NEUMANN JÁNOS

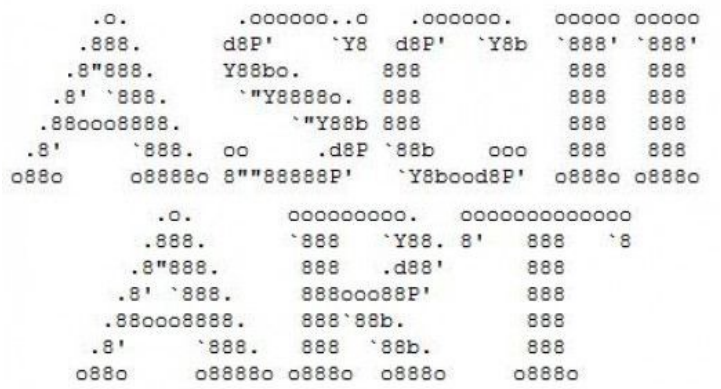
INFORMATIKAI KAR

Szoftverfejlesztés Párhuzamos Architektúrákra

OE-NIK	Hallgató neve:	**Nguyen Van Nam Tamás**
2022	Neptun azonosító:	**AUJZ9U**

Feladat rövid leírása

A féléves beadandónak kitalált feladatomban egy olyan programot fogok megvalósítani, amely egy úgynevezett ASCII-Art stílusú kép generálását fogja véghez vinni. Ehhez szükségem lesz képekre (és/vagy mozgóképekre), mint bemenetek. A képeket elsődlegesen valamilyen szöveges állománnyá szeretném alakítani.



Mi is az ASCII művészet?

Az ASCII művészet alapvetően a művészet szövegalapú formája.

Programokban vagy dokumentumokban használt grafikák vagy szövegek, amelyek ASCII - karakterekből vett karakterekből és szimbólumokból állnak.

Kezdetleges rövid működési elv

A kép állományok átalakítása úgy történik, hogy veszünk egy tetszőleges képet bemenetként és ezt beolvassuk pixelenként és ha sikerült, akkor ezután fekete-fehérré kell ezt alakítanunk valamilyen RGB átalakító képlet alapján. Az új monokróm képünk pixeleit meg kell feleltessük egy stringgel, aminek célja, hogy az abban tárolt karaktereket hozzá tudjuk rendelni egy bizonyos „sötétséghez”.

Figyelni kell arra is, hogy a kép egyes részei sötétebbek, és vannak világosabbak. Pontosan ezt akarjuk szimulálni az ASCII grafikák létrehozásakor, egyes karakterek egy sötét képpontot is képviselhetnek, például egy szóközt vagy egy pontot és némelyik világos képpontot jelenthet, például „@” vagy „W”. Ez a két példa szemléletes, mivel minél sűrűbb a betű, annál fényesebb lesz sötét háttéren. Majd ezeket a sorokat egy fájlba írjuk.

Ennek megmagyarázásához meg kell értenünk, miből áll egy kép. Egy kép, pontosabban a digitális kép pixelekből áll, a kép felbontása alapján. Tehát egy 400 * 600 (magasság\szélesség) felbontású kép 240 000 pixeles. A pixel a digitális kép legkisebb összetevője. Minden pixel tartalmazhat adatokat az adott pixel színéről, a pixelek az őket reprezentáló bitek különböző mennyiségétől, ahol minden bit egy színopciót jelent,

- 1 bpp, 2^1 = 2 szín („monokróm”)
- 2 bpp, 2^2 = 4 szín
- 3 bpp, 2^3 = 8 szín
- 4 bpp, 2^4 = 16 szín
- 8 bpp, 2^8 = 256 szín
- 16 bpp, 2^16 = 65 536 szín („Highcolor”)
- 24 bpp, 2^24 = 16 777 216 szín („Truecolor”)

Szín készítésének módja:

Számos módja van a szín meghatározásának színtér használatával, az egyik legnépszerűbb az sRGB. Az sRGB a „Standard Red Green Blue” rövidítése és egy színtér vagy meghatározott színek halmaza, amelyet a HP és a Microsoft 1996-ban hozott létre azzal a céllal, hogy szabványosítsa az elektronika által ábrázolt színeket.

Más szóval, hogy a piros zöld és kék szín kombinációja hozza létre ezt a színt. Minden érték 0 és 255 között változhat, tehát egy példa a színmegjelenítéshez:

- (R:255, G:0, B:0) = Piros
- (R:0, G:255, B:0) = zöld
- (R: 0, G:0, B:255) = kék
- (R:255, G:255, B:255) = fehér
- (R:0, G:0, B:0) = fekete
- (R:100, G:100), B: 100) = Szürke

Létezik olyan RGBA is, amelynek alfa értéke van, amely meghatározza, hogy az adott szín mekkora átlátszóságú, de ez nem releváns a mi kis projektünknél.

A feladat részletes bemutatása

Ahhoz, hogy egy képet át tudjunk alakítani Ascii képpé, először is meg kell érteni, hogy miként is tudunk képeket átalakítani. Az első lépés mindig, ahhoz, hogy a képeket manipulálhassuk, hogy beolvassuk azokat pixelenként. Az input beolvasást követően minden egyes pixelt szürkeárnyalatosítunk egy előre definiált képlet segítségével.

```
Grayscale = 0.299R + 0.587G + 0.114B
```

Létre kell hoznunk egy olyan stringet, amivel képesek leszünk leírni a kép sötétségét. Így mondhatjuk, hogy ezzel a karakterláncnak az elemeivel megfeleltethetjük a kép kockáinak a sötétség faktorát. Ezt hívjuk majd az úgynevezett „lookup table”-nek.

```
asciiCharsType1 = " .:-=+*#%@";
asciiCharsType2 = "$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,\"^`'. ";
```

Látni lehet, hogy az asciiCharsType2 karaktersor eleje sokkal nagyobb „sűrűséggel” rendelkezik, mint a sor vége felé, így tudjuk meghatározni, hogy bizonyos színekhez milyen karakter tartozzon. Ez jelenleg annyit jelent, hogy ha van mondjuk egy fekete pixel sorunk a képen, akkor az megfeleltethető lesz mondjuk egy „@”-ból álló sornak, mert úgy választottuk azt meg. Ezekből a karakterekből álló sorokat majd egy specifikus fájlba írjuk.

A leírásban szereplő mozgóképet, az idő- és a célhardver hiánya végett, gif-ekkel fogom helyettesíteni. A gif-ek átalakítása Ascii művészetté hasonlókép történik, mint a képeké, viszont van benne egy fontos lépés, ami nélkül az egész nem működne, az előfeldolgozás. Lényegében ez annyit jelent, hogy ezt a mozgóképet felbontjuk különböző képekre, ezeket elmentjük és hasonlóképpen, mint a kép konvertálásánál, Ascii képpé alakítjuk. Az animációt viszont nem tudjuk egy txt fájlba írni, így a végeredményünk másnak kell lennie. Ezért a memóriában lesznek a képek eltárolva. A megjelenítéséhez a konzolt fogjuk használni, ahol ezen képeket egymás után megjelenítjük, így imitálva egy animációt.

Párhuzamosítási kérdések

Mivel a képekből és az animációkból készített Ascii „művek” alapvető része ugyanaz, képek beolvasása és azok elemi részeinek feldolgozása, így ez adja magát, hogy akár 2 különböző fajta párhuzamosítást is alkalmazhatunk. Ahol célszerű ezt végrehajtani az a beolvasása a képeknek, illetve ezen objektumok átalakítása. Ebből adódik adódik, hogy 4 féle képen lehet ezt a párhuzamosítási problémát meghatározni. De mivel a szekvenciális műveletet sort meg lehet valósítani a triviális 2 „for” cikluson kívül egyetlen egy ciklussal is, így ez még kettő különböző megoldást ad nekünk. Tehát lényegében 6 különböző megoldást készítettem el.

Párhuzamosítási megoldások felépítése

	Beolvasás módja	Feldolgozás módja
1	Szekvenciális	Szekvenciális (2 ciklus)
2	Szekvenciális	Szekvenciális (1 ciklus)
3	Szekvenciális	Párhuzamos
4	Párhuzamos	Szekvenciális

5	Párhuzamos	Párhuzamos
6	Párhuzamos	Adat Párhuzamos

A párhuzamos képfeldolgozás alapjául az egy ciklusos szekvenciális feldolgozást választottam és azt készítettem el párhuzamosan. Az adat párhuzamos megoldásnál a különálló szálakon szekvenciálisan hajtom végre a megoldást a különálló elemeken és azokat mentem el.

Mivel a képek feldolgozásánál a szekvenciális algoritmusok lefutása rengeteg időbe telik, így csak a párhuzamos és az adat párhuzamos megoldást fogom összehasonlítani, ellenben a gifekkel, ahol a gifek általában relatív kisfelbontású képeket tartalmaznak.

Megvalósítás

A felhasználónak döntenie kell, hogy képet akar karakterekkel kirajzolni, vagy egy gifet.

Ascii kép készítésének menete

A felhasználónak egy „OpenFileDialog”-on keresztül ki kell választania egy képet, amit Ascii művé szeretne alakítani. Ezután automatikusan elindul a két különböző párhuzamos algoritmus és egy „StopWatch” timer, ami segít mérni a futási időt. Amikor befejeződött a párhuzamos és az adatpárhuzamos algoritmus, kiírjuk az azokhoz tartozó egyéni futási időt a konzolra és az átalakított képet pedig a szövegesfájlba.

Ascii animáció készítésének menete

Ebben az esetben a felhasználónak egy „OpenFileDialog”-on keresztül ki kell választania egy gif fájlt, amit Ascii animációvá szeretne alakítani. Legelőször inicializálunk egy karakter listát/tömböt, amibe a gif képkockái fognak kerülni. Ezután a konzol ablakot a legnagyobbra állítjuk, hogy kiférjen rajta az animáció. Ha ez sikerült, akkor a gifen az előfeldolgozást végrehajtjuk, aminek végeredményeként a gifet szétdaraboljuk különböző jpg kiterjesztésű képekre és a jobb átláthatóság végett kimentjük őket egy mappába. Ezek után meghívjuk az Ascii kép készítésének meneténél használt algoritmusokat. Mindegyik előtt indítunk egy StopWatch-ot, ami segít nekünk a futási időt mérni és a végén kimentjük őket egy változóba, majd reseteljük ezeket. A program futásának a legvégén ezeket az eredményeket a konzolra kiírjuk és egy gombnyomással le is játszhatjuk a kiválasztott Ascii animációt.

Képfeldolgozás

Az egyciklusos képbeolvasás menete során a bemeneti képünket beolvassuk egy „Bitmap” -ba és a „Marshal” -al illetve egy úgynevezett pointerrel átmásoljuk a „képet” egy egydimenziós byte tömbbe. Ebben a tömbben 3-asával kell lépkednünk, mert minden egyes mezőhöz 3 érték tartozik, mégpedig a kék, zöld és a piros. A tömbön végig haladva külön byte típusú változókba ki kell mentenünk a pixeleink színcsatornáit, majd ezek után, úgynevezett „grayscaling”-et (szürkeárnyalatosítást) kell alkalmaznunk a megadott képlet alapján.

```
Grayscale = 0.299R + 0.587G + 0.114B
```

Minden szürkéssé alakított pixelt visszamentünk az eredeti pozíciójának megfelelően egy új stringbe. Egyszerű moduló osztással tudjuk ellenőrizni, hogy a sor végén vagyunk e, és ha 0-t ad vissza, akkor új sort kezdünk.

Szekvenciális 1 ciklusos megoldással az alábbiról van szó:

```
for (int j = 0; j < pixels.Length - 3; j += 3)
{
    byte b = pixels[j];
    byte g = pixels[j + 1];
    byte r = pixels[j + 2];

    double grayScale = (r * 0.299) + (g * 0.587) + (b * 0.114);

    asciiImage += GetSpecificCharacterForEachPixel(grayScale, selectedCharset);

    if ((j + 3) % width == 0)
    {
        asciiImage += "\n";
    }
}
```

Míg párhuzamosan:

```
Parallel.For(0, charPixels.Length, new ParallelOptions()
{
    MaxDegreeOfParallelism = Environment.ProcessorCount
}, i =>
{
    int index = i * step;

    byte b = pixels[index];
    byte g = pixels[index + 1];
    byte r = pixels[index + 2];

    double grayScale = (r * 0.299) + (g * 0.587) + (b * 0.114);

    charPixels[i] = GetSpecificCharacterForEachPixel(grayScale, selectedCharset);
});
```

Adatpárhuzamosság esetében:

```
Task[] tasks = new Task[cpuCount];

for (int i = 0; i < cpuCount; i++)
{
    int j = i;
    byte[] fractionalArray = pixels.Skip(j * fractionalArrayCount).Take(fractionalArrayCount).ToArray();

    Task t = new Task(() => asciiImage[j] = CalcCharacters(fractionalArray, width));
    t.Start();
    tasks[j] = t;
}

Task.WaitAll(tasks);
```

Maga az új sorok beszúrása azért nem látszódik a snipetben, mert az egy másik művelet lesz, amit később kell végrehajtani.

Elért eredmények

A számítógépes rendszer specifikáció, amelyen a program futott:

Processzor	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz 3.19 GHz
Memória mérete	16,0 GB
Videókártya	NVIDIA GeForce GTX 1080
Kiadás	Windows 11 Pro
Verzió	22H2
Operációs rendszer buildszáma	22621.900

Eredmények:

Magyarázat:

- Selected file: Kiválasztott gif
- Selected Ascii characterset: Kiválasztott karakterkészlet
- Algorithm: Algoritmus neve a kódban
    - **SeqAsciiGen_SeqA_OneFor** = Szekvenciális képbeolvasás, szekvenciális képfeldolgozás 1 ciklussal
    - SeqAsciiGen_SeqA_TwoFor = Szekvenciális képbeolvasás, szekvenciális képfeldolgozás 2 ciklussal
    - ParAsciiGen_SeqA = Párhuzamos képbeolvasás, szekvenciális képfeldolgozás
    - SeqAsciiGen_ParA = Szekvenciális képbeolvasás, párhuzamos képfeldolgozás
    - ParAsciiGen _ParA = Párhuzamos képbeolvasás, párhuzamos képfeldolgozás
    - ParAsciiGen _ParDataPar = Párhuzamos képbeolvasás, adatpárhuzamos képfeldolgozás
- Runtime: Futási ideje az algoritmusnak
- Differencial: Kiinduló algoritmushoz (SeqAsciiGen_SeqA_OneFor**)** képest az eltérés
- Acceleration rate: Elért gyorsulás

Eredmények az első gifre:

```
Selected file: gif_acrobatic.gif
Selected Ascii characterset:  .:-=+*#%@
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 1247,0799ms   | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 1262,3458ms   | 15,2659000000001   | 98,79%
ParAsciiGen_SeqA            | 1999,8786ms   | 752,7987           | 62,36%
SeqAsciiGen_ParA            | 102,3058ms    | -1144,7741         | 1218,97%
ParAsciiGen_ParA            | 67,3219ms     | -1179,758          | 1852,41%
ParAsciiGen_ParDataPar      | 320,9445ms    | -926,1354          | 389%
```

```
Selected file: gif_acrobatic.gif
Selected Ascii characterset: $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,"^`'.
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 1232,7895ms   | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 1228,2348ms   | -4,55470000000014  | 100,37%
ParAsciiGen_SeqA            | 2004,1433ms   | 771,3538           | 61,51%
SeqAsciiGen_ParA            | 107,6624ms    | -1125,1271         | 1145,05%
ParAsciiGen_ParA            | 70,5023ms     | -1162,2872         | 1748,58%
ParAsciiGen_ParDataPar      | 333,2099ms    | -899,5796          | 370%
```

Eredmények a második gifre:

```
Selected file: gif_dance.gif
Selected Ascii characterset:  .:-=+*#%@
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 884,9867ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 853,9939ms    | -30,9928           | 103,63%
ParAsciiGen_SeqA            | 1405,7957ms   | 520,809            | 62,95%
SeqAsciiGen_ParA            | 86,9331ms     | -798,0536          | 1018,01%
ParAsciiGen_ParA            | 45,6262ms     | -839,3605          | 1939,65%
ParAsciiGen_ParDataPar      | 221,0212ms    | -663,9655          | 400%
```

```
Selected file: gif_dance.gif
Selected Ascii characterset: $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,"^`'.
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 909,6241ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 894,4297ms    | -15,1944           | 101,7%
ParAsciiGen_SeqA            | 1483,9609ms   | 574,3368           | 61,3%
SeqAsciiGen_ParA            | 68,6285ms     | -840,9956          | 1325,43%
ParAsciiGen_ParA            | 45,4284ms     | -864,1957          | 2002,32%
ParAsciiGen_ParDataPar      | 216,287ms     | -693,3371          | 421%
```

Eredmények a harmadik gifre:

```
Selected file: gif_fight.gif
Selected Ascii characterset:  .:-=+*#%@
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 974,1014ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 943,3772ms    | -30,7242           | 103,26%
ParAsciiGen_SeqA            | 1516,6799ms   | 542,5785           | 64,23%
SeqAsciiGen_ParA            | 86,8382ms     | -887,2632          | 1121,74%
ParAsciiGen_ParA            | 65,2357ms     | -908,8657          | 1493,2%
ParAsciiGen_ParDataPar      | 241,9715ms    | -732,1299          | 403%
```

```
Selected file: gif_fight.gif
Selected Ascii characterset: $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,"^`'.
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 1062,9964ms   | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 1103,1662ms   | 40,1698000000001   | 96,36%
ParAsciiGen_SeqA            | 1634,7485ms   | 571,7521           | 65,03%
SeqAsciiGen_ParA            | 81,5243ms     | -981,4721          | 1303,9%
ParAsciiGen_ParA            | 63,6105ms     | -999,3859          | 1671,1%
ParAsciiGen_ParDataPar      | 249,8572ms    | -813,1392          | 425%
```

Eredmények a negyedik gifre:

```
Selected file: gif_giant.gif
Selected Ascii characterset:  .:-=+*#%@
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 849,9709ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 820,5632ms    | -29,4077           | 103,58%
ParAsciiGen_SeqA            | 1303,0856ms   | 453,1147           | 65,23%
SeqAsciiGen_ParA            | 69,338ms      | -780,6329          | 1225,84%
ParAsciiGen_ParA            | 66,7833ms     | -783,1876          | 1272,73%
ParAsciiGen_ParDataPar      | 214,46ms      | -635,5109          | 396%
```

```
Selected file: gif_giant.gif
Selected Ascii characterset: $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,"^`'.
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 789,3588ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 782,7597ms    | -6,59910000000002  | 100,84%
ParAsciiGen_SeqA            | 1322,6397ms   | 533,2809           | 59,68%
SeqAsciiGen_ParA            | 66,7138ms     | -722,645           | 1183,2%
ParAsciiGen_ParA            | 56,3175ms     | -733,0413          | 1401,62%
ParAsciiGen_ParDataPar      | 192,0133ms    | -597,3455          | 411%
```

Eredmények az ötödik gifre:

```
Selected file: gif_worldcup.gif
Selected Ascii characterset:  .:-=+*#%@
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 213,1679ms    | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 216,0886ms    | 2,92070000000001   | 98,65%
ParAsciiGen_SeqA            | 383,5103ms    | 170,3424           | 55,58%
SeqAsciiGen_ParA            | 31,3931ms     | -181,7748          | 679,03%
ParAsciiGen_ParA            | 54,4796ms     | -158,6883          | 391,28%
ParAsciiGen_ParDataPar      | 71,3442ms     | -141,8237          | 299%
```

```
Selected file: gif_worldcup.gif
Selected Ascii characterset: $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/|()1{}[]?-_+~<>i!lI;:,"^`'.
Algorithm                   | Runtime       | Differencial       | Acceleration rate
SeqAsciiGen_SeqA_OneFor     | 215,072ms     | 0                  | 100%
SeqAsciiGen_SeqA_TwoFor     | 228,882ms     | 13,81              | 93,97%
ParAsciiGen_SeqA            | 376,8879ms    | 161,8159           | 57,07%
SeqAsciiGen_ParA            | 29,0431ms     | -186,0289          | 740,53%
ParAsciiGen_ParA            | 39,4964ms     | -175,5756          | 544,54%
ParAsciiGen_ParDataPar      | 67,5671ms     | -147,5049          | 318%
```

A megjelenített eredményekből egyértelmű, hogy a ParAsciiGen_ParA (Párhuzamos képbeolvasás, párhuzamos képfeldolgozás) teljesít szinte mindig a legjobban. Az kiindulóállapotnak választott 1 ciklusos megoldáshoz képest ez átlagosan 1431,743%-os gyorsítást, ami nagyjából 14szeres gyorsítást jelent.

Fontosnak tartom megjegyezni, hogy ha jobban megfigyeljük a kapott eredményeket, akkor észre lehet venni, hogy a dobogó második helyén nem a teljesen párhuzamos megoldások közül szerepel ismét egy, hanem egy hibrid megoldás, mégpedig a SeqAsciiGen_ParA (Szekvenciális képbeolvasás, párhuzamos képfeldolgozás).

Továbbfejlesztési lehetőségek

Természetesen lehetne még gyorsítani a programon, mégpedig úgy, ha:

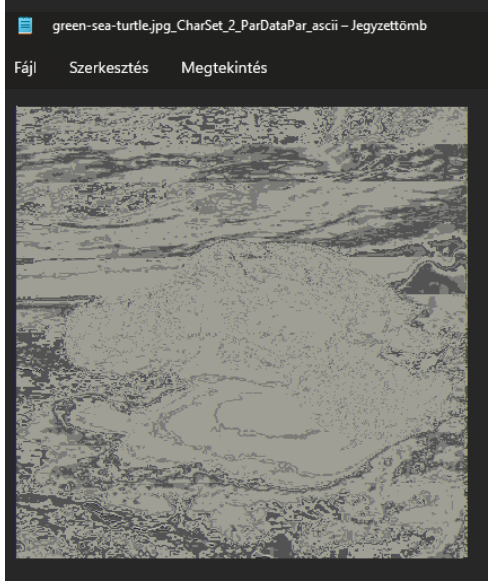
- az alkalmazás előfeldolgozását is párhuzamosítanánk.
- adatpárhuzamosítás során a szálak munkája, amint a fractionalArray-eken végez nem szekvenciális lenne, hanem párhuzamos.
- GPU párhuzamos programozással megvalósítani
- iOS-re lefejleszteni (következő projektem)

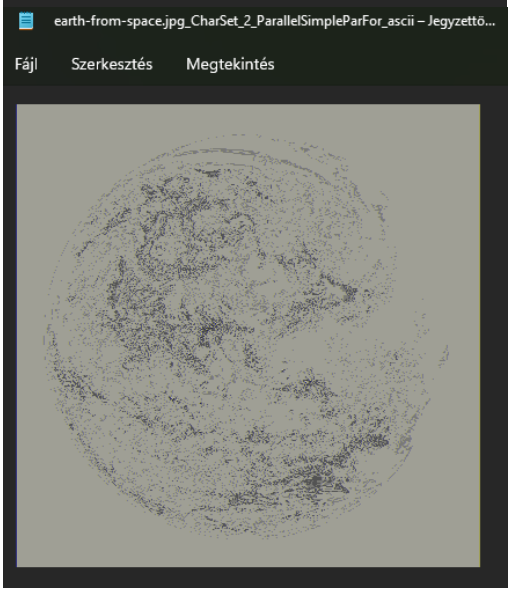
Megjegyzés

- Különböző színek – kontrasztarányokhoz jobban működik az egyik illetve a másik lookup table / karakterkészlet
- A terminált / parancssort állítsuk be maximum méretre.
- A terminál / parancssor betűmérete legyen 5-ös, hogy kiférjen az animáció.
- Nagyméretű gifek-et a gépemmel nem tudtam tesztelni, mert rengeteg idő volt a feldolgozás.
    - Van egy hibaüzenet ami felugrik, annak a jobbfelső sarkára csak nyomjunk rá
        - Lényegében nem is hibaüzenet, csak annyit jelez a VS, hogy a futása az alkalmazásnak sokáig tart.

Képek a végeredményről

Konvertált képek:

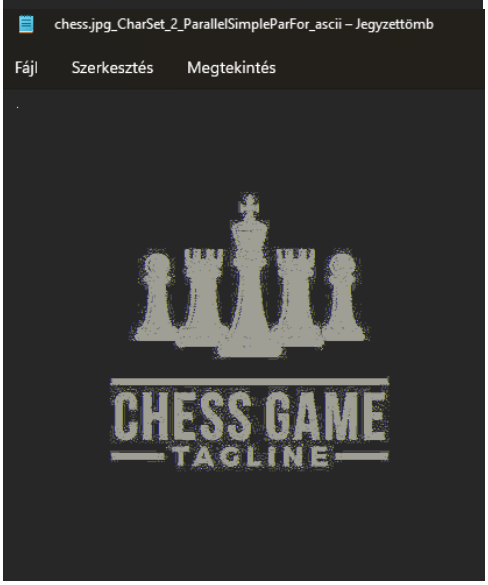
Eredeti fájl neve	Lookup-table	Kép
green-sea-turtle.jpg	-	
green-sea-turtle.jpg	AsciiCharSetType1	
green-sea-turtle.jpg	AsciiCharSetType2	

earth-from-space.jpg	-	
earth-from-space.jpg	AsciiCharSetType1	
earth-from-space.jpg	AsciiCharSetType2	

chess.jpg -

chess.jpg AsciiCharSetType1

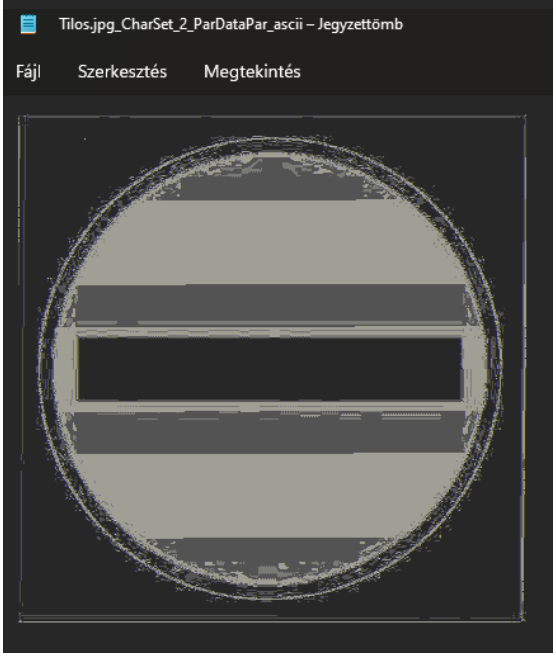
chess.jpg AsciiCharSetType2



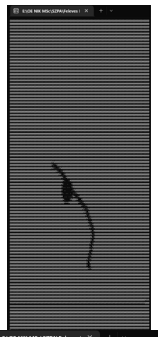
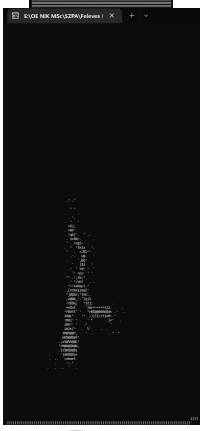
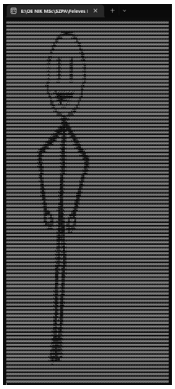
Tilos.jpg -

Tilos.jpg AsciiCharSetType1

Tilos.jpg AsciiCharSetType2



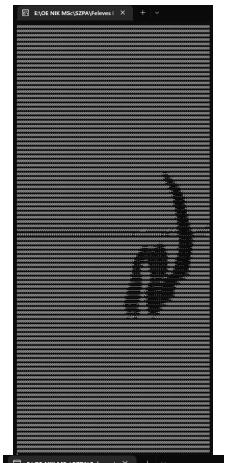
Konvertált animációk:

Eredeti név	Lookup-table	Animáció
gif_acrobatic.gif	-	
gif_acrobatic.gif	AsciiCharSetType1	
gif_acrobatic.gif	AsciiCharSetType2	
gif_dance.gif		
gif_dance.gif	AsciiCharSetType1	

gif_dance.gif AsciiCharSetType2

gif_fight.gif

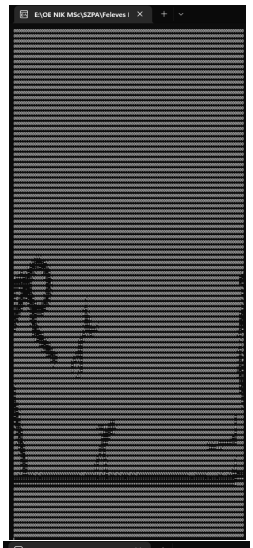
gif_fight.gif AsciiCharSetType1



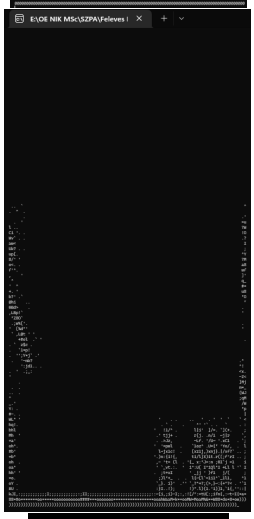
gif_fight.gif AsciiCharSetType2

gif_giant.gif

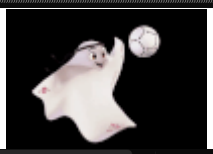
gif_giant.gif AsciiCharSetType1



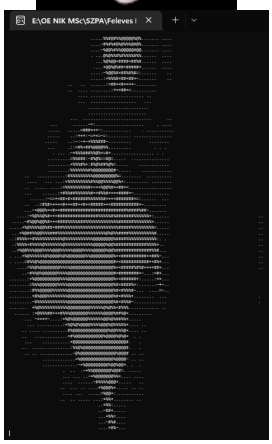
gif_giant.gif AsciiCharSetType2



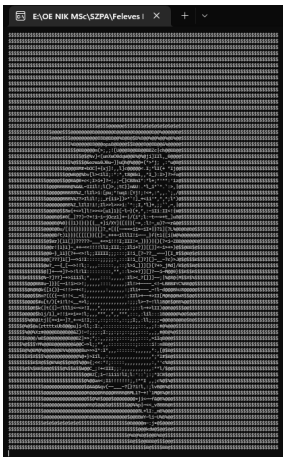
gif_worldcup.gif



gif_worldcup.gif AsciiCharSetType1



gif_worldcup.gif AsciiCharSetType2



Forrás

http://szfi.nik.uni-obuda.hu/elearning/

https://en.wikipedia.org/wiki/Grayscale

https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-space-conversion/

https://hu.wikipedia.org/wiki/Graphics_Interchange_Format