ZÁKLADY POČÍTAČOVÉ GRAFIKY

Počítačová cvičení

Ústav počítačové grafiky a multimédií

Téma cvičení

Omezení barevného prostoru

Cíle cvičení IZG

- Prakticky si vyzkoušet a ověřit znalosti získané na přednáškách

Teorie (přednášky = jak to funguje)
X
Praxe (cvičení = jak se to dělá)

Organizace cvičení

- 6 témat po dvou týdnech = 12 týdnů
- V každém cvičení úkol za 0..3 body
- Jedno téma mají na starosti 1 – 2 cvičící
- Různá témata mohou mít různí cvičící

- **Průběh každého tématu:**
  - Výklad a společná práce
  - Samostatná bodovaná práce

Kompilace a editace programů

- **Kompilace v prostředí MS Visual Studio:**
  - Spustit soubor izg_lab.sln
  - Ctrl + F5 = build + run
- **Kompilace pomocí MinGW:**
  - Spustit konzoli MinGW: Q:\mingw\run mingw
  - V konzoli přepnout na adresář se cvičením
  - Pro překlad použít příkaz mingw32-make
- **Kompilace v CentOS**
  - Použít příkaz „make“ v adresáři se cvičením

Framework

- Implementace v C++ (C rozšířené o prvky C++)
- Použití GLUT – nadstavba OpenGL pro snadné, rychlé a multiplatformní aplikace s podporou OpenGL
- Stavové globální proměnné
- Štábní kultura kódu, komentáře, popisy

- Funkce programu:
  - Inicializace GLUT a OpenGL (RGBA, DoubleBuffering)
  - Registrace funkcí pro vykreslení, změnu velikosti okna, stisknutí klávesnice, stisknutí tlačítka myši, pohyb myši

Implementace úkolu pouze v souboru „student.cpp“

Framebuffer

- Kreslení probíhá do vlastního paměťového framebufferu
- Důvod:
  - Názornost funkce a implementace základních grafických aplikací
  - Prakticky nevhodné
- Lineární paměťové pole RGBA složek 32 bit barvy, chápáno jako 2D matice o rozměrech okna

Framebuffer

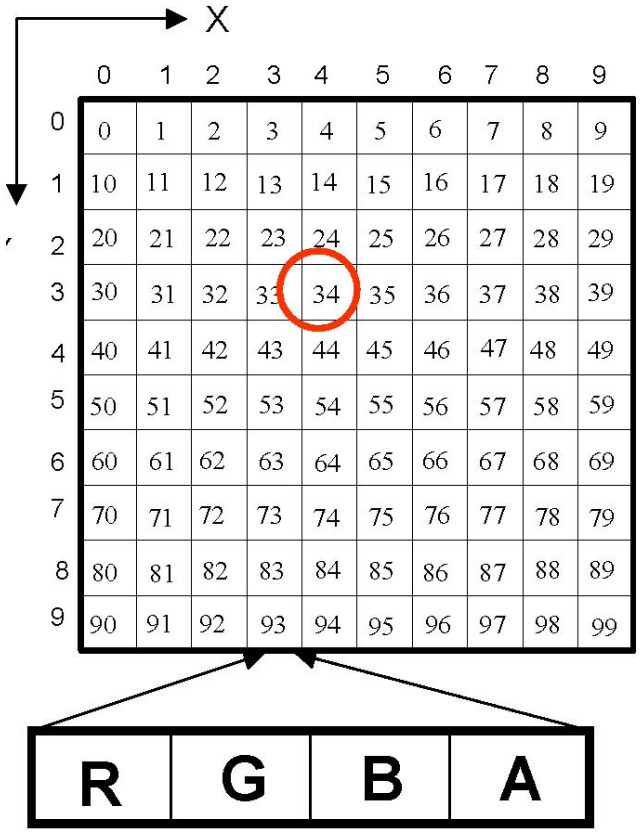
- Typ pole – struktura S_RGBA

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

- GLUT pouze zajišťuje vykreslení framebufferu na obrazovku
- Všechny grafické operace ve cvičeních (kromě posledního) jsou postaveny na přímém zápisu S_RGBA barev do paměti framebufferu

Vykreslení pixelu

- Vstup:
  - Souřadnice pixelu
  - S_RGBA barvy
  - Rozměry framebufferu
  - Ukazatel na začátek framebufferu
- Postup:
  - Test souřadnic vůči rozměrům
  - Výpočet ukazatele (offsetu)
  - Zápis RGBA barvy pixelů do paměti
- Pomocné funkce:




```
frame_buffer[frame_w*y + x] = color(R, G, B, A)
*(frame_buffer + frame_w * y + x] = color(R, G, B, A)
```

Vykreslení pixelu

- **Implementovat funkce (každá za 0,5 bodu):**

```
void PutPixel(int x, int y, S_RGBA color)
void GetPixel(int x, int y, S_RGBA & color)
```

- **Pomocné proměnné (pozor na hranice!):**

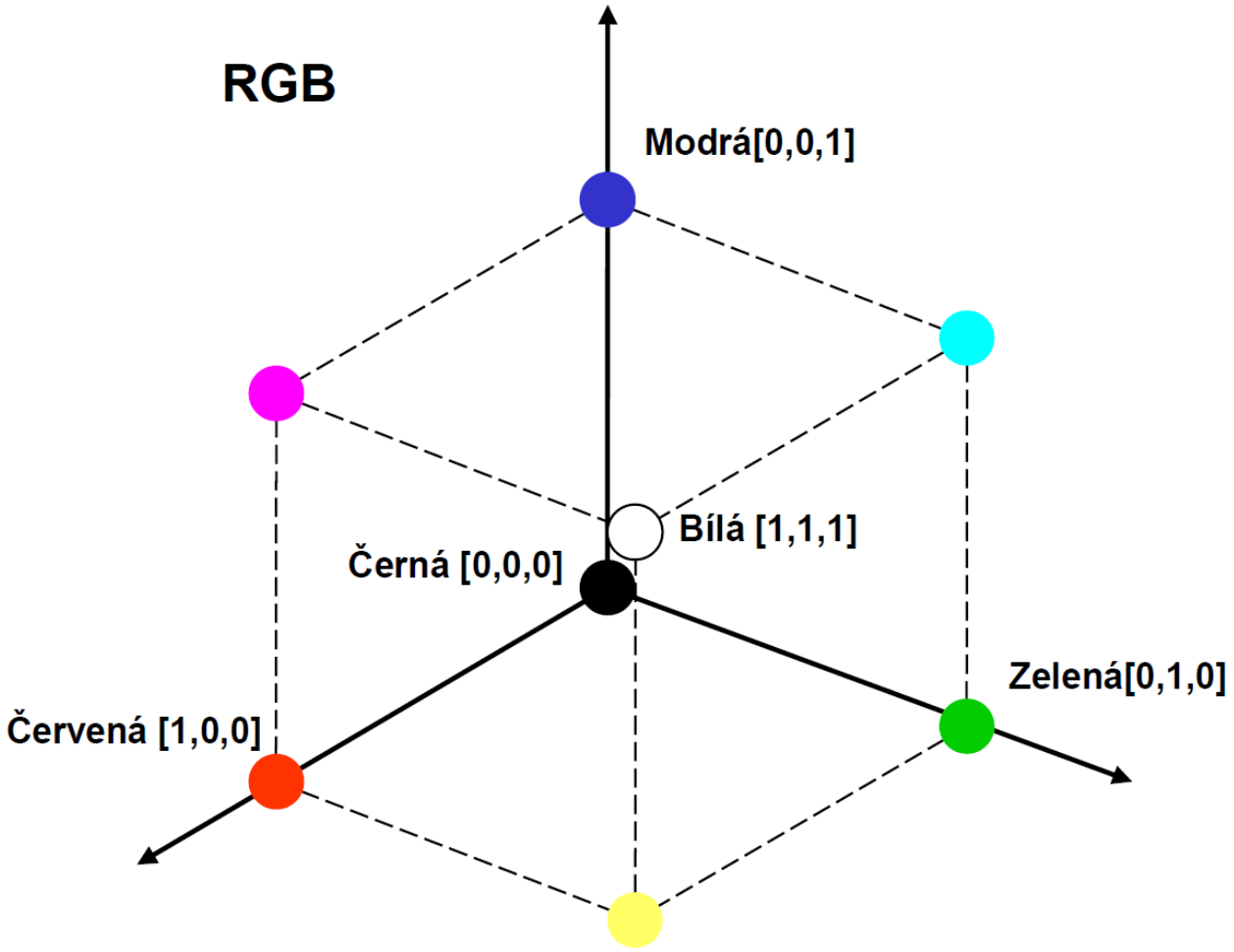
```
frame_h
frame_w
```

- **Pomocné funkce:**

```
frame_buffer[frame_w*y + x] = color(R, G, B, A)
*(frame_buffer + frame_w * y + x] = color(R, G, B, A)
```

- **Testování:**
  - PutPixel:
    - Klávesa „L“ načte obrázek
    - Tažení s pravé tlačítkem myši kreslí modré obdélníky
  - GetPixel:
    - Po kliknutí na obrázek vypisuje konzole hodnoty pixelů

Opakování – barevný model RGB



Převod do stupňů šedi

- **Vstup**:
  - Red (0..255)
  - Green (0..255)
  - Blue (0..255)
- **Výstup**
  - Greyscale (0..255)
  - GS = 0.299R + 0.587G + 0.114B

Převod do stupňů šedi

Převod do stupňů šedi

- **Implementovat funkce (společný úkol):**

```
void GreyScale()
```

- **Pomocné proměnné**

```
frame_h
frame_w
```

- **Pomocné funkce a makra**

```
void PutPixel(int x, int y, S_RGBA color)
void GetPixel(int x, int y, S_RGBA & color)
ROUND(x)
GS = 0.299R + 0.587G + 0.114B
```

- **Testování:**
  - „L“ načtení testovacího obrázku
  - „G“ převod do odstínů šedi

Převod do stupňů šedi

```
void GreyScale()
{
    int y, x;
    S_RGBA color;
    unsigned char intensity;
    for (y = 0; y < frame_h; y++)
        for (x = 0; x < frame_w; x++)
        {
            GetPixel(x, y, color);
            intensity = ROUND(color.red * 0.299 + color.green * 0.587
                              + color.blue * 0.114);
            color.red = intensity;
            color.green = intensity;
            color.blue = intensity;
            PutPixel(x, y, color);
        }
}
```

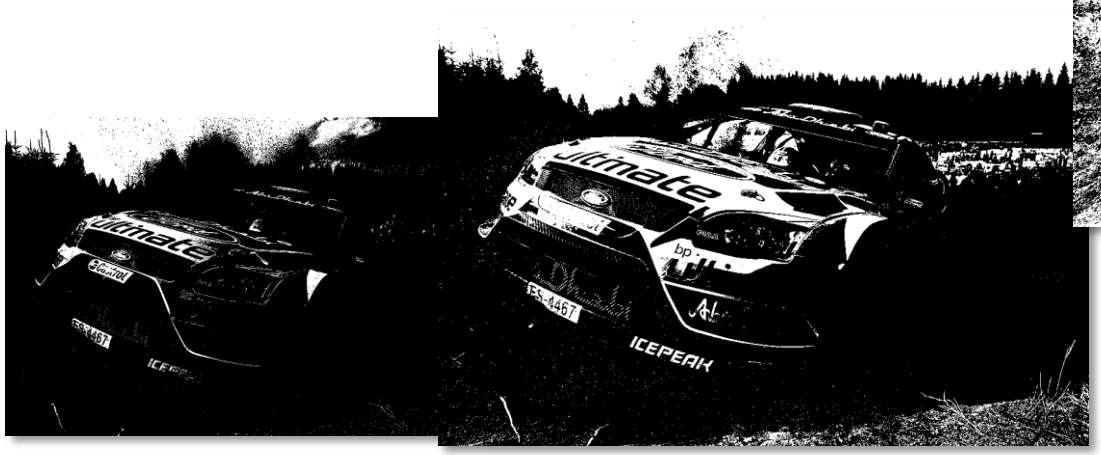
Prahování

- Převod prostoru odstínů šedi na černobílý
- Prahování intenzity, ne jedné ze složek RGB

- Nastavitelný práh t

- Výstupní barva $c_1 = \begin{cases} bílá, když\ c>t \\ černá, když\ c \leq t \end{cases}$

Prahování



Prahování

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

```
void Thresholding(int Threshold)
```

- **Pomocné proměnné**

```
COLOR_WHITE
COLOR_BLACK
frame_h
frame_w
```

- **Pomocné funkce a makra:**

```
ROUND(x)
GreyScale()
```

- **Testování:**
  - „L“ načtení testovacího obrázku
  - „1“..„4“ prahování s různou úrovní

Distribuce chyby

- Distribuce chyby okolním pixelům
- $Chyba\ \mathrm{E} = \begin{cases} I(x,y) - \mathrm{I}_{\max}, & pokud\ G(x,y)=1 \\ I(x,y), & pokud\ G(x,y)=0 \end{cases}$

I - Vstupní hodnota pixelu
G – Výstupní hodnota pixelu

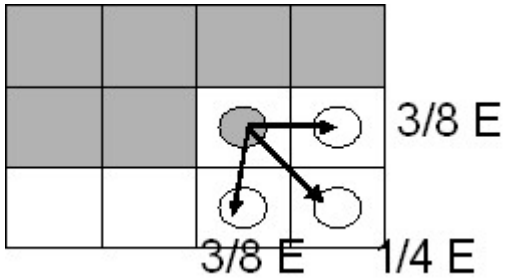
- Pro každý pixel je vypočtena chyba a distribuována do okolních pixelů
- Viz přednášky IZG

Distribuce chyby

Distribuce chyby

- **Implementovat funkci (za 1 bod)**
  `void` ErrorDiffusion()
- **Pomocné proměnné**
  `COLOR_WHITE`
  `COLOR_BLACK`
  `frame_h`
  `frame_w`
- **Pomocné funkce a makra:**
  `ROUND(x)`
  `GreyScale()`

$$Chyba\ E = \begin{cases} I(x,y) - I_{max}, & pokud\ G(x,y)=1 \\ I(x,y), & pokud\ G(x,y)=0 \end{cases}$$



- **Testování:**
  - „L“ načtení testovacího obrázku
  - „D“ spustí algoritmus distribuce chyby