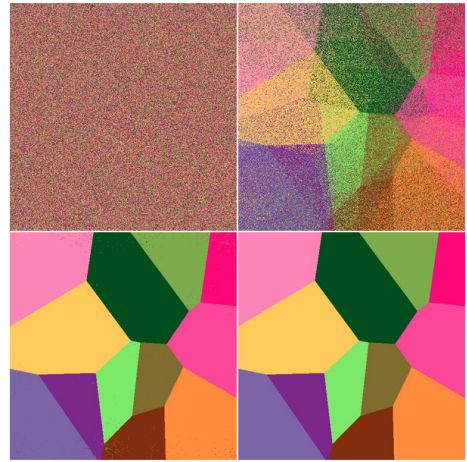
GPU-Accelerated Jump Flooding Algorithm for Voronoi Diagram in $log \star (n)$

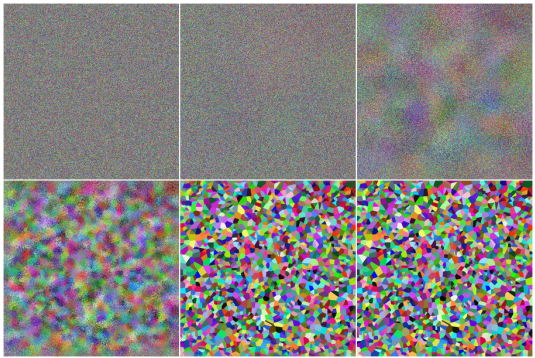
JFA$\star$, OpenCL

Maciej A. Czyzewski

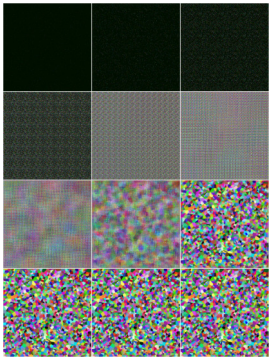
May 27, 2019



Our $log \star (n)$ vs. Current $log(p)$ (where n - seeds in diagram, p - resolution in pixels)

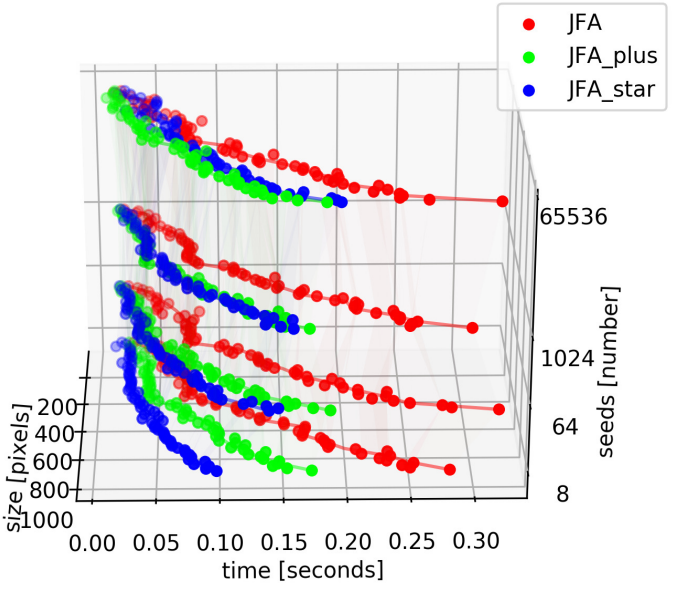


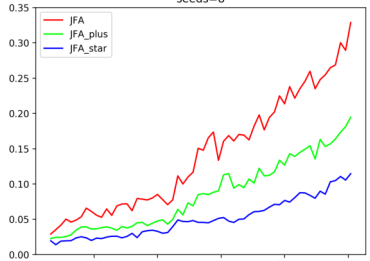
(a) JFA$\star$+1 (4 steps)

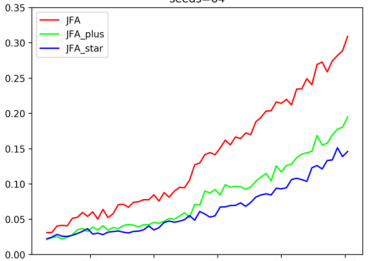


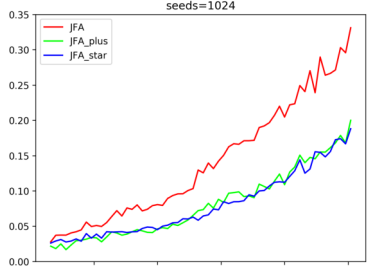
(b) JFA+1 (10 steps)

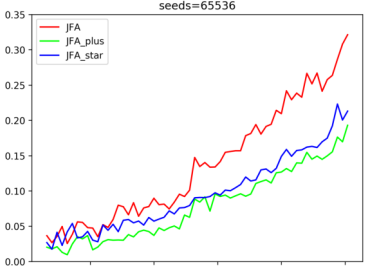
Figure: x $= 720$; y $= 720$; seeds $= 2000$ (read as n $= 2000$; p $= 720$).


JFA
JFA_plus
JFA_star
65536
1024
64
8
seeds [number]
200
400
600
800
1000
size [pixels]
0.00
0.05
0.10
0.15
0.20
0.25
0.30
time [seconds]


seeds=8
JFA
JFA_plus
JFA_star
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
200
400
600
800
1000


seeds=64
JFA
JFA_plus
JFA_star
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
200
400
600
800
1000

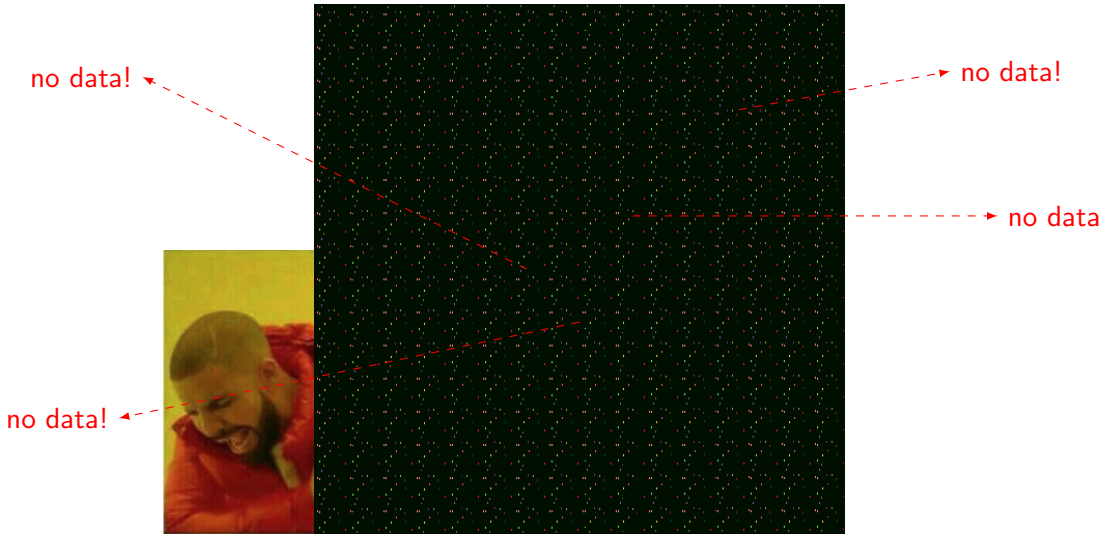

seeds=1024
JFA
JFA_plus
JFA_star
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
200
400
600
800
1000


seeds=65536
JFA
JFA_plus
JFA_star
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
200
400
600
800
1000

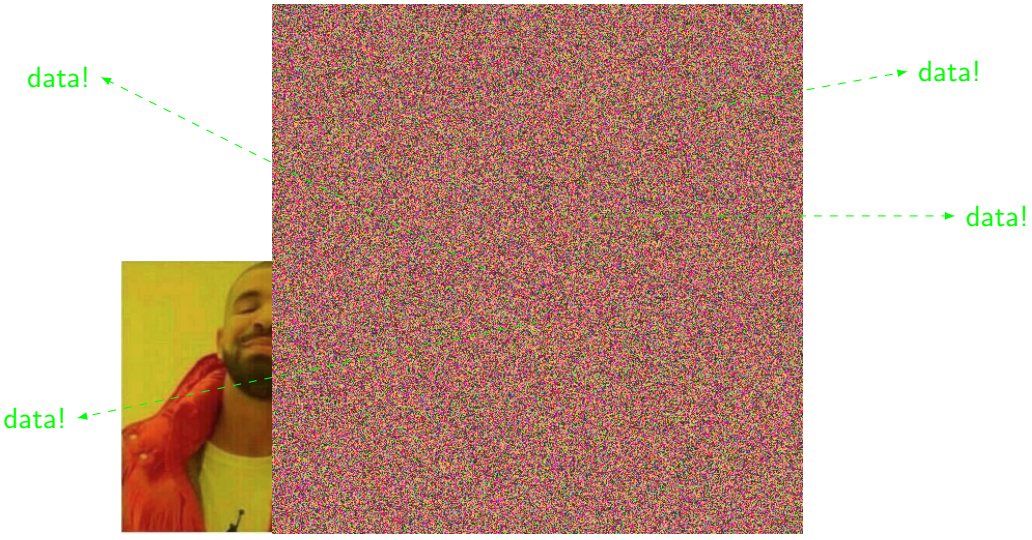
JFA: classic approach on GPU

```
int gid = get_global_id(0);
int y =  gid    % Y_size;
int x = (gid-y) / X_size;
#define POS(X, Y) ((X)*X_size + (Y))

int best_0  = M_g[ gid], // input
    best_1a = P1_g[gid],
    best_1b = P2_g[gid];
float bestS = metric(best_1a, best_1b,
                          x,       y);
if (best_0 == 0)
  bestS = 4294967296; // +inf

int pos[] = {-step, 0, step};

// SELECTION ////////////////////////
for(int i = 0; i < 3; i++) // problem #2
for(int j = 0; j < 3; j++) {
  int idx = POS(x+pos[i], y+pos[j]);
  if (P1_g[idx] == 0 && \ // problem #1
      P2_g[idx] == 0) continue;
  float s2 = metric(
    P1_g[idx], P2_g[idx], x, y);
  if (bestS >= s2) {
    best_0  = M_g[ idx];
    best_1a = P1_g[idx];
    best_1b = P2_g[idx];
    bestS   = s2;
  }}
//////////////////////////////////////

M_o[ gid] =  best_0; // output
P1_o[gid] = best_1a;
P2_o[gid] = best_1b;
```

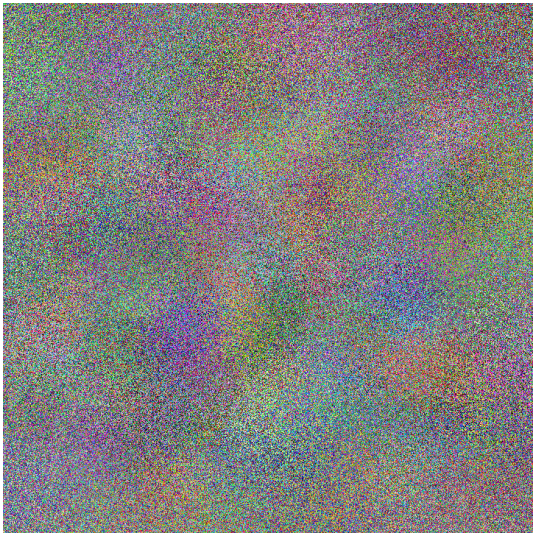
Problem #1: wasting time in empty areas



Solution #1: apply random noise



Solution #1: apply random noise (why?)



1. In areas where the classic JFA does not perform any calculations, JFA$\star$ performs **passive grouping** (noise reduction), often on the correct seeds, therefore small corrections are needed in the next steps
2. In this way, the same algorithm behaves like as if performing weighted quick-union with path compression. Therefore, in a smaller number of steps, the result is obtained

Solution #1: noise in-place (JFA+)

```
// SELECTION ////////////////////////
for(int i = 0; i < 3; i++)
for(int j = 0; j < 3; j++) {
  int idx = POS(x+pos[i], y+pos[j]);
  int m  = M_g[ idx], // current
      p1 = P1_g[idx],
      p2 = P2_g[idx];
  // if no information, get from noise
  if (p1 == 0 && p2 == 0) {
    int ridx = NOISE_g[idx];
    m  = IDS_g[ridx];
    p1 = PTS_g[ridx].x;
    p2 = PTS_g[ridx].y;
  }
  float s2 = metric(p1, p2, x, y);
  if (bestS >= s2) {
    best_0  = m;
    best_1a = p1;
    best_1b = p2;
    bestS   = s2;
  }}
/////////////////////////////////////
```

Solution #1: applying mask with noise (JFA⋆)

```
int gid = get_global_id(0);
int m  = M_g[ gid],
    p1 = P1_g[gid],
    p2 = P2_g[gid];

// if no information,
//     get from noise
if (p1 == 0 && p2 == 0) {
  int ridx = NOISE_g[gid];
  m  = IDS_g[ridx];
  p1 = PTS_g[ridx].x;
  p2 = PTS_g[ridx].y;
}


M_o[ gid] =  m; // output
P1_o[gid] = p1;
P2_o[gid] = p2;
```

Problem #2: selection is too regular

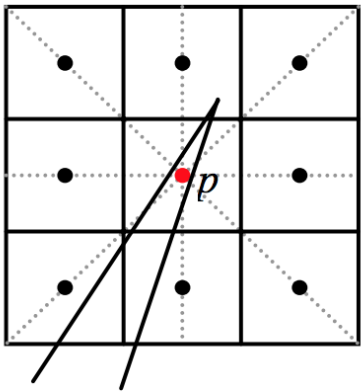


Figure: A Voronoi cell shown partially as a wedge can "steal" a center pixel p without including any of the eight neighboring pixels. Such a Voronoi cell looks disconnected when displayed on screen. [Guodong 2006]

Solution #2: random points in circle

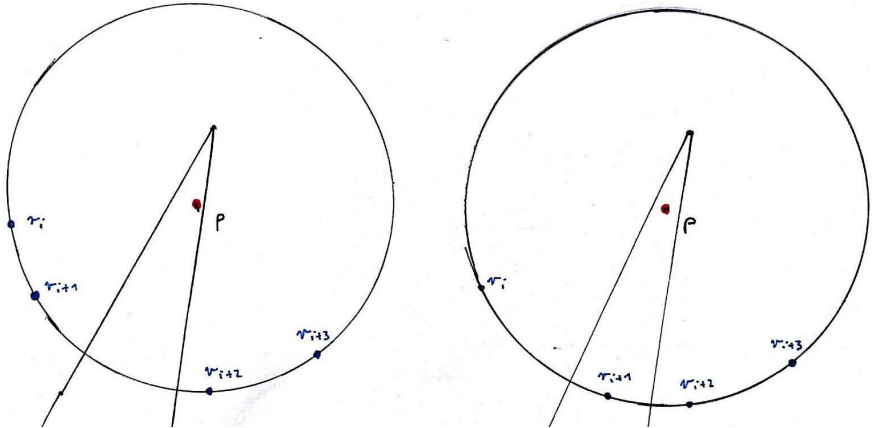
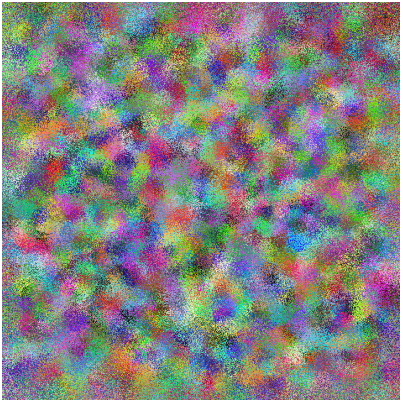
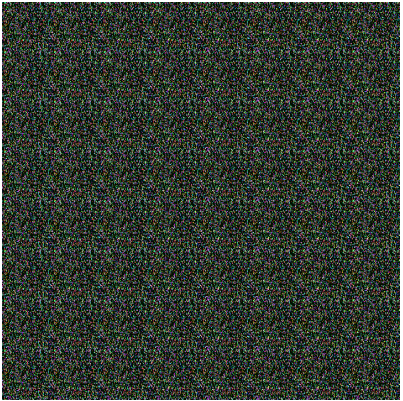


Figure: The solution to this problem is to select points on a circle during each step. In this figure, it can be seen that thanks to that we can choose the area of interest (i.e. r_{i+i}).

Solution #2: the difference at the same step



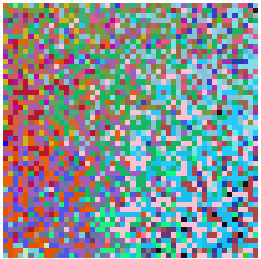
(a) JFA⋆+1 (circle: random 12 points)



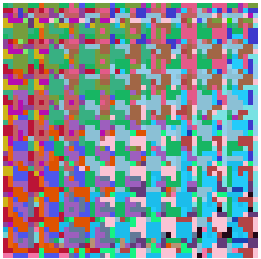
(b) JFA+1 (grid: regular 9 points)

Figure: Please note that Voronoi cells are already clearly defined in our method.

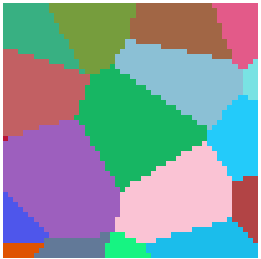
Solution #2: difference in entropy



(a) JFA⋆+1



(b) JFA+1



(c) Voronoi

Figure: The difference presented above in for these methods.

Question #1: how many steps do we need now?

I DON'T NOW!

no mathematical proof

Question #1: empirical observations

	JFA$\star$	JFA+	JFA
used improvement	noise+selection	noise	–
num. of needed steps	log$\star$(n)	log4(p)	log2(p)
step size	$\frac{p}{3^i}$	$\frac{p}{2^i}$	$\frac{p}{2^i}$

Recommended reading

1. “Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform”, Guodong Rong, Tiow-Seng Tan, 2006
2. “Facet-JFA: Faster computation of discrete Voronoi diagrams”, Talha Bin Masoodi, Hari Krishna Malladi, Vijay Natarajan, 2014