IPA - pokročilé assemblery
2017/2018

Grafický editor: Rotace obrázku s antialiasingem

Tomáš Pazdiora
login: xpazdi02

Obsah

1 Zadání

Zadáním tohoto projektu bylo implementovat a následně optimalizovat rotaci obrázku bez aliasingu. Optimalizovaná část má být zapsána v jazyce symbolických instrukcí s pomocí instrukcí SSE, AVX a případně AVX2.

1.1 Aliasing

Při rotaci či jiné transformaci obrázku může vznikat aliasing, a to právě pokud se pro každý zdrojový pixel hledá cílový (transformovaný) pixel. Pro odstranění aliasingu je nutné provést opačný postup a pro každý cílový pixel hledat pixel zdrojový.

Použitím následujícího kódu vzniká aliasing:

```
1  for(x = 0; x < width; x++){
2      for(y = 0; y < height; y++){
3         [tx,ty] = transform([x,y]);
4         Destination[tx,ty] = Source[x,y];
5      }
6  }
```

Následující kód aliasing eliminuje:

```
1  for(x = 0; x < width; x++){
2      for(y = 0; y < height; y++){
3         [tx,ty] = inverse_transform([x,y]);
4         Destination[x,y] = Source[tx,ty];
5      }
6  }
```

2 Řešení

Pro řešení byl využit výše popsaný algoritmus který nezpůsobuje aliasing. Tento algoritmus sestaví inverzní transformační matici, pomocí které mapuje souřadnice zdrojového obrázku na cílový, transformovaný. Poté už jen přesune pomocí těchto transformovaných souřadnic data ze zdrojového do cílového obrázku.

2.1 Transformační matice

Při otáčení obrázku je důležitý střed rotace (origin) a úhel otočení (angle). Případně můžeme obrázek i škálovat (scale). Dílčí transformační matice si tedy označíme jako M_T pro matici posunu ke středu rotace, M_R jako matici rotace a M_S jako matici škálování. Výslednou inverzní transformační matici M^{-1} tedy vytvoříme následovně:

$$M^{-1} = M_T^{-1} \cdot M_S^{-1} \cdot M_R^{-1} M_T$$

Matice můžeme rozepsat podrobněji jako:

$$M^{-1} = \begin{bmatrix} 1 & 0 & -T_x \\ 0 & 1 & -T_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x^{-1} & 0 & 0 \\ 0 & S_y^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

Vynásobením dílčích matic získáme následnou inverzní transformační matici:

$$M^{-1} = \begin{bmatrix} \frac{\cos(\alpha)}{S_x} & \frac{\sin(\alpha)}{S_x} & \frac{\cos(\alpha) \cdot T_x}{S_x} - T_x + \frac{T_y \cdot \sin(\alpha)}{S_x} \\ -\frac{\sin(\alpha)}{S_y} & \frac{\cos(\alpha)}{S_y} & \frac{\cos(\alpha) \cdot T_y}{S_y} - T_y - \frac{T_x \cdot \sin(\alpha)}{S_y} \\ 0 & 0 & 1 \end{bmatrix}$$

Samotná transformace souřadnic pak vypadá takto:

$$\begin{bmatrix} \bar{x} \\ \bar{y} \\ 1 \end{bmatrix} = M^{-1} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Vzhledem k tomu, že nás třetí složka vektoru nezajímá, tak je v programu transformační matice zapsána pouze jako **matice 2x3**.

2.2 Interpolace

Zadání nijak nespecifikovalo interpolaci a tak pro jednoduchost byla zvolena interpolace nejbližším sousedem (nearest-neighbour). Tato metoda zaokrouhluje souřadnici na nejbližší souřadnici pixelu. Ostatní metody mohou poskytnout kvalitnější výsledky, jako například bikubická nebo alespoň bilineární interpolace. Tyto metody ale zase o něco více komplikují implementaci a požadují více přístupů do paměti.

3 Optimalizace

Pro optimalizaci kódu byly použity instrukce SSE, AVX, AVX2 a **FMA**. Vzhledem k tomu, že použití FMA instrukcí nebylo v zadání, je možné je v případě nekompatibility při kompilaci pomocí NASM vypnout přepínačem *WITHOUT_FMA.*

3.1 Struktura algoritmu

4 Spuštění programu

5 Závěr