Eötvös Loránd Tudományegyetem

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Többváltozós Lagrange interpolációs polinom előállítása és ábrázolása párhuzamos programmal.

Dr. Hajder Levente

docens

Lonkai Kristóf Dániel

Programtervező Informatikus BSc.

Budapest, 2018

Tartalomjegyzék

1 Bevezetés

A szakdolgozatom témája egy olyan alkalmazás elkészítése, ami segítségével képeket tudunk nagyítani, úgy, hogy szín átmenetet próbálunk rekonstruálni az adott képből. Sok esetben, ha képeket szeretnénk nagyítani nem áll rendelkezésünkre olyan eszköz, amely a pixelek nagyítása helyett, átmenetet képezne, illetve a legtöbb erre képes program komplex képszerkesztő és áruk igen magas is lehet. Az eljárás folyamán a nagyítani kívánandó képből, egy kétváltozós polinomot készítünk el, így közelítjük az eredeti tartalmat. A kapott függvényből ezután egyszerűen lehet a képpontok közötti értékeket számolni és tetszőleges méretű képet generálni. A feladat elsőre igazán bonyolultnak tűnt, de ez csak még több motivációt adott. A problémára és a megoldásra a Numerikus Módszerek 2. előadás adott ihletet. Akkor körvonalazódott meg bennem egy ötlet, hogy interpolációval akár képeket is tudunk közelíteni, nem csak függvényeket. A kezdetleges megoldást a képben minden sort egy egyváltozós függvényként fogok fel, ezt interpolálom és számolok belőle egy nyújtott képet majd az így kapott képet ugyanígy interpolálom, de nem sorok, hanem oszlopai szerint. Ezt megint megnyújtom és kapok egy új, nagyított képet. Ahogy haladtunk a tananyaggal egyre több interpolációs eljárást ismertünk megami arra ösztönzött, hogy az interneten is rákeressek. Ekkor találtam rá egy cikkre, amelyben többváltozós interpolációról írtak. Ekkor döntöttem, hogy ez lesz a szakdolgozatom témája. A programhoz használt technikák is igazán felkeltették az érdeklődésemet. A jövőben az a célom, hogy elosztott programokat írjak, így pont kapóra jött, hogy az eljárást lehet párhuzamosítani, persze ez egy kicsit másképp végződött, de összeségében nagyon élveztem az alkalmazásom fejlesztésének összes fázisát. Végezetül úgy gondolom az alkalmazás egy hasznos kis program lett, amely, habár nem mindenható, tényleg fel tudjuk használni, akár, hogy alapokat szolgáljon a program, akár kifejezetten alkalmazni szeretnénk valahol.

2 Felhasználói dokumentáció

2.1 Feladat

Egy olyan alkalmazás készítése, amelynek segítségével egyszerűen generálhatunk képekből képeket. A felhasználó a programmal egyszerűen egy grafikus felületen keresztül kommunikálhat, amely figyel a beviteli adatok helyességére. Az alkalmazás a programnak egy, a felhasználó által megadott fekete-fehér képből, és egy egész számból kell egy olyan kétváltozós polinomot számolni, melynek a fokszáma legfeljebb az előbb említett szám. Majd az így kapott függvénybe behelyettesítve megkapjuk a nagyított képet, amit később lementünk.

2.2 Módszerek

Az alkalmazás a feladat megoldására több módszert ötvöz. A függvényt többváltozós interpoláció segítségével számoljuk ki, és ezt egy kétváltozós függvényt reprezentáló osztályba csomagoljuk. Ez a folyamat a felépítéséből adódóan párhuzamosítható. A párhuzamosítást nem a CPU végzi, hanem a GPU nagyobb erőforrásait használjuk ki. A felhasználói felület egyrétegű objektumorientált szemléleten alapszik, amely a számítást közvetlen a számító modulból importáljuk. A program Windows alatt .NET keretrendszert és NVIDIA CUDA-t támogató rendszereken fut. Az alkalmazás könnyen telepíthető a specifikációnak megfelelő számítógépeken.

2.3 Rendszerkövetelmények

Minimális rendszerkövetelmény.

- NVIDIA CUDA 9.0-t támogató grafikus kártya
- 500 MB RAM
- 400 MB szabad hely
- Dual Core 1 GHz órajelű x64 architektúrájú processzor
- Windows 7/8/8.1/10

Optimális rendszerkövetelmény

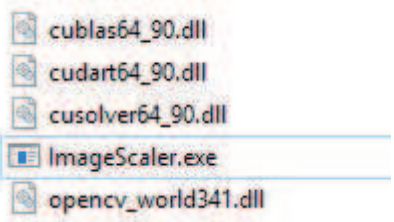
- NVIDIA CUDA 9.0-t támogató grafikus kártya
- 2 GB RAM
- 2 GB szabad hely
- Dual Core 2.4 GHz órajelű x64 architektúrájú processzor

- Windows 7/8/8.1/10

2.4 Használati útmutató

2.4.1 Telepítés

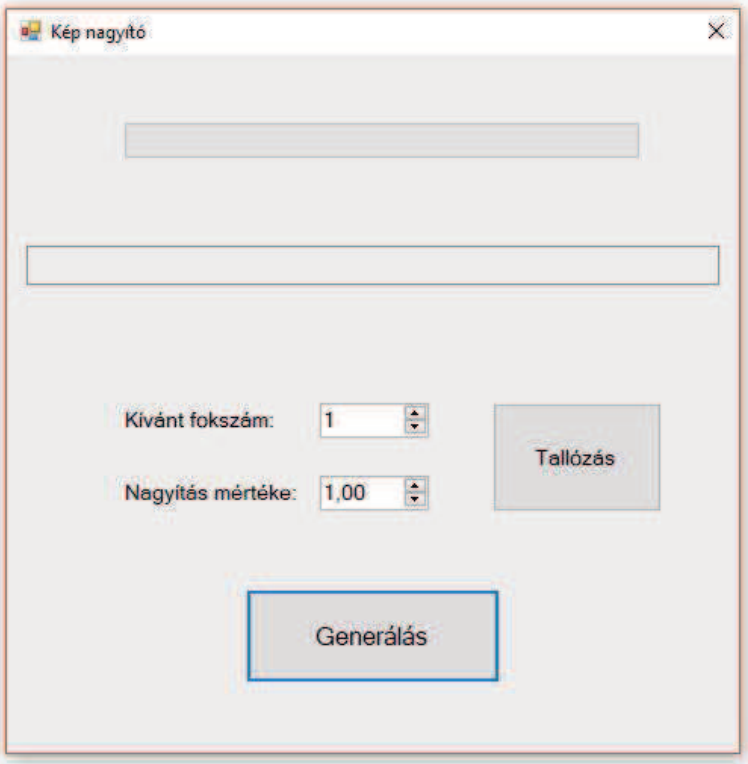
Az alkalmazás telepítése egyszerű. Elhelyezzük a fájlokat egy általuk kiválasztott mappába.



1. ábra: Szükséges fájlok

2.4.2 Használat

Az alkalmazás elindítása után megjelenik a felhasználói felület. Ezen megtalálható a programot vezérlő összes elem.

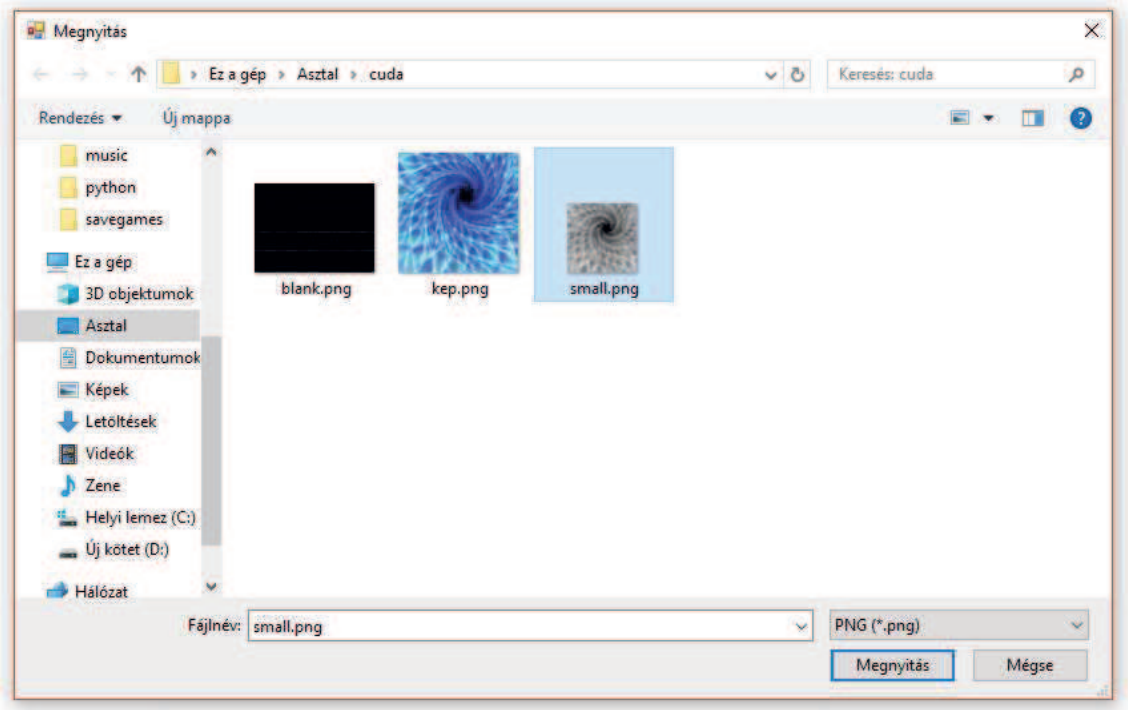


2. ábra: Felhasználói felület. Ez biztosítja a felhasználói interakciók egyszerűségét és helyességét

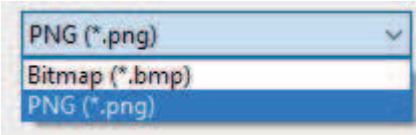
A felhasználó négy beállítást tud konfigurálni és két információs vezérlő segíti a tájékozódásban

2.4.2.1 Vezérlők

- „Kívánt fokszám”
    - A vezérlő a generált polinom maximális fokszámáért felel. Ez egy 1 és 100 közötti egész szám lehet.
- „Nagyítás mértéke”
    - A vezérlő a generált kép mérete és az eredeti kép méretének aránya. Ennek az értéke 1 és 100 közötti kéttizedes jegyre kerekített racionális szám.
- „Tallózás”
    - A gombra kattintva egy párbeszéd ablak nyílik meg. (3. ábra) Ebben az ablakban a felhasználó böngészhet a gépen levő fájlokból. Alapértelmezetten bitképfájlokat tudunk megnyitni, (.bmp) de .PNG kiterjesztésű képeket is ki tudunk választani. (4. ábra)
    - Az ablak ellenőrzi a fájl meglétét, illetve a „OK” gomb megnyomásával az alkalmazás egyéb validációkat is elvégez. Az esetleges hibákat felugró ablak formájában jelzi a felhasználónak. (6. ábra)

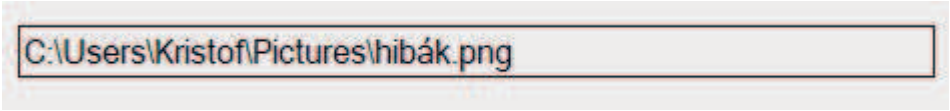


3. ábra Tallózás a fájlok között.



4. ábra Formátum választó

- Elérési útvonal
  - A vezérlő alapértelmezetten üres, (2. ábra) de ha érvényes fájlt választottunk ki, akkor a mezőjébe belekerül annak a teljes elérési útvonala. (5. ábra)

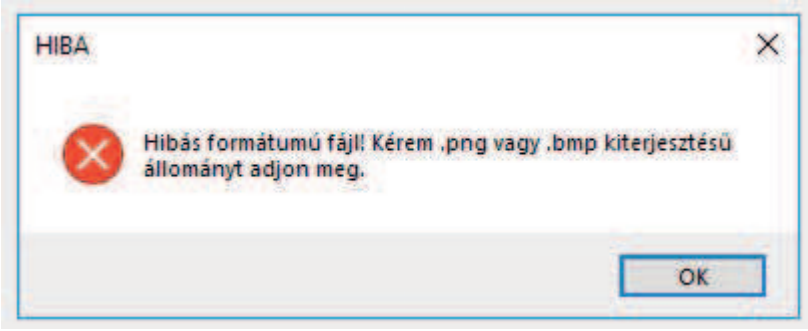


5. ábra Teljes elérési út

  - A tartalma kijelölhető, másolható, de kézzel nem módosítható.
- Töltősáv
  - Ez az elem alapértelmezetten üres, de a program futása során változik. Így a felhasználó egy közelítő érzést kap az eljárás menetének állapotáról.
  - Ha a képgenerálás befejeződött, akkor amíg nem kezdünk új kép alkotásba, addig teljesen betöltve marad, majd kiürül.
- „Generálás”
  - A gomb felel a polinom és kép generálásért. Megnyomásakor ellenőrződik a beviteli adatok helyessége. Elindulás után a gomb eltűnik mind addig amíg a vezérlés vissza nem tér, nem látszik.

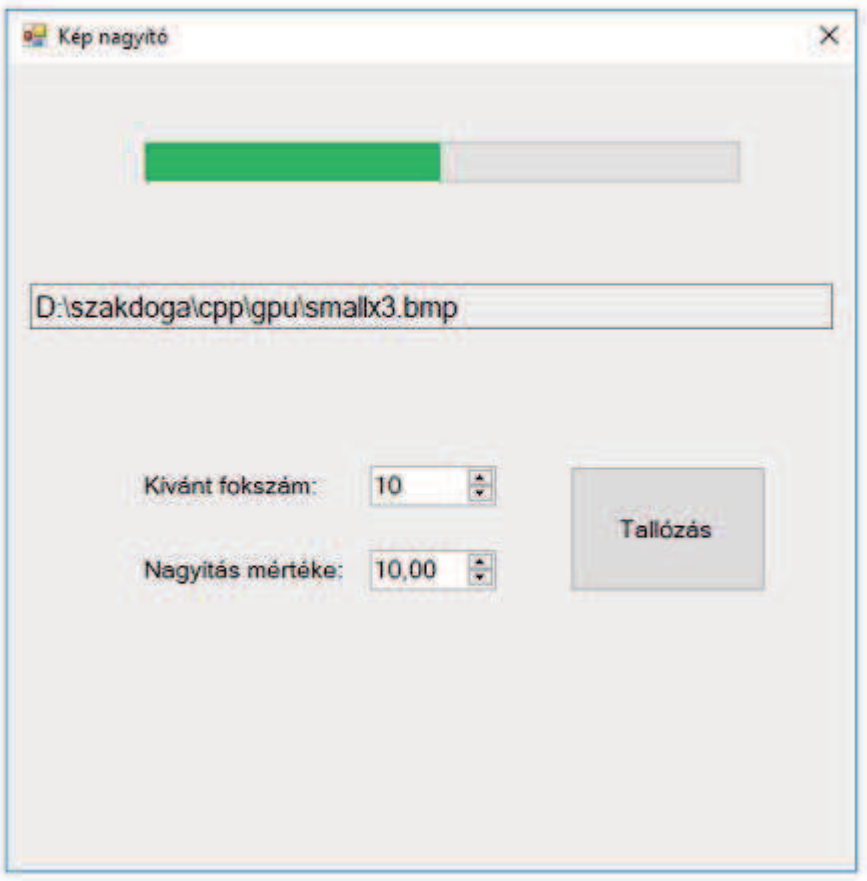
2.4.2.2 Működés

Az alkalmazás az elindítása után a felhasználónak minden funkcióra lehetősége van. A beviteli adatok kitöltése után a „Generálás” gomb megnyomásának a hatására elkezdődik a képalkotási folyamat. Ez a tevékenység időtartalomban a bemenő paraméterektől függ.

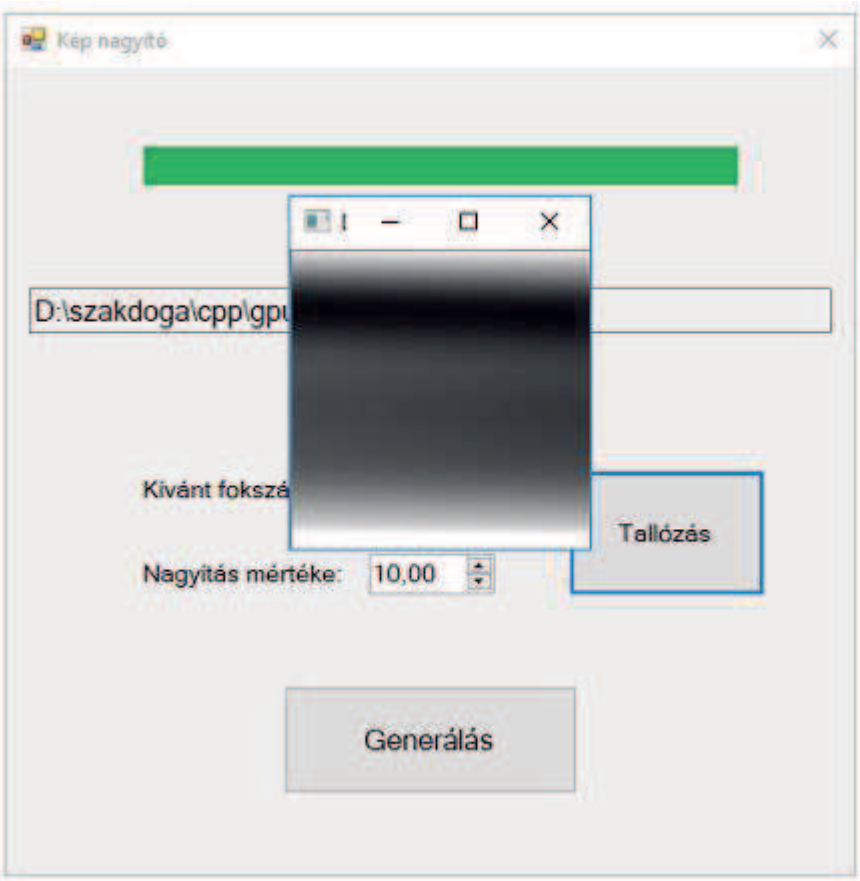


6. ábra Rossz kiterjesztésű fájl

A futásidő függ a bemeneti kép méretétől, generálandó polinom sokszámától és az új kép méretéről, azaz a nagyítás nagyságától. A felhasználó a nagyítani kívánandó képet a „Tallózás” feliratú gomban tudja kiválasztani. A gomb megnyomása után felugrik egy fájlmegnyitó párbeszédablak. Ezen a felületen a felhasználónak lehetősége nyílik arra, hogy a gép fájljai között böngészve ki tudjon választani egy képet, amelynek a kiterjesztése .png-nek illetve .bmp-nek kell lennie. Különben az alkalmazás nem veszi figyelembe a kérést és hibaüzenetet ad. (6. ábra) Ha nem történt hiba, akkor az alkalmazás elkezdi beolvasni a képet és összeállítani az elkészítendő függvényt. Ezt azzal láthatjuk, hogy a „Generálás” gomb láthatatlanná válik. A folyamat során a töltőcsík elkezd nőni. Ezek a folyamat egy-egy fázisát jelzik. Miután végzett az eljárás a vezérlést visszakapjuk és újra lehet használni a „Generálás” gombot. Az alkalmazás az elkészült képet kimenti a futtatható állomány mellé „result.png” névvel. Az alábbi képsorok mutatják a működés közbeni programot.



7. ábra A szoftver képernyője futás közben



8. ábra Az elkészült kép megjelenik a képernyőn is, de lementődik a merevlemezre

9. ábra Nagyítás Paint-tel

10. ábra Nagyítás a programmal

2.4.2.3 Speciális indítási lehetőségek

A program lehetőséget nyújt az interpolációs módszer implementálásának megváltoztatása. Parancssori bemenetként, ha „-qr” akkor az alkalmazás egy másik módszert használ, amely lassabb, numerikusan stabilabb módszer. Ennek módja egy parancsikon létrehozása és ott a paraméter megadása a tulajdonságok menüpontnál a cél mező végére. Másik módszer, hogy parancssorral indítjuk az alkalmazást. Ezt a program nevének beírásával és opcionálisan „-qr” kapcsolóval látjuk el. Ez mindenféleképp első paraméternek kell lennie, különben nem veszi figyelembe. Ha parancssorból indítjuk az alkalmazás, akkor is a felhasználói felületet kapjuk meg.

2.5 Határok és célközönség

A program a módszer tulajdonságai és a hardware korlátjai miatt, nem képes teljes rekonstruálásra. Olyan felhasználóknak ajánlom a program használatát, akik viszonylag kicsi képeket szeretnének egyszerűen nagyítani. A legtöbb egyszerű szín átmenettel készült kép kis hibával nagyítható, így ezeknek a textúráknak kicsinyített változatait helytakarékosan tudjuk tárolni. A nagyobb képeket, illetve a azokat a képeket, amelyek bonyolultak (például: emberi arc, panellakások este) nem tudunk rekonstruálni. A tesztelés folyamán, viszont kiderült, hogy a legoptimálisabb felhasználás az apró, átmenetes kép, nagyítása. (15-20 szoros)

3 Fejlesztői dokumentáció

3.1 Követelményelemzés

3.1.1 Feladat

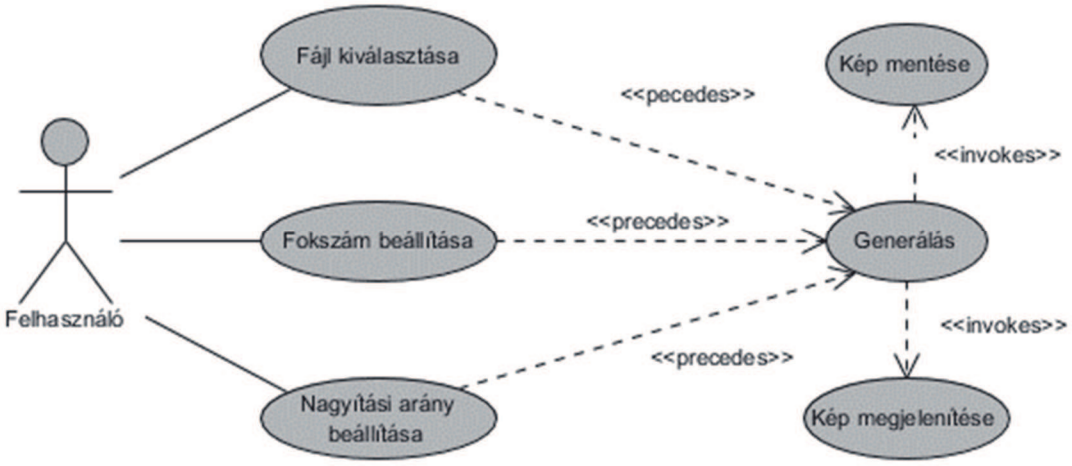
A szakdolgozó feladata interpolációs polinomot előállító program párhuzamos implementációja GPU segítségével. A programnak működése során egy kétváltozós függvényt kell előállítani, amelyet a Lagrange interpoláció többváltozós alakjából számol. Mintavételezéshez egy tetszőleges szürkeárnyalatos kép adhat alapot, amelyről az egyes pixelek sor- és oszlopindexei határozzák meg az interpolációs függvény alappontjainak értékét, a hozzárendelt szint pedig az alappontokban felvett függvényértékből származik. A program a kiszámolt függvény segítségével a mintának használt kép egy tetszőleges korlátos halmazának egy nagyított változatát állítsa elő úgy, hogy a nagyított képről megmondhatjuk, hogy hányszoros nagyításban szeretnék látni. A pixelek közötti értéket az interpolációs függvényből mintavételezéssel kell számolni.

3.1.2 Felhasználói történetek

	esemény	**állapot**	**mikor**	**akkor**
1	alkalmazás indítása	nem fut az alkalmazás	felhasználó futtatja a programot	a program elindul és megjelenik a felhasználói felület
2	kép kiválasztásának szándéka	az alkalmazás elindítva	a felhasználó rákattint a „Tallózás” gombra	megjelenik egy párbeszédablak amint kiválaszthatjuk a fájlt
3	hibás képválasztás	képválasztó ablak aktív	a felhasználó hibás inputot ad meg	a hiba jelződik a felhasználó számára

4	helyes képválasztás	képválasztó ablak aktív	a felhasználó helyes inputot ad meg	a program bejegyzi az inputot
5	helyes fokszám választás	a felhasználói felület fut csak	felhasználó a programba bejegyzi az inputot, beírja a kívánt fokszámot	a program bejegyzi az inputot
6	helyes nagyítás választás	a felhasználói felület fut csak	felhasználó a programba bejegyzi az inputot, beírja a kívánt nagyítást	a program bejegyzi az inputot
7	a generálási folyamat elkezdése	az inputok helyesen be vannak állítva	a felhasználó rákattint a „Generálás” gombra	A program előállítja az új képet
8	a generálási folyamat elkezdése	az inputok helytelenül be vannak állítva	a felhasználó rákattint a „Generálás” gombra	A program szól a hibáról.
9	helyes fokszám választás	a felhasználói felület fut csak	felhasználó a programba bejegyzi az inputot, beírja a kívánt fokszámot	A program szól a hibáról.
10	helyes nagyítás választás	a felhasználói felület fut csak	felhasználó a programba bejegyzi az inputot, beírja a kívánt nagyítást	A program szól a hibáról.

3.1.3 Felhasználói esetek



11. ábra Felhasználói esetek diagramja

3.2 Tervezés

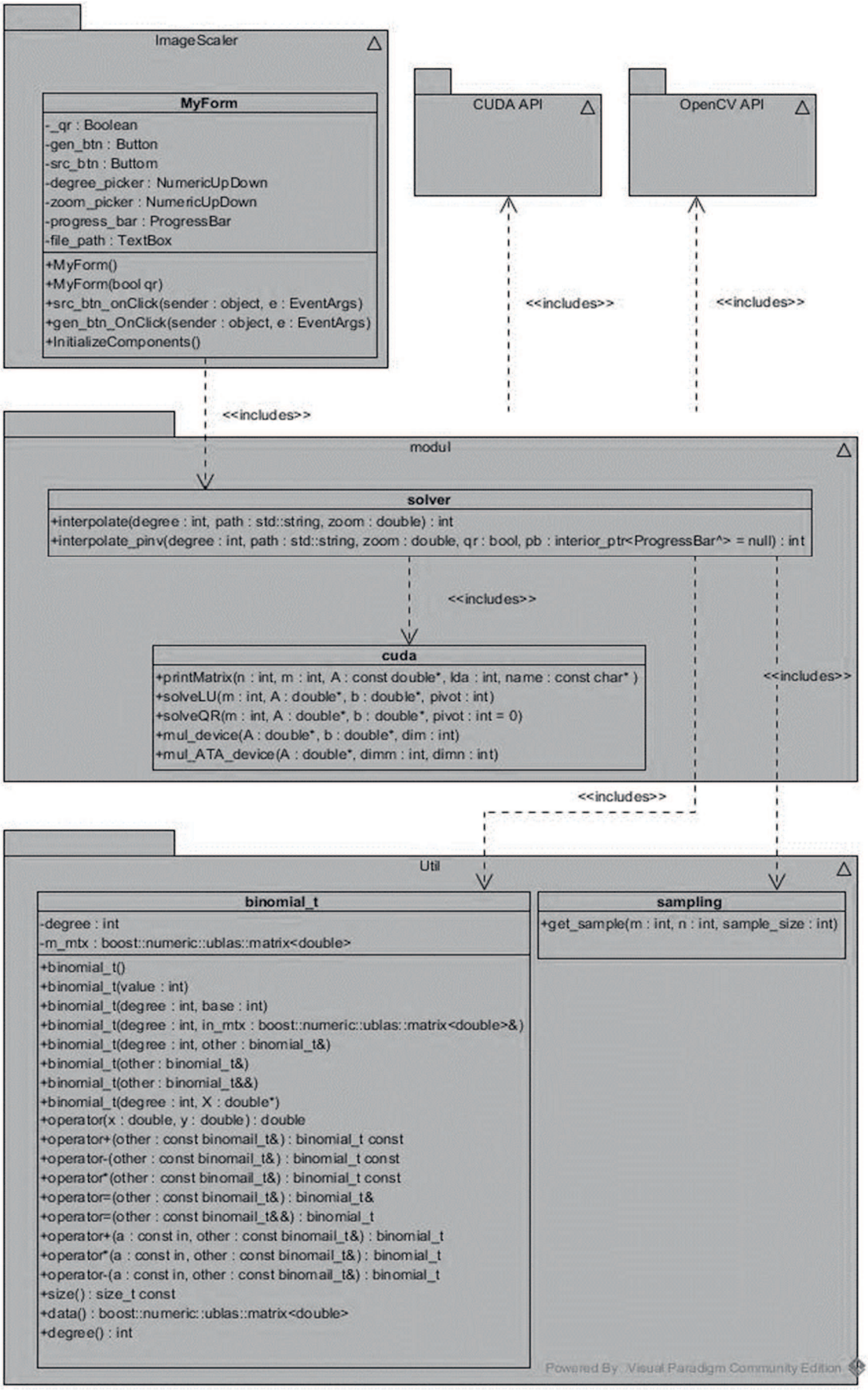
Az alkalmazást Windows alapú rendszerekhez és NVIDIA CUDA technológiát támogató videókártyákhoz írjuk. Amely az egyszerű telepítést és a helyigényt szolgálják.

3.2.1 Software architektúra

Az alkalmazás egyszerű egyrétegű programként valósítjuk meg. A megvalósításhoz több programozási paradigmát is felhasználunk. A program magja egy procedurális modul, amely kommunikál a grafikus kártyával és amely magában hordozza a feladat matematikai problémájának megoldását, megoldásait. Erre a modulra illesztünk egy grafikus felhasználói interfészt, amelyet objektum orientált szemléletben valósítunk meg. A rendszer jól működésént tesztelés és gondosan kiválasztott bevált könyvtárak biztosítják.

A program ezáltal két fő részből fog állni. Egyik része a megjelenítésért és a felhasználói interakciókért felel. (későbbiekben ezt a felhasználói felületnek nevezzük) A másik a matematikai problémát oldja meg. (későbbiekben ezt a modulnak nevezzük) A programot c++ nyelven írjuk és több külső könyvtárat is segítségül hívunk

A program szerkezetét osztálydiagrammokkal reprezentálhatjuk.



12. ábra A program osztálydiagramja

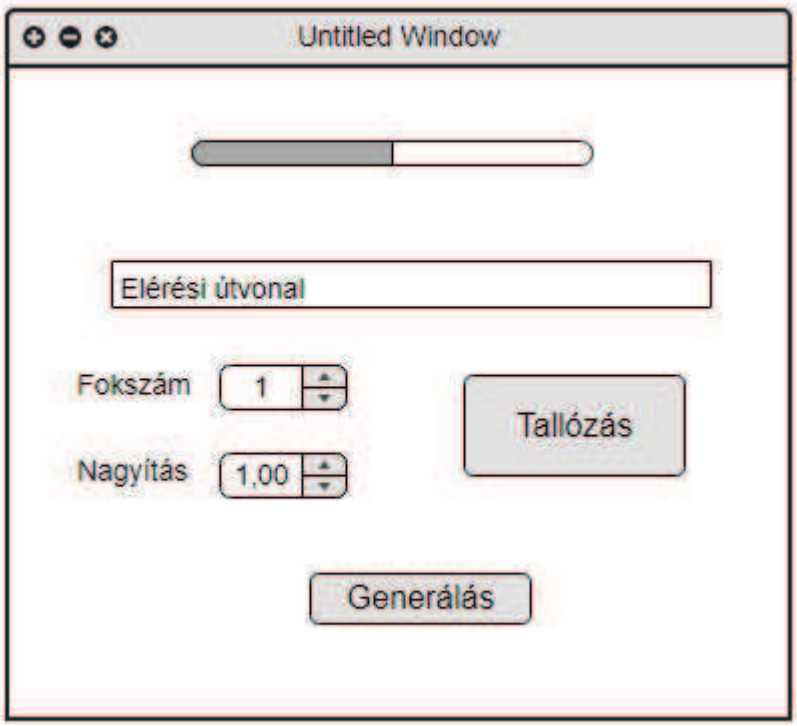
3.2.2 Felhasználói felület

A felhasználói felületet egy egyszerű ablak reprezentálja. Ezen az ablakon elhelyezzük a vezérlő és jelző elemeket, amelyeket a felhasználót segíti. A felhasználói felület motorja a .NET keretrendszer, amihez a hozzáférést a Visual C++ fordítója (cl.exe) ad interfészt.

A továbbiakban részletezzük:

- A felhasználói felület vizuális tervét
- A felhasználói felület statikus tervét
    - Osztálydiagrammal
    - Részletes leírással

3.2.2.1 A felhasználói felület vizuális terve:

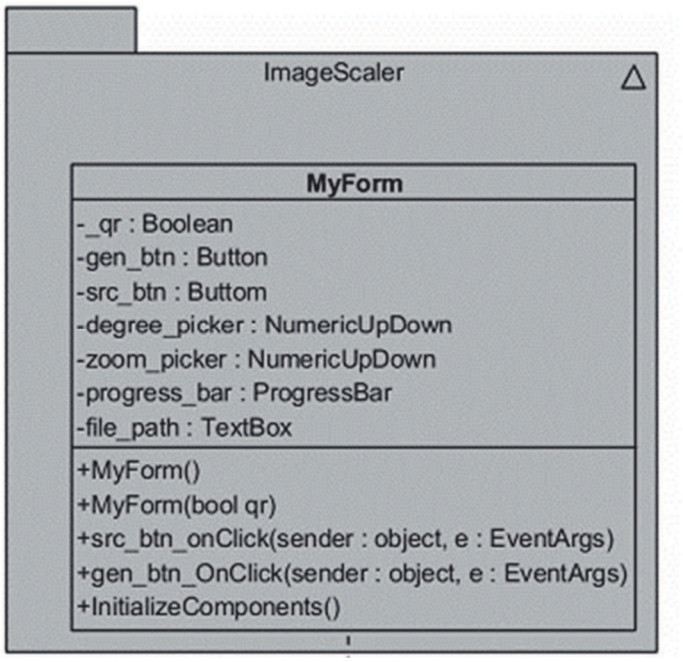


13. ábra Felhasználói felület terve

A felhasználói felületet egy egyszerű kompakt, letisztult felületként képzeljük el, amin csak a működéshez elengedhetetlen adatokat, elemeket jelenítjük meg. A mérete fix és nem igényel semmilyen hardware-es gyorsítást

3.2.2.2 A felhasználói felület statikus terve

A felhasználói felületet egy *MyForm* nevű osztály valósítja meg (13. ábra). Ez az osztály a NET keretrendszerre épül, mivel leszármazik a *System::Windows::Forms::Form* osztályból amely megvalósítja egy ablak kirajzolását, kezelését. Ebbe az osztályba privát adattagként vesszük fel a felületen megjelenő elemeket, vezérlőket.



14. ábra A felhasználói felület osztálydiagramja

MyForm:

- **Attribútumok**
  - gen_btn
    - típus: System::Windows::Forms::Button^
    - leírás. Egy szemétgyűjtővel kezelt gomb elem
    - feladat: A számító folyamat elindítása
  - src_btn
    - típus: System::Windows::Forms::Button^
    - leírás: Egy szemétgyűjtővel kezelt gomb elem
    - feladat: A fájlkiválasztó folyamat elindítása
  - file_path
    - típus: System::Windows::Forms::TextBox^

        - leírás: Egy szemétgyűjtővel kezelt szöveges beviteli mező elem, amelyet kézzel nem módosíthatunk
        - feladat: A fájl elérési útjának megjelenítése és tárolása
    - degree_picker
        - típus: System::Windows::Forms::NumericUpDown^
        - leírás: Egy szemétgyűjtővel kezelt numerikus adat bevitelére szolgáló elem
        - feladat: A generálandó függvény legnagyobb fokszámának megjelenítése és tárolása
    - zoom_picker
        - típus: System::Windows::Forms::NumericUpDown^
        - leírás: Egy szemétgyűjtővel kezelt numerikus adat bevitelére szolgáló elem
        - feladat: A generálandó függvény nagyításának arányának megjelenítése és tárolása

    - progress_bar
        - típus: System::Windows::Forms::ProgresBar^
        - leírás: Egy szemétgyűjtővel rendelkező vizális megjelenítő
        - feladat: A számító folyamat állapotát reprezentáló objektum

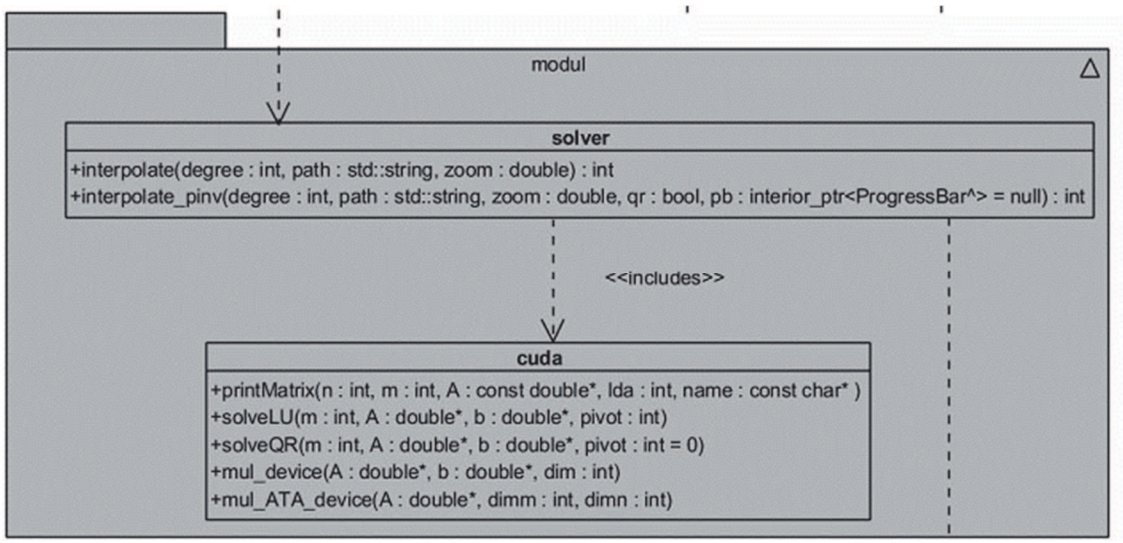
- **Konstruktorok**
    - MyForm**()**
        - leírás: defaut konstruktor
        - tevékenység: elindítja az inicializáló folyamatot
    - MyForm**(**bool)
        - leírás: paraméretes konstruktor
        - tevékenység: elindítja az inicializáló folyamatot és bejegyzi, hogy akarunk-e más implementációt használni
- **Metódusok**
    - InitializeComponents()
        - Leírás: Inicializálja a komponenseket
            - Szemétgyűjtőt rendel a .NET osztályokhoz
            - beállítja a komponens osztályok attribútumait

- src_btn_OnClick(object,EventArgs)
  - Leírás: dinamikusan létrehoz egy párbeszédablakot és elindítja azt, majd validálja a formátumot.
- gen_btn_OnClick(object,EventArgs)
  - Leírás: dinamikusan létre hoz egy párbeszédablakot és elindítja azt, majd validálja a formátumot.

3.2.3 A Modul

A modul fő feladata, hogy a problémára való megoldást enkapszulálja. (részletesebben a megvalósítás pont alatt) A modulnak tartalmaznia kell egy olyan megvalósítást, amely megfelel a feladat specifikációjának. A modult procedurális paradigmának megfelelően írjuk meg. A c++ nyelvi elemeit kihasználva az építőelemeket névterekbe fogjuk becsomagolni. A modul fő alkotóelemeit külön csomagokba helyezzük. A modul tartalmazni fog egy megoldó csomagot és egy CUDA API-t használó csomagot. Ezen felül egy kétváltozós polinomokat megvalósító osztályt is implementálunk, amely segítségével könnyen végezhetünk műveleteket a későbbiekben.

3.2.3.1 A modul statikus terve



15. ábra A modul csomag felépítése

A modul csomag két alnévtérrel rendelkezik, ezek rendre egy-egy feladatrészt írnak le. Az alnévtereket ezen felül a későbbi implementációs érdek miatt is külön választjuk, funkcionálisan a *solver* névtér OpenCV API-t használ fel, míg a *cuda* névtérbe tartozó

függvények a CUDA API-t veszik igénybe és ebből kifolyólag a GPU közeli a kód kerül bele.

Megoldandó részfeladatok specifikációja:

- interpoláció

$A = \underset{fok}{\mathbb{N}} \times \underset{kép}{B^{n \times m}} \times \underset{f}{\mathbb{P}}$, ahol B egy bájtot *jelent* és

$\mathbb{P} = \mathbb{Q} \times \mathbb{Q} \mapsto \mathbb{Q}$ függvény és n és m pozitív egészek

$Q = (\text{fok} < 100 \wedge \text{fok} = \text{fok}' \wedge \text{lép} = \text{kép}' \wedge \text{n} = \text{n}' \wedge \text{m} = \text{m}')$

$R = (Q \wedge (\forall x \in [1,n], \forall y \in [1,m] : f(x,y) = \sum_{i=0}^{fok} \sum_{j=0}^{i} a_{i,j} * x^j * y^{i-j} = kép_{x,y}))$,

ahol $a_{i,j} \in \mathbb{Q}$

- Képnagyítás

$A = \underset{nagy}{\mathbb{Q}} \times \underset{ki\_kép}{B^{k \times l}} \times \underset{f}{\mathbb{P}}$, ahol B egy bájtot *jelent* és

$\mathbb{P} = \mathbb{Q} \times \mathbb{Q} \mapsto \mathbb{Q}$ függvény és k, l, n és m pozitív egészek

$Q = (\text{nagy} = \text{nagy'} \wedge \text{nagy} \in [1,100] \wedge \text{n} = \text{n}' \wedge \text{m} = \text{m}' \wedge \text{f} = \text{f}')$

$R = (Q \wedge k = \text{nagy} * n \wedge l = \text{nagy} * m \wedge$

$\wedge \forall i \in [1,k], \forall j \in [1,l] : \text{ki\_kép}_{i,j} = f(i/\text{nagy}, j/\text{nagy}))$

A két feladatot úgy adja ki az eredeti feladatot, hogy az interpoláció során kapott függvényt használjuk fel a nagyításhoz. Ezt a *solver* névtérben fogjuk megvalósítani az *interpolate* függvénnyel.

A fent specifikált programot interpolációval fogjuk megoldani. A matematikában az interpoláció lényege az, hogy egy adott függvényt (pl.: szinusz) a lehető legpontosabban közelítsük meg egy olyan másik függvénnyel, amely át megy az előre megadott pontokon. A jelen esetben ezeket az előre megadott pontokat egy képből vesszük. Ezáltal azt mondhatjuk hogy közelítendő függvényünk nem más mint maga a kép, amelynek leírása is maga a kép, úgy, hogy az értelmezési tartomány a kép koordinátái, és az értékkészlet pedig 0-255.

Az interpolálást többváltozós Lagrange interpoláció módszerével oldjuk meg, amelyet részletez Kamron Saniee a „A Simple Expression for Multivariate Lagrange Interpolation” című írásában. [1] Ez kimondja, hogy ha $\binom{n+m}{n}$ darab pontot veszünk, ahol n az interpoláció során kapott függvény legmagasabb lehetséges fokszáma és m a változók száma, akkor képesek vagyunk így egy függvényt konstruálni. Az esetünkben n-et a felhasználó adja meg és m pedig 2 lesz. A binomiális együtthatók tulajdonságából adódik, hogy $\binom{n+2}{n} = \binom{n+2}{2}$, ezért elegendő „(n+2)*(n+1)/2”-t írnunk az implementáláskor.

Az írás leírja hogy az interpolációs függvényt úgy kapjuk hogy minden mintaként felhasznált (x,y) párhoz készítünk egy $\sum_{i=0}^{n} \sum_{j=0}^{i} a_{i,j} * x^{i-j} * y^{j}$ vektort, ahol x-et és y-t behelyettesítjük és a vektor elemei az összeg tagjai, illetve az $a_{i,j}$ együtthatók ismeretlenek. Ezeknek a vektoroknak a hosszai egyenként (n+2)*(n+1)/2. Így kapunk egy (n+2)*(n+1)/2×(n+2)*(n+1)/2-es mátrixot ahol az oszlopokban közös elem a megfelelő $a_{i,j}$ együttható.

Ugyanebben a sorrendben a mintáinkból összeállítjuk azt a vektort amelyet úgy kapunk,hogy az i-edik elemhez értékül adjuk az i-edik minta x és y koordinátából származó képpontot, azaz kivesszük az x-edik sor y-odik oszlopához tartozó pixelt. A mátrixból kiemelünk egy (n+2)*(n+1)/2 elemű vektort amelynek az elemei rendre

$$a_{0,0}, a_{1,0}, a_{1,1}, a_{2,0}, a_{2,1} \dots a_{n,n-1}, a_{n,n}$$

lesz. Így $a_{i,j}$ elemekre kapunk egy lineáris egyenletrendszert. Az írat szerzője azt írja le hogy ezt az egyenletrendszert nem kell megoldani és ad egy képletet hogyan tudjuk az Lagrange alap polinomokat felírni. Ez a folyamat (n+2)*(n+1)/2+1 db (n+2)*(n+1)/2×(n+2)*(n+1)/2 mátrix determinánsának kiszámolására van szükségünk, ahol (n+2)*(n+1)/2 db olyan mátrixunk van, ahol az elemek nem valós számok, hanem kétváltozós polinomok. Ezeket a kiszámolt értékeket el kell osztani az eredeti lineáris egyenletrendszer determinánsával, vagyis az eljárás elvárja, hogy ez nem lehet 0.

Az alappontokat egymástól függetlenül lehet számolni így lehet párhuzamosítani a folyamatot, amit GPU-n fogunk elvégezni. Azonban a determináns nem nulla létéből fakadóan egyszerűen (grafikus kártya szempontjából) a lineáris egyenletrendszert is megoldhatjuk, amelynek létezik párhuzamosított konstrukciója. Erre használható a „cusolver” külső modul. Ezek közül választhatjuk a QR felbontással történő megoldást,

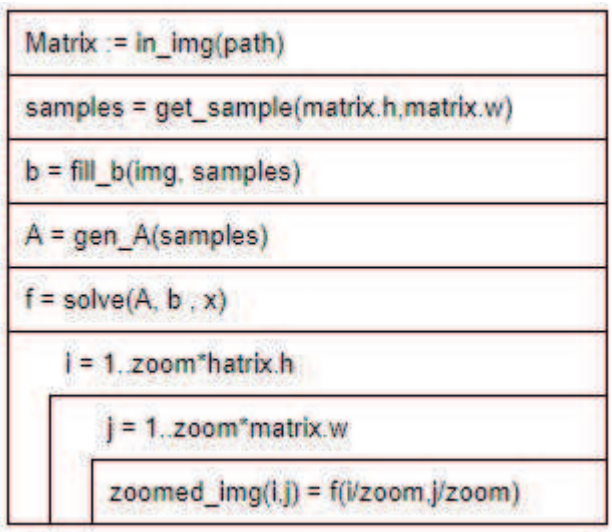
ahol az A mátrixot egy ortogonális mátrix és egy felső háromszögmátrixra bontjuk majd az eredeti $\mathrm{Ax} = \mathrm{b}$ helyett az $\mathrm{Rx} = Q^T\mathrm{b}$ egyenletrendszert oldjuk meg. Egy másik lehetséges megoldás, hogy LU felbontást alkalmazunk. Majd az így kapott x vektorból felépítünk egy kétváltozós polinomokat reprezentáló osztályt. Ebből fogjuk számolni a nagyitáshoz a meglévő pontok közötti pontokat.

A *solve* modul felépítése:

Az *interpolate* függvény

- paraméterek
    - degree
        - pozitív egész szám
        - az képet közelítő függvény fokszámának legnagyobb lehetséges értéke
    - path
        - karakterlánc
        - a feldolgozandó képfájl elérési útvonala
            - lehet relatív és abszolút
    - zoom
        - pozitív lebegőpontos szám
        - a kép kívánt nagyításának mértékét írja le
    - visszatérési érték
        - egész szám
        - 0 ha minden renden történt
- a függvény feladata az, hogy a kapott paraméterekből először beolvassa a képet, mintát vegyen, majd megcsinálja a keresett függvényt, majd összeállítsa az új képet.

- absztrakt program:



16. ábra interpolate függvény absztrakt stuktogramja

A *cuda* modul felépítése

A solveQR függvény

- paraméterek
  - m
    - egész szám
    - az egyenletek száma
  - A
    - duplapontosságú lebegőpontos számra mutató pointer
    - a lineáris egyenletrendszer mátrixa oszlopfolytonos megfeleltetésben (m*m hosszú)
  - b
    - duplapontosságú lebegőpontos számra mutató pointer
    - a lineáris egyenletrendszer jobb oldala (m hosszú)
      - lehet relatív és abszolút
  - pivot
    - egész szám
    - függvény kompatibilitási szempontból szerepel (nem csinál semmit, szerepe a függvényszignatúra megegyezése a *solveLU* függvénnyel)
  - visszatérési érték
    - duplapontosságú lebegőpontos számra mutató pointer

        - m hosszú területre mutató mutató, amely az egyenletrendszer megoldását tartalmazza

    a függvény feladata az, hogy a kapott paraméterekből QR felbontás módszerével állítsa elő az egyenletrendszer megoldását.

A solveLU függvény

- paraméterek
    - m
        - egész szám
        - az egyenletek száma
    - A
        - duplapontosságú lebegőpontos számra mutató pointer
        - a lineáris egyenletrendszer mátrixa oszlopfolytonos megfeleltetésben (m*m hosszú)
    - b
        - duplapontosságú lebegőpontos számra mutató pointer
        - a lineáris egyenletrendszer jobb oldala (m hosszú)
            - lehet relatív és abszolút
    - pivot
        - egész szám
        - ha az értéke 1 akkor $A = LUP$ felbontást alkalmazunk
    - visszatérési érték
        - duplapontosságú lebegőpontos számra mutató pointer
        - m hosszú területre mutató mutató, amely az egyenletrendszer megoldását tartalmazza

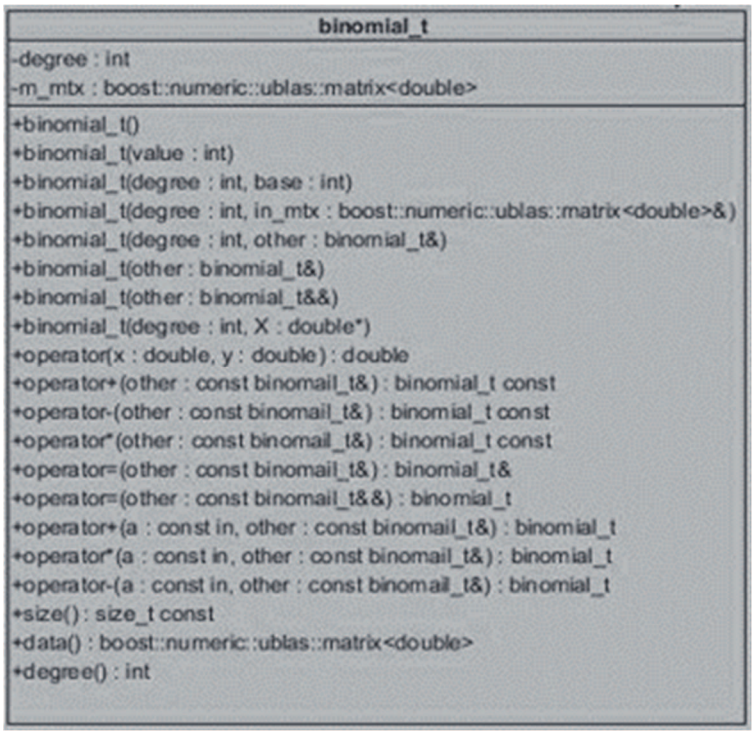
a függvény feladata az, hogy a kapott paraméterekből LU felbontás módszerével állítsa elő az egyenletrendszer megoldását.

3.2.4 Segéd osztályok

sampling

Ennek a modulnak az a feladata, hogy a mintavételezéshez előállításon egy (x,y) párokból álló sorozatot. **binomial_t** osztály

Ez az osztály képes tárolni egy kétváltozós függvényt. (bivariate polinom) Az osztály c++14-es szabvány alapján van írva. A legtöbb operátor felül van definiálva, számos parametrikus konstruktor meg van írva a könnyed használat érdekében.



17. ábra binomial_t osztálydiagramja

- **attribútumok**
  - degree
    - egész szám
    - a polinom foka
    - size() függvénnyel kérhető le
  - m_mtx
    - duplapontosságú lebegőpontos elemeket tartalmazó mátrix
    - a polinom együttható
    - data() függvénnyel kérhető le

- **konstruktorok**
  - alapértelmezett konstruktorok(ctor)
    - C++14-es szabványvak megfelelő [3]
    - default ctor
    - copy ctor
    - move ctor
  - parametrikus konstruktorok
    - value : int

        - nulladfokú (konstans) polinom
    - degree: int, base int
        - feltölti az együttható mátrixot a base tartalmával egy degree fokszámú polinomhoz
    - degree: int, other binomial_t&
        - egy degree fokszámú polinomot hoz létre, amelyben a másik polinom elemei találhatóak meg. Ha a fokszám kisebb mint a másik polinom fokszáma akkor csonkol, többlet helyeknél 0 kerül beírásra.
    - degree: int , x :double*
        - egy degree fokszámú polinomot hoz létre az x tömbből úgy, hogy $\langle \mathrm{x}^{degree}y^0, \mathrm{x}^{degree-1}y^1, \ldots, x, y, 1\rangle$ sorozathoz tartozó együtthatók.
        - az x bemenő paraméter típusából adódik, hogy a tömb mérete:
          $$(\mathrm{n}+2)*(\mathrm{n}+1)/2+1 \leq \mathrm{length}(\mathrm{x})$$
          ellenkező esetben szegmentálási hibát kapunk!
    - degree: int, in_mtx : boost::numeric::ublas::matrix<double>&
        - lemásolja a paramétereket a saját tárolóiba
- **operátorok**
    - megvalósítjuk az osztályhoz:
        - értékadó operátorokat
        - aritmetikai operátorokat
            - összeadás
            - szorzás
            - kivonás
        - kiíró operátort
    - az operátorok a C++14-es szabvány [4] szempontjaival megfelelően írjuk meg
- **getter függyvények**
    - size()
        - a degree értékéről ad egy másolatot
    - data()
        - az m_mtx egy másolatát adja vissza
    - kiíró operátort is hozunk létre a tesztelés számára, hogy tesztelés közben lehessen ellenőrizni a kapott eredményt, különben nincs szükség rá
    - beolvasó operátort nem implementálunk.

3.3 Megvalósítás

3.3.1 Implementációs döntések

Ebben a fejezetben kerül leírásra, hogy az egyes részek megvalósítása közben, milyen eltérések keletkeztek a tervhez képest. A megvalósítás során először a *modult*

implementáljuk, majd ehhez készítjük el a grafikus felhasználói felületet, amit a *modul* fő függvényére kötünk rá.

3.3.1.1 Modul implementációs döntései

A modul a megvalósítás során volt az a rész, ahol a legtöbb dolog eltér a tervekben leírtaktól. A megoldás közben 3 fő problémát kellet felülszárnyalni

1. Hogyan válaszunk pontokat a kép rácsai közül, hogy a belőlük generált mátrix sorai, illetve oszlopai lineárisan függetlenek legyenek?
2. Hogyan fogjuk át a képet, ha csak azt tudjuk garantálni, hogy a képből a legrövidebb oldal kétszeresnyi pontot tudunk kiválasztani.
3. Hogyan kezeljük a numerikus hibákat és hogyan legyen az egész kép interpolálva
    a. a beviteli adat kerekített
    b. a mátrix nem feltétlenül jól kondicionált
    c. a függvény szélén kifutunk az értékkészletből

Az első problémát úgy próbáltam megoldani, hogy a rácsból úgy veszünk ki pontokat, hogy azok közül 3 nem esik ugyan arra a pontra. Először egy visszalépéses kereséssel működő algoritmust terveztem meg és implementáltam, de ennek az alkalmazását futásidő rohamos növekedése miatt el kellett vetni. A „nincs három egy vonalon” probléma a kombinatorikus matematika is vizsgálja. Amimből léteznek sejtések felső korlátra. Ez az ág nyújt megoldást számunkra, ami egy alsó korlátot ad a kiválasztható pontokra. [2]. Ezzel megoldható az egzakt lineáris egyenletrendszerrel működő algoritmus.

A második problémát az egzakt lineáris egyenletrendszer okozza. Ha van egy 2n széles és 2n magas képünk akkor a fenti módszerrel a generált polinom az garantálja, hogy legfeljebb $3(n-1-\varepsilon)$ ponton keresztül lesz a függvényünk pontosan ugyan az, mint a mintánk. Ezt a problémát úgy hidaljuk át, hogy az egzakt lineáris egyenletrendszer helyett egy túlhatározott egyenletrendszert oldunk meg. Ebben az esetben az egész képet használjuk fel mint minta és a megoldás a legkisebb hibájú x vektort próbálja meg megtalálni

A harmadik probléma szintén akkor merült fel amikor az egzakt lineáris egyenletrendszer tesztelése során merült fel. A probléma az volt, hogy egy egyszerű

$$f(x,y) = 2x + y + y^2$$

függvényt az elvártaknak megfelelően interpolált, addig egy képet csak egy kis szeletén közelített, de nem az egészen. Egy ehhez tartozó hiba volt még, hogy egy komplexebb függvényből készített kép is hibás eredményt szült. Erre a problémára is megoldást nyújt a túlhatározott egyenletrendszerrel való megközelítés.

Azért választottam a túlhatározott egyenletrendszer megoldását, mert ezzel az eredeti feladatot is meg tudjuk oldani, mert az ennek a feladatnak egy részét képzi, mivel az egyenletrendszert megoldó algoritmus nem változik, csak a beadott mátrix meg az egyenletrendszer jobboldala.

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$$

Az implementáláshoz három külső könyvtárat használtam. Ezek:

- boost [5]
- OpenCV [6]
- CUDA Runtime [7]

Ahogy a tervben van írva az absztrakt program (solver.cpp) (16. ábra) megvalósítása nem változott. A képet az OpenCV könyvtár segítségével *cv::imread()* olvassuk be egy mátrix típusba. Ennek a létrehozásakor nem csak szürkeárnyalatos képeket tudunk feldolgozni, hanem a színes képeket is fekete fehér képpé konvertáljuk.

Ezután a *sample* (sampling.hpp) modulban található *sample_img_v5()* mintavevő függvényt használjuk. (ezen felül megtaláljuk még a [2] es forrásbal lévő minta generáló függvényt).

A következő lépésben megalkotjuk az egyenletrendszert OpenCV segítségével megalkotjuk a transzformált egyenletrendszert. Ennek a rendszernek megoldására két implementáció áll rendelkezésünkre(qr.cpp) *solveLU()* , *solveQR()* ezek szignatúrája megegyezik így könnyedén válthatunk a kettő között. Alapértelmezetten LU felbontást használunk, és külön indítási paraméterként szólhatunk a programnak, hogy QR felbontást használjunk. Eredménye egy *double* tömb.

Ezt a tömböt át konvertáljuk, becsomagoljuk egy *binomial_t* típusba így egybefoglaltuk egy ellenőrzött objektumba. Majd ebből a példányból könnyedén számolhatunk egy nagyított mátrixot, amelyet szintén az OpenCV segítségével kiírunk fájlba.

A *solveLU()* és *solveQR()* implementációja során CUDA nyelven keresztül kommunikálunk a grafikus processzorral. Két fajta megközelítéssel dolgozhatunk. Az első az hogy megírjuk saját magunk, minden GPU utasítást és azt használjuk fel. A másik, hogy a CUDA API lineáris algebra megoldó könyvtárát (CuSolver) használjuk fel. Ezek közül az utóbbit választottam, mert ez a könyvtár optimalizálva van a grafikus kártyára, illetve biztonságos interfészt biztosít.

3.3.1.2 A felhasználói felület megvalósítása

A tervezés során eldöntött irányelvek mentén alkotjuk meg a grafikus felületet. Az alkalmazás ezen része is C++ nyelven van írva. A VisualC++ fordítója lehetőséget nyújt hogy .NET keretrendszer osztályait felhasználjuk. A felhasználói felület WinForms technológián alapszik. Ezt legkönnyebben Visual Studio 2015, illetve magasabb verziószámú alkalmazással tudjuk fejleszteni. Ebből adódóan egy „solution”-be helyeztük el az egész alkalmazást. Így a program fejlesztéséhez elégendő fájlokat egyszerűen tudjunk migrálni.

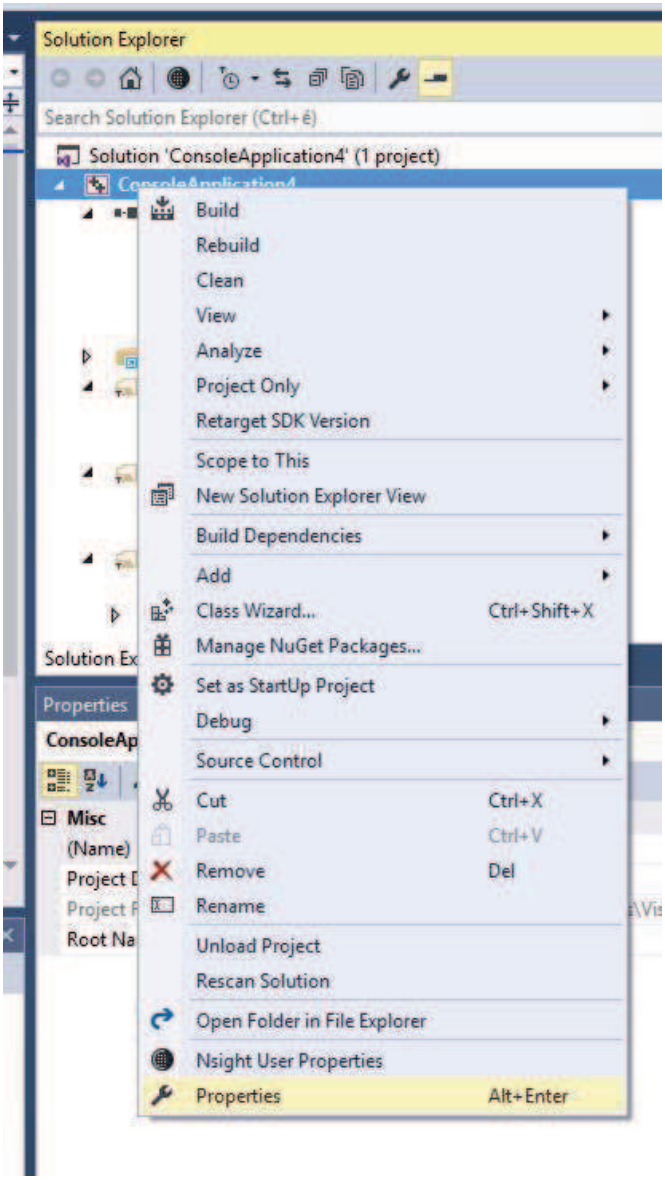
A felhasználói felület megvalósítása során a fő osztályba felvettem egy „openFileDialog1” nevű adattagot, amely megvalósít egy olyan párbeszédablakot amely, egy beágyazott Windows Intézővel rendelkezik, képes kiterjesztés alapján szűrni és igazán letisztult.

3.3.2 Fejlesztői környezet

Az implementáláshoz a választott nyelv a C++ illetve ennek egy változata a VisualC++. Utóbbit a legkönnyebben és felhasználó (jelen esetben fejlesztő) baráti módon a Microsoft IDE-jét használjuk. A projekthez tartozó „solution” fájlt kell betölteni és konfigurálni a saját gépünkre.

3.3.2.1 Build konfiguráció

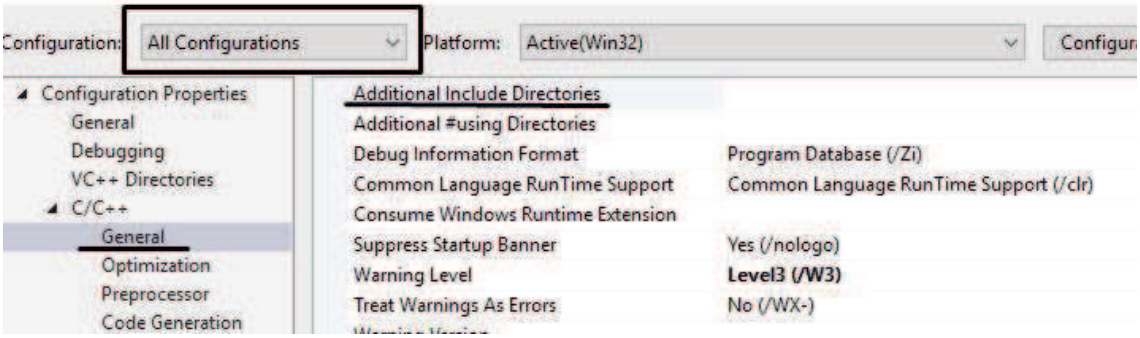
Mivel három külső könyvtárat is használunk. (.NET belsőnek lehet venni) ezért ezeket meg kell mondanunk a fordítónak, hogy a szükséges fájlok melyek és, hogy hol vannak a gépünkön.



18. ábra Properites

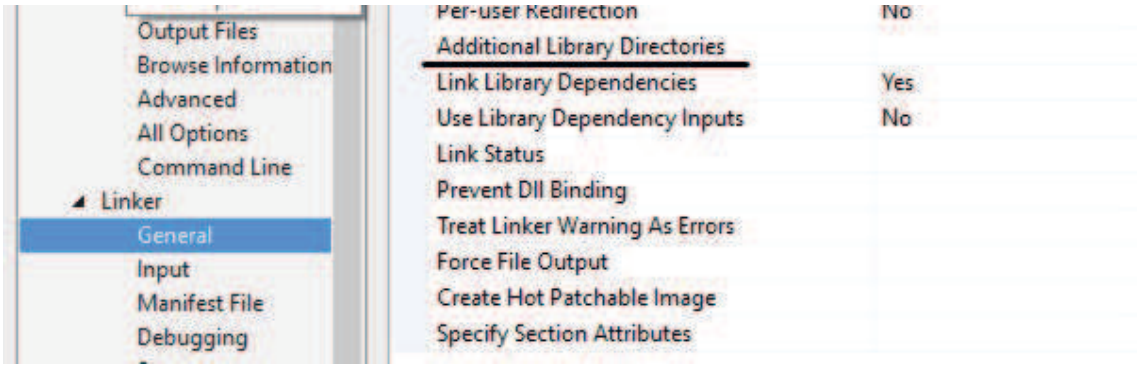
Be kel mennünk a projekt beállításaiba, majd a felugró ablakon a „configuration” menüpontot „All Configurations”-ra állítani . ezután „C/C++” menüpont alatt a „General” részre kattintva megkeressük az „Additional Include Directories” pontot.

Ide kell beírnunk a BOOST könyvtár az OpenCV és a CUDA könyvtár header fájlainak helyét.



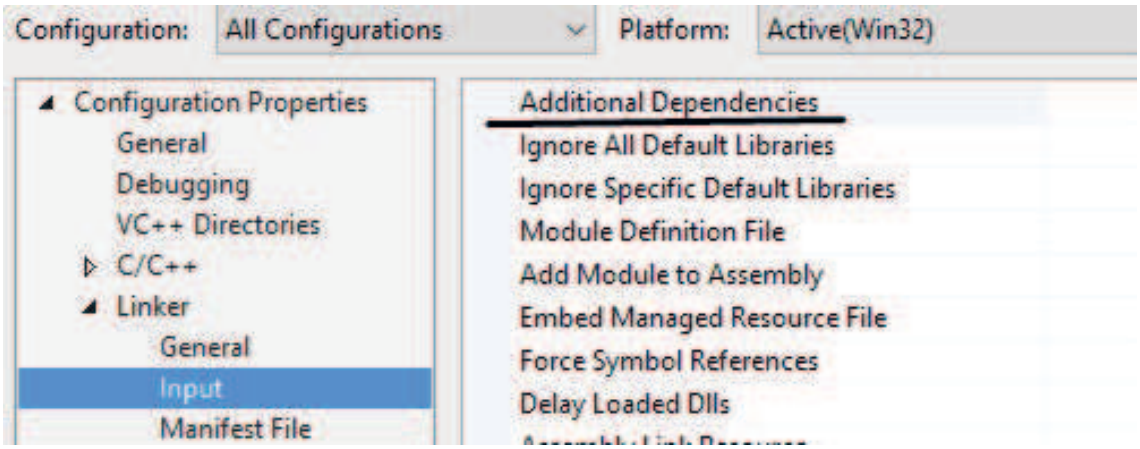
19. ábra Include fájlok beállítása

Be kell állítani hogy a linker mely binárisokat kell összekötnie a projekttel. Ezt a „Linker” „General” részben lehet az „Additional Library Directories” pontnál tehetjük meg. Ide azt írjuk be hogy hol vannak a linkelni kívánandó fájlok.(OpenCV,CUDA)



20. ábra lib fájlok helyének megadása

Az „Input” menüpont alatt pedig az „Additional Dependencies” pontnál kell megadni a linkelendő binárisokat. (Ebben az esetben cudart.lib, cusolver.lib, cublas.lib opencv_world341.lib, opencv_world341d.lib (debug opciókhoz)).



21. ábra Könyvtárak megadása

3.3 Tesztelés

3.3.1 Felhasználói történetek

A program felhasználói felületét a felhasználói történetek alapján teszteljük (Fekete doboz teszteléssel)

	esemény	**mikor**	**akkor**	**eredmény**
1	alkalmazás indítása	felhasználó futtatja a programot	a program elindul és megjelenik a felhasználói felület	a program elindul és megjelenik a felhasználói felület
2	kép kiválasztásának szándéka	a felhasználó rákattint a „Tallózás” gombra	megjelenik egy párbeszédablak amint kiválaszthatjuk a fájlt	megjelenik a párbeszédablak amint kiválaszthatjuk a fájlt
3	hibás képválasztás	a felhasználó hibás inputot ad meg	a hiba jelződik a felhasználó számára	felugrik egy ablak a hibáról és közli a felhasználóval a helyes használatot
4	helyes képválasztás	a felhasználó helyes inputot ad meg	a program bejegyzi az inputot	A felület szöveges mezőjébe belekerül a teljes elérési útvonal
5	helyes fokszám választás	felhasználó a programba bejegyzi az inputot, beírja a kívánt fokszámot	a program bejegyzi az inputot	A felület megfelelő számbeviteli mezőjébe belekerül a teljes elérési útvonal

6	helyes nagyítás választás	felhasználó a programba bejegyzi az inputot, beírja a kívánt nagyítást	a program bejegyzi az inputot	A felület megfelelő számbeviteli mezőjébe belekerül a teljes elérési útvonal
7	a generálási folyamat elkezdése	a felhasználó rákattint a „Generálás” gombra	A program előállítja az új képet	Elindul a folyamat a töltő csík változik, majd megjelenik a kép
8	a generálási folyamat elkezdése	a felhasználó rákattint a „Generálás” gombra	A program szól a hibáról.	Az alkalmazás szól a hibáról és megmondja a helyes módot
9	helyes fokszám választás	felhasználó a programba bejegyzi az inputot, beírja a kívánt fokszámot	A program szól a hibáról.	Az alkalmazás nem engedi beírni, illetve, ha nincs 1-100 között akkor odakerekíti
10	helyes nagyítás választás	felhasználó a programba bejegyzi az inputot, beírja a kívánt nagyítást	A program szól a hibáról.	Az alkalmazás nem engedi beírni, illetve, ha nincs 1-100 között akkor odakerekíti

3.3.2 A modul tesztelése

A modul tesztelése során teszteljük, hogy az adott építőegység mennyire hibamentes, (pl.: nem omlik össze) arra is kitérünk, hogy az egyes komponensek, funkcionálisan megfelelőek-e (fehér doboz teszteléssel). A teszek megvizsgáljuk a futásidőt, és az elvárt és kapott eredmény (fekete doboz teszteléssel).

- minden adat helyes
  - létező fájl
  - fok 0 és 100 között van nagyítás 1-100 között van
    - egy nagyításszoros képet kapunk, ha megfelelő számú minta szerepel. ((fok+1)(fok+2)/2)
- hibás fájl
  - nem létezik a fájl
    - hibaüzenettel tér vissza, hogy hibás az input fájl
  - rossz kiterjesztésű fájl
    - A kép beolvasásra kerül, ha a megfelelő DLL-ek jelen vannak a gépen, különben hibaüzenet hogy nem olvasható a fájl.
  - szerkesztésben lévő fájl
    - hibaüzenetet ad hogy nem lehet megnyitni a fájt
- hibás fokszám
  - fokszám = 0
    - hibával tér vissza
  - fokszám < 0
    - hibával tér vissza
  - fokszám > 101
    - hibával tér vissza
  - fokszám = 101
    - hibával tér vissza
- fokszám határa
  - fokszám = 1
    - megkapjuk a képet, ha megfelelő számú minta szerepel ((fok+1)(fok+2)/2)
  - fokszám = 100
    - megkapjuk a képet, ha megfelelő számú minta szerepel ((fok+1)(fok+2)/2)
- hibás zoom
  - hibás = 0
    - hibával tér vissza
  - hibás < 0
    - hibával tér vissza
  - hibás > 101
    - hibával tér vissza
  - hibás = 101
    - hibával tér vissza
- hibás határa

- fokszám = 1
  - megkapjuk a képet, ha jó a fokszám
- fokszám = 100
  - megkapjuk a képet, ha jó a fokszám

3.3.3 A binomial_t osztály tesztelése

Konstruktorok testelései

- azokat a konstruktokot amlelyek definíciója „= default”, nem kell tesztelni, mivel a szabvány [3] szerint történik a generálásuk, illetve nem kell mélyen másolni, mert minden adattag member szinten van definiálva és tesztelve a másoló konstruktoraik [5]
- binomial_t(int degree, int base)
  - létre jön egy (degree+1) × (degree+1) dimenziós mátrix és minden eleme a base-zel lesz egyenlő
- binomial_t(int value)
  - létrejön egy 1×1 es mátrix és az eleme a value értéke
- binomial_t(int, double* x)
  - x mérete = $\binom{\text{degree}+2}{2}$
    - az elemek a specifikációnak (statikus tervezés) megfelelően kerülnek bele az elemek. Ténylegesen a polinomot reprezentálja.

Operátorok tesztelései

Az operátorok teszteléseit matematikai műveletekkel végeztük. Ennek folyamata a következő. Különböző kombinációkat írunk ki majd a 2 fajta ellenőrzést végzünk

- A kapott polinomot, mint függvény fogjuk fel és az (x,y) behelyettesítési értékét hasonlítjuk össze az általunk előre kiszámol értékkel.
- A másik tesztelési lehetőség az, hogy kiírjuk a polinomot az együtthatóival reprezentálva és ezt hasonlítjuk össze az általunk várt eredménnyel.

Ezzel egyibőben próbára tesszük a kiíró operátort és a gömbölyű zárójel operátort is

Tesztesetek

- Konstruktorok
  - konstanst polinom

    - első fokú polinom
    - ötöd fokú polinom
    - tized fokú polinom
- összeg
    - konstans + konstans
    - konstans + polinom
    - polinom + polinom
        - azonos fokszámú
        - különböző fokszámú
    - polinom + konstans
    - polinom + polinom + polinom
- különbség
    - konstans - konstans
    - konstans - polinom
    - polinom - polinom
        - azonos fokszámú
        - különböző fokszámú
    - polinom - konstans
    - polinom - polinom - polinom
- szorzat
    - konstans * konstans
    - konstans * polinom
    - polinom * polinom
        - azonos fokszámú
        - különböző fokszámú
    - polinom * konstans
- vegyes műveletek
    - itt ellenőrizni kell, hogy a műveletek között érvényesül-e az aritmetikai műveletek között a precedencia.
        - Ezt biztosítja a nyelv konstrukciója, amely garantálja a műveletek sorrendjének helyességét. [4]
- kiírás
    - külön tesztelési figyelmet fordítunk a konstans polinomokra

- teszteljük, hogy bármely helyen a behelyettesítési érték megfeleljen a konstans értéknek.

- A kiíró operátor implementációjában a polinomot, mint egy szöveg írjuk ki ezért figyelni kell, hogy a végére is kerül egy „+” jel!

3.3.4 Performancia:

Itt vizsgáljuk a program futásának idejét a bemenő paraméterek viszonylatában

Fokszám szerinti tesztelés

LU és QR felbontással

15 kísérletből átlagolva

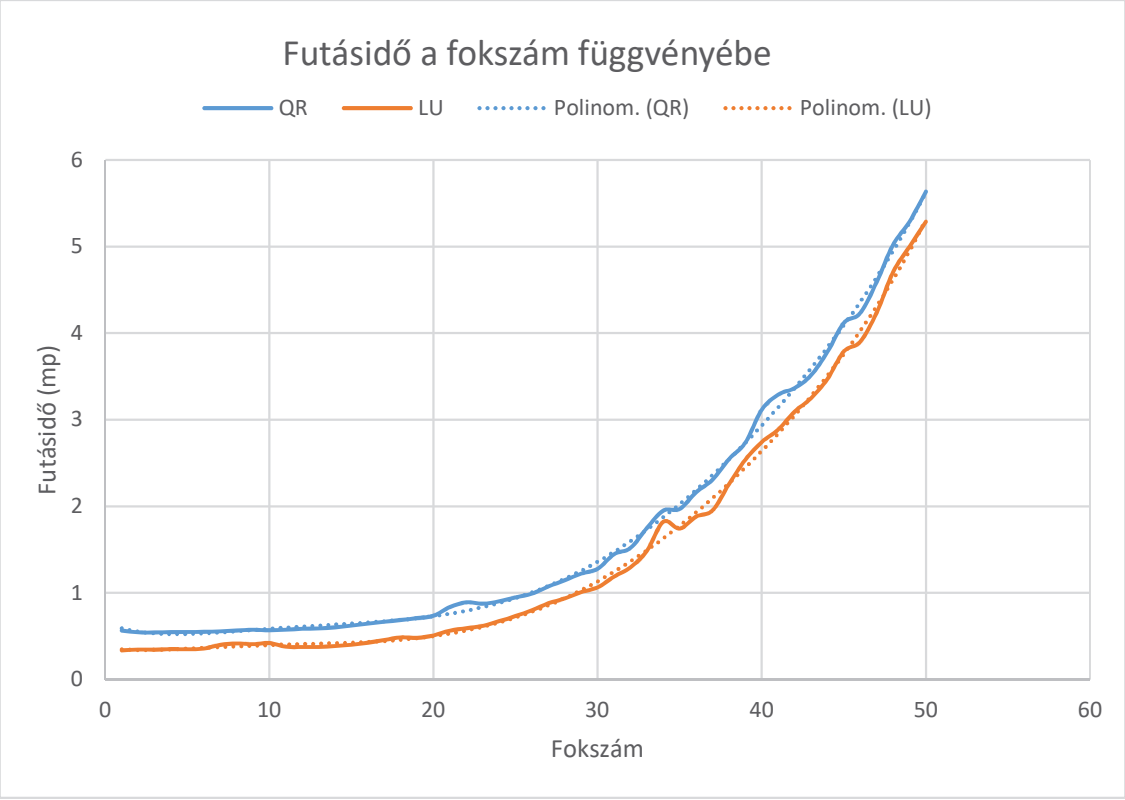
ötszörös nagyítás

75×75-ös kép

Specifikáció:

NVIDIA GeForce GTX 950m

Intel® Core™ i7 4720HQ 2.6GHz up to 3.6GHz



Látszik, hogy polinomiális futásideje van. A használt trendvonalak hatodfokú polinomok.

Nagyítás szerinti tesztelés

15 kísérletből átlagolva

huszadfokú interpoláció

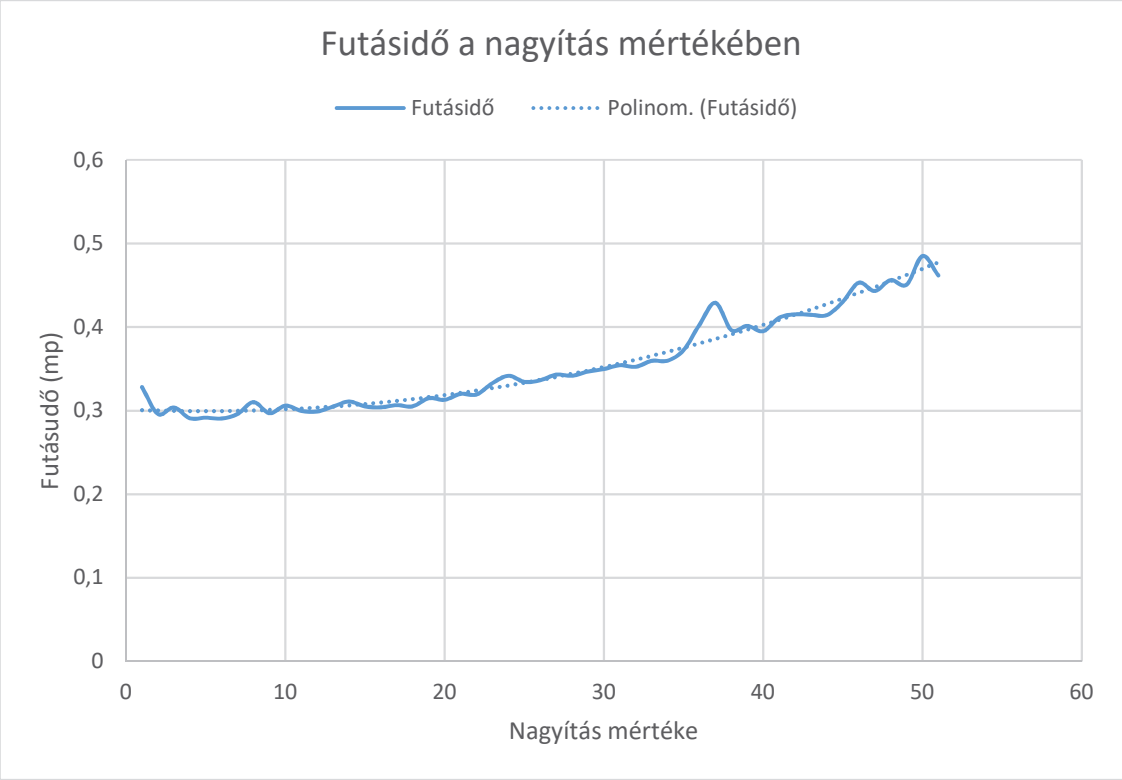
0,5-ös lépésköz

75×75-ös kép

Specifikáció:

NVIDIA GeForce GTX 950m

Intel® Core™ i7 4720HQ 2.6GHz up to 3.6GHz



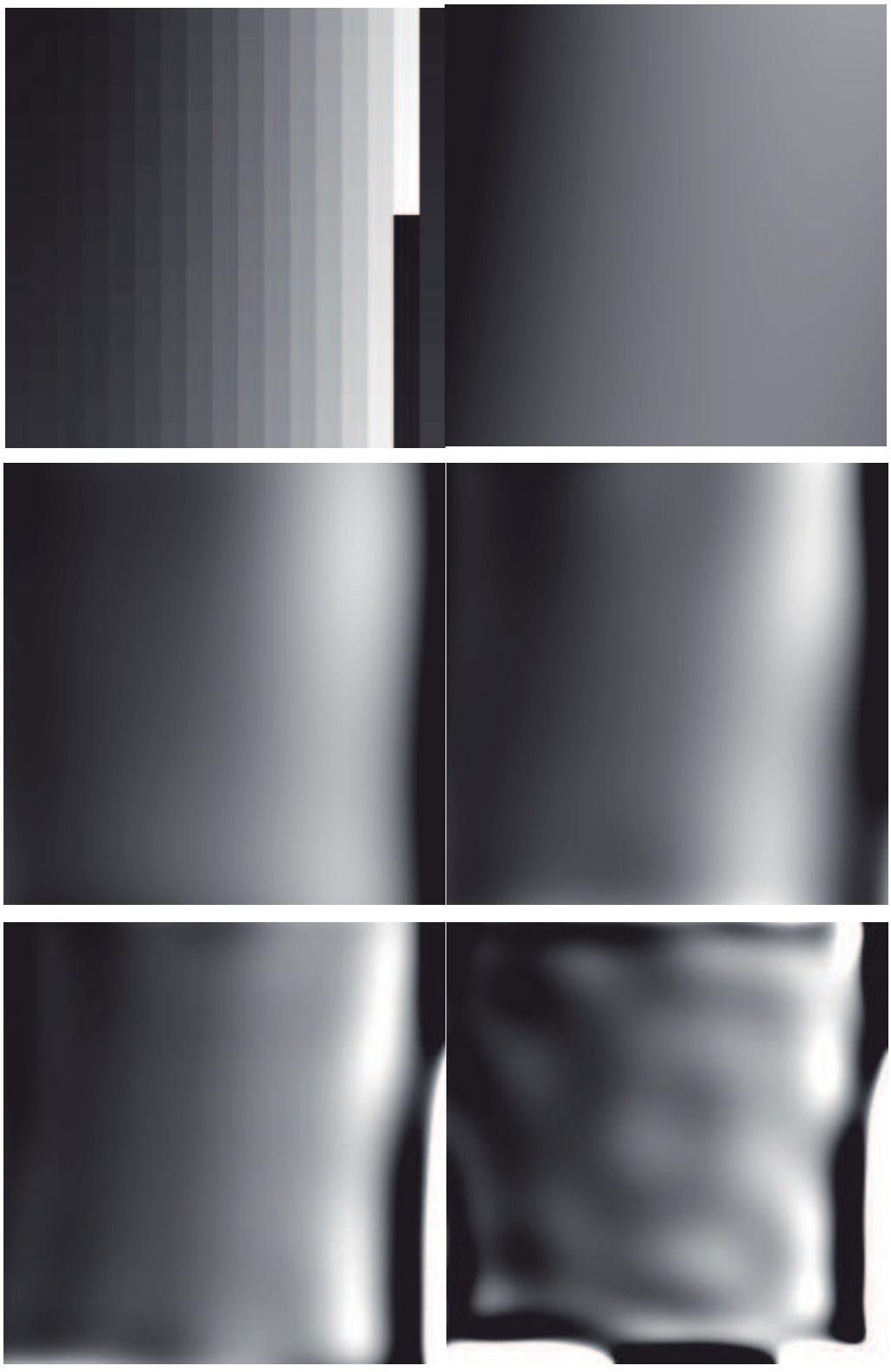
A tesztből látszik, hogy a futásidő négyzetes a nagyítás mértékével. A trendvonal egy másodfokú polinom.

3.3.5 Képes tesztek

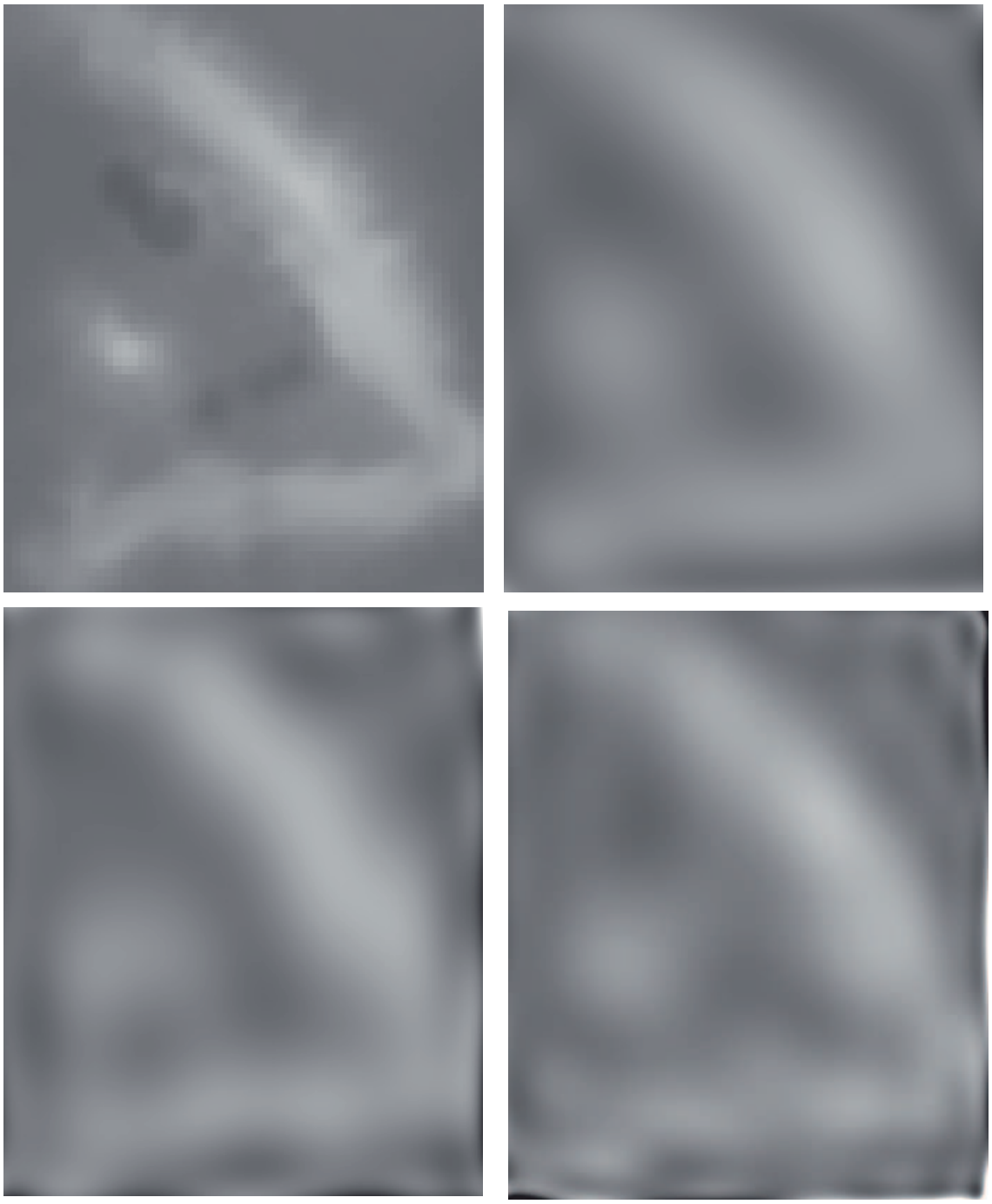
Balról jobbra, fentről lefelé indexelődik.

Egy harmadfokú polinomból generált kép 15x nagyitás fokok: 2,5,7,10,15

A teszt azt vizsgálta, hogy mennyire zavar bele két oszlopnyi radikálisan eltérő minta

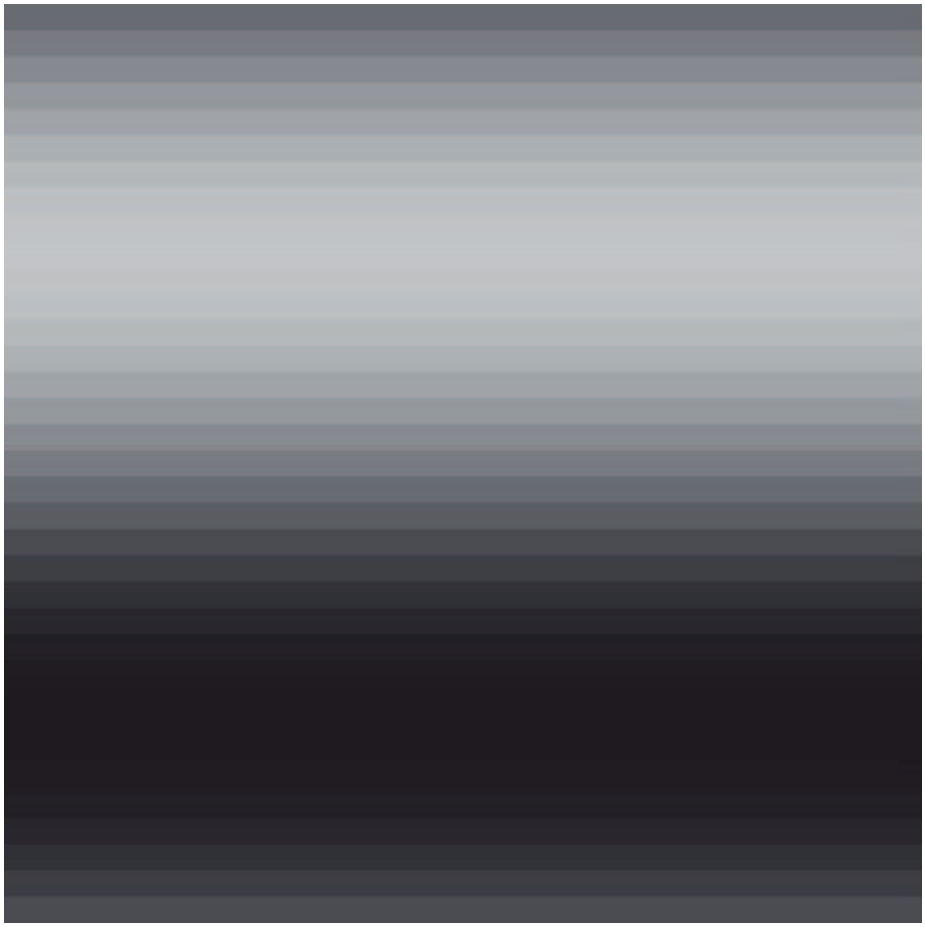


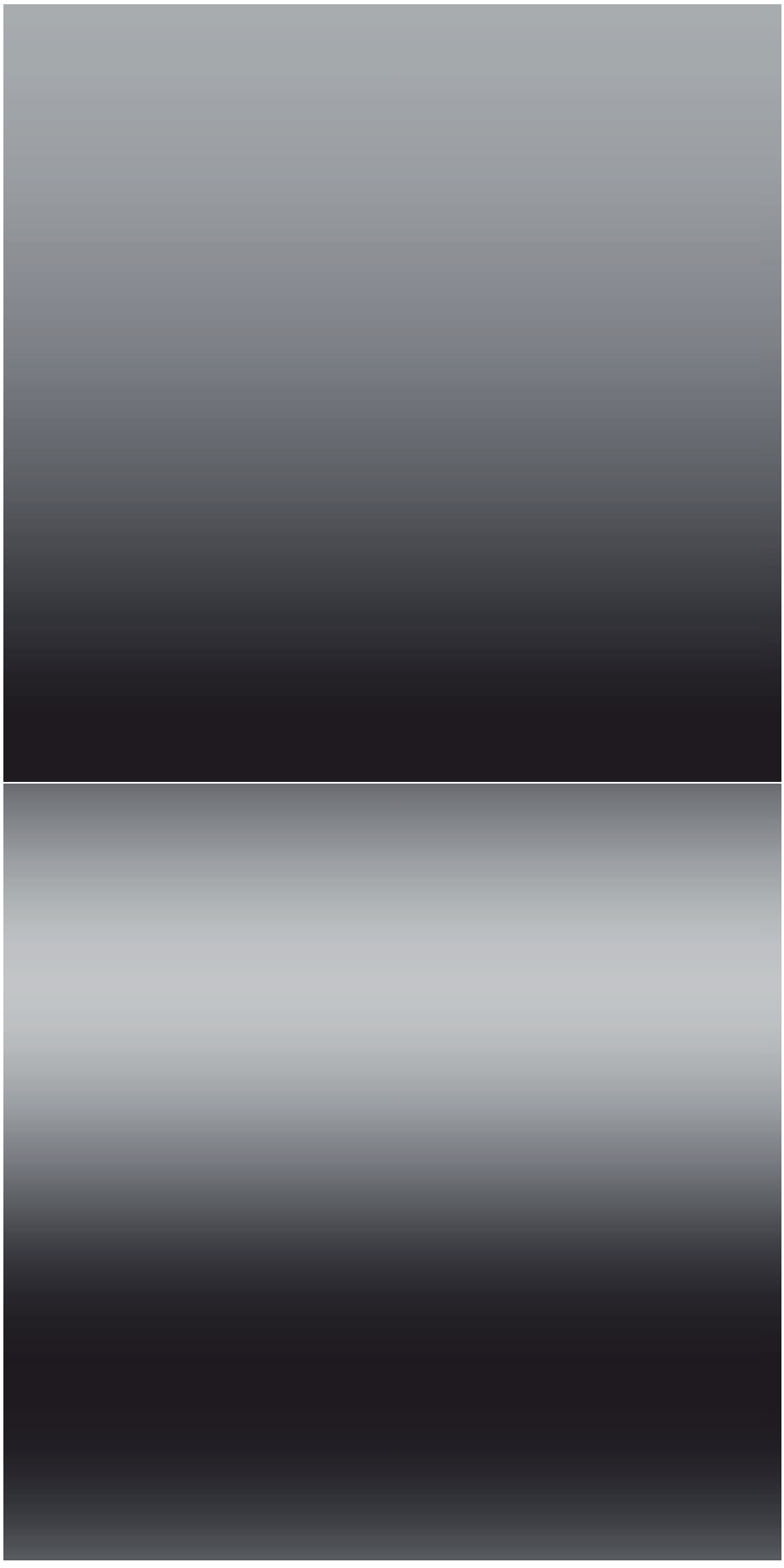
Erdély topologikus képe 3x nagyitás fok: 10,15,30 (laphoz igazítva)



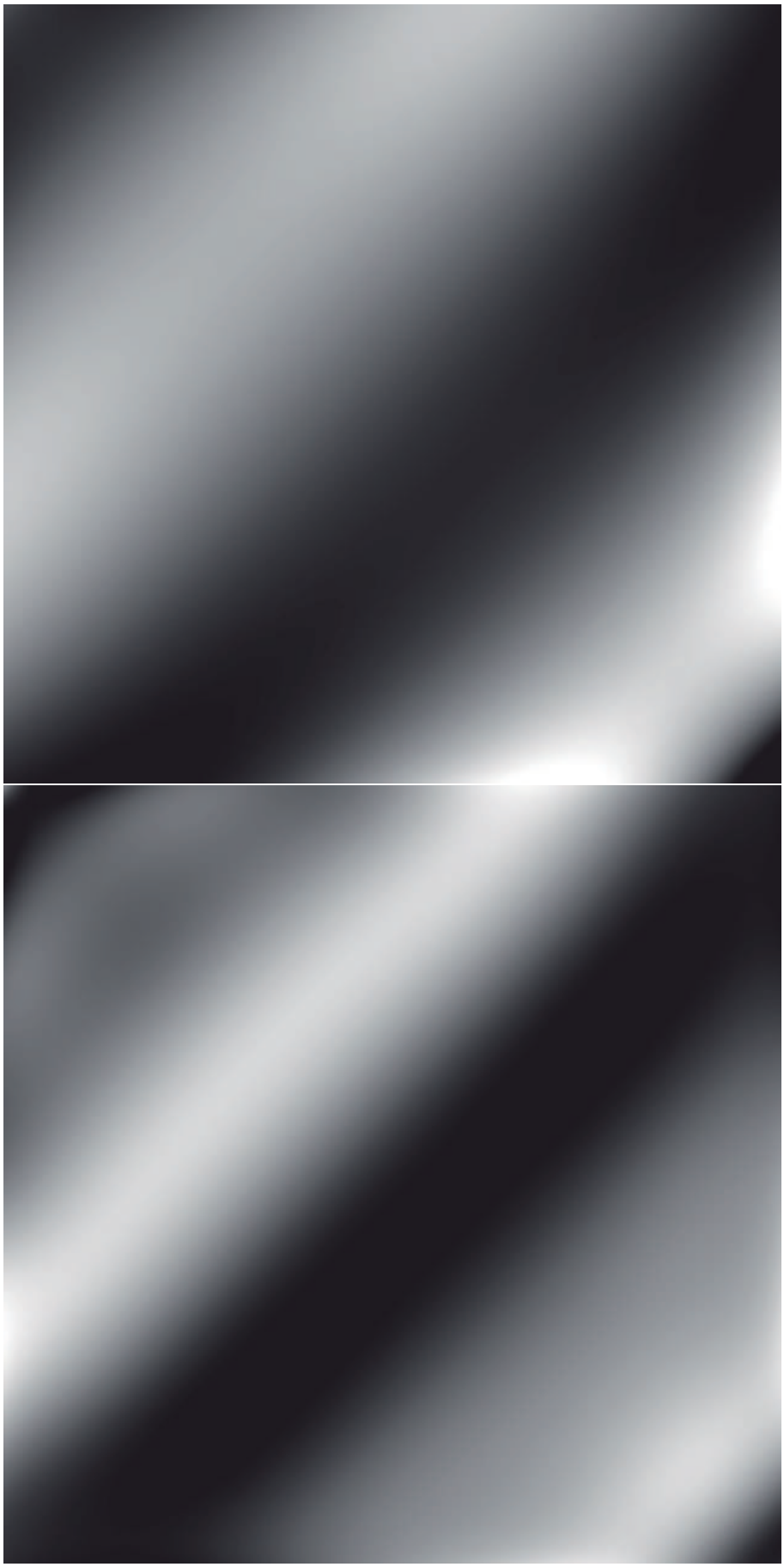
Szinusz függvényből generált függvény 15x fok: 2,7 (laphoz igazítva)

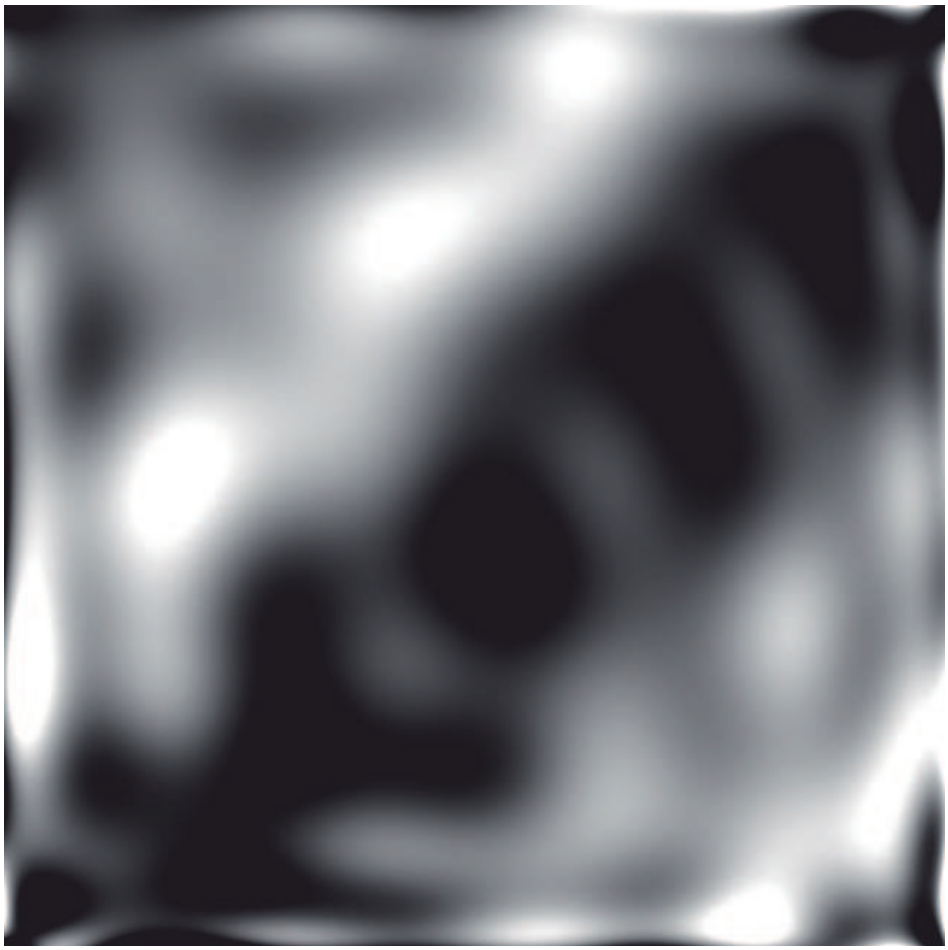
A teszt során azt vizsgáltuk meg, hogy mennyire tudjuk közelíteni a szinusz függvényt egy periódusán. Ez egy tipikusan interpolációs feladat.





Egy programmal generált átmenetes mintázat. nagyítás:2.5x fok:3,5,10,20

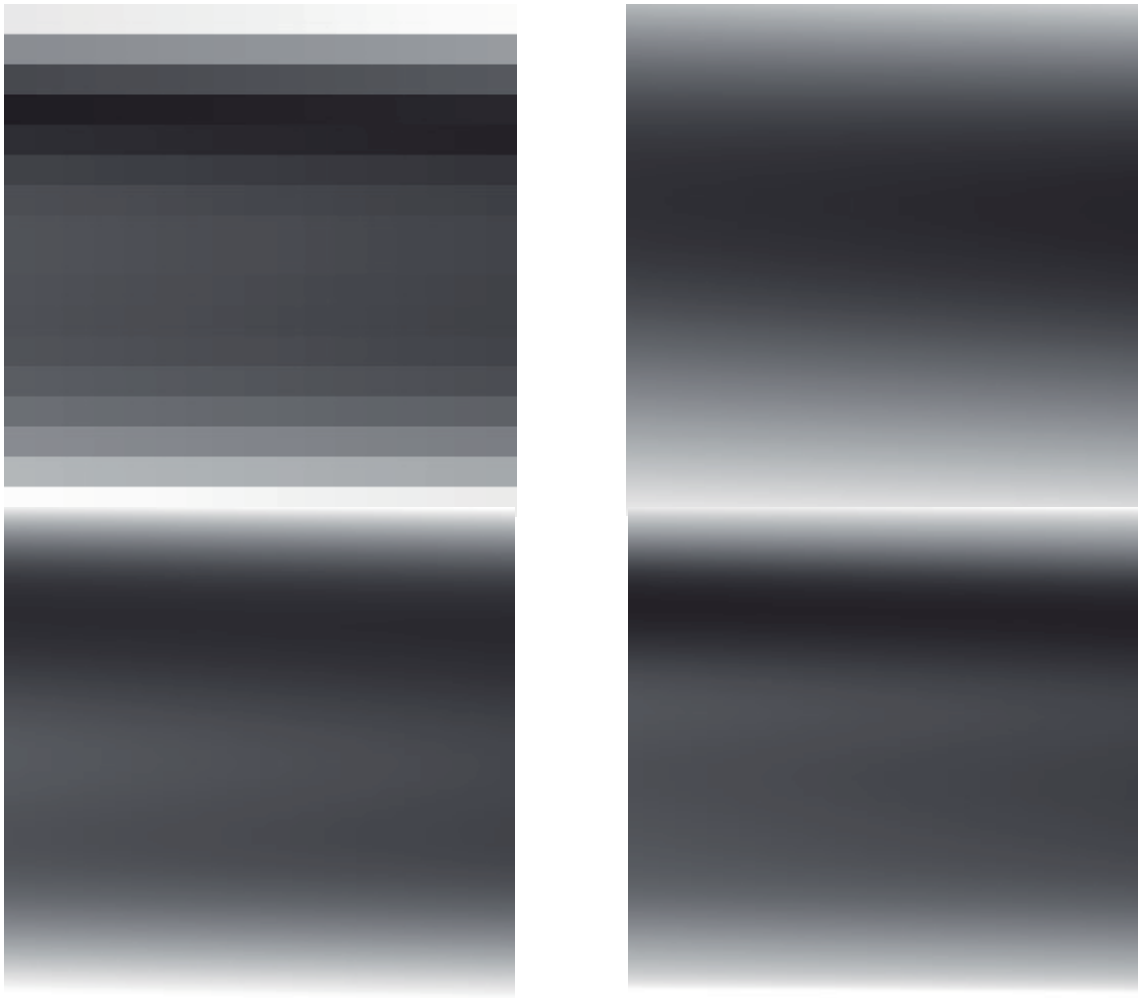




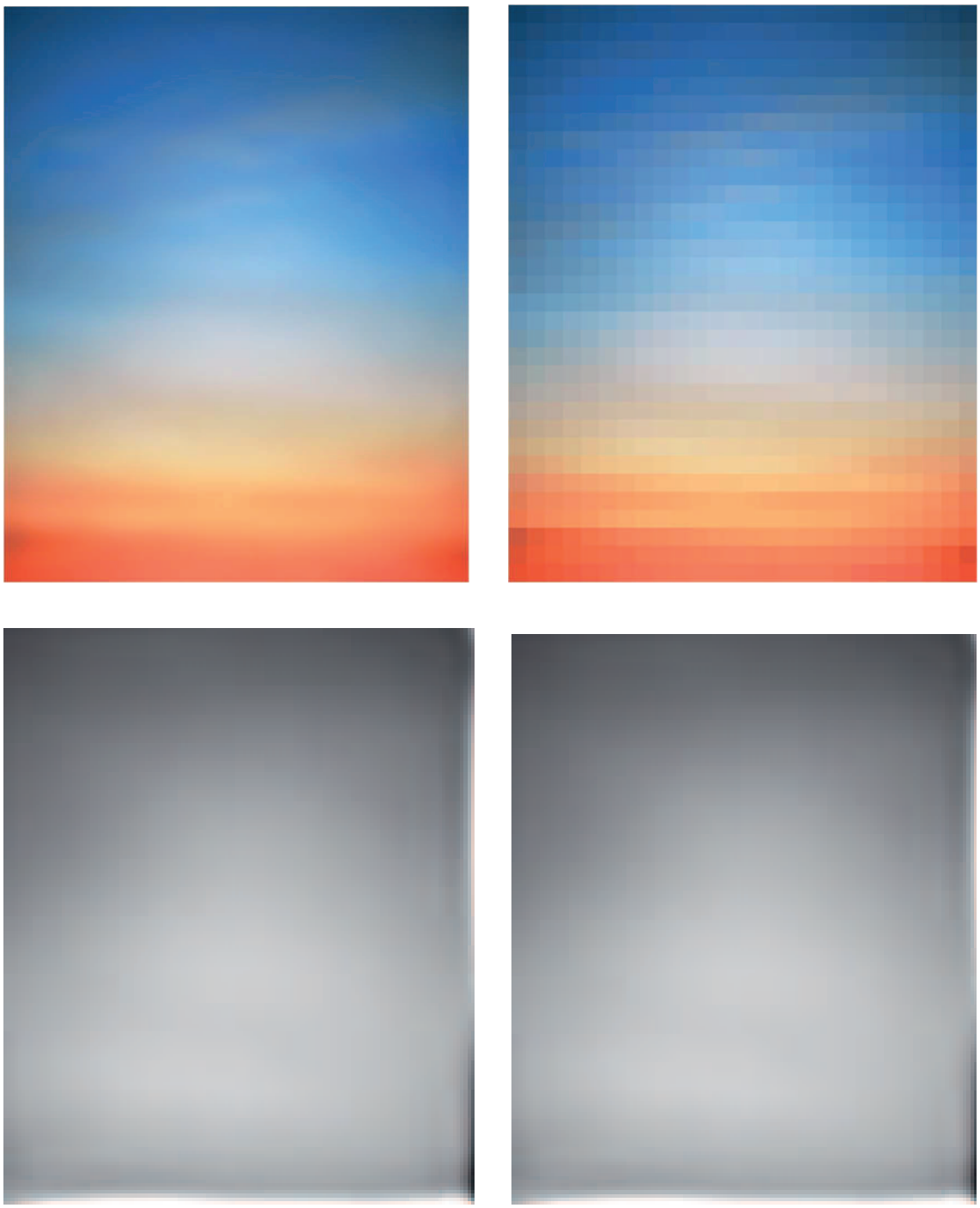
A tesztesetből látszik, ha egyre növeljük a fokszámot a kapott függvény hibája nő.

Másodfokú polinom, melynek minden egész (x,y) pontpárra egész értéket ad.(azaz nincs kerekítésből számított hiba) nagyítás: 15x fok 1,2,30 (laphoz igaítva)

Harmadfokú polinom abszolút értékkel kompozícióban nagyítás 15x fokszám: 3,6,10



Színes naplemente kép nagyítás x5 fokszám: 5,10 (bal első kép köbös interpolációval előállított)



A tesztelés során láthatjuk, hogy az alkalmazás nagyol jól közelít folytonos, átmenetes képeket. A legjobb eredményt akkor kapjuk, ha a mintának használt kép folytonos, illetve minden egész koordinátához 0 és 255 közötti értéket rendeltünk, azaz a mintánkon nem jelenik közelítésből származó hiba. A tesztelés során kiderült, hogy a kisebb nagyítások, illetve a kisebb polinomokkal való becslések gyorsak. A programnak polinomiális futásideje van. Óriási képekre nem hatékony. Az alkalmazás körülbelül maximálisan negyvened fokú polinommal tud dolgozni, efelett a GPU számítás NAN értéket ad vissza.

4 Irodalomjegyzék

[1] https://www.siam.org/students/siuro/vol1issue1/S01002.pdf

2018.05.09

[2] R. R. Hall, T. H. Jackson, A. Sudbery , K. Wild: Some advances in the no-three-in-line problem, Journal of Combinatorial Theory. Series A, 18 (3), 1974, 336–341

[3] http://en.cppreference.com/w/cpp/language/classes

2018.05.09

[4] http://en.cppreference.com/w/cpp/language/expressions

2018.05.09

[5] https://www.boost.org/doc/libs/

2018.05.09

[6] https://docs.opencv.org/3.4.1/

2018.05.09

[7] https://docs.nvidia.com/cuda/index.html

2018.05.09

5 Függelék

LU felbontás 15 kísérletből táblázatos formában. Növekvő paraméterekkel a Tesztelés résztől különböző

Specifikáció:

NVIDIA GeForce GTX 660

Intel® Core™ i5 4670 3.4GHz

Méret	**Fokszám**	**Nagyítás**	**Idő(átlag)**
23×23	1	10	0.2707
23×23	1	20	0.2505
23×23	1	30	0.2997
23×23	2	30	0.2978
23×23	3	30	0.3105
23×23	5	30	0.3271
23×23	10	30	0.3404
23×23	20	30	0.4743
27×27	1	10	0.2105
27×27	1	20	0.2458
27×27	1	30	0.3060
27×27	2	30	0.2917
27×27	3	30	0.3391
27×27	5	30	0.3500
27×27	10	30	0.3935
27×27	20	30	0.5712
35×35	1	10	0.2154

35×35	1	20	0.2712
35×35	1	30	0.3601
35×35	2	30	0.3697
35×35	3	30	0.3757
35×35	5	30	0.4006
35×35	10	30	0.4993
35×35	20	30	0.7973
52×52	1	10	0.2812
52×52	1	20	0.3867
52×52	1	30	0.5584
52×52	2	30	0.5997
52×52	3	30	0.6644
52×52	5	30	0.7354
52×52	10	30	0.8574
52×52	20	30	1.5247
256×256	1	10	1.2040
256×256	1	20	4.0886
256×256	1	30	9.0381
256×256	2	30	9.7212
256×256	3	30	9.9123
256×256	5	30	11.4551
256×256	10	30	15.1473
256×256	20	30	30.547

QR felbontás 15 kísérletből.

Specifikáció:

NVIDIA 950M grafikus kártya

Intel I7 4720HQ up to 3.6GHz

Méret	**Fokszám**	**Nagyítás**	**Idő(átlag)**
23×23	1	10	0.3985
23×23	1	20	0.4287
23×23	1	30	0.4444
23×23	2	30	0.4295
23×23	3	30	0.4257
23×23	5	30	0.4487
23×23	10	30	0.4841
23×23	20	30	0.7112
27×27	1	10	0.3619
27×27	1	20	0.4250
27×27	1	30	0.4339
27×27	2	30	0.4425
27×27	3	30	0.4412
27×27	5	30	0.5296
27×27	10	30	0.5400
27×27	20	30	0.7639
35×35	1	10	0.3997
35×35	1	20	0.4120

35×35	1	30	0.5328
35×35	2	30	0.5012
35×35	3	30	0.5417
35×35	5	30	0.5858
35×35	10	30	0.6775
35×35	20	30	0.9158
52×52	1	10	0.3856
52×52	1	20	0.5078
52×52	1	30	0.7995
52×52	2	30	0.7911
52×52	3	30	0.7587
52×52	5	30	0.9090
52×52	10	30	1.3974
52×52	20	30	1.6032
256×256	1	10	1.3578
256×256	1	20	4.3251
256×256	1	30	9.2281
256×256	2	30	9.5573
256×256	3	30	10.8410
256×256	5	30	11.4510
256×256	10	30	15.0348
256×256	20	30	31.4278