projekt do předmětu **GMU – Grafické a multimediální procesory**

Urychlení zpracování obrazu založené na modifikaci histogramu OpenCL/CUDA

řešitelé: **Lukáš Piják**, xpijak00
Filip Zapletal, xzaple27
Jan Vybíral, xvybir05

Zadání

- Implementovat výpočet histogramu vstupního obrázku pomocí OpenCL.
- Každý člen týmu implementovat jednu metodu využívající histogramu, a to jak s pomocí OpenCL tak CPU implementaci.
- Vstup aplikace zpracuje vždy oběma způsoby a vypíše čas každé z nich.
- V dokumentaci zdůvodnit použití paralelismu a přizpůsobení algoritmu pro běh na GPU.
- Zvolené metody (algoritmy):
    - Ekvalizace histogramu
    - Prahování s pomocí metody Otsu
    - Segmentace obrazu na základě prahování histogramu

Použité technologie

- OpenCL 1.2
- SDL 1.2
- Microsoft Visual Studio 2010 (Projekt)
- C++

Použité zdroje

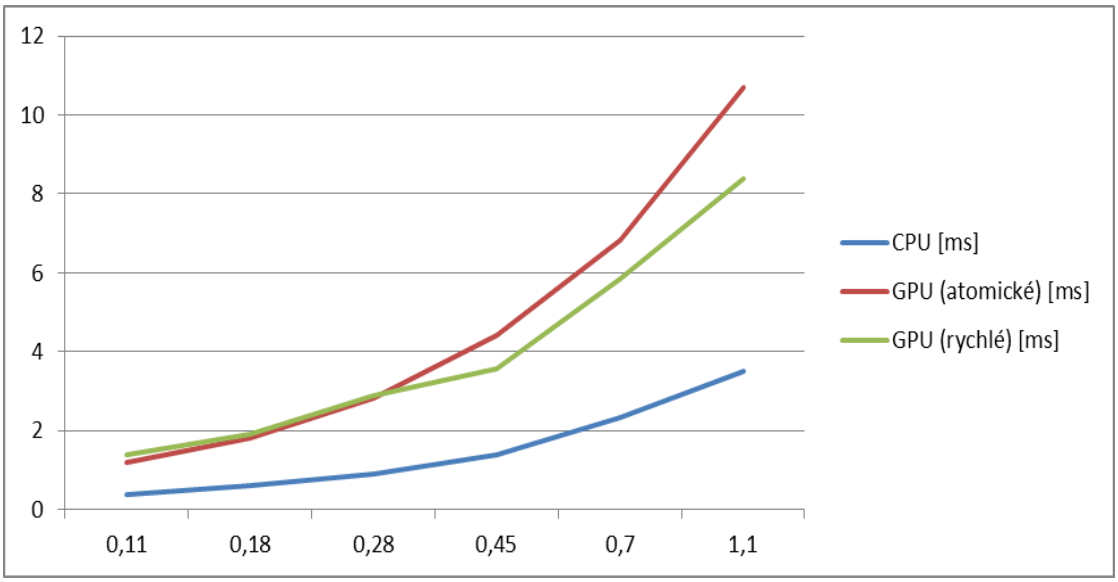
- Histogram
    - http://www.rapidsnail.com/Tutorial/t/2012/116/63/23959/opencl-learning-step-by-step-7-grayscale-histogram-computation-1.aspx
- Otsu
    - http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf
    - http://www.codeproject.com/Articles/38319/Famous-Otsu-Thresholding-in-C
    - http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html
    - Inspirace z projektu z HSC (FIT VUT Brno)
- Ekvalizace
    - http://fourier.eng.hmc.edu/e161/lectures/contrast_transform/node2.html
- Segmentace
    - L. Lucas, Image Segmentation. München : Technische Universität München, 2010.
    - F. Kurugollu, B. Sankur, A.E. Harmanci. Color image segmentation using histogram multithresholding and fusion. 2001.
    - Rajagopal, Venugopal. Image Segmentation by Histogram Thresholding. 2002.
- Další
    - http://www.khronos.org/files/opencl-1-2-quick-reference-card.pdf

- Šablona ze cvičení GMU

Nejdůležitější dosažené výsledky

Výpočet histogramu šedotónového obrázku

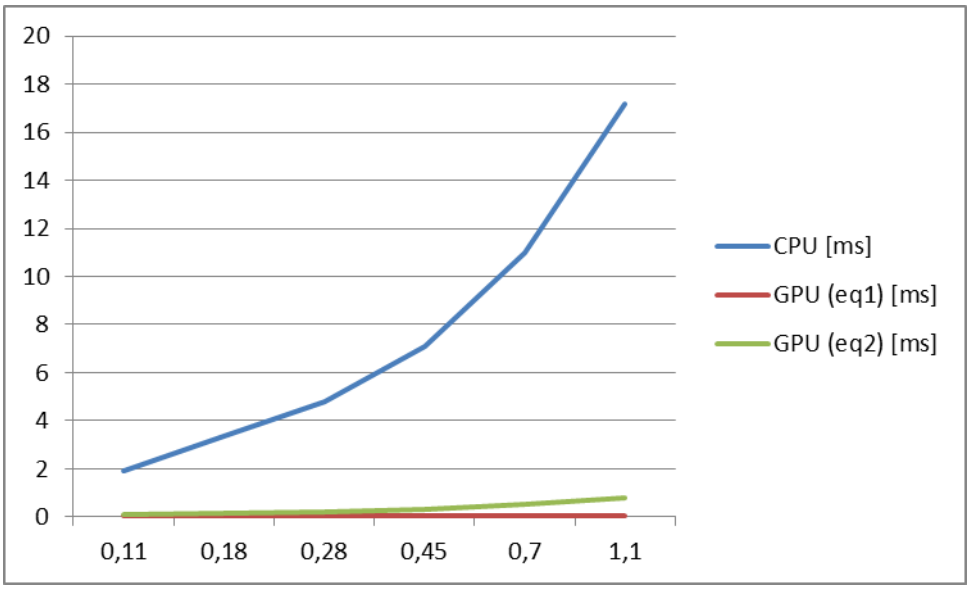
Implementovali jsme jak na CPU tak GPU. Na GPU jsme udělali dvě různé implementace. Ta první používá atomické instrukce a je velmi pomalá. Poté jsme implementovali jinou metodu, která se ukázala být o něco efektivnější, a navíc se obešla bez použití atomických instrukcí.



Graf 1: Srovnání rychlosti CPU implementace a obou GPU implementací výpočtu histogramu

Ekvalizace histogramu

Implementovali jsme metodu ekvalizace histogramu jak na CPU tak GPU a podařilo se nám dosáhnout velmi výrazného urychlení na GPU. Výpočet na GPU je rozdělen do dvou částí (kernelů).

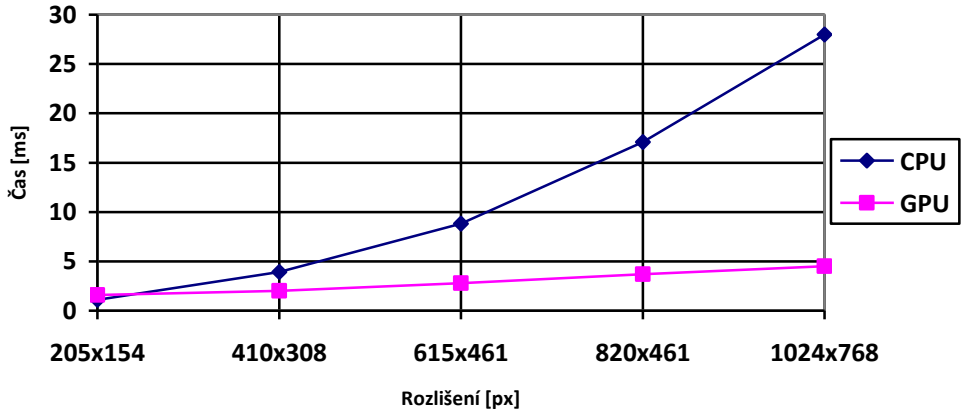


Graf 2: Srovnání rychlosti CPU implementace a obou částí GPU implementace ekvalizace histogramu

Obrázek 1: Ukázka ekvalizace, vlevo vstup a vpravo výstup

Prahování pomocí metody Otsu

Metoda je rozdělena na 2 části. První je výpočet prahu pomocí metody Otsu, kde jsou paralelizovány součty histogramu pomocí sčítacího stromu. Druhá část je zaměřená na samotné prahování, kde je obraz rozdělen na části, které jsou paralelně zpracovávané. Na grafu je patrné zrychlení GPU implementace v závislosti na velikosti obrazu.

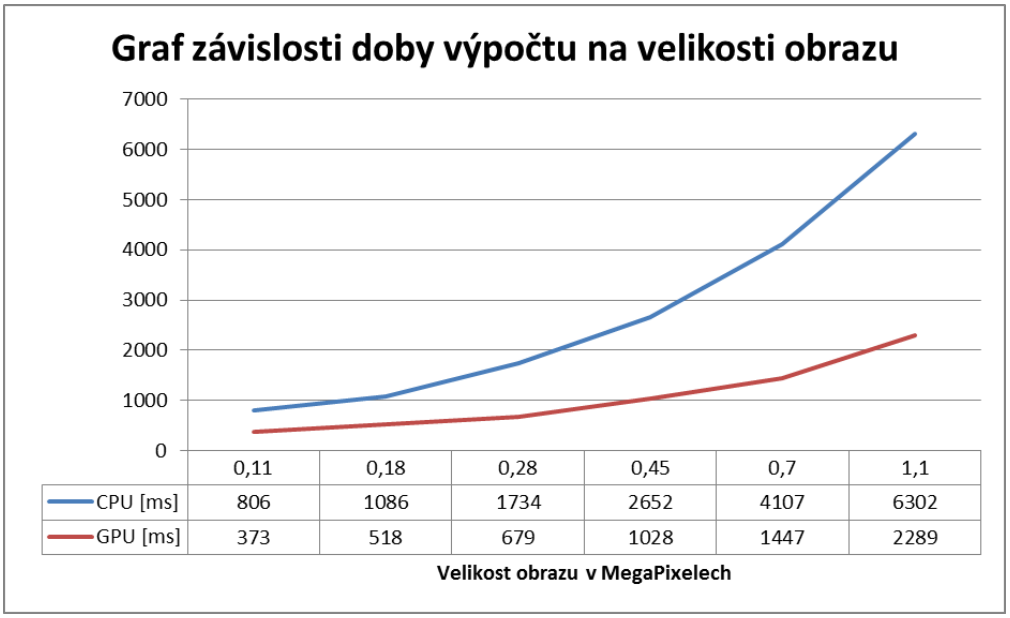


Graf 3 – Závislost času na velikosti obrazu pro CPU a GPU

Obrázek 2- Výsledek metody Otsu (http://cs.wikipedia.org/wiki/Škoda_13T)

Segmentace obrazu s využitím prahování histogramu

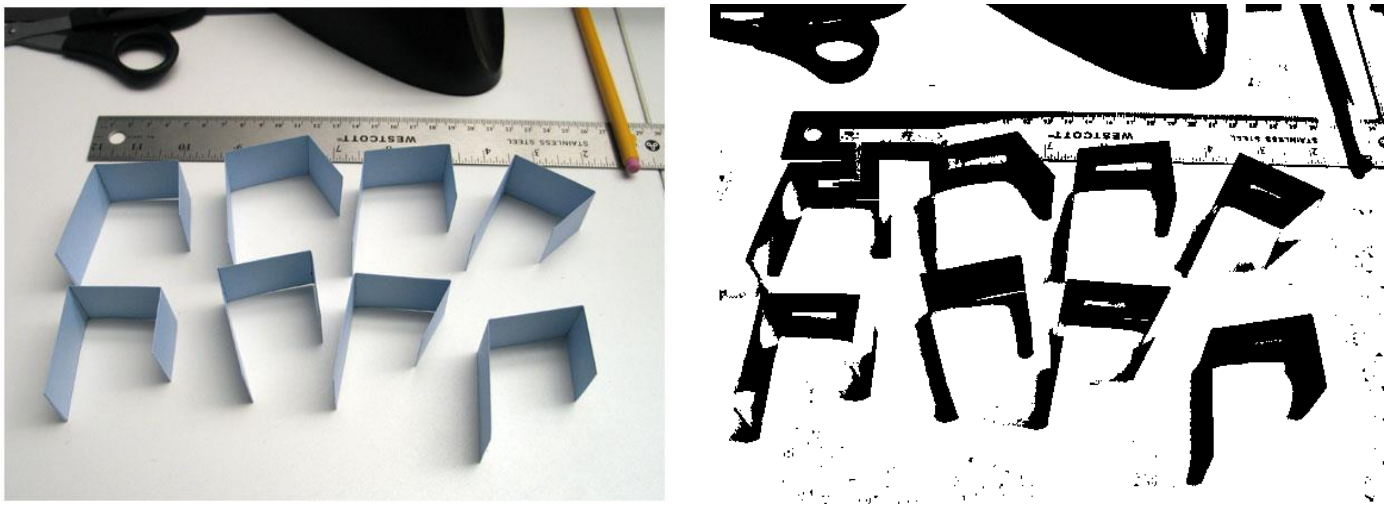
Na testovacím stroji (CPU – Intel Core i7 2.44GHz; GPU NVIDIA GeForce GT620M) bylo dosaženo cca 3 násobné akcelerace. Tato hodnota se může zdát malá, ale je ovlivněna poměrně výkonným CPU oproti nevýkonnému grafickému adaptéru. Ten taky obsahuje poměrně malé množství výpočetních jednotek, a tedy na lepším zařízení by bylo možné použít vyšší úroveň paralelizace tím získat i vyšší výkon. Nelze opomenout ani algoritmus obsahující mimo výpočtu poměrně velké množství řídící logiky (podmínky, skoky), které výpočet na GPU brzdí.



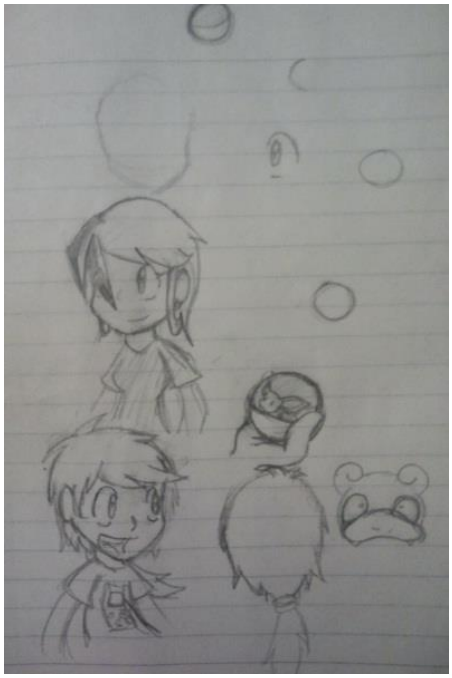
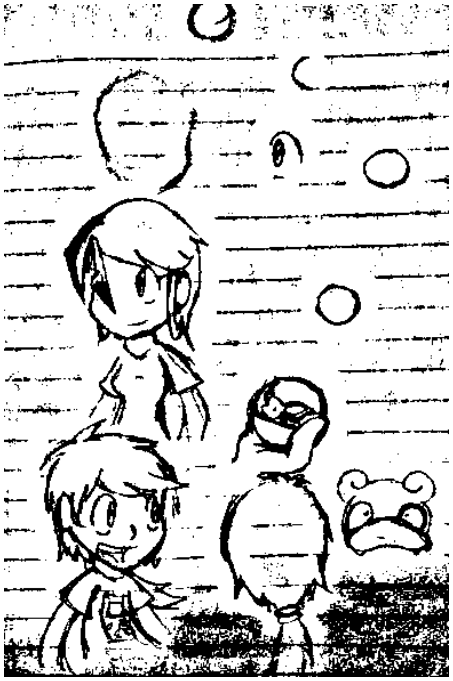
	0,11	0,18	0,28	0,45	0,7	1,1
CPU [ms]	806	1086	1734	2652	4107	6302
GPU [ms]	373	518	679	1028	1447	2289

Graf 4 – Závislost času na velikosti obrazu pro CPU a GPU

Výstupy algoritmu velmi závisí na celkové členitosti obrazu. Adaptivní metoda se totiž hodí primárně pro velmi členité obrázky, kde jsou objekty po obraze rozmístěny rovnoměrně. Pokud budeme mít velkou jednolitou plochu pozadí, tak metoda selhává, protože v takto omezeném regionu není z čeho určit práh. Tam se hodí nějaká globální metoda.



Obrázek 3: Ukázka segmentace – tmavě jsou objekty v popředí, světle v pozadí (zdroj: avoision.com)

Obrázek 4: Ukázka využití segmentace pro odělení hran jednotlivých objektů (zdroj: paper-bob-omb.deviantart.com)

Ovládání vytvořeného programu

Ovládání je více než standardní. Program je spouštěn z konzole, kdy je potřeba zadat 3 vstupní parametry. Viz následující schéma:

```
gmu.exe <metoda výpočtu histogramu> <metoda> <cesta k obrázku>
```

Metodu výpočtu histogramu lze vybrat ze dvou možností:

- `hist1`
- `hist2`

Metodu lze vybrat ze tří možností:

- `equalize`
- `otsu`
- `segmentation`

Zvláštní použité znalosti

Výpočet histogramu

Výpočet histogramu je implementován dvěma různými způsoby, ze kterých si uživatel může vybrat:

1. V první metodě zpracovává každé vlákno jeden pixel vstupního obrázku. Vlákna jsou uspořádána do dvourozměrných workgrup o velikosti 16x16. Samotný výpočet probíhá tak, že nejprve úplně první vlákno vynuluje histogram, potom první vlákno každé workgrupy vynuluje lokální paměť (každá workgrupa má jednu). Poté, co mají všechny workgrupy vynulovanou lokální paměť, přistoupí se k samotnému výpočtu. Každé vlákno získá hodnotu svého korespondujícího pixelu a atomicky inkrementuje odpovídající položku lokální paměti. Potom, co všechna vlákna ve workgrupě inkrementovala lokální paměť, první vlákno ve workgrupě atomicky přičte lokální pamět ke globálnímu histogramu.

2. Druhá metoda je rozdělena do dvou kernelů, které se vykonávají po sobě. V prvním kernelu zpracovává každé vlákno 255 pixelů (stejně, jako je velikost histogramu). Vlákna jsou uspořádána do jednorozměrných workgrup o velikosti 128. Každá workgrupa sdílí lokální paměť o velikosti 128x255. Každá workgrupa si nejprve vynuluje svoji lokální paměť a poté si do ní každé vlákno spočte svůj subhistogram z jemu korespondujících 255 pixelů. Tím získáme v každé workgrupě pole 128 subhistogramů. Poté se pro každý index histogramu sečtou hodnoty na odpovídajících pozicích těchto subhistogramů a výsledek se přiřadí do výstupního pole finálních subhistogramů o velikosti počet workgrup x 255. Výstupem prvního kernelu tedy je pole subhistogramů jednotlivých workgrup. Toto pole se poté předá na vstupu druhému kernelu. Ten má 256 vláken. Každé z nich sečte hodnoty na odpovídajících indexech vstupního pole subhistogramů a výsledek uloží do výsledného histogramu.

Ekvalizace histogramu

Cílem ekvalizace histogramu je zvýšit globální kontrast obrázku, čehož se dosáhne rovnoměrnějším rozložením intenzit v histogramu obrázku. Implementace ekvalizace je rozdělena do dvou kernelů, které se vykonávají po sobě. První část slouží k určení nových hodnot pro všechny možné hodnoty pixelů vstupního obrázku (kterých je 255) a zpracovává ji 255 vláken. První vlákno nejprve spočte ze vstupního histogramu kumulativní histogram, ve kterém je na každém indexu kromě prvního součet hodnot z předchozích indexů původního histogramu. Výsledek se získá tím, že každé vlákno vynásobí sobě odpovídající hodnotu kumulativního histogramu hodnotou 255 / počet pixelů obrázku. Výsledek z tohoto kroku je potom na vstupu druhého kernelu, který využívá dvourozměrné workgrupy o velikosti 16x16 vláken. Jedno vlákno odpovídá jednomu pixelu vstupního obrázku. V každém vlákně se pro daný pixel zjistí nová hodnota z pole získaného v předchozí části a ta se mu přiřadí.

Segmentace obrazu s využitím prahování histogramu

Cílem segmentace je rozdělení jednotlivých částí obrazu do několika podmnožin. Těmi může být například popředí a pozadí objektu na obrázku.

Pokud se tedy omezíme pouze na rozdělení do dvou takovýchto skupin, lze využít jednoduchou metodu prahování. Problémem je ale určení prahu. Zvolená metoda práh získává s využitím histogramu tak, že se v něm snaží najít separační hodnotu s využitím následujícího iterativního algoritmu:

1. Zvolíme inicializační práh T.
2. Za pomoci T rozdělíme histogram na dvě podmnožiny – pod a nad prahem.
3. Pro každou podmnožinu vypočteme vážený průměr (mean1 a mean2) všech bodů v podmnožině.
4. Vypočítáme nový práh T jako (mean1+mean2)/2
5. Opakujeme kroky 2-4 dokud se práh dramaticky mění

Pro účely tohoto projektu byla navíc zvolena adaptivní varianta tohoto algoritmu, kdy se práh (a i histogram, ze kterého vychází) nepočítá z celého obrázku, ale pouze z okolí aktuálně zpracovávaného bodu.

Při výpočtu na GPU (OpenCL) je pak využito toho, že jsou na sobě pak jednotlivé pixely obrazu nezávislé a je možné je provádět paralelně, tedy v jednom kroku například pro region 32x16 bodů, čímž je získána potřebná akcelerace. Ta ovšem není až tak výrazná (relativně), protože algoritmus obsahuje poměrně velké množství „řídící logiky“, tedy podmíněných příkazů a skoků, kde GPU jádra nad těmi standardními zaostávají.

Rozdělení práce v týmu

Společně: Výpočet histogramu

Jan Vybíral: kostra aplikace, ekvalizace histogramu

Lukáš Piják: prahovani pomoci metoty Otsu

Filip Zapletal: Segmentace na základě prahování histogramu

Co bylo nejpracnější

Nejpracnější bylo určitě dekomponovat jednotlivé části algoritmu a získat představu o tom, které části algoritmu lze paralelizovat, tak aby nedocházelo k časově závislým chybám. Dále vhodně zvolit počet výpočetních jednotek pro jednotlivé části obrazu. Také je důležité vhodně zvolit velikost zpracovávaného obrazu pro jedenu výpočetní jednotku. Posledním závažným problémem byla atomicita operací, nebo synchronizace taková, aby nebyly atomické operace potřebné.

Zkušenosti získané řešením projektu

Snad nejdůležitější zkušeností získanou v tomto projektu je porozumění OpenCL a jeho využití v praxi. Také je dobré zdůraznit pochopení problematiky počítání na grafickém akcelerátoru a problém paralelizace. Důležité bylo také uvědomění si, že grafický akcelerátor lze použít i na jiné aplikace než je vykreslování grafiky na obrazovku počítače.

Autoevaluace

Technický návrh: 70% (analýza, dekompozice problému, volba vhodných prostředků, …)

Každý měl implementovat vlastní algoritmus, který pracuje nad daty v histogramu. Projekt je rovnoměrně rozložen mezi všechny účastníky a to odpovídá návrhu řešení.

Programování: 70% (kvalita a čitelnost kódu, spolehlivost běhu, obecnost řešení, znovupoužitelnost, …)

V kódu jsou občas části, kde se kód opakuje a bylo by dobré udělat nějaké rozhraní, tak aby se se nemuselo ladit vše pořád dokola, ale vzhledem k tomu, že naše zkušenosti s OpenCL jsou poměrně nové, tak styl kód odpovídá procesu učení dané technologie.

Vzhled vytvořeného řešení: 60% (uvěřitelnost zobrazení, estetická kvalita, vhled GUI, …)

Jedná se o aplikaci s konzolovým a obrazovým výstupem. O nějakém perfektně propracovaném GUI tu nemůže být řeč, ale to nebylo ani cílem tohoto projektu. Program zpracuje z konzole parametry a spustí výpočet a následně vykreslí obraz a vyhodnocení v konzoli.

Využití zdrojů: 65% (využití existujícího kódu a dat, využití literatury, …)

Vycházeli jsme z internetových zdrojů, zejména jsme čerpali ideu sekvenčně implementovaných algoritmů, které jsme převáděli do co největšího možného paralelizmu. Využili jsme předlohu kódu z cvičení GMU

Hospodaření s časem: 80% (rovnoměrné dotažení částí projektu, míra spěchu, chybějící části řešení, …)

Dle našeho názoru s časem nebyl problém. Každý zvládl vše, co měl zadané bez větších potíží.

Spolupráce v týmu: 95% (komunikace, dodržování dohod, vzájemné spolehnutí, rovnoměrnost, …)

Spolupráce byla naprosto skvělá. Problémy nenastaly a vše na čem jsme se domluvili, fungovalo.

Celkový dojem: 85% (pracnost, získané dovednosti, užitečnost, volba zadání, cokoliv, …)