Posztert a szobámba!

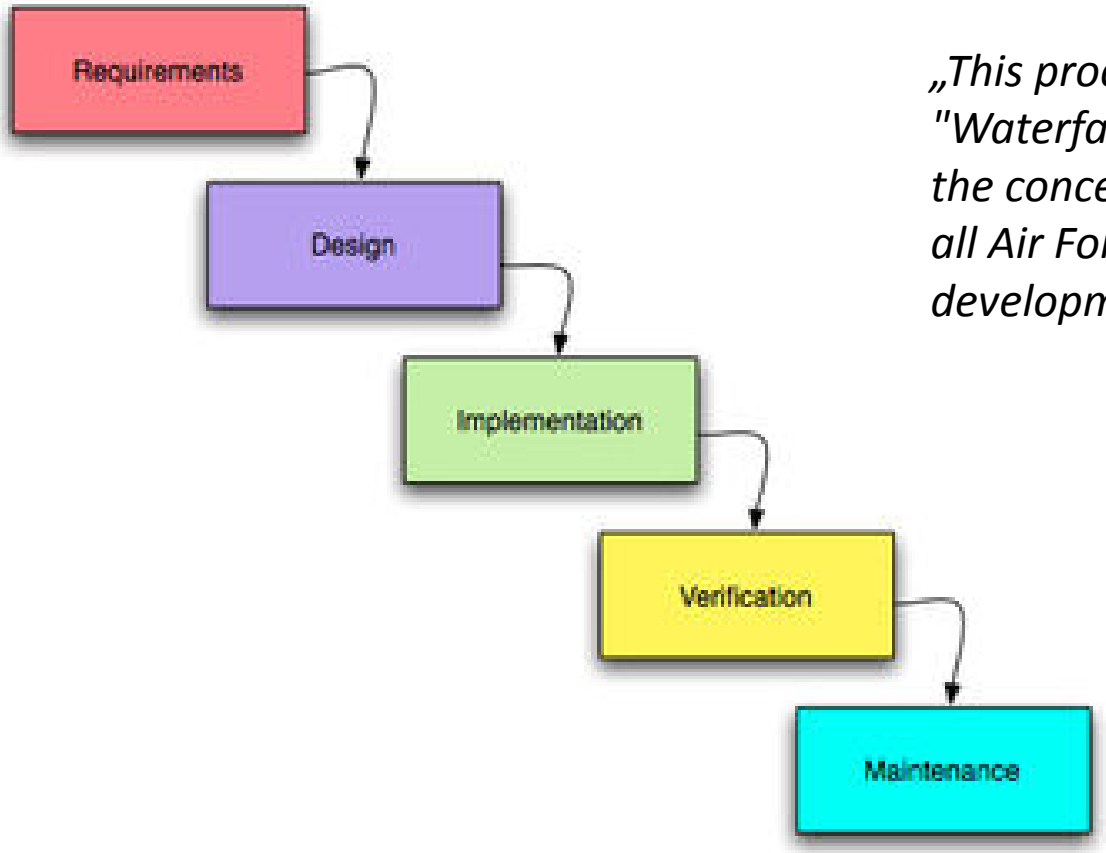
Speciális fejezetek informatikából

PROJEKT

2016-17 / II

Szoftverfejlesztés miniben

- Vízesés modell (Royce, 1970)



„This process is represented by the "Waterfall" Model, which serves as the conceptual guideline for almost all Air Force and NASA software development.” (www1.jsc.nasa.gov)

A **vízesés modell** néhány előnye/hátránya

- A jól érthető és kevésbé komplex projektek esetén szabályozottan rögzíti a komplexitást, és jól megbirkózik azzal;
- Strukturáltságot biztosít még a kevésbé képzett fejlesztők számára is;
- A projekt menedzser számára könnyű a tervezés és a szereplők kiválasztása;
- Nincs visszacsatolás, iteráció.

1 Projekt alapítás: célok kitűzése és hitelesítése

- Cél kitűzése
  - Poszter-kollekció rangsorolása adott szoba-képhez, színösszetétel alapján
- Kockázat elemzés
- Megtérülés vizsgálat
- Humán és eszköz erőforrások projekthez rendelése
  - Projektvezető: KZ
  - Tanácsadó: ID
  - Fejlesztő csapat:
    - „Infósok” / „Villamosok” / „Gépészek” 3/4-es csoportokba rendezve
- Ütemterv elkészülés
  - n x (1 előadás + 2 szeminárium)

2 Szoftver specifikáció

- Igényspecifikáció
  - A felhasználó megadja a szobakép, illetve a poszter kollekciót tartalmazó mappa elérési útját, a szoftver pedig kilistázza a rangsort
    - A képek .ppm kiterjesztésűek
- Funkcionális specifikáció
  - 1. modul: adott mappa ppm-képeinek listázása
  - 2. modul: adott kép szín-számának csökkentése
  - 3. modul: a poszterek rangsorolása színtávolság alapján

2 Szoftver specifikáció

- Rendszerterv
  - 1. modul: adott mappa ppm-képeinek listázása
    - egy mappában / hierarchikus al-mappa rendszerben
  - 2. modul: adott kép szín-számának csökkentése
    - adott kép RGB-kockájának létrehozása
    - adott kép RGB-spectrumának létrehozása
    - adott szín-spektrum redukálása
      - k-mean algoritmus, stb
    - adott kép színredukált változatának kreálása
  - 3. modul: a poszterek rangsorolása
    - két kép színtávolságának meghatározása különböző metrikák révén

3 A szoftver fejlesztése

- 1. hét (2 óra)
  - void list_all_entries (const char *rootDirName);
    - Kilistázza a rootDirName mappa tartalmát
    - Extra: hierarchikus mappaszerkezet
- #include <dirent.h>
  - DIR *dp; // akárcsak FILE *fp;
    - opendir / closedir
    - struct dirent * readdir(DIR *)
      - struct dirent
        - » mezők: d_name, d_ino

3

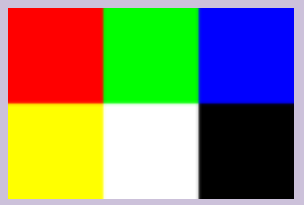
```
void list_all_entries(const char *entryname, int n){
    DIR *dp;
    struct dirent *ep;
    dp = opendir(entryname);
    if (dp != NULL){
        while (ep = readdir(dp)){
            if(strcmp(ep->d_name,".") && strcmp(ep->d_name,"..")){
                char id[101];
                strcpy(id, entryname);
                strcat(id, ep->d_name);
                strcat(id, "/");
                int i; for(i = 0 ; i < n ; ++i) { printf("\t"); }
                printf ("%s\n", ep->d_name);
                list_all_entries(id, n+1);
            }
        }
        closedir(dp);
    }
}
```

3 A szoftver fejlesztése (2 óra)

- PPM (**p**ortable **p**ix**m**ap format) formátum (en.wikipedia.org/wiki/Netpbm_format)
  - File format description
    - Each file starts with a two-byte magic number (in ASCII) that identifies the type of file it is (PBM, PGM, and PPM) and its encoding (ASCII or binary)

Type	Magic number		Magic number		Extension	Colors
Portable BitMap	P1	ASCII	P4	binary	.pbm	0–1 (black & white)
Portable GrayMap	P2	ASCII	P5	binary	.pgm	0–255 (gray scale)
Portable PixMap	P3	ASCII	P6	binary	.ppm	0–255 (RGB)

```
P3
# The P3 means colors are in ASCII, then 3 columns and 2 rows,
# then 255 for max color, then RGB triplets
3 2
255
255 0 0 0 255 0 0 0 255
255 255 0 255 255 255 0 0 0
```



3 A szoftver fejlesztése (2 óra)

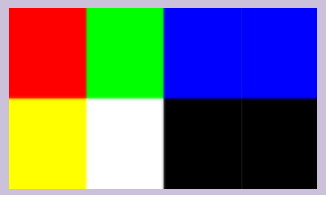
- PPM (**p**ortable **p**ix**m**ap format) formátum (en.wikipedia.org/wiki/Netpbm_format)
  - File format description
    - Each file starts with a two-byte magic number (in ASCII) that identifies the type of file it is (PBM, PGM, and PPM) and its encoding (ASCII or binary)

Type	Magic number		Magic number		Extension	Colors
Portable BitMap	P1	ASCII	P4	binary	.pbm	0–1 (black & white)
Portable GrayMap	P2	ASCII	P5	binary	.pgm	0–255 (gray scale)
Portable PixMap	P3	ASCII	P6	binary	.ppm	0–255 (RGB)

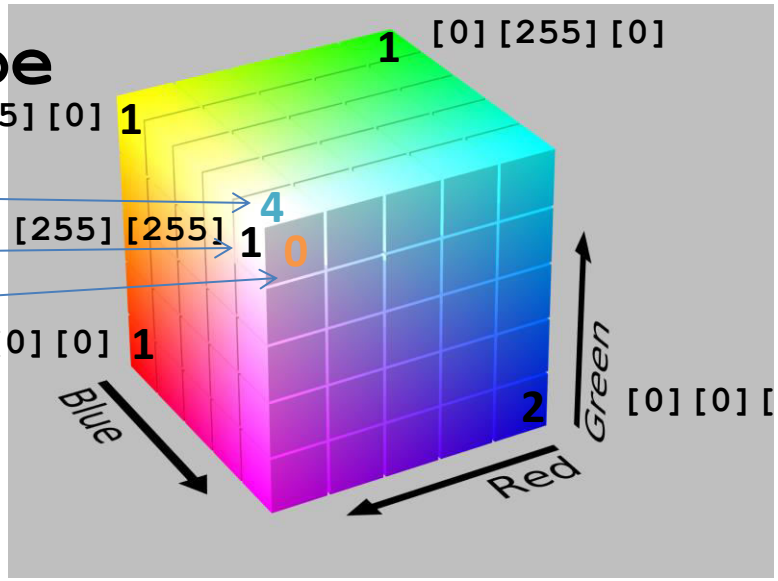
fscanf

fread

```
P6
#The P6 means colors are in binary, then 4(x) columns and 2(y) rows,
#then 255(maxc) for max color, then RGB triplets
4 2
255
FF0000 00FF00 0000FF FFFF00 FFFFFF 000000000000
```



3



picture

	0(r)	1(g)	2(b)
0	FF	00	00
1	00	FF	00
2	00	00	FF
3	00	00	FF
4	FF	FF	00
5	FF	FF	FF
6	00	00	00
7	00	00	00
...			

spectrum

	0(r)	1(g)	2(b)
0	FF	00	00
1	00	FF	00
2	00	00	FF
3	FF	FF	00
4	FF	FF	FF
5	00	00	00
...			

centroids

	0(r)	1(g)	2(b)
0	?	?	?
1	?	?	?
...			

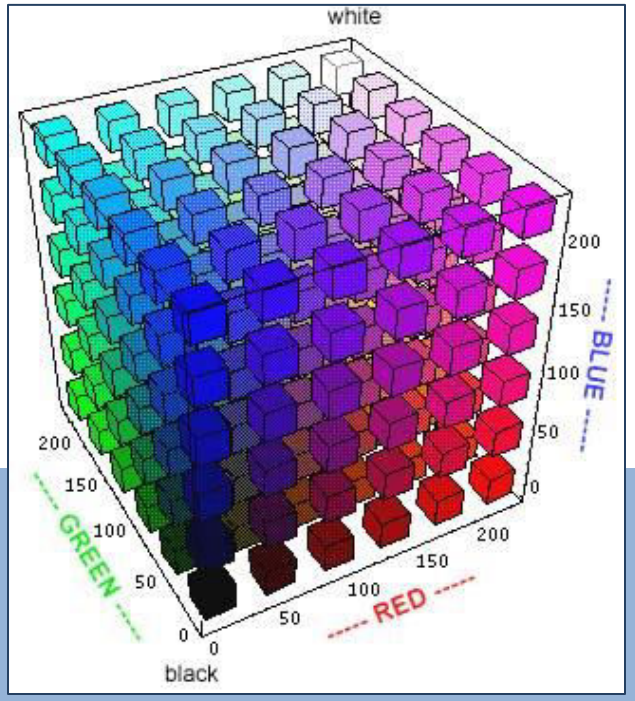
3

```
typedef struct{
    int spect_index, nr_pixels, centr_label;
}COLOR;
typedef struct{
   int x,y,maxc,nr_colors,k;
   unsigned char (*picture)[3]; //x*y rows
   unsigned char (*spectrum)[3]; //nr_colors rows
   unsigned char (*centroids)[3]; //k rows
   COLOR cube[256][256][256];
}PICTURE;
int main(){
   PICTURE *p = (PICTURE *)calloc(1,sizeof(PICTURE)); p->k = 64;
   ReadOldPicture("poster.ppm",p);
   CreateColorStatistics(p);
   CreateSpectrum(p);
   CreateCentroids(p);
   CreateReducedImage("newimage.ppm",p);
}
```

3

```
void ReadOldPicture(const char* fname, PICTURE *p){
   FILE *fin = fopen(fname, "rb");
   char line[101];
   fscanf(fin, "%[^\n]\n", line); // P6

   // skip comment line
   while( fscanf(fin, "%[^\n]\n", line), line[0] == '#');

   sscanf(line, "%i %i\n", &p->x, &p->y);
   fscanf(fin, "%i\n", &p->maxc);

   p->picture = (unsigned char (*)[3])
                malloc(p->x * p->y * 3 * sizeof(unsigned char));

   fread(p->picture, sizeof(unsigned char[3]), p->x*p->y, fin);
   // create and display the inverted picture; c = 255 - c
   fclose(fin);
}
```

3




```
void CreateColorStatistics(PICTURE *p){
    int i;
    for(i = 0 ; i < p->x * p->y ; ++i){

        ++(p->cube[ p->picture[i][0] ]
                  [ p->picture[i][1] ]
                  [ p->picture[i][2] ].nr_pixels);

        if( p->cube[ p->picture[i][0] ]
                    [ p->picture[i][1] ]
                    [ p->picture[i][2] ].nr_pixels == 1 ){
            ++p->nr_colors;
        }
    }
}
```

3

```
void CreateSpectrum(PICTURE *p){
    int r,g,b, ind = 0;
    p->spectrum = (unsigned char (*)[3])
                malloc(p->nr_colors * 3 * sizeof(unsigned char));

   for(r = 0 ; r < 256 ; ++r){
        for(g = 0 ; g < 256 ; ++g){
            for(b = 0 ; b < 256 ; ++b){
                if (p->cube[r][g][b].nr_pixels){
                    p->spectrum[ind][0] = r;
                    p->spectrum[ind][1] = g;
                    p->spectrum[ind][2] = b;
                    p->cube[r][g][b].spect_index = ind;
                    ++ind;
                }
            }
        }
    }
}
```

3

Color Quantization (1)

- Common color resolution for high quality images is 256 levels for each Red, Greed, Blue channels, or $256^3 = 16777216$ colors.
- How can an image be displayed with fewer colors than it contains?
- Select a subset of colors (the colormap or pallet) and map the rest of the colors to them.

3 Color Quantization (2)

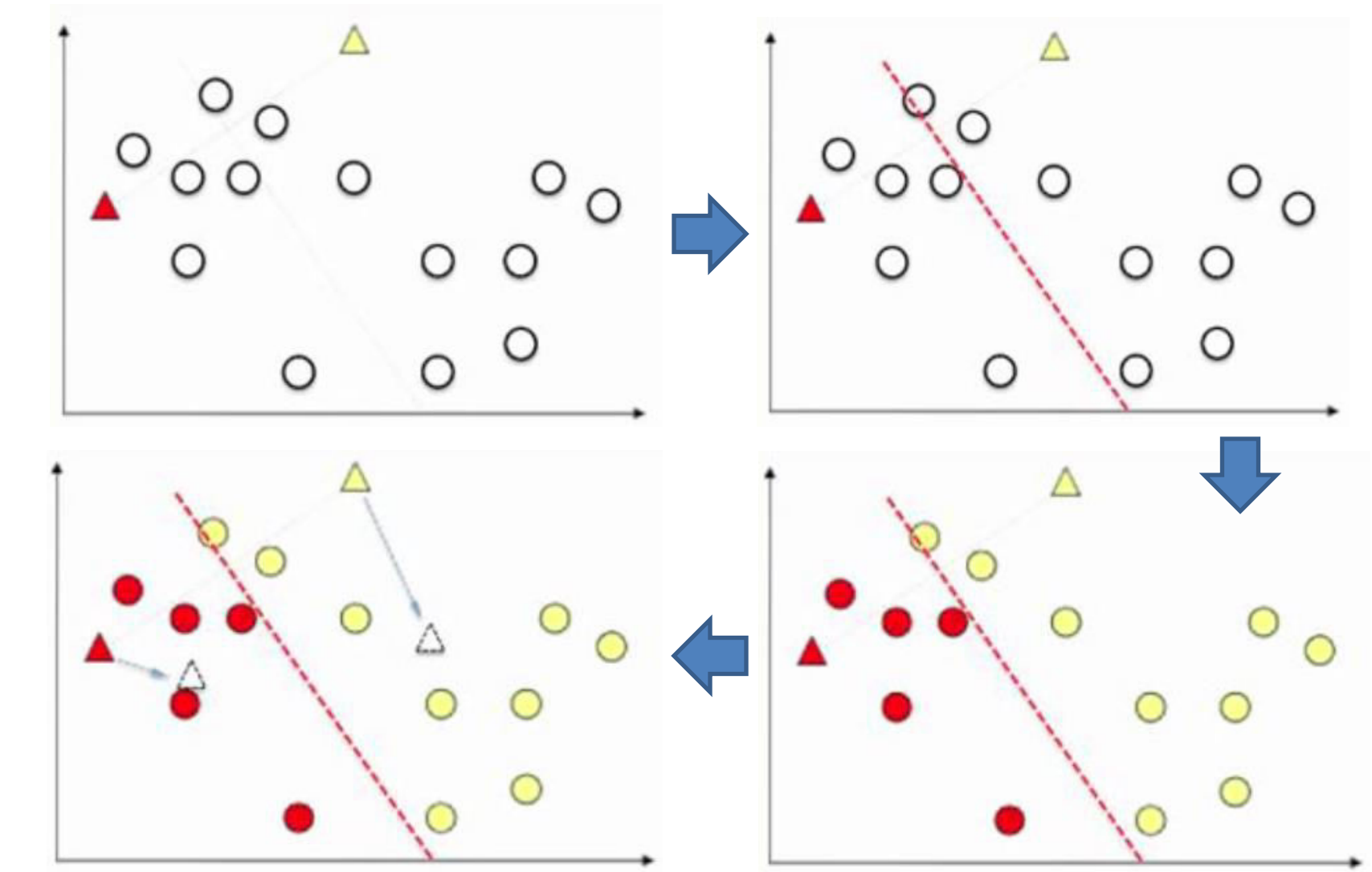
2 colors

16 colors

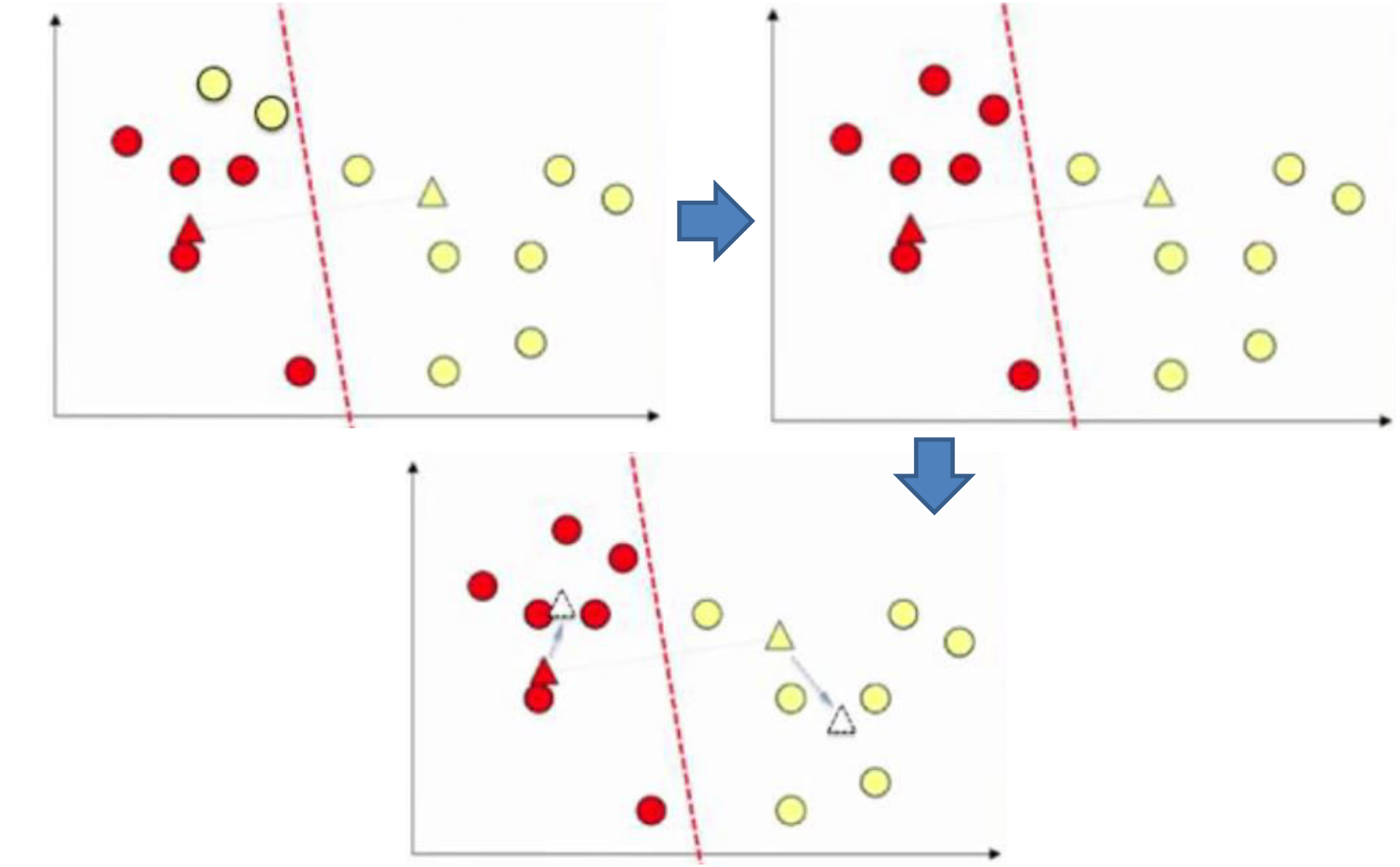
4 colors

256 colors

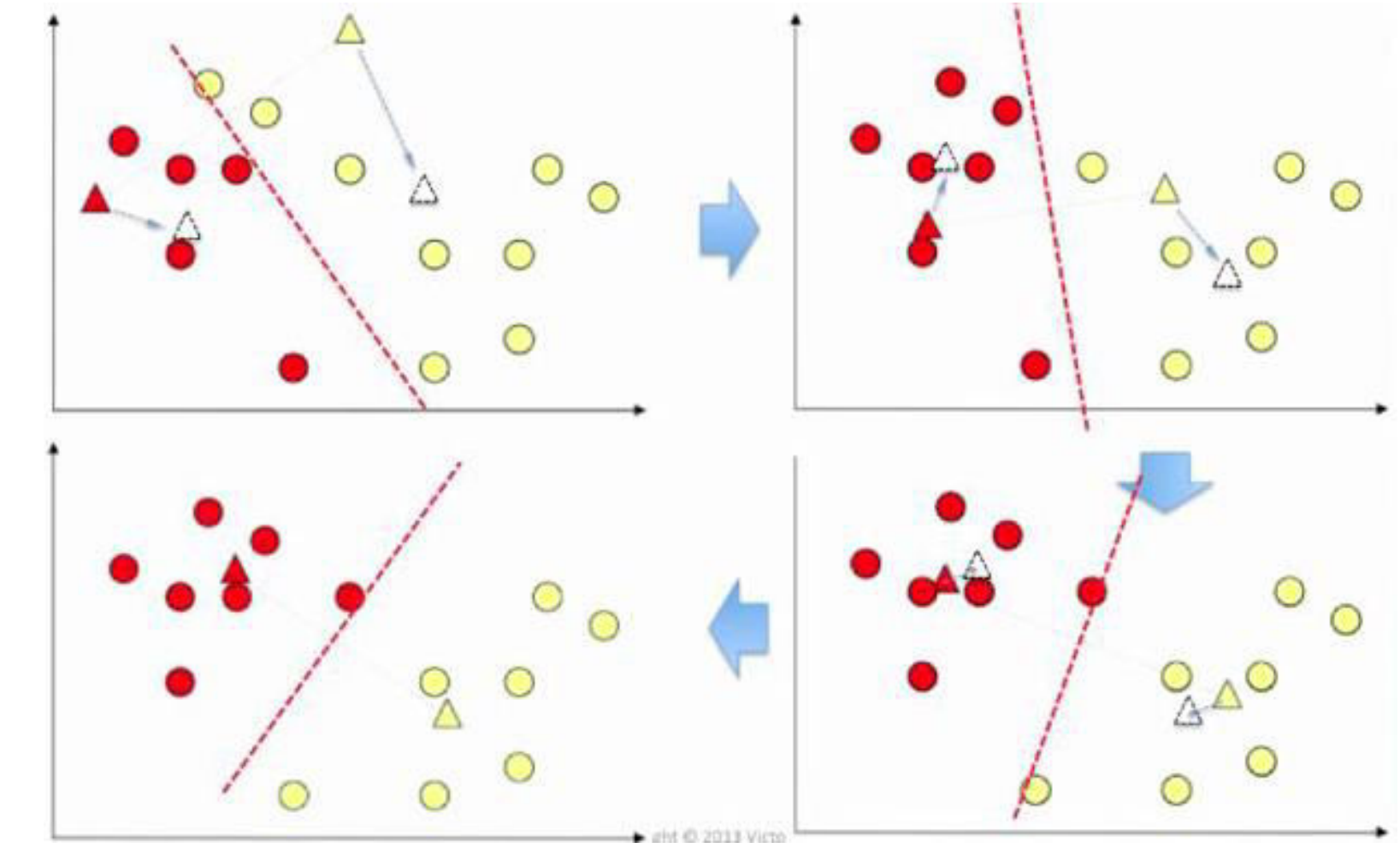
3 K-mean algoritmus (1)



3 K-mean algoritmus (2)



3 K-mean algoritmus (3)



https://www.youtube.com/watch?v=_aWzGGNrcic

3 A szoftver fejlesztése (2 óra)

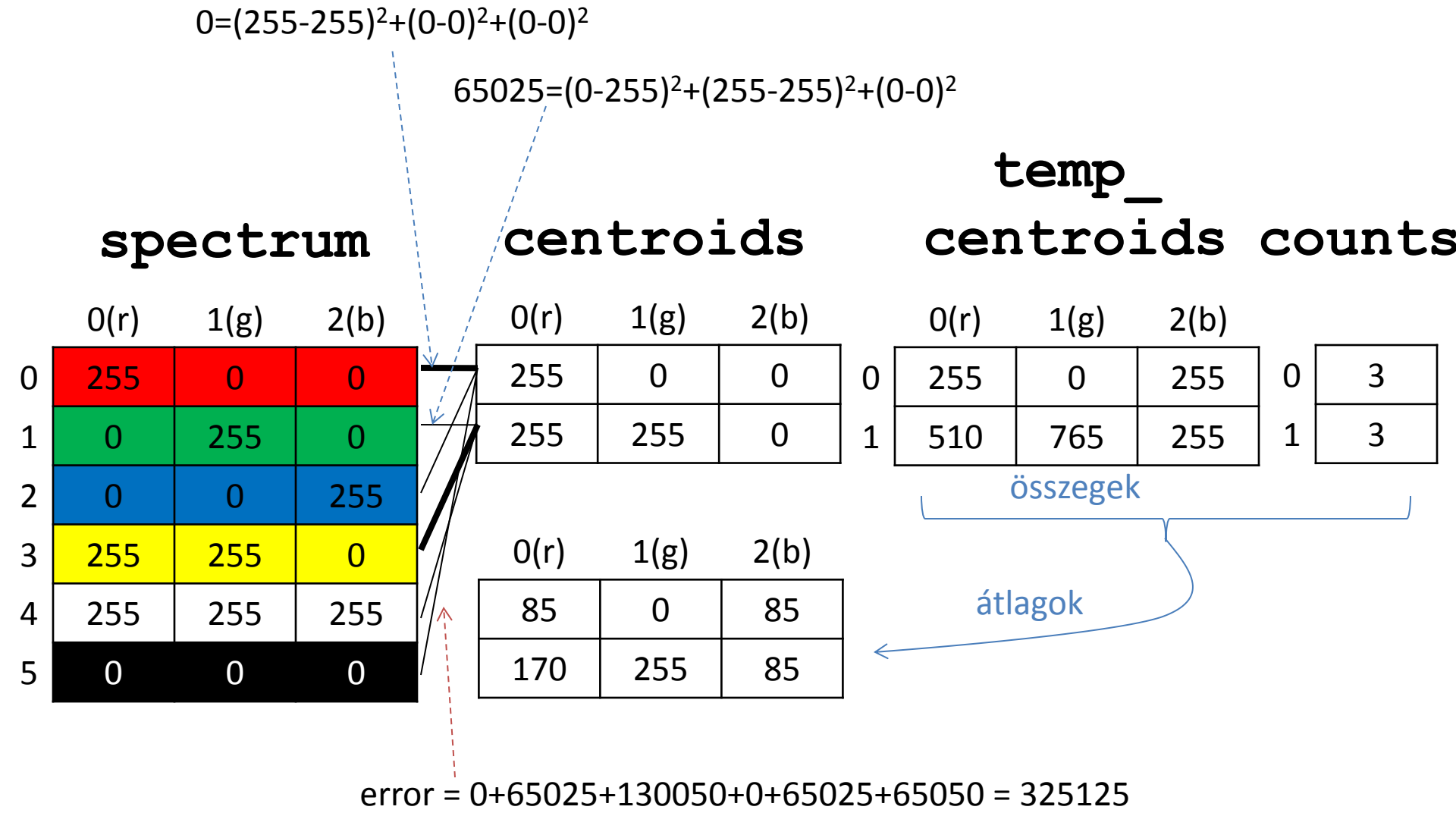
K-means clustering algorithm

- Input: K, set of points $x_1 \ldots x_n$
- Place centroids $c_1 \ldots c_K$ at random locations
- Repeat until convergence:
  - for each point x_i:
    - find nearest centroid c_j $\arg\min_j D(x_i, c_j)$
    - assign the point x_i to cluster j
  - for each cluster j = 1 ... K: $c_j(a) = \frac{1}{n_{j x_i \to c_j}} \sum x_i(a)$ for a = 1..d
    - new centroid c_j = mean of all points x_i assigned to cluster j in previous step
- Stop when none of the cluster assignments change

distance (e.g. Euclidian) between instance x_i and cluster center c_j

https://www.youtube.com/watch?v=_aWzGGNrcic

1. iteráció


0=(255-255)²+(0-0)²+(0-0)²
65025=(0-255)²+(255-255)²+(0-0)²
spectrum
centroids
temp_centroids
counts
0(r) 1(g) 2(b)
0 255 0 0
1 0 255 0
2 0 0 255
3 255 255 0
4 255 255 255
5 0 0 0
0(r) 1(g) 2(b)
255 0 0
255 255 0
0(r) 1(g) 2(b)
85 0 85
170 255 85
0(r) 1(g) 2(b)
0 255 0 255
1 510 765 255
0 3
1 3
összegek
átlagok
error = 0+65025+130050+0+65025+65050 = 325125

3

```
void CreateCentroids(PICTURE *p){
  p->centroids = (unsigned char (*)[3])
                    calloc(p->k * 3, sizeof(unsigned char));

  unsigned int (*temp_centroids)[3], *counts;
  counts = (int*)calloc(p->k, sizeof(int));
  temp_centroids = (unsigned int (*)[3])
                    calloc(p->k * 3, sizeof(unsigned int));

  int h, i, j;
  for ( h = i = 0 ; i < p->k ; h += p->nr_colors / p->k, i++ ) {
    copy(p->centroids[i], p->spectrum[h]);
  }
  …
```

3

```
void CreateCentroids(PICTURE *p){
  …
  int old_error, error = INT_MAX;
  do {
    old_error = error, error = 0;
    clear(temp_centroids, counts, p->k);
    for ( h = 0 ; h < p->nr_colors ; h++ ) {
      int closest_centroid, min_distance = INT_MAX;
      for ( i = 0 ; i < p->k ; i++ ) {
        int distance = eucl_dist(p->spectrum[h], p->centroids[i]);
        if ( distance < min_distance ) {
          closest_centroid = i; min_distance = distance;
        }
      }
      add(temp_centroids[closest_centroid], p->spectrum[h]);
      ++counts[closest_centroid];
      error += min_distance;
      p->cube[p->spectrum[h][0]][p->spectrum[h][1]]
              [p->spectrum[h][2]].centr_label = closest_centroid;
    }
    update_centroids(p->centroids, temp_centroids, counts, p->k);
  } while (error != old_error);
  …
}
```

3

```
void CreateReducedImage(const char* fname, PICTURE *p){
  FILE *fout = fopen(fname, "wb");

  fprintf(fout, "P6\n");
  fprintf(fout, "%i %i\n%i\n", p->x, p->y, p->maxc);

  int i;
  for( i = 0 ; i < p->x * p->y ; ++i ){
    int c_lable = p->cube[p->picture[i][0]][p->picture[i][1]]
                         [p->picture[i][2]].centr_label;
    fwrite(p->centroids[c_lable],sizeof(unsigned char[3]),1,fout);
  }

  fclose(fout);
}
```

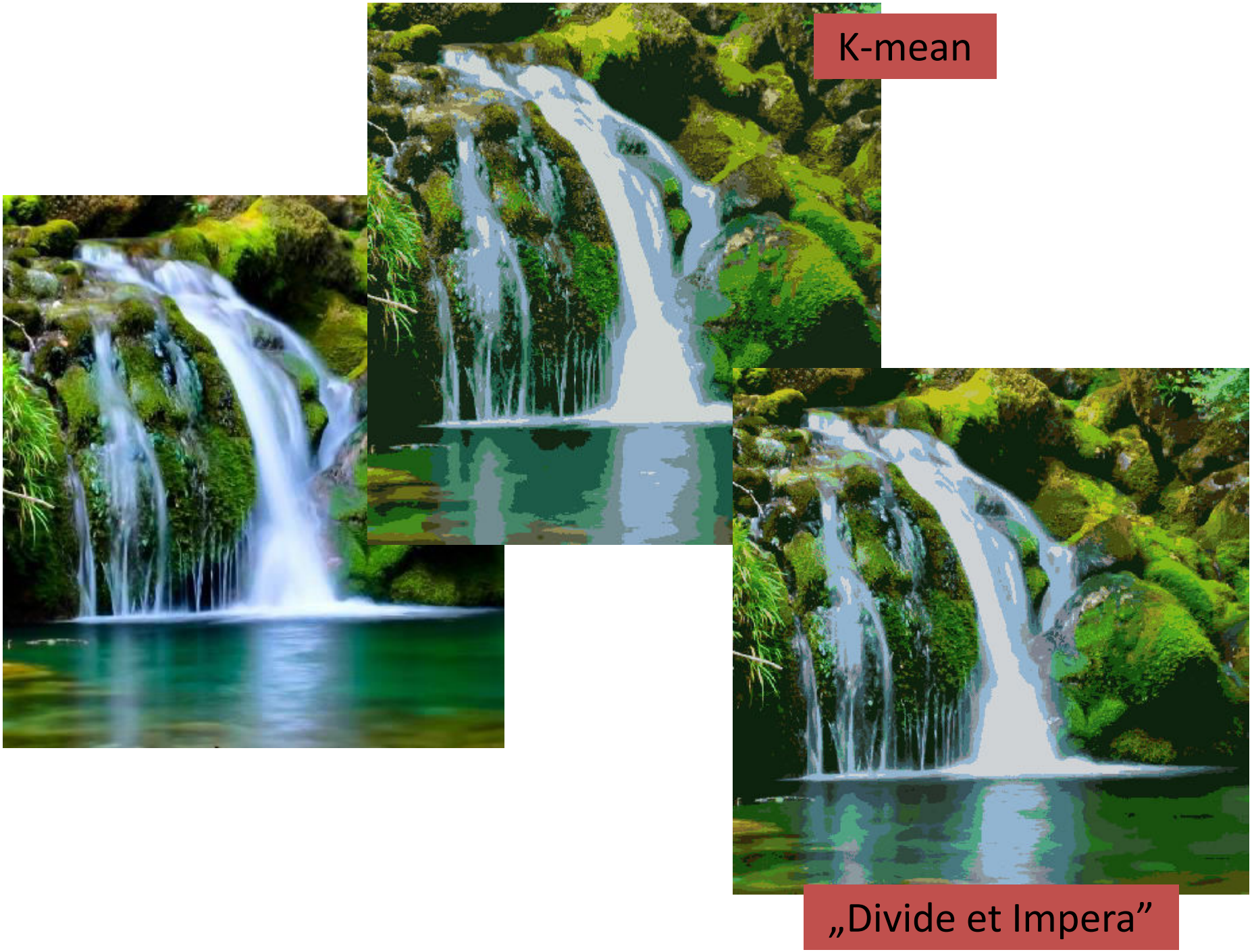
3 Divide et impera módszer

```
void CreateCentroids1(PICTURE *p){
    p->centroids1 = (unsigned int (*)[3])calloc(p->k * 3,
                     sizeof(unsigned int));
    divide(p,0,255,0,255,0,255,0,p->nr_colors);
}
```

Szint (level)

```
void divide(PICTURE *p, int rmin, int rmax, int gmin, int gmax,
            int bmin, int bmax, int l, int nr){
 int i,j,k,x,s;
  if ( l == p->n ){ // ha k=16, akkor n=4 (mélység)
   for(i = rmin ; i <= rmax ; ++i){
    for(j = gmin ; j <= gmax ; ++j){
     for(k = bmin ; k <= bmax ; ++k){
      if (p->cube[i][j][k].nr_pixels){
       p->centroids1[p->c_ind][0] += i;
       p->centroids1[p->c_ind][1] += j;
       p->centroids1[p->c_ind][2] += k;
       p->cube[i][j][k].centr_label1 = p->c_ind;
      }
     }
    }
   }
   p->centroids1[p->c_ind][0] /= nr;
   p->centroids1[p->c_ind][1] /= nr;
   p->centroids1[p->c_ind][2] /= nr;
   ++(p->c_ind);
  }
```

```
void divide(PICTURE *p, int rmin, int rmax, int gmin, int gmax,
            int bmin, int bmax, int l, int nr){
 …
  else{
   x = maxi(rmax-rmin+1, gmax-gmin+1, bmax-bmin+1); // 0/1/2, r/g/b
    switch (x) {
     case 0:{ // r-irányba a leghosszabb
      s = 0;
      for(i = rmin ; i <= rmax ; ++i){
       for(j = gmin ; j <= gmax ; ++j){
        for(k = bmin ; k <= bmax ; ++k){
         if (p->cube[i][j][k].nr_pixels){
          ++s;
         }
        }
       }
       if ( s >= nr/2 ){ break; }
      }
      divide(p,rmin,i,gmin,gmax,bmin,bmax,l+1,s);
      divide(p,i+1,rmax,gmin,gmax,bmin,bmax,l+1,nr-s);
      break;
    }
    case 1:{ … // g-irányba a leghosszabb
  }
 }
```


K-mean
„Divide et Impera”

Poszter a falon (1)

Poszter a falon (2)

Képtávolság – 1

```
s = 0;
for(i = 0 ; i < k ; ++i){
   for(j = 0 ; j < k ; ++j){
       d = dist(p1->centroids[i], p2->centroids[j]);
       s += d*(nr_pixels(i)+nr_pixels(j))
   }
}
s /= ((meret1+meret2)*k);
```

1. kép i. centroid-színére lecserélt pixelek száma

2. kép j. centroid-színére lecserélt pixelek száma

A képek méretei pixel-számban

Poszter a falon (3)

28782

1

37429

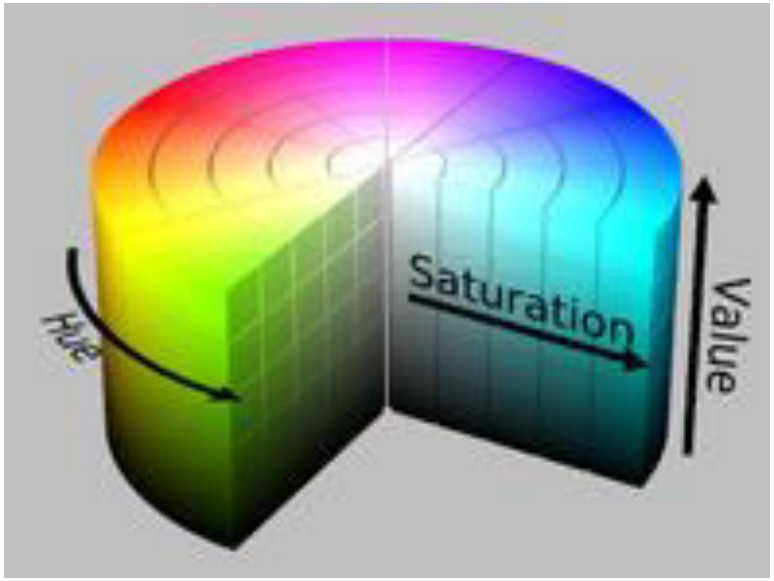
3

29253

2

HSV szín-tér

- **HSL** and **HSV** are the two most common cylindrical-coordinate representations of points in an RGB color model.
- The two representations rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation.



RGB >> HSV

[0..255] → [0..1]

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$Cmax = \max(R', G', B')$$

$$Cmin = \min(R', G', B')$$

$$\Delta = Cmax - Cmin$$

Hue calculation:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

[0..360]

Saturation calculation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

[0..1]

Value calculation:

$$V = Cmax$$

[0..1]

RGB >> HSV

```
void RGBtoHSV
(float r, float g, float b, float *h, float *s, float *v){
    float min, max, delta;
    min = MIN( r, g, b ); //r,g,b ∈ [0,1]
    max = MAX( r, g, b );
    *v = max;
    delta = max - min;
    if( max != 0 ) {*s = delta / max;}
    else {
        *s = 0;
        *h = -1;
        return;
    }
    if( r == max ) { *h = ( g - b ) / delta; }
    else if( g == max ) { *h = 2 + ( b - r ) / delta; }
    else { *h = 4 + ( r - g ) / delta; }
    *h *= 60;
    if( *h < 0 ) { *h += 360; } //h ∈ [0,360]; s,v ∈ [0,1]
}
```

Color similarity

- HSV coordinates of colors i and j
  - (h_i, s_i, v_i), (h_j, s_j, v_j)
- The distance between color-i and color-j
  - $a_{ij} = 1-(1/\sqrt{5})*\mathbf{sqrt(}(v_i-v_j)^2 + (s_i*\cos(h_i)-s_j*\cos(h_j))^2 + (s_i*\sin(h_i)-s_j*\sin(h_j))^2\mathbf{)}$

Comparing histograms

- A color histogram is a vector where each entry stores the number of pixels of a given color in the image.
- All images are scaled to contain the same number of pixels before histogramming, and the colors of the image are mapped into a discrete colorspace containing n colors.
- Typically images are represented in the RGB colorspace, using a few of the most signicant bits per color channel to discretize the space.

Naïve Color Quantization

24 bit to 8 bit:

Retaining 3-3-2 most significant bits of the R,G and B components.

Comparing histograms

Figure 1: Two images with similar color histograms

Comparing histograms

- It is common practice to use bin-to-bin distances comparing histograms.
  - This practice assumes that the histogram domains are aligned.
- However this assumption is violated in many cases due to quantization, …

http://mkweb.bcgsc.ca/color-summarizer/

- http://stats.stackexchange.com/questions/109618/computing-image-similarity-based-on-color-distribution