

ZÁKLADY POČÍTAČOVÉ GRAFIKY

Redukce barevného prostoru - cvičení 1

Petr Bobák (Rostislav Hulík)

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
ibobak@fit.vutbr.cz



- Organizace cvičení
- Seznámení s kostrou aplikace používanou v IZG
- Redukce barevného prostoru
- Samostatná práce

1. putPixel()	– 0.25 bodu
2. getPixel()	– 0.25 bodu
3. greyScale()	– 0.50 bodu
4. orderedDithering()	– 1.00 bod
5. errorDistribution()	– 1.00 bod

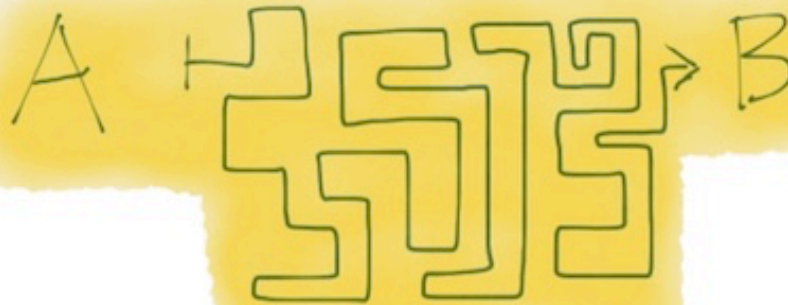
- **6 témat po dvou týdnech:**
 - *Redukce barevného prostoru*
 - Generování základních objektů v rastru
 - Vyplňování uzavřených objektů ve 2D
 - Zobrazování 2D křivek v rastru
 - 3D Transformace
 - Zobrazování 3D scény a základy OpenGL
- Každé cvičení úkol za 3 body
- **Průběh cvičení:**
 - Výklad a společná práce
 - Samostatná bodovaná práce

- Prakticky si vyzkoušet a ověřit znalosti získané na přednáškách

Theory:



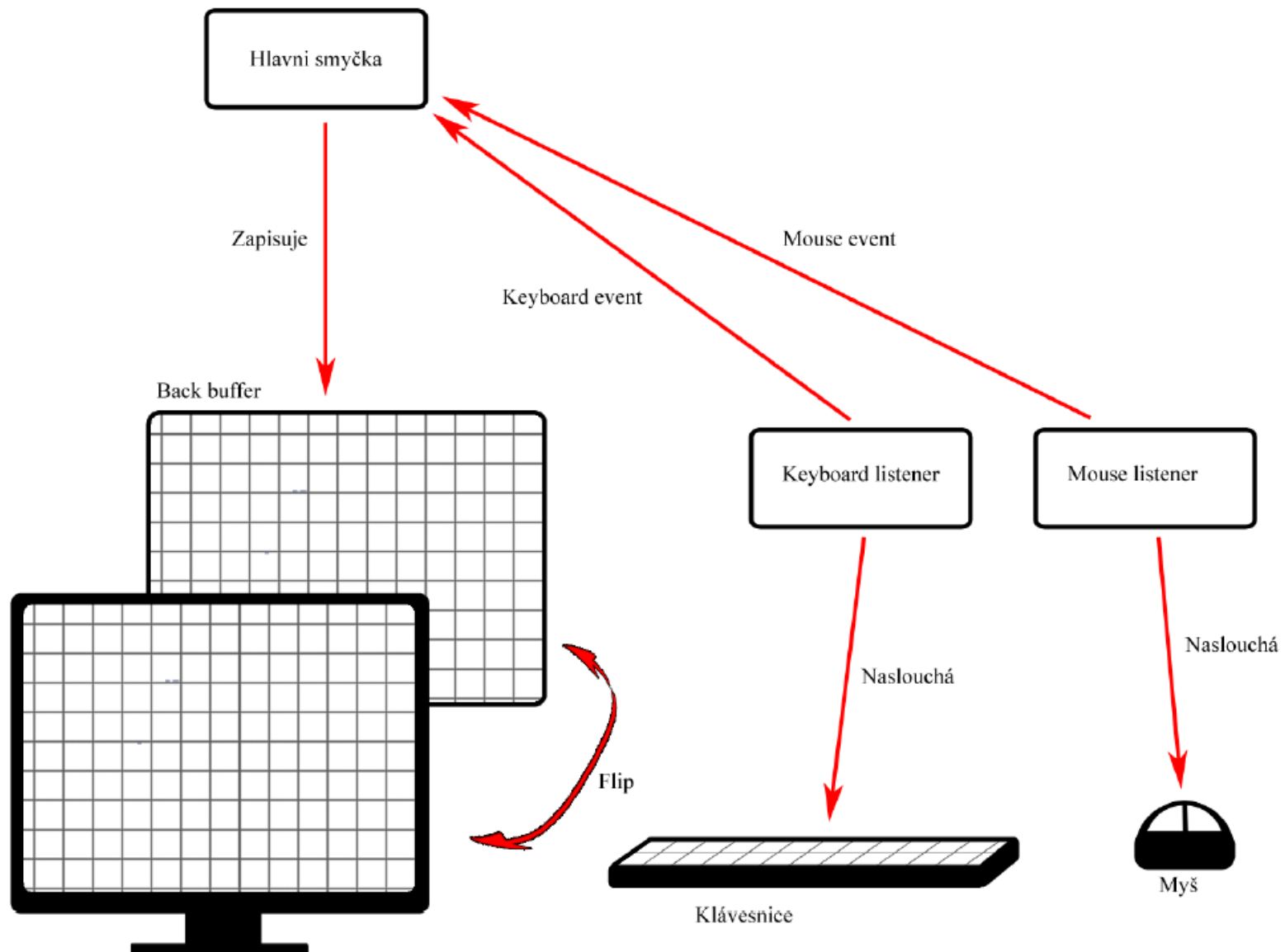
Practice:



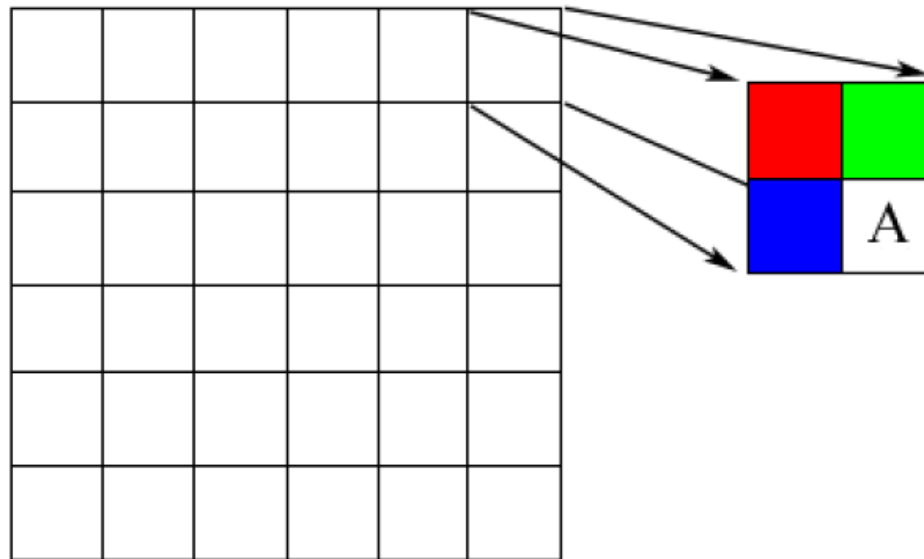


- Implementace v jazyce C
- Použití knihovny **SDL** (verze 1.2.15)
 - <http://www.libsdl.org/>
 - Knihovna pro multiplatformní programování médií (zvuk, grafika, vstupní zařízení) s HW akcelerací
 - Dostupné tutoriály
- Globální stavové proměnné
- Implementace úkolu **pouze** v souboru „student.c“

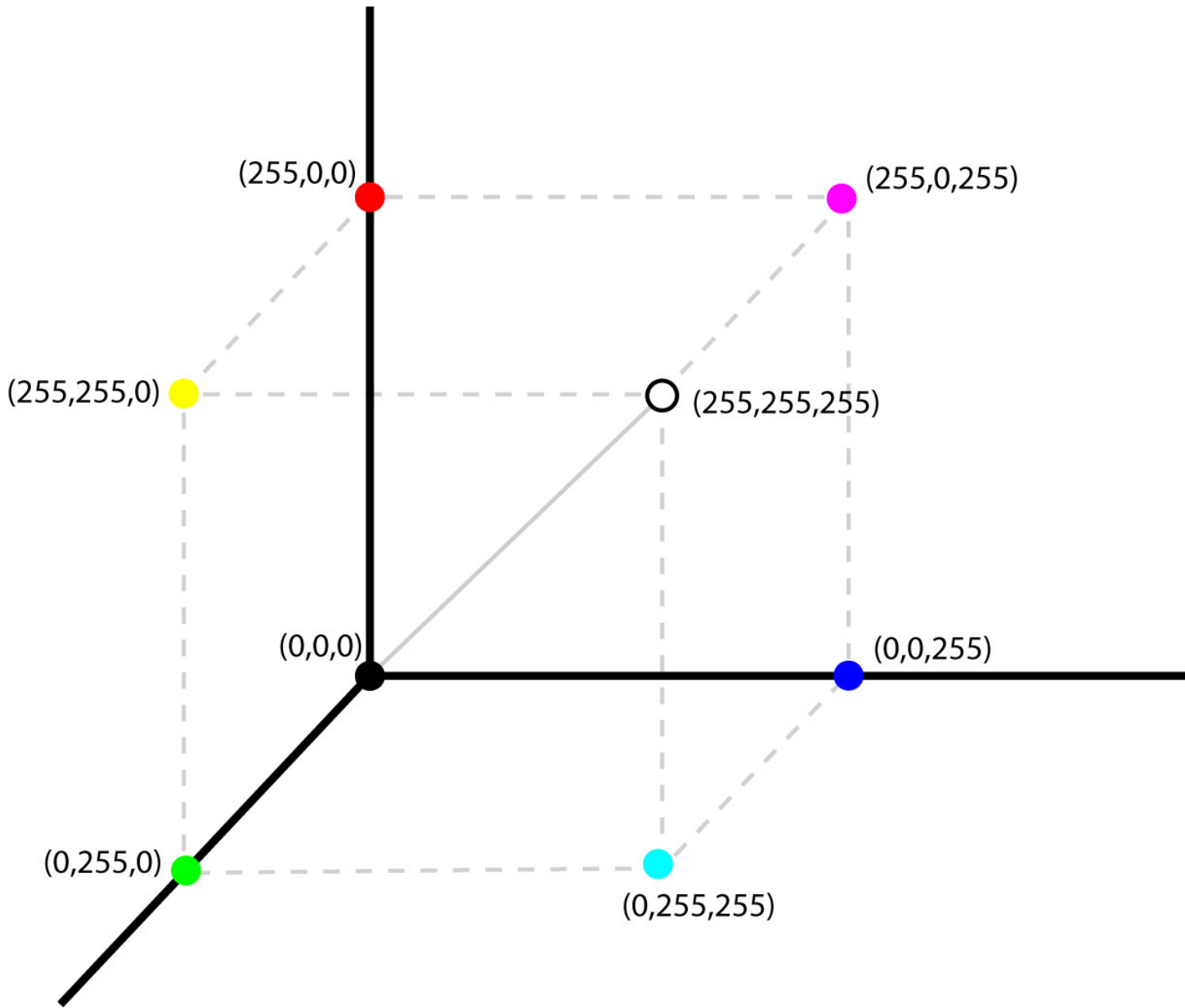
- **base.h**
 - Pomocné definice, funkce, makra
- **color.h + color.c**
 - Definice datového typu RGBA + funkce pro práci s ním
- **globals.h**
 - Stavové globální proměnné
- **io.h + io.c**
 - Funkce pro čtení/zápis z/do souboru .bmp
- **student.h + student.c**
 - Soubory s úkoly
- **main.c**
 - Hlavní tělo programu



- Pole *pixelů* reprezentující obraz
- Vnímáme jako 2D pole
- V IZG efektivnější implementace pomocí 1D pole



Pixel = **pix** (in the sense “pictures”) + **el**(ement), 1965-70



- Jeden pixel = *struktura S_RGBA*

```
struct S_RGBA {  
    unsigned char red;  
    unsigned char green;  
    unsigned char blue;  
    unsigned char alpha;  
};
```

- Framebuffer

```
S_RGBA *frame_buffer
```

- Většina operací ve cvičeních jsou postaveny na přímém zápisu do paměti *framebufferu*.

```
void putPixel(int x, int y, S_RGBA color)
```

- **Vstup:**

- Souřadnice x, y
- Pixel (S_RGBA)
- Velikost framebufferu (pomocné proměnné)

- **Postup:**

1. Test souřadnic x, y
2. Výpočet offsetu (2D index \rightarrow 1D index)
3. Zápis pixelu do framebufferu

- **Implementovat funkce (každá za 0,25 bodu)**
 - `S_RGBA getPixel(int x, int y)`
 - `void putPixel(int x, int y, S_RGBA color)`
- **Pomocné proměnné**
 - `int width`
 - `int height`
 - `S_RGBA *frame_buffer`
 - `COLOR_BLACK`
- **Testování**
 - Klávesa L načte obrázek ze souboru image.bmp
 - Klávesa S uloží scénu do souboru out.bmp
 - Kliknutí do scény vypisuje hodnotu pixelu

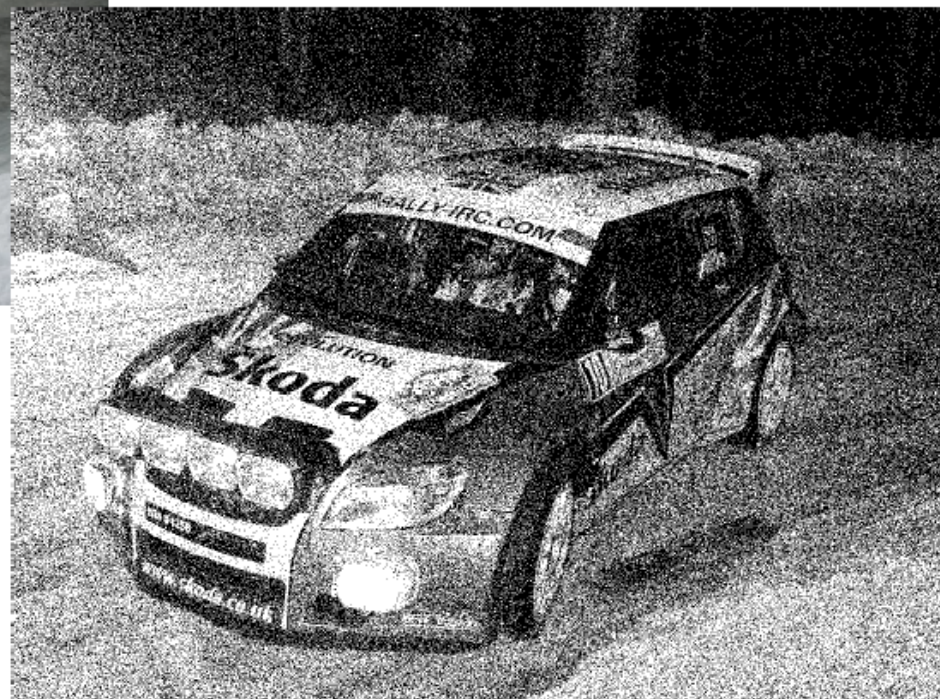
- Převod tříslložkového barevného modelu RGB do jednosložkového *šedotónového*
- Pixel má pouze jednu hodnotu – intenzitu
- **Vstup:**
 - $R(0..255), G(0..255), B(0..255)$
- **Výstup:**
 - $\text{Intensity} = 0.299R + 0.587G + 0.114B$
 - $\text{Intensity}(0..255)$
- **Simulace v RGB:**
 - $R = G = B = \text{Intensity}$



- **Implementovat funkci (úkol za 0.5 bodu)**
 - `void greyScale()`
- **Pomocné proměnné**
 - `int height`
 - `int width`
- **Pomocné funkce a makra**
 - `void putPixel(int x, int y, S_RGBA color)`
 - `S_RGBA getPixel(int x, int y)`
 - `ROUND(x)`
- **Testování**
 - Klávesa L načte obrázek, G převede obrázek do stupňů šedí

- Nutné vycházet z šedotónového obrazu
- Různé metody:
 - Prahování
 - Náhodné rozptýlení
 - Maticové rozptýlení
 - Distribuce chyby aj.
- Demo (implementováno):
 - Klávesa 1..4 prahuje obraz s různými hodnotami prahu
 - Klávesa R provádí algoritmus náhodného rozptýlení
 - Funkce k nahlédnutí v souboru student.c





- **Maticové rozptýlení**

- Porovnání pixelů obrazu s odpovídajícími hodnotami distribuční (rozptylovací) matice a prahování

$$M = \begin{bmatrix} 0 & 204 & 51 & 255 \\ 68 & 136 & 187 & 119 \\ 34 & 238 & 17 & 221 \\ 170 & 102 & 153 & 85 \end{bmatrix}$$

- Index pro matici M

$$\begin{aligned} i &= x \bmod n \\ j &= y \bmod n \end{aligned} \quad n \dots \text{side of M}$$

- Výstupní intenzita

$$G(x, y) = \begin{cases} I_{max} & \text{for } I(x, y) > M(i, j) \\ 0 & \text{else} \end{cases}$$

- Maticové rozptýlení

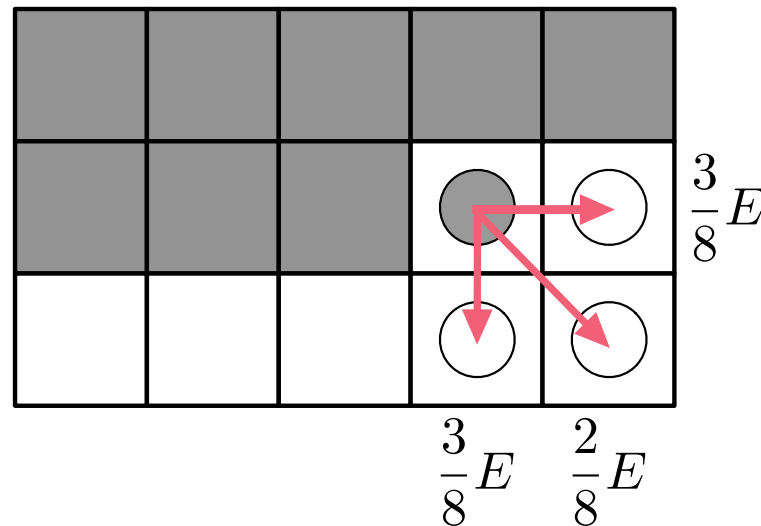


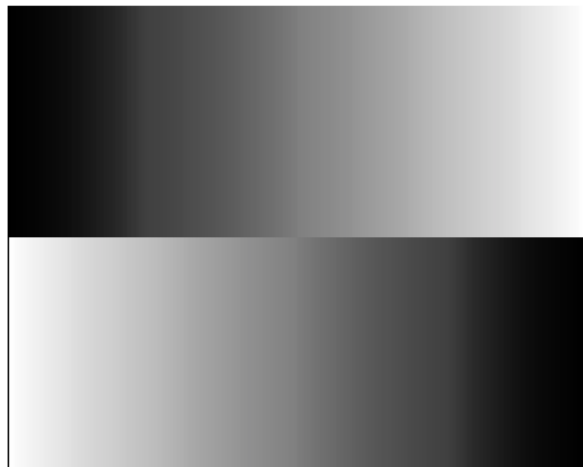
- **Implementovat funkci (za 1 bod)**
 - `void orderedDithering()`
- **Pomocné proměnné:**
 - `COLOR_WHITE`
 - `COLOR_BLACK`
 - `int height`
 - `int width`
 - `int M[]` (Matice rozptýlení)
 - `int M_SIDE` (Velikost strany čtvercové matice)
- **Pomocné funkce a makra:**
 - `void greyScale()`
- **Testování:**
 - „L“ načtení testovacího obr., „M“ spustí algoritmus mat. rozptýlení

• Distribuce chyby

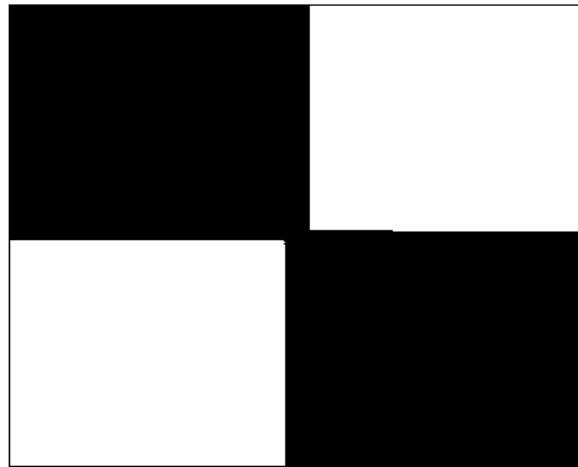
- Prahování, náhodné i maticové rozptýlení pracují pouze s jedním pixelem!
- Pro lepší kvalitu je vhodné sledovat i okolí

$$\bullet \text{ Chyba } E = \begin{cases} I(x,y) - I_{\max}, & \text{pokud } G(x,y) = I_{\max} \\ I(x,y), & \text{pokud } G(x,y) = 0 \end{cases}$$

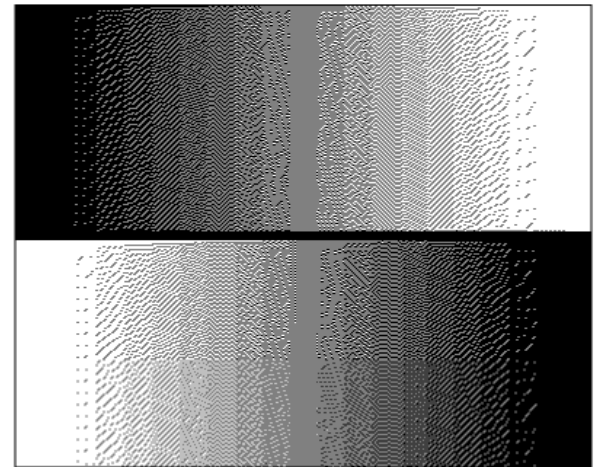




Vstupní obraz



Prahování



Distribuce chyby

- Distribuce chyby



- Implementovat funkci **(za 1 bod)**

- void errorDistribution()

- Pomocné proměnné:

- COLOR_WHITE
- COLOR_BLACK
- int height
- int width

- Pomocné funkce a makra:

- ROUND(x)
- void greyScale()

- $$Chyba\ E = \begin{cases} I(x,y) - I_{max}, & \text{pokud } G(x,y) = I_{max} \\ I(x,y), & \text{pokud } G(x,y) = 0 \end{cases}$$

- Testování:

- „L“ načtení testovacího obr., „D“ spustí algoritmus distribuce chyby

