Vladimír Klement, Tvorba softwaru, FJFI ČVUT

Školitel: Ing. Tomáš Oberhuber Ph.D.

ROZPOZNÁVÁNÍ OBRAZU POMOCÍ GPGPU A TECHNOLOGIE CUDA

GRAFICKÁ KARTA

- Speciální kus hardwaru sloužící k výpočtům v 3D hrách
- Řádové zrychlení pro výpočetně náročné paralelní úlohy
- Lepší možnost růstu výkonu do budoucna, než u procesorů

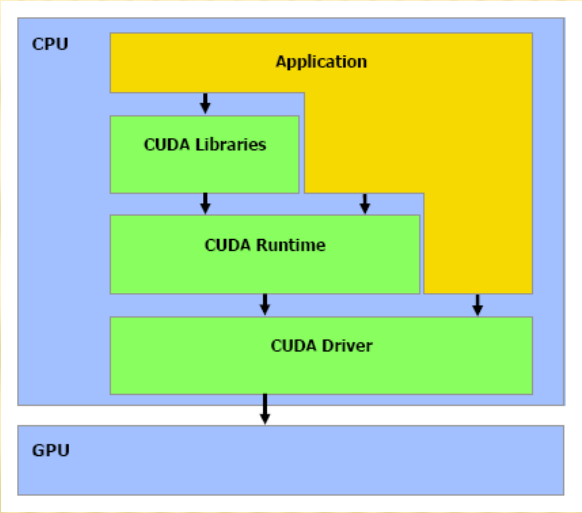
TECHNOLOGIE CUDA

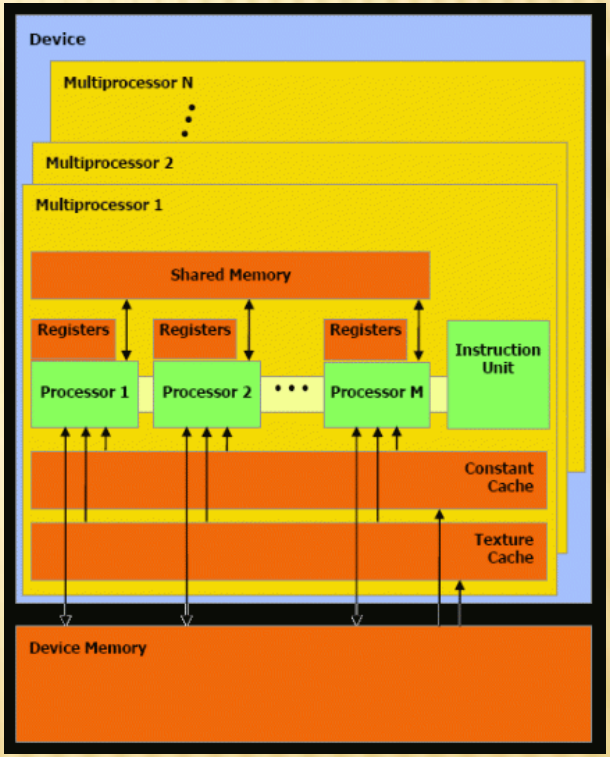
- Moderní technologie pro výpočty na grafických kartách od firmy Nvidia
- Odstraňuje nutnost používat grafické API
- Jazyk pro psaní programů na grafickou kartu podobný jazyku C
- Jen karty od firmy Nvidia, do budoucna OpenCL

TECHNOLOGIE CUDA

Důležité pojmy:

- Kernely
- Vlákna
- Síť - Grid





CUDA – UKÁZKA KÓDU

```
//Deklarace kernelu
__global__ void kernel(float* odata, int height, int width)
{
   unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
   unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
   float c = tex2D(tex, x, y);
   odata[y*width+x] = c;
}
```

```
//Zavolání kernelu
cudaBindTextureToArray(tex, cu_array);
dim3 blockDim(16, 16, 1);
dim3 gridDim(width / blockDim.x, height / blockDim.y, 1);
kernel<<< gridDim, blockDim, 0 >>>(d_odata, width, height);
cudaUnbindTexture(tex);
```

LEVEL SET FUNKCE VE ZPRACOVÁNÍ OBRAZU

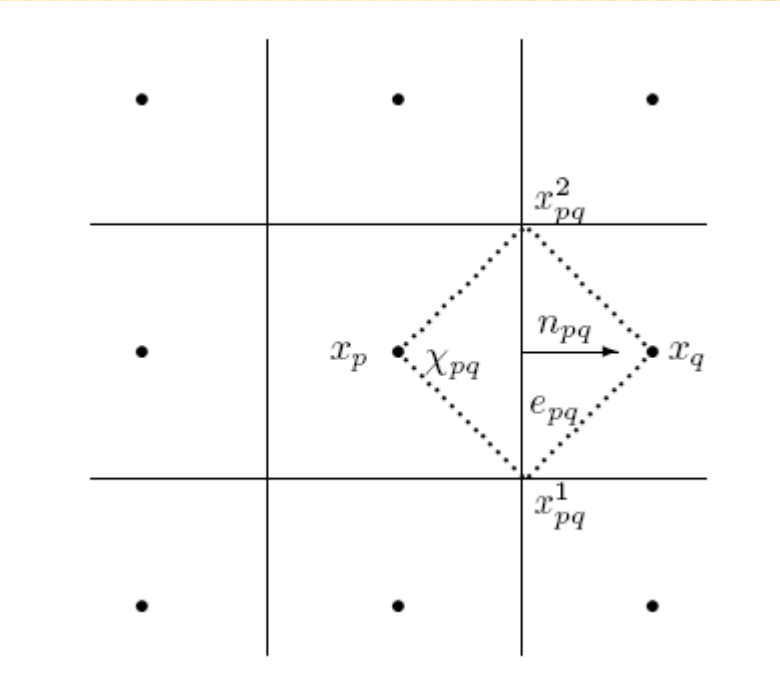
- Vymyšlena Osherem a Sethianem
- Metoda vývoje křivky pomocí implicitní funkce
- Využití dynamika tekutin, segmentace obrazu …

Regularizovaná verze pro segmentaci obrazu

$$u_t = \sqrt{\varepsilon^2 + |\nabla u|^2}\,\nabla \cdot \left(g^0 \frac{\nabla u}{\sqrt{\varepsilon^2 + |\nabla u|^2}} \right)$$

METODA KOMPLEMENTÁRNÍCH KONEČNÝCH OBJEMŮ

- Metoda pro prostorovou diskretizaci úlohy
- Základní myšlenka stejná jako konečné objemy



EXPLICITNÍ SCHÉMATA

Eulerova metoda

- Nejjednodušší metoda pro rovnice

$$\frac{du_{i,j}}{dt} = f(t,u)$$

- Výpočet dle

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \tau \cdot f(t,u^n)$$

Mersonova metoda

- Metoda čtvrtého řádu
- Automatické určení kroku

$$k_{i,j}^{1} = \tau \cdot f(t,u^n)$$

$$k_{i,j}^{2} = \tau \cdot f(t+\frac{1}{3}\tau, u^n + \frac{1}{3}k^1)$$

$$k_{i,j}^{3} = \tau \cdot f(t+\frac{1}{3}\tau, u^n + \frac{1}{6}k^1 + \frac{1}{6}k^2)$$

$$k_{i,j}^{4} = \tau \cdot f(t+\frac{1}{2}\tau, u^n + \frac{1}{8}k^1 + \frac{3}{8}k^3)$$

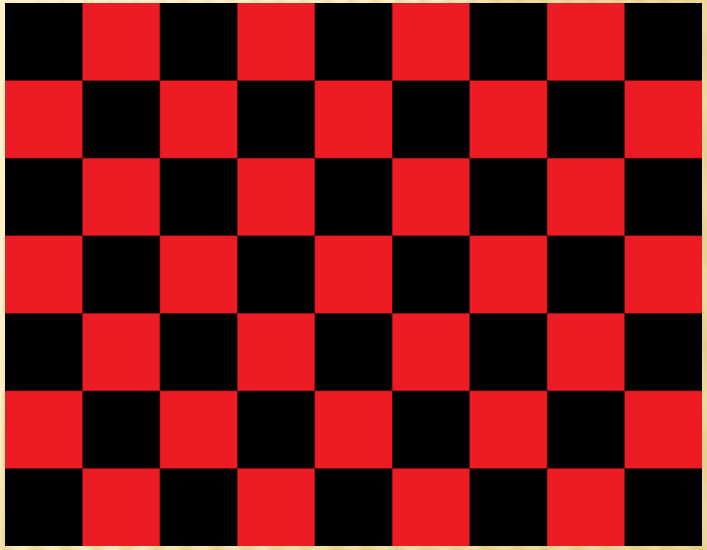
$$k_{i,j}^{5} = \tau \cdot f(t+\tau, u^n + \frac{1}{2}k_{i,j}^{1} - \frac{3}{2}k^3 + 2k^4)$$

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{1}{6}k_{i,j}^{1} + \frac{2}{3}k_{i,j}^{4} + \frac{1}{6}k_{i,j}^{5}$$

SEMIIMPLICITNÍ - RED-BLACK SOR METODA

- Systém rovnic řešíme SOR metodou

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij}x_j^{(k)} - \sum_{j<i} a_{ij}x_j^{(k+1)}\right)$$



- Potřeba úpravy do paralelizovatelné verze
- Speciální tvar matice

Čas	0,02	0,05	0,1
Klasická	4 915	12 926	25 022
Red-Black	5 429	14 275	27 710

VÝSLEDKY - ZRYCHLENÍ

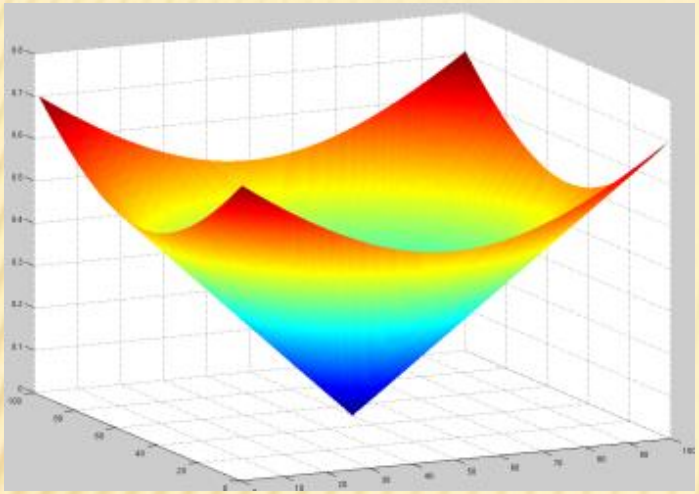
Dosažené urychlení pomocí GPU

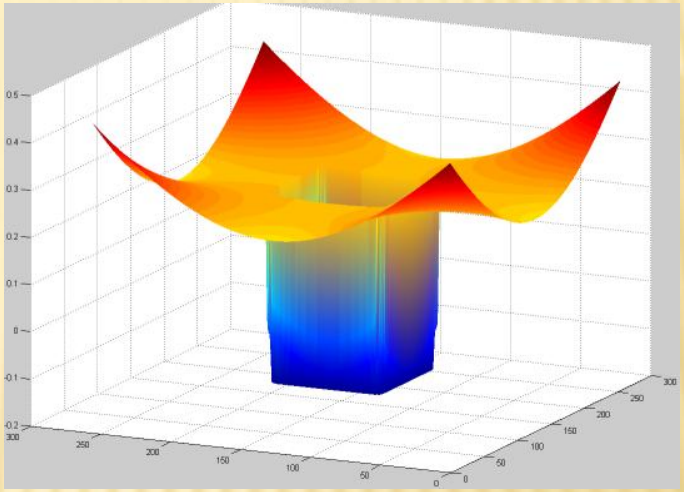
	Eulerova metoda	Mersonova metoda	Red-Black SOR
128x128 px	11x	10x	2x
256x256 px	18x	18x	6x
512x512 px	18x	18x	6x

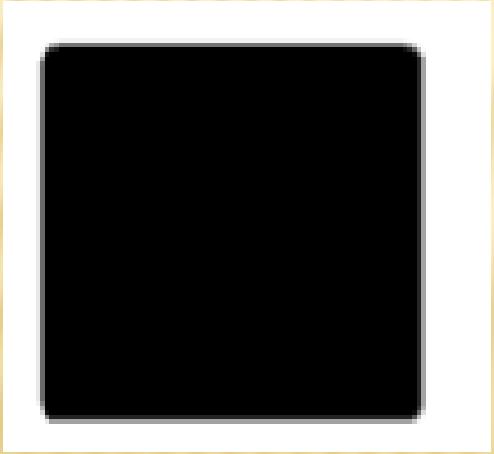
Porovnání jednotlivých metod

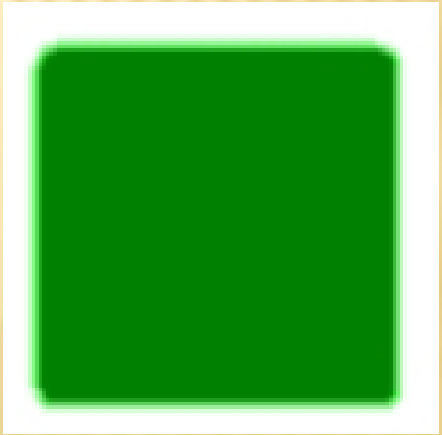
Metoda	Čas GPU	Čas CPU
Eulerova	282 s	5083 s
Mersonova	317 s	5706 s
Red-Black SOR	203 s	1214 s

VÝSLEDKY – SEGMENTACE 1

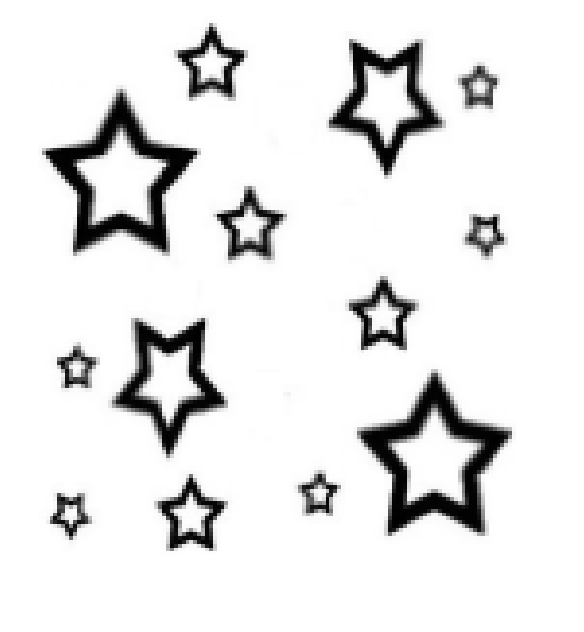






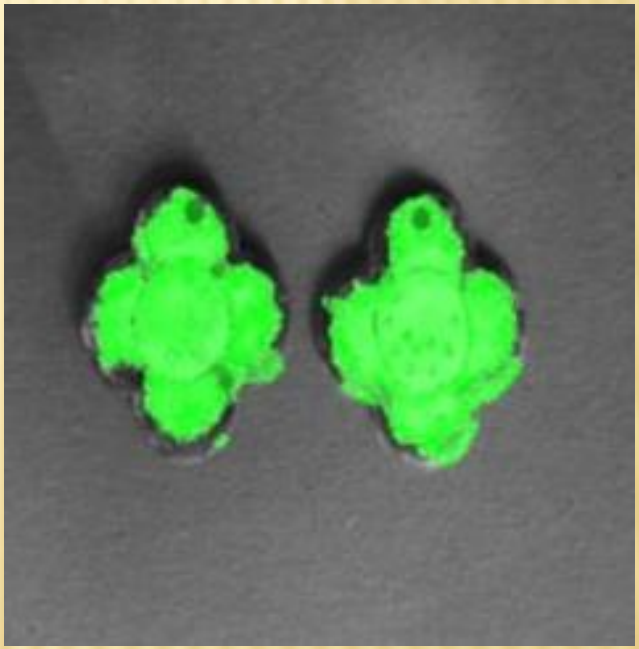


VÝSLEDKY SEGMENTACE 2



VÝSLEDKY SEGMENTACE 3

VÝSLEDKY SEGMENTACE 4



KONEC

- Děkuji za pozornost.