Szakdolgozat

Lugosi László

Programozó informatikus szakirányú továbbképzési szak
Levelező tagozat

Kétirányú szórás-eloszlás függvény mérőkészülék vezérlő és adatfeldolgozó programjának fejlesztése

Neumann János Egyetem
GAMF Műszaki és Informatikai Kar
Kecskemét
2018

Neumann János Egyetem
GAMF Műszaki és Informatikai Kar
Informatika Tanszék

Adatlap a szakdolgozat beadásához

Név: Lugosi László Hallgatói kód: TVLETF

A szakdolgozat címe:

Kétirányú szórás-eloszlás függvény mérőkészülék vezérlő és adatfeldolgozó programjának fejlesztése

Tulajdonosi nyilatkozat

Ez a szakdolgozat a saját munkám eredménye. Dolgozatom azon részeit, amelyeket más szerzők munkájából vettem át, egyértelműen megjelöltem. A dolgozat elektronikusan feltöltött illetve bekötött változatának tartalma azonos. Ha kiderülne, hogy ez a nyilatkozatom valótlan, tudomásul veszem, hogy a záróvizsga-bizottság a záróvizsgáról kizár, és záróvizsgát csak új szakdolgozat készítése után tehetek. Hasznosítás esetén a szerzőnek és az NJE GAMF Karának külön megállapodást kell kötnie.

Kecskemét, 20.....

.....
hallgató

A külső konzulens véleménye a szakdolgozatról

....., 20.....

.....
külső konzulens

A belső konzulens véleménye a szakdolgozatról

Kecskemét, 20.....

.....
belső konzulens

A tanszéki bizottság véleménye a dolgozat elfogadásáról

Kecskemét, 20.....

.....
tanszékvezető

Neumann János Egyetem
GAMF Műszaki és Informatikai Kar
Informatika Tanszék

Szakdolgozati feladatlap

Név: Lugosi László Kód, munkarend: TVLETF, levelező tagozat

Szak: Programozó informatikus szakirányú továbbképzési szak

A szakdolgozat címe:

Kétirányú szórás-eloszlás függvény mérőkészülék vezérlő és adatfeldolgozó programjának fejlesztése

Az elkészítés helye: Optimal Optik Optikai, Elektronikai és Finommechanikai Tervező és Fejlesztő Kft.

Belső konzulens: Dr. Pásztor Attila
Beosztása: Főiskolai tanár

Külső konzulens: Dr Szarvas Gábor

A feladat részletezése:

1. Fizikai modellen alapuló mérési módszer kidolgozása, a szükséges matematikai formalizmus implemetálása
2. Azimutális és radiális irányokban alkalmazott Arduino léptetőmotorok COM-vezérléséhez modul létrehozása
3. FieldMax lézer teljesítmény mérő műszerrel USB alapú kommunikáció megvalósítása NI műszermeghajtó felhasználásával
4. Mérési adatok naplózása standard BSDF adatállományba és az előállított szóráskép megjelenítése hőtérképen
5. MFC felhasználói felület kialakítása, valamint diffúzor anyagokra a mérési adatok összevetése a Lambert-Phong modell eredményével

Kecskemét, 2018. március 19.

P. H.

....
belső konzulens tanszékvezető

Tartalomjegyzék

1. Bevezetés

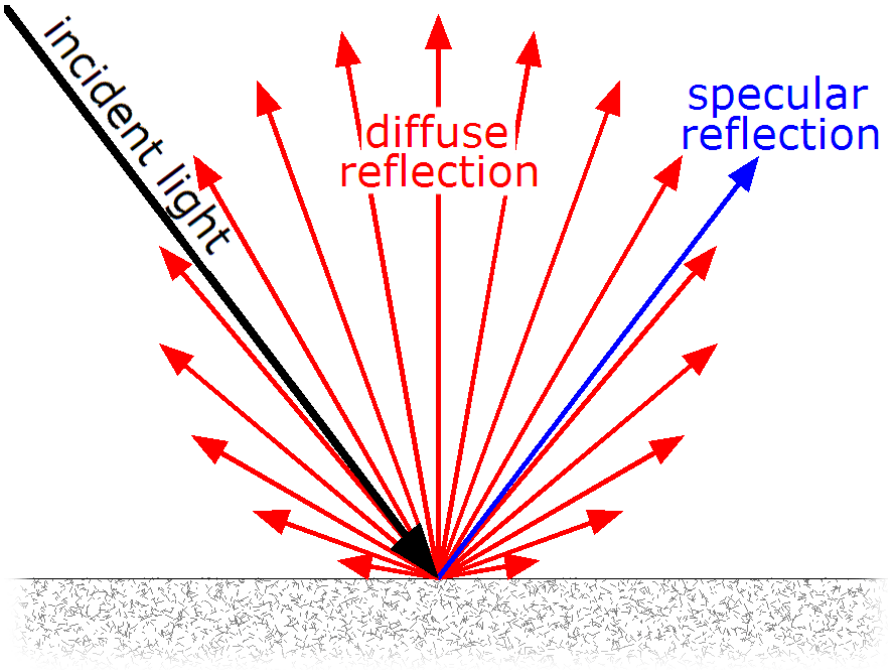
Az optikai anyagmodellek egy alapvető mennyisége a kétirányú szórás-eloszlás függvény (*Bidirectional Scattering Distribution Function, BSDF*), amely a fenomenologikus leírás szerint azt adja meg, hogy adott hullámhosszú fénynyaláb és adott beesési irány mellett mekkora a felületről visszavert (reflektált), illetve az anyagi rétegen áteresztett (transzmittált) normalizált fényerősség a vizsgált irányban. A BSDF első paramétere a fény bejövő irányát, a második paramétere pedig a kilépő irányt azonosítja szteradiánban. A számítógéppel létrehozott 3D grafika és a gépi látás területén nagy igény mutatkozik valósághű képek előállítására, valamint már elkészült felvételek utólagos módosítására. A képszintézist végző programokkal modell alapú virtuális világot definiálunk, ami tartalmazza a tárgyak helyét és geometriáját, a felületek optikai tulajdonságait, továbbá a fényforrások és kamerák helyzetét. Ezekből az adatokból az algoritmusoknak ki kell számítania a fény által szállított energiát több (legalább három) hullámhosszon, hogy a tárgy pontjainak színét megadhassák. A numerikus számítások elvégzéséhez a BSDF függvény ismeretére van szükség, mivel az - a hullámhossz, a megvilágítási és nézeti irányok mellett - minden felületi ponthoz megadja a visszaverő, illetve áteresztő képességet. A fotorealisztikus képszintézis megköveteli a megbízható és precíz műszerekkel mért BSDF adatok meglétét az alkalmazott anyagokra vonatkozóan. Ennek megvalósítása azonban eléggé költséges, ezért a számítógépes grafikában sokszor alkalmaznak erősen közelítő, illetve félempirikus BSDF modelleket, amelyek viszont megkönnyítik a számításokat. Az Optimal Optik Kft. munkatársaként részt vettem egy diffúzor anyagok szórásképét meghatározó BSDF mérőkészülék fejlesztésében. A feladatom ezen mérőrendszer mérésvezérlő és adatfeldolgozó szoftver moduljainak a megalkotása volt Microsoft Visual Studio fejlesztő környezetben, C/C++ programozási nyelvet alkalmazva. Dolgozatom első részében a megvalósított, elméleti fizikai modellen alapuló méréstechnikai módszert mutatom be. A detektálás irányát meghatározó gömbi polárkoordináta szögeket *Arduino* típusú léptetőmotorok segítségével állítjuk be. A folyamatos, kétirányú pásztázó mérést lehetővé tevő, soros portot (COM) használó vezérlő modult a második részben ismertetem. A beállított detektálási irányokban a szórt fényteljesítmény mérését egy *Coherent FieldMaxII-TO* típusú lézeres teljesítménymérő műszerrel végezzük. A műszerrel való USB alapú kommunikációt a National Instruments vállalat által forgalmazott műszermeghajtó tesz lehetővé, amelyhez kapcsolódó program

modulom működését a harmadik fejezetben részletezem. A negyedik rész a mérési adatok standard BSDF állományba történő naplózását, valamint a szóráskép hőtérképen való megjelenítését tárgyalja. Az ötödik részben az MFC (*Microsoft Foundation Class Library*) keretrendszerben elkészített asztali alkalmazás használatának bemutatása található, valamint egy adott diffúzor mintára kapott mérési adatok összevetése a Lambert-Phong modell eredményével.

2. Az alkalmazott méréstechnikai módszer

2.1. Elméleti háttér

Ideális tükröződés esetében csak a visszaverődési törvény szerinti irányban történik fényvisszaverés. Diffúz anyagok a beérkező fényt minden irányban egyenletesen szétszórják, vagyis a felület színe nem függ a nézőponttól. Nagy felületi érdességű fényvisszaverő anyagokról sem tisztán reflexív módon, sem tisztán diffúz módon verődik vissza a fény, hanem spekulárisan, azaz a beeső fény nagy hányada az ideális tükröződési irányába és annak meghatározott környezetébe verődik vissza (a gyártástechnológiának megfelelően) és a fényintenzitás eloszlása irányfüggő [1][2][3].



1. ábra. Diffúz szórás és spekuláris visszaverődés [4].

A fény alapvető radiometriai mennyisége az adott hullámhosszon λ egységnyi idő alatt kisugárzott fényenergia a tér egy adott $\boldsymbol{r}$ helyvektorral megadott pontjában, a sugárzási fluxus [1, 2]:

$$\Phi_{e,\lambda}(\mathbf{r}) \equiv P_{\lambda}(\mathbf{r}) = \frac{\mathrm{d}Q_{e,\lambda}(\mathbf{r})}{\mathrm{d}t}. \tag{1}$$

A sugárzási fluxus meghatározott irányban és határfelületen átadott része a radiometriai fényerősség [1, 2]

$$\mathrm{I}_{\mathrm{e},\lambda}(\mathbf{r}) = \frac{\mathrm{d}\Phi_{\mathrm{e},\lambda}(\mathbf{r})}{\mathrm{d}\,\Omega}. \tag{2}$$

ahol a határfelület kiterjedését a térszög határozza meg. A sugárzási fluxus egységnyi merőleges felületre eső része a radiometriai kisugárzás, másnéven irradiancia [1, 2]:

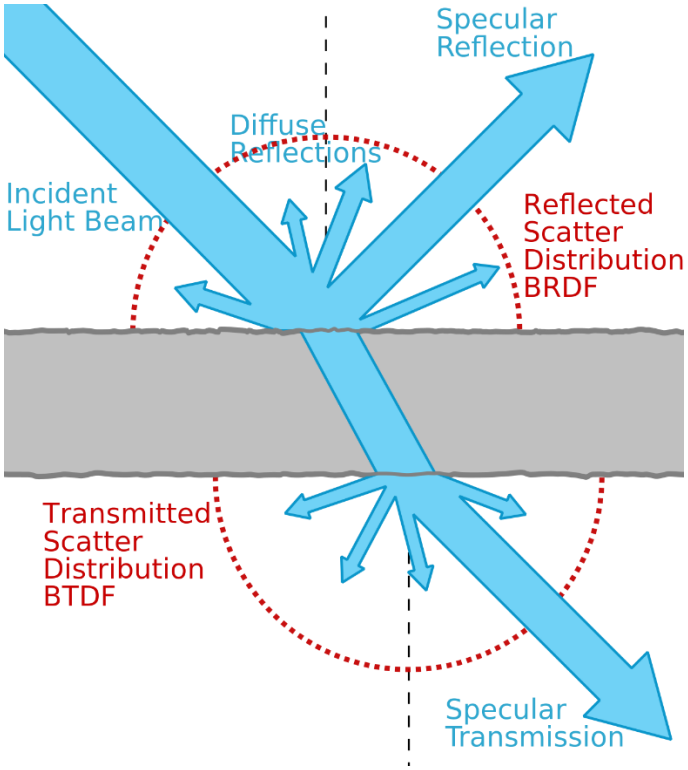
$$\mathrm{E}_{\mathrm{e},\lambda}(\mathbf{r}) = \frac{\mathrm{d}\Phi_{\mathrm{e},\lambda}(\mathbf{r})}{\mathrm{d}\,A}. \tag{3}$$

A sugárzási fluxus egy pontra eső része a radiometriai fénysűrűség, illetve radiancia, amely a fényerősség és a térszög irányából látható felület vetületének hányadosa [1, 2]:

$$\mathrm{L}_{\mathrm{e},\lambda}(\mathbf{r},\Omega) = \frac{\mathrm{dE}_{\mathrm{e},\lambda}(\mathbf{r})}{\mathrm{d}\Omega} = \frac{\mathrm{d}^2\Phi_{\mathrm{e},\lambda}(\mathbf{r})}{\mathrm{d}\Omega\mathrm{d}A_{\perp}} = \frac{\mathrm{d}^2\Phi_{\mathrm{e},\lambda}(\mathbf{r})}{\mathrm{d}\Omega\mathrm{d}A\cos\Theta}. \tag{4}$$

A különböző anyagok tükrözik, szétszórva visszaverik, megtörik vagy elnyelik a fénysugarakat. Ezeket a jelenségeket mind visszavert, mind átmenő λ hullámhosszúságú fényre a kétirányú szórás eloszlás függvénnyel írhatjuk le elméletileg, ami a következő alakban van definiálva [1, 2, 3]:

$$\mathrm{BSDF}(\Phi_{\mathrm{i}},\Theta_{\mathrm{i}};\Phi_{\mathrm{s}}\Theta_{\mathrm{s}}) \doteq \frac{dL_s(\Omega_i,\Omega_s;E_i)}{dE_i(\Omega_i)} = \frac{\mathrm{d}L_s(\Omega_i,\Omega_s;E_i)}{L_i(\Omega_i)\cos(\Theta_i)\mathrm{d}\Omega_i}, \tag{5}$$



2. ábra. Reflexióban (BRDF) és transzmisszióban (BTDF) vizsgálható szóráseloszlások [5].

ahol $\Omega_i = (\Phi_i, \Theta_i)$ a beeső, $\Omega_s = (\Phi_s, \Theta_s)$ pedig a kilépő térszögek gömbi polárkoordináta rendszerben, valamint L_i a belépő, L_s a kilépő felületi fényáram sűrűségek. Ezen négydimenziós numerikus függvény az anyagi jellemzőknek megfelelően arról ad számot, hogy bizonyos irányból beérkező foton mekkora valószínűséggel lép ki a

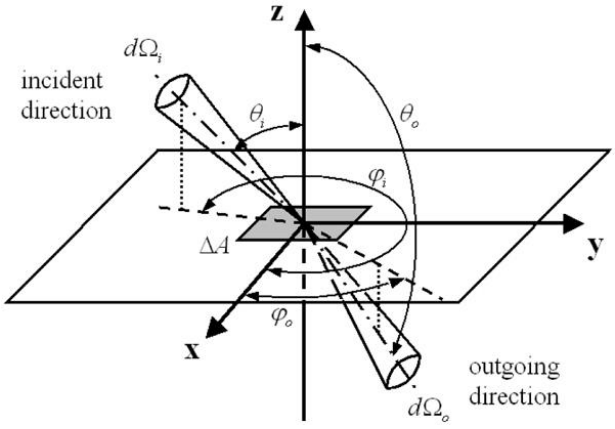
megfigyelési irányba spekuláris reflexiót (*BRDF*), illetve transzmissziót (*BTDF*) követően (2. ábra).

2.2. Gyakorlati megvalósítás

Egy adott konfiguráció $\{\Phi_i \equiv 0^\circ,\ \Theta_i = AOI;\ \Phi_s = \Phi_{az},\ \Theta_s = \Theta_{rad}\}$ - itt *AOI* a beesési szöget (*Angle Of Incidene*) jelöli, míg Φ_{az} és Θ_{rad} a detektálás pozíciójának azimutális és radiális szögeit - mellett a BSDF numerikus értékeit a mérések során a fenti (5) definícióból következő átlagos értékkel közelítjük a [3] munkában megadott összefüggés szerint,

$$BSDF_{meas}(AOI;\Phi_{az},\Theta_{rad}) \approx \frac{P_{meas,s}(AOI;\Phi_{az},\Theta_{rad})/\Omega_s}{P_{meas,i}} = \frac{R_D^2}{A_D P_{meas,i}} \cdot P_{meas,s}(AOI;\Phi_{az},\Theta_{rad}) \quad (6)$$

mivel a gyakorlatban véges nyaláb foltméret, detektor apertúra felület (A_D) és térszög értékek (Ω_s) állnak rendelkezésre adott minta és detektor távolság (R_D) mellett. A (6) kifejezésben $P_{meas,i}$ és $P_{meas,s}$ a bejövő és szórt fénynyalábokhoz tartozó teljesítményeket jelöli, továbbá $\Omega_s = A_D / R^2_D$. A szórt fényteljesítmény mérésekor a különböző azimutális és radiális szögpárokat az ideális fényvisszaverődés, illetve fényáteresztés irányához, az úgynevezett *direkt* irányhoz viszonyítva állítjuk be, amelyet egyértelműen meghatároz a beesési szög.



3. ábra. Mérések során alkalmazott gömbi polárkoordináta rendszer [3].

A természetes félgömbi (*HemiSperical, HS*) és a szórásképről több információt nyújtó direkt (*Directional, D*) gömbi polárkoordináta rendszerek között az alábbi (7) transzformációs képlettel számíthatok át az egyes szög pozíciók koordinátái:

$$\underline{\mathrm{P}}_{\mathrm{Direct}} = \left[\underline{\underline{R}}^{\mathrm{T}}(-\mathrm{AOI})\right]^{-1} \underline{P}_{\mathrm{HS}} \Rightarrow \underline{\mathrm{P}}_{\mathrm{HS}} = \underline{\underline{R}}(\mathrm{AOI})\underline{\mathrm{P}}_{\mathrm{Direct}} \quad (7)$$

Itt $P_{HS} = [X_{HS}\ Y_{HS}\ Z_{HS}]$ a *HS*, és $P_D = [X_D\ Y_D\ Z_D]$ a *D* koordináta rendszerekben a pozíció vektorok, ahol

$$\underline{\mathrm{P}}_{\mathrm{Direct}} = [\mathrm{X_D Y_D Z_D}]^{\mathrm{T}} = \begin{bmatrix} \sin(\Theta_D)\cos(\Phi_D) \\ \sin(\Theta_D)\sin(\Phi_D) \\ cos(\Phi_D) \end{bmatrix}, \tag{8}$$

$\underline{\underline{R}}$ pedig az $\boldsymbol{R}$*: HS → D* *Y*-tengely körüli *AOI* szögű forgatás operátorának mátrixa Descartes reprezentációban felírva:

$$\underline{\underline{R}}(\mathrm{AOI}) = \begin{bmatrix} cos(AOI) & 0 & sin(AOI) \\ 0 & 1 & 0 \\ -sin(AOI) & 1 & cos(AOI) \end{bmatrix} \tag{9}$$

A (Φ_{HS}, Θ_{HS}) azimutális és radiális szögek ennek megfelelően a következő formulákkal számíthatóak:

$$\Phi_{\mathrm{HS}} = arg\big(\mathrm{X_{HS}}(\Phi_D, \Theta_D), \mathrm{Y_{HS}}(\Phi_D, \Theta_D)\big),\ \ \Theta_{\mathrm{HS}} = arccos\big(\mathrm{Z_{HS}}(\Phi_D, \Theta_D)\big). \tag{10}$$

A BSDF felvétele során pásztázó (*2D-scanning*) méréstechnikát alkalmaztunk, azaz a direkt azimutális és radiális szögek értékét egy-egy előre megadott $[\Phi_{D,min} \ldots \Phi_{D,max}]$ és $[\Theta_{D,min} \ldots \Theta_{D,max}]$ tartományban a megválasztott $[\Delta\Phi_D,\ \Delta\Theta_D]$ lépésközök szerint változtattuk:

$$\Phi_{D,i} = \Phi_{\mathrm{D,min}} + (\mathrm{i} - 1) \cdot \Delta\Phi_{\mathrm{D}}, \quad \mathrm{i} = 1, \ldots, \mathrm{N} \tag{11}$$

$$\Delta\Phi_{\mathrm{D}} = \frac{\Phi_{\mathrm{D,max}} - \Phi_{\mathrm{D,min}}}{\mathrm{N}-1} \tag{12}$$

$$\Theta_{\mathrm{D},j} = \Theta_{\mathrm{D},min} + (j - 1) \cdot \Delta\Theta_{\mathrm{D}}, \quad j = 1, \ldots, M \tag{13}$$

$$\Delta\Theta_{\mathrm{D}} = \frac{\Theta_{\mathrm{D,max}} - \Theta_{\mathrm{D,min}}}{\mathrm{M}-1}, \tag{14}$$

majd ezen $(\Phi_D,\ \Theta_D)$ szögérték párokból a fenti (7)-(10) transzformációs egyenletek segítségével meghatároztuk azon félgömbi $(\Phi_{HS},\ \Theta_{HS})$ polár koordinátákat, melyek kijelölik a mérési pontokat a BSDF adatok felvételéhez. A (11)-(14) egyenletekben *N* és *M* a mérés során beállított különböző azimutális, illetve radiális irányú szögek számát jelöli. Síkszimmetriával bíró, illetve aszimmetrikus mintákra a direkt azimutális szög értékkészlete:

$$PlaneSymmetrical\colon \quad 0° \leq \Phi_{\mathrm{D}} \leq 180° \tag{15}$$

$$Asymmetrical\colon \quad 0° \leq \Phi_{\mathrm{D}} \leq 360° \tag{16}$$

Míg a radiális szögtartomány mindkét esetben:

$$0° \leq \Theta_{\mathrm{D}} \leq 90° \tag{17}$$

A koordináta transzformáció során előállhat, hogy a *HS* radiális szög értéke nagyobb a (17) által definiált maximális értéknél. Ilyenkor a mért teljesítmény értékét azonosan zérusnak írjuk elő:

$$P_{meas}(\Phi_{HS}, \Theta_{HS} > 90°) \equiv 0. \quad (18)$$

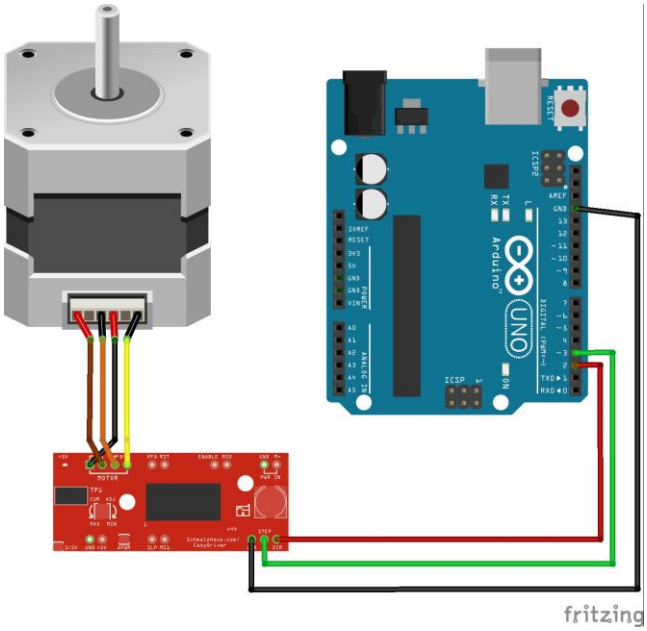
A BSDF mérőkészülékben alkalmazott *Arduino* típusú azimutális és radiális léptető motorok a mintatartó asztalt és az optikai mérőfejet az alábbi táblázat szerinti „inverz”

$$\left\{\underline{\underline{\Phi}}_{HS}, \underline{\underline{\Theta}}_{HS}\right\}_{NxM} = \begin{bmatrix} (\Phi_{11}, \Theta_{11}) & \cdots & (\Phi_{1N}, \Theta_{1M}) \\ \vdots & \ddots & \vdots \\ (\Phi_{N1}, \Theta_{N1}) & \cdots & (\Phi_{NM}, \Theta_{NM}) \end{bmatrix} \quad (19)$$

pozíciókba állítják be, majd a pozícionálási időnek megfelelő késleltetés után egy *Coherent FieldMaxII-TO* típusú lézer teljesítménymérő műszer a szórt fényteljesítményt meghatározza. Fényforrásként egy Optotronic gyártmányú LED modul szolgál. A különböző beesési szögek mellett elvégzett mérések eredményeit szabványos formátumú BSDF állományba naplózzuk. Ezeket az adatállományokat a vállalatunknál is használt ZEMAX [8] optikai tervező szoftver képes beolvasni, és a bennük lévő tabulált adatokat felhasználni olyan modellszámításokhoz, melyek egy tárgy felülete szórási tulajdonságainak a meghatározását célozzák. A mérésvezérlést és adatfeldolgozást egy általam fejlesztett C/C++ alkalmazás végzi. Az aktuális mérési eredmények táblázatos és grafikus megjelenítésén kívül ezen program felhasználói felületének segítségével állíthatóak be a pásztázási tartományok és mérési paraméterek a vizsgálni kívánt minták esetében.

3. Kétirányú pásztázó mérések megvalósítás léptetőmotorokkal

A fényteljesítmény adatok mérésekor az optikai szenzort a detektálás irányát meghatározó azimutális és radiális szögeknek megfelelő helyzetbe kell pozícionálni, amelyet két *Arduino* típusú léptetőmotor segítségével végzünk el (4. ábra). Az általunk alkalmazott megoldásban az első léptetőmotor a vízszintes helyzetben lévő tárgyasztalt forgatja, ahol is a vizsgálandó minta van rögzítve. Míg a második léptetőmotorhoz egy mechanikus tartó keretet képes elforgatni amire a szenzort helyeztük el. A tartókeret síkja a kezdeti, kiinduló pozícióban merőleges a tárgyasztal síkjára.

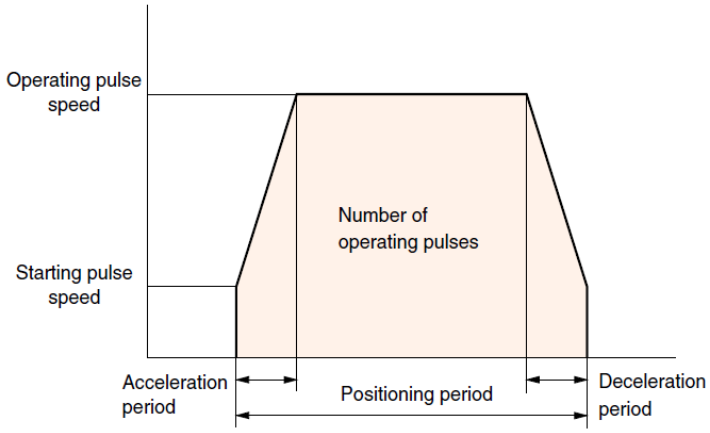


4. ábra. Megfigyelési irányt beállító Arduino léptetőmotor [6].

A léptetőmotorokról általánosságban elmondható, hogy mikrokontrollerek segítségével vezérelhető elektromágneses aktuátorok, melyek a kiadott jel (vezérlő impulzus) hatására előre meghatározott pozícióba állítják be a forgórészüket. Lényegi feladatuk a folyamatos forgás helyett a megadott szögelfordulással járó lépések végrehajtása. A léptetőmotorok fontos tulajdonsága a felhasználási szempontból, hogy egy teljes körül forduláshoz hány lépés (impulzus) szükséges. Ez megadja az egy lépésre eső szögelfordulás nagyságát az úgynevezett a lépésszöget, amelynél kisebb szögbe nem képes a léptetőmotor a beállítást elvégezni, vagyis csak diszkrét helyzetekben állhat meg. Természetesen fontos paraméterek ezen kívül a motor terhelhetősége, teljesítménye és nyomatéka is, amelyeket szintén figyelembe vettünk az alkalmazni kívánt típus kiválasztásakor. A léptetőmotorok előnye, hogy a határértéken belüli terhelés esetén a megfelelő pozícióba fordulnak, viszont

hátrányaik közé sorolható, hogy szögsebességük nem állandó a gyorsítási, illetve lassítási időtartományok alatt (5. ábra). A BSDF mérőkészülékbe beépített Arduino léptetőmotorok mikrovezérlőjének hardver közeli C programját villamosmérnök kollégám írta meg.

Az általunk használatba vett léptetőmotorok által beállítható legkisebb szögelfordulás 0.15 fok, ami megfelel a céljainknak, mert 0.5 foknál kisebb lépésközben nem kívánunk 2D-scanning méréseket végezni. Mivel nincs visszajelzés arról, hogy a léptetőmotorok az aktuális pozíciójukból beálltak a kívánt pozíciókba, ezért kísérleti úton kellett meghatároznom az ehhez szükséges időt. Ehhez a $t_{total} = (N_{az}\ t_{az} + N_{rad}\ t_{rad}) / N_{rev} + t_{delay}$ összefüggést használtam fel, ahol N_{az} és N_{rad} az azimutális, illetve radiális lépésszámok; t_{az} és t_{rad} az egy körül-forduláshoz szükséges lemért időtartamok, N_{rev} (=360˚/0.15˚) pedig az egy körül forduláshoz tartozó lépésszám. A t_{delay} szintén lemért, járulékos időtartam azt veszi figyelembe, hogy a mozgó keret a léptetőmotor leállása után rövid idejű csillapodó lengést végez.



5. ábra. Léptetőmotor frekvencia karakterisztika [6].

2D pásztázó mérés során a mérést végző személy számára biztosítani kell, hogy a kialakított felhasználó felületen be tudja állítani az azimutális és radiális szögek egy-egy értéktartományát, valamit az egyes lépésközöket fokban megadva. Ezért szükség van egy olyan köztes szoftver modulra, ami kapcsolati réteget valósít meg a felhasználói felület adatbeviteli és vezérlő egysége, valamit a léptetőmotorok mikrovezérlő programja között. Ezen általam fejlesztett modul feladatai közé tartozik, hogy soros (COM) port kommunikációt használva -a mérőszámítógép és az Arduino mikrovezérlője között- a forgásirányra és lépésszámra vonatkozó adatokat küldjön a léptetőmotorok vezérlő programjának. Továbbá tárolnia kell a léptetőmotorok aktuális pozícióit, és minden sorban következő azimutális és radiális szöggel meghatározott detektálási helyzet esetében az újra

számolt lépésszámokat, illetve frissített forgási irányokat kell elküldenie. Ebben a „*MotorControl.h*” és „*MotorControl.cpp*” forráskódok által implementált egységben található függvények működését fogom ismertetni a következőkben, kitérve a használt adatszerkezetekre is.

A lépetetőmotor vezérlő modul szigorúan ellenőrzi, hogy a megadott azimutális és radiális szögek a helyes értéktartományukon belül változhassanak, különben előfordulhatna, hogy az optikai szenzort mozgató keret a tárgyasztal vízszintes síkja alá mozogjon, ugyanis erre nincs hardveres védelem. A fénynyaláb beesési szöge, amit manuálisan kell beállítani (és értékét felhasználói felületen megadni) szintén nem haladhatja meg a 85 fokot, mivel az ilyen közel súrlódó beesést a berendezés nem képes kezelni. Továbbá pásztázó méréskor nem állítható be olyan azimutális vagy radiális lépésköz, ami kisebb, mint a léptetőmotor szöglépése. A BSDF alkalmazás lehetőséget biztosít -az összes egyéb mérési beállítást változatlanul hagyva- egymás után több beesési szögre való mérésre, de korlátozza ezek számát. Ezen túlmenően a mérési pontok számára (ami kis szög-lépésköz megadása esetén igen nagy is lehetne) is van felső korlát, mivel a léptetőmotorok pozicionálása időt vesz igénybe, melyek összeadódva aránytalanul megnövelhetik a mérési időt. A vezérlő modulban a két léptetőmotor kezdeti, kiindulási pozícióját a [0, 0] lépésszámok definiálják. Ugyanekkor az ezen „parkírozó” helyzethez tartozó radiális és azimutális szögek a [$\Phi_{az,init}$, $\Theta_{rad,init}$] értékeket veszik fel, ami a berendezés alapbeállításától függ (a modul fejléc állományában ezek konstansként szerepelnek).

Mivel a mérőkészülék ipari körülmények között, hosszú időtartamokig folyamatosan használatban lehet, ezért nagy hangsúlyt helyeztem a hibakezelésre. Minden külső modulból hívott függvény visszaad egy státusz kódot, ami a „*Global.h*” fejléc állományban definiált (*Success = 0, Warning = 1, Failure = -1*) értékeket veheti fel. Hiba fellépésekor a hívott függvény beállít egy a modulban statikus egésznek deklarált hibakód váltózót, ami a hibát azonosítja. Ha a státusz kód nem „*Success*” értékű, akkor hívó függvény köteles lekérdezni a hibaüzenetet a modul *GetMCErrorMsg* függvényével és azt automatikusan naplózni a „*Global.h*” állományban definiált *WriteToLogFile* függvény segítségével, ami az alkalmazás nevével azonos és „*log*” kiterjesztésű szöveges állományba írja a hibaüzenetet. A *GetMCErrorMsg* függvény a hibakód alapján olvassa ki egy listából a megfelelő hibaüzenetet. Ha WinAPI függvény hívás esetén történik hiba, akkor szintén a „*Global.h*” fejléc állományban definiált *GetLastErrorMsg* függvénytől a modulban lekért rendszerszintű hibaüzenettel is kiegészül a *GetMCErrorMsg* által szolgáltatott információ.

Ezen hagyományos, C nyelvben megszokott hibakezelés mellett a C++ nyelv nyújtotta *try{...}catch(...){...}* blokkok használatára épülő haladottabb kivételkezelést is alkalmazom a vezérlő modul függvényeit hívó programrészben. Kritikus hiba („*Failure*”) esetében a felhasználó üzenőablakban kap értesítést a hibáról. A pásztázó méréstechnikát lehetővé tevő függvények hierarchikus kapcsolatban állnak egymással, azaz megfelelő sorrendben kell őket hívni, annak érdekében, hogy biztosítva legyen az egymás közötti helyes adatátadás, illetve a COM porton történő adatküldés. Ezen hívási lánc sorrendje is ellenőrizve van az *SFlags* bitmező segítségével, ami a „*MotorControl.cpp*” forráskódban van definiálva és a hívott függvények ezen mezőket tudják lekérdezni, illetve beállítani.

A felhasználói felületről származó adatoknak és a modul függvényei elvárt működésének az ellenőrzését az alább definiált konstansok teszik lehetővé.

Konstans definíciók:

#define

NUMOFMOTORS 2

Léptetőmotorok száma.

NUMOFCOMMANDS 2

Léptetőmotor mikrovezérlő programjának küldendő parancsok száma.

MINPHI 0.0

Azimutális szög értéktartományának alapértelmezett alsó határa ($\Phi_{az,min} = 0^{\circ}$),

($\Phi_{az,min} <= \Phi_{az}$).

MAXPHI 360.0

Azimutális szög értéktartományának alapértelmezett felső határa ($\Phi_{az,max} = 360^{\circ}$),

($\Phi_{az} <= \Phi_{az,max}$).

MINTHETA -90.0

Radiális szög értéktartományának alapértelmezett alsó határa ($\Theta_{rad} = -90^{\circ}$),

($\Theta_{rad,min} <= \Theta_{rad}$).

MAXTHETA 90.0

Radiális szög értéktartományának alapértelmezett felső határa ($\Theta_{rad} = 90^{\circ}$),

($\Theta_{rad} <= \Theta_{rad,max}$).

INIPHI 0.0

Azimutális szög alapértelmezett kezdeti értéke ($\Phi_{az,init} = 0^{\circ}$).

INITHETA 0.0

Radiális szög alapértelmezett kezdeti értéke ($\Theta_{rad,init} = 0^{\circ}$).

```
MINAOI 0.0
```

Manuálisan beállítandó beesési szög megengedett minimális értéke ($AOI_{min} = 0^{\circ}$).

```
MAXAOI 85.0
```

Manuálisan beállítandó beesési szög megengedett maximális értéke ($AOI_{min} = 85^{\circ}$).

```
DEGSTEPMIN 0.15
```

Felhasználói felületen beállítható legkisebb szög-lépésköz pásztázó mérés során.

```
DEFSETP 10.0
```

Felhasználói felületen beállított alapértelmezett szög-lépésköz pásztázó mérés során.

```
MAXNUMOFAOI 256
```

A mérés előtt beállítandó fénynyaláb beesési szögek maximális száma.

```
MAXMEASPOINT 16384
```

Az összes mérési pont maximális számára felső korlát.

Felsorolás típusok:

```
enum EMotorID : int { First = 0, Second = 1 };
```

Az első (azimutális szöget) és a második (radiális szöget) beállító léptetőmotor azonosítója.

```
enum EDirection : int { Forward = 0, Backward = 1 };
```

Léptetőmotor előre, illetve hátra irányba történő forgására vonatkozó igényt azonosítja.

Adattárolók:

```
typedef struct SAngleData
{       float MinPhi, MaxPhi, MinTheta, MaxTheta;
        float InitPhi, InitTheta;
        int NminPhi, NmaxPhi, NminTheta, NmaxTheta;
        SAngleData
        (
        float MinPhiValue = MINPHI, float MaxPhiValue = MAXPHI,
        float MinThetaValue = MINTHETA, float MaxThetaValue = MAXTHETA,
        float InitPhiValue = INIPHI, float InitThetaValue = INITHETA
        ) {...}
        } TSAngleData;
```

Struktúra típus, ami az azimutális és radiális szögek értéktartományát, a kezdeti értéküket és az ezekhez tartozó lépésszámok értékeit tárolja.

```
typedef const char Tcommand[NUMOFMOTORS][NUMOFCOMMANDS];
```

Kétdimenziós konstans tömb, ami a mikrovezérlő programjának COM porton kiküldendő parancs kódokat tárolja. Az első indexet az EMotorID, míg a második indexet az EDirection felsorolás típusú változókkal állíthatjuk be.

```
static TCommand Commands =
{ // Forward (előre forgás), Backward (hátra forgás)
        { 't', 'g' }, // First Motor (azimutális szög)
        { 'r', 'f' } // Second Motor (radiális szög)
};
```

```
typedef int TPosition[NUMOFMOTORS];
```

A léptetőmotorok kezdő pozíciótól számított aktuális lépésszámát tároló statikus tömb.

A vezérlő modulban megvalósított függvények:

```
int DegRangeToStep(const float& MinDeg, const float& MaxDeg);
```

Meghatározza, hogy hány lépésszám szükséges ahhoz, hogy a léptetőmotor segítségével a paraméterként fokban megadott kezdőhelyzetből a véghelyzetbe jussunk.

```
void InitPositions(void);
```

A léptetőmotorok aktuális lépésszámát a kezdő pozíciónak megfelelő értékekre állítja.

```
bool SetStepValues(TSAngleData& AngleData);
```

Beállítja a léptetőmotorok adott kezdő pozíciótól számított lépésszámait a *TSAngleData* struktúrában tárolt azimutális és radiális szögtartományoknak, valamint azok kezdeti értékeinek megfelelően a *DegStep* (lépésszög) konstans értéket felhasználva. A visszatérési érték igaz, ha számolás eredménye helyes, egyébként hamis.

```
bool PhiToStep(const TSAngleData& AngleData, const float& Phi, int& NPhiStep, EDirection& PhiDirection);
```

bool ThetaToStep(const TSAngleData& AngleData, const float& Theta, int& NThetaStep, EDirection& ThetaDirection);

Ezen két függvény meghatározza, hogy melyik irányban ás hány lépést kell végezni az első/második léptetőmotornak, ahhoz, hogy beálljon a megadott azimutális/radiális szögbe. Ehhez felhasználja az *TAngleData* struktúrában tárolt adatokat, valamint a *TPosition* típusú tömbben lévő aktuális azimutális/radiális pozíció értéket. A visszatérési érték igaz, ha számolt értékek helyes értéktartományba esnek, egyébként hamis.

int InitCOMPort(LPCWSTR pcCommPort);

Standard WinAPI függvényeket használva megnyitja és inicializálja a számítógép és léptetőmotorok mikrovezérlője közötti kommunikációra használt soros portot.

int MoveMotor(const TSAngleData& AngleData, EMotorID MotorID, EDirection Direction, int NumOfStep);

A kiválasztott léptetőmotorral rögzített forgási irányban megadott számú léptetést hajtat végre. Ennek során COM porton keresztül adja át a mikrovezérlő programnak a *TCommand* típusú mátrixból kiolvasott parancsot, valamint a lépésszám értékét, majd aktualizálja *TPosition* típusú tömbben a motor pozíció értékeket.

int ResetPositions(const TSAngleData& AngleData);

Az *AngleData* struktúraban tárolt *InitPhi* és *InitTheta* értékek által definiált kiinduló helyzetű pozícióba állítja mindkét léptetőmotort, majd hívja az *InitPositions* függvényt.

int GetPositioningTime(int NPhiStep, int NthetaStep);

Meghatározza, hogy a léptetőmotorok az aktuális helyzetükből várhatóan mennyi idő elteltével állnak be a kívánt stabil véghelyzetbe.

int CloseCOMPort(void);

Standard WinAPI függvényeket alkalmazva zárja a soros port kommunikációt.

std::string GetMCErrorMsg(void);

A belső hibakódnak megfelelő hibaüzenetet adja vissza.

4. A szórt fényteljesítményt mérő műszer vezérlése

A BSDF adatok előállításához szükséges megmérni, hogy a léptetőmotorok segítségével beállított, adott detektálási irányokban mekkora a diffúzor anyag által reflexió vagy transzmisszió útján szórt fény teljesítménye. Erre az optikai mérésre természetesen a léptetőmotorok pozicionálási idejének letelte után kerülhet sor minden egyes mérési pontban, amelyet az azimutális és radiális szögpárok határoznak meg. A szórt fényteljesítmény méréséhez egy *Coherent FieldMaxII-TO* (6. ábra) típusú lézer-teljesítmény mérő műszert használtunk fel. A mérésvezérlő számítógép és a műszer közötti adatátvitel USB kommunikációs csatornán keresztül történik. Ahhoz, hogy saját C++ modulból lehessen vezérelni a műszert, a mérőgépre telepíteni kell a National Instruments cég által fejlesztett műszermeghajtó szoftver környezetet. Továbbá csatolni kell a mérésvezérlő modult megvalósító projekthez a *FieldMax2Lib.dll* dinamikus hivatkozású könyvtárat (*Dynamic Link Library, DLL*), valamint a *FieldMax2Lib.h* fejléc állományt, amelyben a DLL-ben tárolt függvények prototípusai, illetve felhasználói adattípus-deklarációk találhatóak. Ez a kapcsolódási felületet teszi lehetővé a mérő számítógépen futó alkalmazás és a FieldMaxII-TO műszer közötti adatátvitelt. Az általam fejlesztett műszer vezérlő modul ezt a DLL-t tölti be a memóriába és annak függvényeit hívva szolgáltatja a mért fényteljesítmény adatokat.

6. ábra. Coherent FieldMaxII-TO lézert eljesítmény mérőműszer [7].

A FieldMaxII-TO lézer teljesítménymérő műszer optikai szenzorok széles választékával működtethető és ezeket használva lehetővé teszi az ultraibolya (UV), látható és infravörös (IR) lézerfény teljesítmények mérését a nW-tól a kW-os tartományig.

Pontos BSDF adatok méréséhez a külső zavaró fényforrások hatásától mentesíteni kell készüléket árnyékoló burkolattal (ideális, ha sötétszobai körülményeket biztosítunk). A mérések megkezdése előtt célszerű a műszeren az *Auto* gombbal beállítani azt, hogy automatikusan a mért értékekhez igazodó legalkalmasabb méréshatárt válassza. A *Zeroing* gombbal a háttérfény korrekcióját végezhetjük el, ami megemeli a nulla teljesítmény értékhez tartozó szintvonalat. Ez biztosítja, hogy az optikai szenzor az alacsony fényteljesítmények mérésénél is pontos adatokat szolgáltasson.

A teljesítmény mérést megvalósító osztály feladata, hogy dinamikusan importálja a gazda interfész DLL-t, majd csatlakozzon a mérőműszerhez annak az operációs rendszer által kiosztott egyedi objektum azonosítóját (*handle*) megszerezve. Ezután végeztesse el az USB újra szinkronizálást. Minden tárgyasztal és optikai mérőfej pozicionálás befejezése után kérje le a teljesítmény mérő műszertől az aktuális mérési adatot és adja át azt az adattárolást végző objektumnak. Az adott mintára történő minden beesési szögre való pásztázó mérés elvégzése után bontsa a kapcsolatot a műszerrel és szabadítsa fel a DLL által lefoglalt memóriát. A mérésvezérlő osztály fel van készítve arra is, hogy egy mérési pontban ne csak egy teljesítmény adatot, hanem egy előre meghatározott számú adat sorozatot kérjen le a műszertől. Ebben az esetben a mérési adatok egy dinamikus tömbben vannak tárolva (és az adatokból számolt átlagérték javíthatja pontosságot a mérési pontokban). Amennyiben sikertelen az adatlekérés - azaz nem érkezik mérési adat a műszertől, vagy az érvénytelen értékű (*NAN*) - akkor az egy adott időkorlátig (*timeout: 5s*) többször egymás után megismétlődik.

A DLL dinamikus importálása esetében a betöltés és felszabadítás műveletéhez WinAPI függvényeket és rendszer specifikus típusokat szükséges felhasználni, amelyek a „*windows.h*” fejléc állományban vannak deklarálva. A könyvtár memóriába való betöltését a *LoadLibrary* függvénnyel végezhetjük el:

```
HINSTANCE LoadLibrary(LPCTSTR lpFileName);
```

Az argumentumban kell megadni a könyvtár nevét, ami nem tartalmazhatja az elérési utat. Ha sikeres a betöltés, akkor visszakapjuk a DLL leíróját (a betöltött modul azonosítóját), ellenkező esetben NULL lesz a visszatérési érték. Ha DLL már betöltődött és újra meghívásra kerül a *LoadLibrary* függvény, akkor az operációs rendszer csak egy belső

számláló értékét növeli meg. A DLL által foglalt memória felszabadítását a *FreeLibrary* függvény segítségével tehetjük meg,

```
BOOL FreeLibrary(HMODULE hLibModule);
```

ami csökkenti a számláló értékét és paraméterként a DLL leíróját várja. A betöltött könyvtár akkor törlődik véglegesen a memóriából, ha a számláló eléri a nulla értéket. A DLL sikeres betöltése után a következő lépés a modulban definiált függvények elérése. A kiválasztott függvényre való hivatkozáshoz meg kell szereznünk a belépési pontjához tartozó címet és ennek alapján elvégezni a hívást. A cím lekérdezéséhez felhasználhatjuk a *GetProcessAddress* függvényt:

```
FARPROC GetProcAddress(HMODULE hLibModule, LPCSTR  lpProcName);
```

amelynek első paramétere a DLL leíró, a második pedig a hívni kívánt függvény neve. Ha a megadott névhez nem található alprogram a könyvtárban, akkor a függvény NULL értékkel tér vissza.

Az interfész DLL betöltését és annak függvényei hívását a *CFieldMaxController* osztály végzi, ami a „*FieldMaxInterface.h*”, illetve „*FieldMaxInterface.cpp*” forrásállományokban van implementálva. Az osztályban a prototípus deklarációk az interfész DLL-ből hívandó függvényekre való hivatkozáshoz a következőképpen vannak megadva:

```
typedef VB_LONG        (__stdcall *TPfm2LibInit)(void);
typedef VB_BOOLEAN     (__stdcall *TPfm2LibDeInit)(void);
typedef VB_LONG        (__stdcall *TPfm2LibOpenDriver)(VB_INTEGER);
typedef VB_BOOLEAN     (__stdcall *TPfm2LibCloseDriver)(VB_LONG);
typedef VB_BOOLEAN     (__stdcall *TPfm2LibSync)(VB_LONG);
typedef VB_BOOLEAN     (__stdcall *TPfm2LibGetData)(VB_LONG,
                                DataPacketType*, VB_INTEGER*);
typedef void           (__stdcall *TPfm2LibFlush)(VB_LONG);
```

Ez természetesen összhangban a „FieldMax2Lib.h” fejléc állományban található függvény deklarációkkal. Az osztály konstruktora

```
CFieldMaxController::CFieldMaxController(void);
```

inicializálja a *private* elérésű adattagokat oly módon, hogy az interfész DLL leíróját *0*, míg a műszerhez rendelt objektum leíróját *-1* értékre állítja, valamint az USB újra szinkronizálást végző (*fm2LibSync*) és adatcsomagot lekérő (*fm2LibGetData*) függvényekre irányuló mutatóknak NULL kezdő értéket ad. A továbbiakban az osztály tagfüggvényeinek feladatát ismertetem. Az *InitLibrary* metódus

```
int InitLibrary(void);
```

memóriába tölti a *LoadLibrary* WinAPI függvénnyel az interfész DLL-t, majd hívja az fm2LibInit alprogramot, amely a további műveletek végrehajtásához szükséges meghajtó szintű inicializálást elvégzi. Az *OpenDriver* metódus csak ezt követően alkalmazható

```
int OpenDriver(short DriverIndex = 0);
```

és feladata, hogy az *fm2LibOpenDriver* DLL függvényt hívja, ami visszaadja a *DriverIndex* paraméternek megfelelő indexű műszerhez tartozó objektum leírót. A *Synchronize* metódus

```
int Synchronize(void);
```

pedig meghívja az *fm2LibSync* DLL függvényt, amelynek segítségével végbemegy az USB szinkronizálás. Ezek után már lehetőség van arra, hogy műszertől lekérjünk mérési adatokat. Amennyiben minden egyes pozícióban egy mérést kívánunk elvégezni akkor a *GetSingleData* metódust kell használtunk, mivel ez

```
int GetSingleData(float* Data);
```

az *fm2LibGetData* DLL függvény segítségével a műszertől lekéri a mért fényteljesítmény értékét. Ezen mérési eredményt a cím szerint átadott Data változóban kapjuk meg. Ha a beállított pozícióban több mérési adat felvételére van szükség, akkor erre a *GetVectorData* metódust használhatjuk,

```
int GetVectorData(float*& DataArray, short Count);
```

mivel ez meghívja az *fm2LibGetData* DLL függvényt olyan paraméterezés mellett, hogy az megadott számú (*Count*) mérést hajtson végre a műszerrel az adott mérési pontban. A mérési eredményeket a referenciaként átadott *DataArray* dinamikus tömbben kapjuk meg. A mérések befejeztével a *CloseDriver* metódust kell hívni,

```
int CloseDriver(void);
```

amely az *fm2LibCloseDriver* DLL függvény segítségével bontja a kapcsolatot a mérésvezérlő számítógép és a mérőműszer között. Végül pedig a *DeInitLibrary* a metódust alkalmazva

```
int DeInitLibrary(void);
```

először meghívjuk az *fm2LibDeInit* DLL alprogramot, ami meghajtó szintű véglegesítő beállításokat hajt végre, majd ezután a *FreeLibrary* WinAPI függvénnyel felszabadítjuk a DLL által foglalt memóriát. Ha adatgyűjtés során hiba történik és többször kell megismételni egy mérést, akkor az erre fordított időtartam nem haladhat meg egy felső korlátot. Az időtartam túllépést vizsgáló *TimeOut* metódus

```
bool TimeOut(DWORD StartTime, DWORD TimeOutLimit);
```

ennek ellenőrzést a *GetTickCount* WinAPI függvényt használva végzi. Az első paraméter az éppen aktuális időponthoz tartozó *TickCount* érték, míg a második a másodpercben megadott időtartam határ.

5. Mérési eredmények feldolgozása és grafikus megjelenítése

5.1. Mért értékek naplózása standard formátumban

Egy adott diffúzor minta BSDF adatainak a felvétele során a merésvezérlő program a léptetőmotorok segítségével a (19) „mátrix” elemei által meghatározott azimutális és radiális szögpároknak megfelelő pozíciókba mozgatja a tárgyasztalt és az optikai szenzort, majd ebben a beállításban műszeres teljesítmény mérést végez. A (6) egyenlet szerint ezek alapján egy adott beesési szög mellett, egy adott pontban meghatározható a mért BSDF érték. Mivel egy pásztázó mérés során (lásd a (11)-(14) összefüggéseket) egymás után több pontban szeretnénk méréseket végezni szükség van olyan adatszerkezetre, ami az összetartozó $\{P_{meas}; \Phi_{az}, \Theta_{rad}\}$ értékeket tárolja.

A „*Global.h*” és „*Global.cpp*” forrásállományokban van megvalósítva az a *CBSDF* osztály, amely a mérési adatokat tárolja és azokat a mérés végeztével, normalizálást követően automatikusan naplózza egyedi névvel ellátott állományokba. Ez utóbbi úgy van biztosítva, hogy az állománynév a mentéskor aktuális dátum és idő adatokat (év, hó, nap, óra, perc, másodperc) is tartalmazza. A *CBSDF* osztályban deklarált *TSMeasurements* struktúra típus segítségével tárolhatók a több beesési szög mellett elvégzett pásztázó mérések adatai:

```
typedef struct SMeasurements
{
    SMeasurements(int AOIs, int Phis, int Thetas) : NumOfAOI(AOIs),
    NumOfPhi(Phis), NumOfTheta(Thetas) {…}
    float* AOI;
    float* Phi;
    float* Theta;
    float** PhiCorrected;
    float** ThetaCorrected;
    std::vector<float**> Power;  […]
} TSMeasurements;
```

Ugyanis a mérések megkezdése előtt a felhasználónak meg kell adnia az alkalmazni kívánt beesési szögek számát (*NumOfAOIs*), valamint az azimutális és radiális szögtartományok alsó és felső határait, továbbá a lépésközök nagyságát. Ezekből a paraméterekből meghatározható, hogy hány különböző azimutális (*NumOfPhis*) és radiális (*NumOfThetas*)

szögbe kell beállni. Így a struktúra konstruktora az összes *AOI*, *Phi* és *Theta* szög értékekhez létrehoz egy dinamikus tömböt, amiben azok értékeit tárolja. A struktúrában szereplő *PhiCorrected* és *ThetaCorrected* kétdimenziós tömbök a (19) által definiált mátrixnak felelnek meg. Ezen mátrixok elemeit a (8)-(14) egyenletek szerint kell meghatározni. A programban a számolások elvégzését a „*CDTMath.h*” és „*CTDMath.cpp*” forrásállományokban megvalósított *CDTMath* egyke osztály *GetTransformedCoordinates* metódusának

```
int GetTransformedCoordinates(float* PhiDV, int PhiCard, float* ThetaDV, int ThetaCard, float AOI, int Symmetry, float** PhiHSM, float** ThetaHSM);
```

a hívásával lehet elvégezni (ami paraméterként kéri direkt irányú radiális és azimutális szögeket tároló *PhiDV* és *ThetaDV* tömböket és azok *PhiCard* és *ThetaCard* számosságait, valamint az *AOI* beesési szöget és a (15)-(16) egyenletekben definiált szimmetriák azonosítóját). A számított félgömbi koordinátákat a *PhiHSM* és *ThetaHSM* mátrixok tartalmazzák. Ezen beállításoknak megfelelően minden beesési szöghöz tartozik egy mért teljesítmény mátrix, melyek az alábbi *Power* vektorban

```
std::vector<float**> Power;
```

kerültek tárolásra a struktúrán belül.

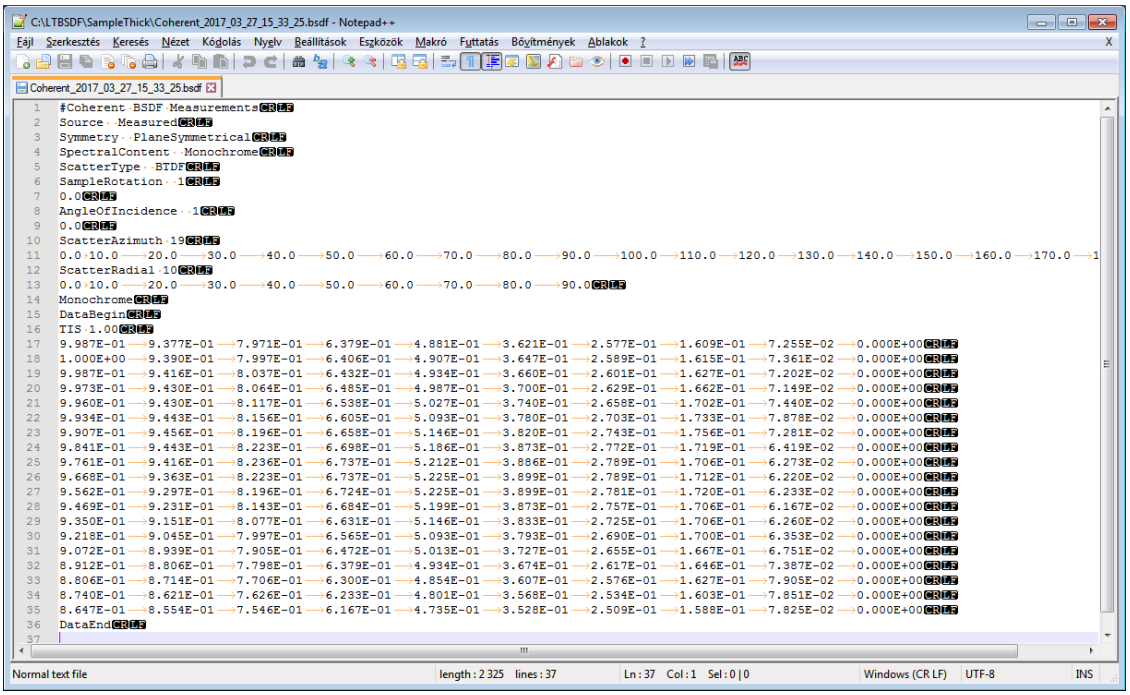
Line #	Description	Values
#	<comments>	<Any line that starts with # is ignored as a comment>
1	Source	Measured
2	Symmetry	PlaneSymmetrical Asymmetrical Asymmetrical4D
3	SpectralContent	Monochrome XYZ
4	ScatterType	BRDF BTDF
5	SampleRotation	<number of different sample rotations>
6	<enum>	<enumeration of sample rotations>
7	AngleOfIncidence	<number of different angles of incidence>
8	<enum>	<enumeration of angles of incidence>
9	ScatterAzimuth	<number of different azimuth angles>
10	<enum>	<enumeration of azimuth angles>
11	ScatterRadial	<number of different radial angles>
12	<enum>	<enumeration of radial angles>
13	<empty line>	
14	<SpectralContent indicator>	Monochrome Tristimulus<XYZ>
15	DataBegin	<Begin Data>
16+	<enum>	<enumeration of scatter data for all angles>
17	DataEnd	<End Data>

7. ábra. ZEMAX LLC. által megadott BSDF állomány formátum specifikáció [8].

A BSDF adatok szöveges állományba való mentése a ZEMAX [8] vállalat által ajánlott formátumban történik, amelyhez tartozó specifikáció lényege a 7. ábrán látható. A mérési adatok mentését a *CBSDF* osztály *SaveToFile*

```
int CBSDF::SaveToFile(TSMeasurements* const Measurements, const std::string&
FileName)
```

tagfüggvénye végzi, amelynek paraméterei a *TSMeasurement* típusú, mérési adatokat tároló struktúra példány és a kimeneti adatállomány neve.



8. ábra. Egy BSDF mérés naplózott adatai.

a 8. ábrán egy diffúzor minta mérési adatait tartalmazó BSDF állomány tartalma látható.

5.2. A felvett szóráskép ábrázolása hőtérképen

A mért BSDF értékek gömbi polárkoordináta rendszerben értelmezett hőtérképen ábrázolhatók szemléletesen, ezzel megkönnyítve az adatkiértékelést is.

Mivel a mérések elvégzése időigényes, viszont finomabb felbontást szeretnénk elérni, ezért a vizuális megjelenítés előtt újra mintavételező (*resampling*) eljárást alkalmaztam az adatsorokon. A legközelebbi szomszéd interpolációs (*Nearest Neighbor Interpolation, NNI*) algoritmus 2D változatával [9] előállítottam a $\{(\Phi_i, \Phi_{i+1}); (\Theta_j, \Theta_{j+1})\}$ mérési alappontok között értelmezett új:

$$\Phi_k = \Phi_i + (k-1)\frac{\Phi_{i+1}-\Phi_i}{n+1}, \quad (k = 1, \dots, n; \quad i = 1, \dots, N-1), \tag{20}$$

$$\Theta_l = \Theta_j + (l-1)\frac{\Theta_{j+1}-\Theta_j}{m+1}, \quad (l = 1, \dots, m; \quad j = 1, \dots, M-1) \tag{21}$$

rácson a BSDF közelítő numerikus értékeit (itt *n* és *m* azt adja meg, hogy hány új Φ_k, illetve Θ_l pontban kívánjuk kiszámolni a BSDF értékeit). Az interpolált fényteljesítmény adatok meghatározásához a (22) gömbi távolság formulát

$$\mathrm{D(d,s)} = \arccos[\cos\Theta_\mathrm{d}\cos\Theta_\mathrm{s}\,(\cos\Phi_\mathrm{d}\cos\Phi_\mathrm{s} + \sin\Phi_\mathrm{d}\sin\Phi_\mathrm{s}) + \sin\Theta_\mathrm{d}\sin\Theta_\mathrm{s}] \quad (22)$$

használtam a legközelebbi szomszéd

$$(i^*, j^*) = \min_{i,j}\{D(\Phi_k, \Theta_l; \Phi_i, \Theta_j), D(\Phi_k, \Theta_l; \Phi_i, \Theta_{j+1}),$$
$$D(\Phi_k, \Theta_l; \Phi_{i+1}, \Theta_j), D(\Phi_k, \Theta_l; \Phi_{i+1}, \Theta_{j+1})\} \quad (23)$$

indexeinek megkereséséhez a mérési alappontok által kifeszített gömbsüvegen (*d* jelöli az új rácspontok (Φ_k, Θ_l) koordinátáit, és *s* pedig a négy (Φ_i, Θ_j), (Φ_i, Θ_{j+1}), (Φ_{i+1}, Θ_j) és (Φ_{i+1}, Θ_{j+1}) mérési alappont egyikét). Ezek alapján felírható, hogy az újonnan felvett rácspontokban az interpolált értékek:

$$\mathrm{BSDF_{interpolated}}(\Phi_\mathrm{k}, \Theta_\mathrm{l}) = \mathrm{BSDF_{meas}}(\Phi_{\mathrm{i}^*}, \Theta_{\mathrm{j}^*}). \quad (24)$$

A fenti (20)-(24) összefüggések szerint definiált interpolációs probléma algoritmizált megoldását a „*CDTMath.h*” és „*CDTMath.cpp*” forrás állományokban megvalósított *CDTMath* egyke (*singleton*) osztály végzi. A (20) és (21) egyenletek által meghatározott, sűrűbb beosztású rácson számított BSDF adatok tárolását, a *CDTMath* osztályon belül deklarált *TSInterpolatedData*

```
typedef struct SInterpolatedData
{
    SInterpolatedData(const SParameters& Parameters, CBSDF::TSMeasurements*
    const MeasuredData) {...}
    float* PhiValues;
    float* ThetaValues;
    std::vector<float**> PowerValues; [ … ]
} TSInterpolatedData;
```

struktúra típus teszi lehetővé. A struktúra *SInterpolatedData* konstruktorának meghívásakor az (11)-(14) eredeti rácson megadott *N* és *M* dimenzióknak megfelelő *(N-1) n + N*, valamint *(M – 1) m + M* méretű *PhiValues* és *ThetaValues* dinamikus tömbök jönnek létre az azimutális és radiális szögek számára, melyek a teljes interpolációs rács alappontjait tárolják. Hasonlóképpen inicializálásra kerülnek a különböző beesési szögekre a teljesítmény értékeket tároló *PowerValues* kétdimenziós dinamikus tömbök is. Természetesen a struktúra konstruktora a *PowerValues* mátrixokba visszaírja az eredeti rácspontoknak megfelelő mérési adatokat.

A többi hiányzó BSDF mátrixelem pedig a *NearestNeighborInterpolation* metódussal

```
int CDTMath::NearestNeighborInterpolation(const SParameters& Parameters,
CBSDF::TSMeasurements* const MeasuredData, TSInterpolatedData*&
InterpolatedData, int MeasCount);
```

van meghatározva, amely a (22)-(24) összefüggések szerint végzi el a számolásokat és az eredményeket tárolja a fenti *TSInterpolatedData* típusú struktúra változóban.

A kapott BSDF adatokat minden egyes pásztázó mérés befejeztével egyedi névvel ellátott adatállományba menti a mérésvezérlő program a *SaveMapDataFile* metódussal

```
int CBSDF::SaveMapDataFile(const NSMeasurement::SParameters& Parameters,
TSMeasurements* const Measurements, int MeasCount, const std::string& FileName);
```

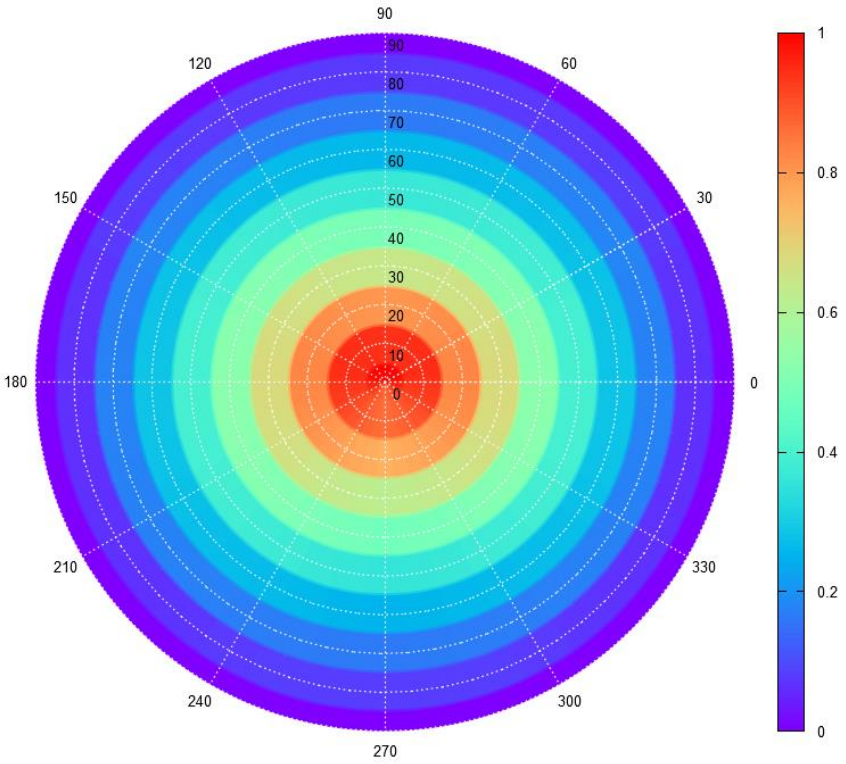
amely „*Global.cpp*” forrásállományban van definiálva a *CBSDF* osztályban.

A hőtérkép előállításhoz felhasználjuk a C programozási nyelven fejlesztett gnuplot nevű parancssor-vezérelt, két és háromdimenziós függvényrajzoló programot. Ezen alkalmazás lehetővé teszi, hogy a saját belső szkript nyelvén megadott parancsok szerint történjen egy diagram elkészítése. Ez a megírt szkript fogadhat külső paramétereket, hasonlóan a batch file programozáshoz. További hasznos lehetőség, hogy a gnuplot alkalmazás könnyen hívható C++ programokból parancssori argumentumokkal ellátva. Az alábbi gnuplot szkriptet felkészítettem arra, hogy a paramétereként megadott nevű – és a mérésvezérlő program által előzőleg elérhetővé tett – adatállományból legenerálja a 9. ábrán látható BSDF hőtérképet PNG képformátumban. (A BSDF adatállomány nevét kiterjesztés nélkül kapja meg a szkript, mivel az alapértelmezés szerint a dat. A gnuplot az elkészített diagramot az adatállomány nevével megegyező png kiterjesztésű képállományba menti el.)

```
#GNUPLOT SCRIPT: Hőtérkép előállítása BSDF adat állományból
set terminal pngcairo background rgb "white" font 'arial,10' size 368,368
set output $0."png"

set lmargin at screen 0.05
set rmargin at screen 0.85
set bmargin at screen 0.1
set tmargin at screen 0.9
set datafile missing "NAN"
set multiplot
```

```
unset border
unset key
set palette rgb 33,13,10
set pm3d map
set mapping cylindrical
set angles degrees
thetamax = 90.0;
set xrange[-thetamax:thetamax]
set yrange[-thetamax:thetamax]
set zrange[0:1]
set colorbox user origin 0.9, 0.1 size 0.03, 0.8
set tics textcolor rgb 'black'
set cbrange [0:1]
unset xtics
unset ytics
unset ztics
set view 0,270
splot $0."dat" using 1:3:2:4 notitle
set style line 10 lc rgb 'white' lw 2 lt 0
set grid polar 30 ls 10
set polar
set rrange[0:thetamax]
unset raxis
set rtics format '' scale 0
set rtics (0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0)
set for [i=0:330:30] label at first (thetamax + 5) * cos(i), first (thetamax + 5) * sin(i) center
sprintf('%d', i) tc rgb 'black'
set for [j=0:90:10] label at first 3, first j - 3 center sprintf('%d', j) tc rgb 'black'
plot NaN w l
unset multiplot
set output
reset
quit
```



9. ábra. Mért BSDF adatokat ábrázoló hőtérkép.

A hőtérkép kirajzolásakor a mérésvezérlő program meghívja a gnuplot alkalmazást oly módon, hogy parancssori argumentumként átadja neki a fenti szkript nevét (*bsdf.cmd*), illetve a lementett BSDF adatállomány nevét (kiterjesztés nélkül). Ezután betölti a gnuplot által létrehozott PNG állományt és a 12. ábrán látható módon megjeleníti azt a felhasználói felületen egy képdobozban.

A rajzoló programmal való kapcsolatot „*HeatMap.h*” és „*HeatMap.cpp*” forrásállományokban implementált „*chart*” névtérben elhelyezkedő gnuplot osztály valósítja meg, amely csővezeték (*pipe*) felhasználásával végzi az információ átadást.

Az osztály konstruktora

```
chart::gnuplot::gnuplot();
```

a *gnuplotpipe* adattagot inicializálja NULL értékkel, amely egy FILE típushoz tartozó mutató. Csővezetékként a rendszer által létrehozott, név nélküli speciális állományt fogunk használni, amelyhez ezen mutatón keresztül férhetünk majd hozzá. Az *engine* private hozzáférésű tagfüggvény

```
int chart::gnuplot::engine(std::string path, std::string command, bool window);
```

a *_popen* könyvtári függvény segítségével létrehozza a csővezetéket. Továbbá A *gnuplotpipe* változó értékét beállítja a *_popen* függvény visszatérési értékére, ami csővezetéket reprezentáló adatállományra irányuló mutató. A *path* argumentumban a pgnuplot.exe segédprogram (ez a gnuplot telepítése után szintén rendelkezésre áll) elérési útvonalát lehet megadni. A kommunikációs csatorna vevő oldalán ez a segédprogram fogadja a csővezetéken az *fprintf* könyvtári függvénnyel küldött gnuplot parancssort, melyet a *command* karakter füzér tartalmaz. A *window* nevű logikai típusú paraméterrel azt lehet beállítani, hogy a gnuplot felhasználói felülete látható legyen-e a kommunikáció során (program tesztelés során lehet hasznos ez a funkció). A publikus hozzáférésű *plotheatmap* tagfüggvény

```
int chart::gnuplot::plotheatmap(std::string datafilename);
```

az *engine* metódus hívása előtt az alábbi formátumban előállítja elő a rajzoló programot vezérlő parancssort:

pgnuplot call bsdf.cmd „datafilename”

amiben a futtatandó gnuplot szkript neve „*bsdf.cmd*”, a BSDF adatállomány nevét pedig a *datafilename* argumentum határozza meg. A „call” prefix jelzi a rajzoló programnak, hogy szkript állományt kell értelmeznie. Az adatállomány nevét a szkript a „*$0*” változójában kapja meg. Az osztály destruktora

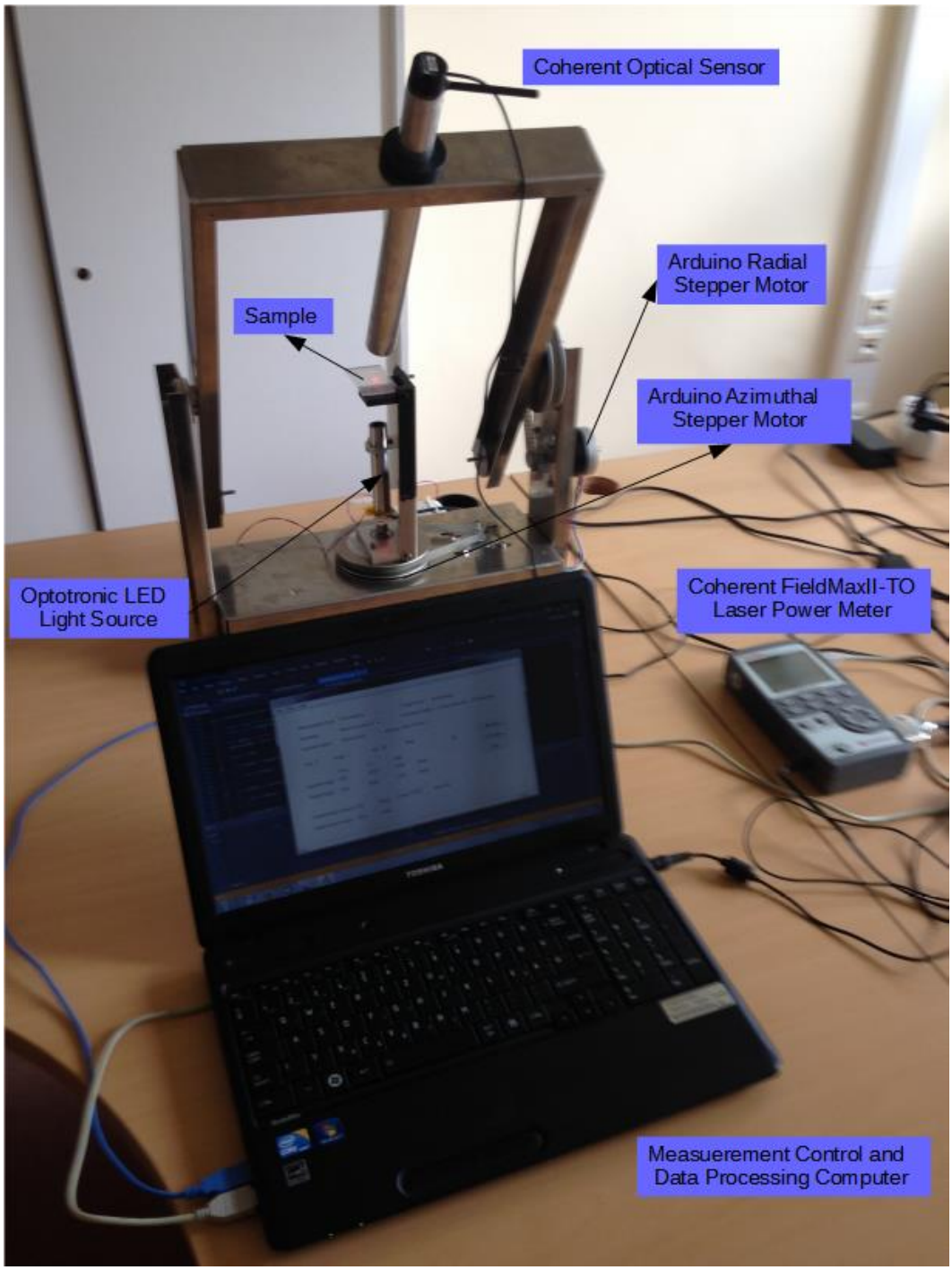
```
chart::gnuplot::~gnuplot()
```

lezárja a csővezetékhez tartozó állományt, majd megvárja az elindított folyamat befejezését.

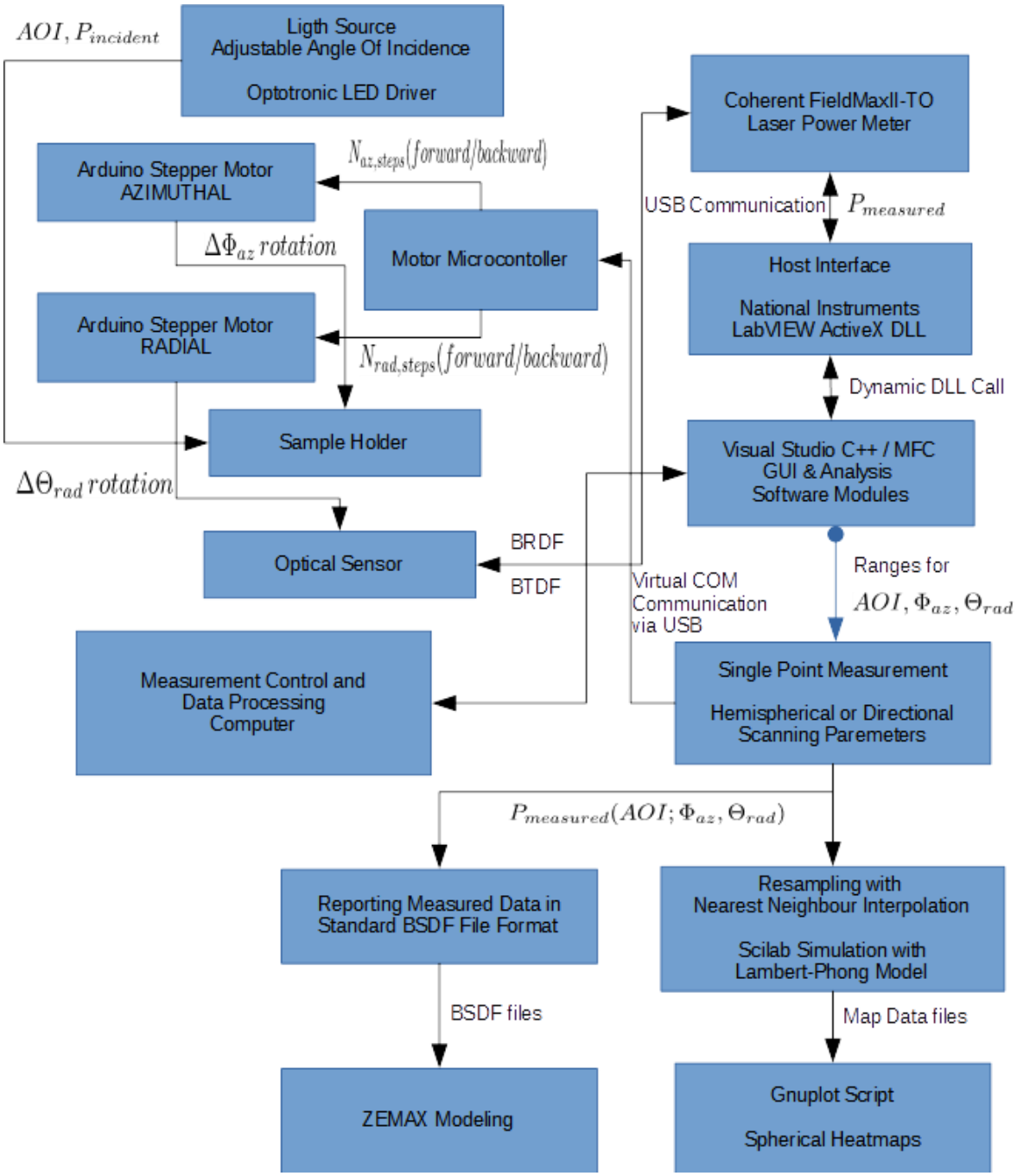
6. A mérésvezérlő asztali alkalmazás

6.1. Felhasználói felület bemutatása

Az Optimal Optik Kft. által kutatás-fejlesztés céljából megépített BSDF mérőkészülék fényképe a 10. ábrán látható.



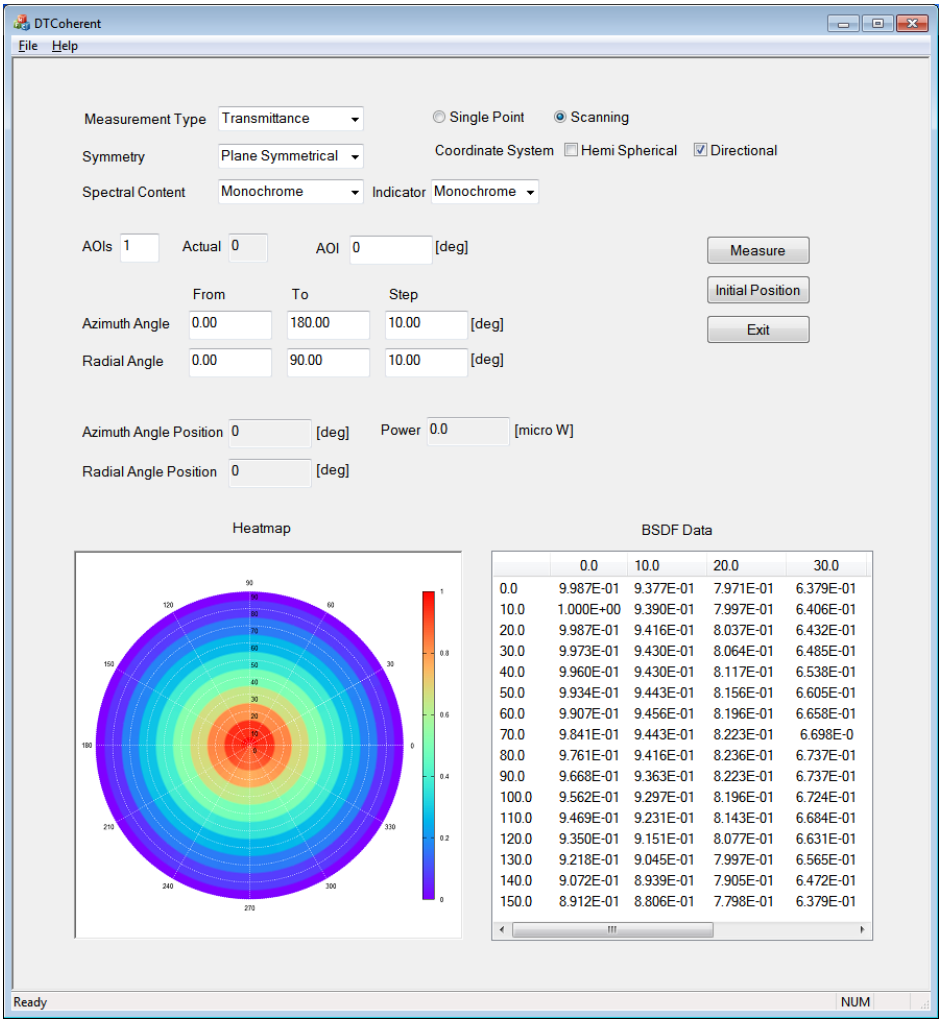
10. ábra. BSDF mérőkészülék.



11. ábra. A mérésvezérlő alkalmazás blokk diagramja.

A mérésvezérlő és adatfeldolgozó számítógépen futó asztali alkalmazás működését a fenti blokk diagramm (lásd 11. ábra) szemlélteti. A program felhasználói felülete (12. ábra) az MFC keretrendszer segítségével készült. A mérésvezérlést megvalósító alkalmazás logikát a „*DTCoherentView.h*” és „*DTCoherentView.cpp*” forrásállományok tartalmazzák. A mérést végző személynek a felhasználói felületen a mérés megkezdése előtt a be kell állítania a mérés típusát, a minta szimmetria tulajdonságát, valamint azt, hogy egyetlen pontban szeretne csak mérni, vagy pásztázó mérés elvégzése a célja. A BSDF adatok felvétele történhet normál félgömbi, illetve direkt irányú gömb polárkoordináta

rendszerekben. Szükséges megadni, hogy hány manuális beesésiszög beállítás történik majd és milyen szögértékekkel, mert a mérésvezérlő program nem kap a hardvertől információt erről. Több beesési szög alkalmazásakor az egyes mérések befejezése után a program értesít arról, hogy várja a következő manuális szögbeállítást és a szög értékének megadását.



12. ábra. Mérésvezérlő alkalmazás MFC felhasználói felülete.

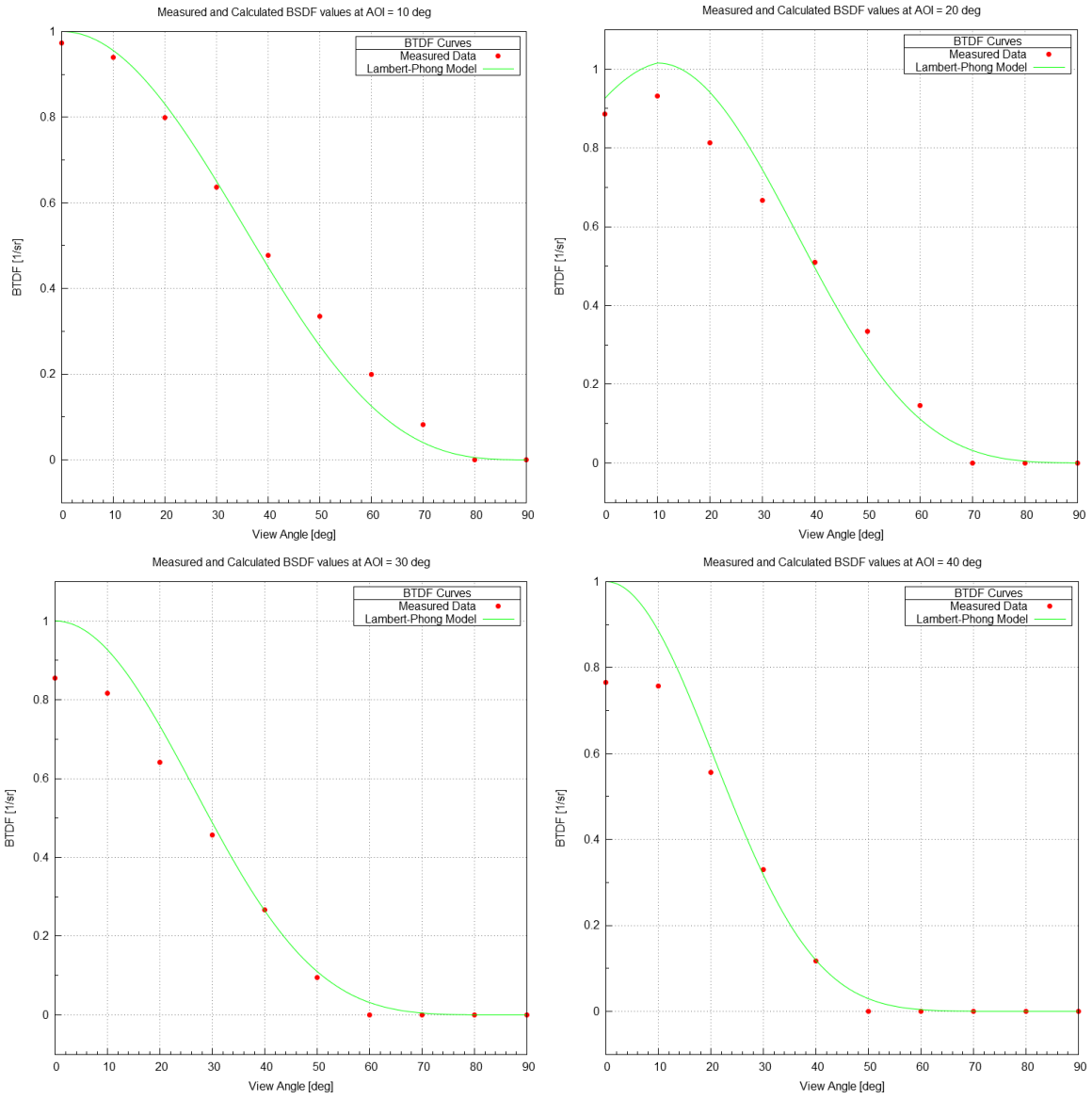
Az azimutális és radiális szögek értéktartományainak és lépésközeinek megadása szintén szükséges, mert ennek hiányában az alapértelmezett értékek kerülnek alkalmazásra Mérés indítása a „*Measure*” gombra való kattintással történhet. A program minden beállított pozícióban kiírja az aktuális azimutális és radiális szögek és a mért teljesítmény értékeit. A pásztázó mérések befejeztével megjelenik az adott beesési szöghöz tartozó szóráskép, illetve a táblázatban a mért BSDF adatok. Az „*Initial Position*” gombra való kattintás olyan eseményt vált ki, ami a parkírozó pozícióba viszi a mérőfejet és alaphelyzetbe állítja a tárgyasztalt. Az alkalmazásból kilépni az „Exit” gombbal lehet.

6.2. Mérési adatok összehasonlítása modell számolás eredményével

A mérési eredményeket összevetettük a szakirodalomból jól ismert és elterjedten használt Phong-Lambert modell [10] egy módosított változatával [11]. Ezen egyszerűsített (azimutális szimmetriát feltételező) empirikus tárgyalásmódban a visszaverődés / áteresztés nagy része a Θ_{peak} direkt irányban és annak környezetében történik:

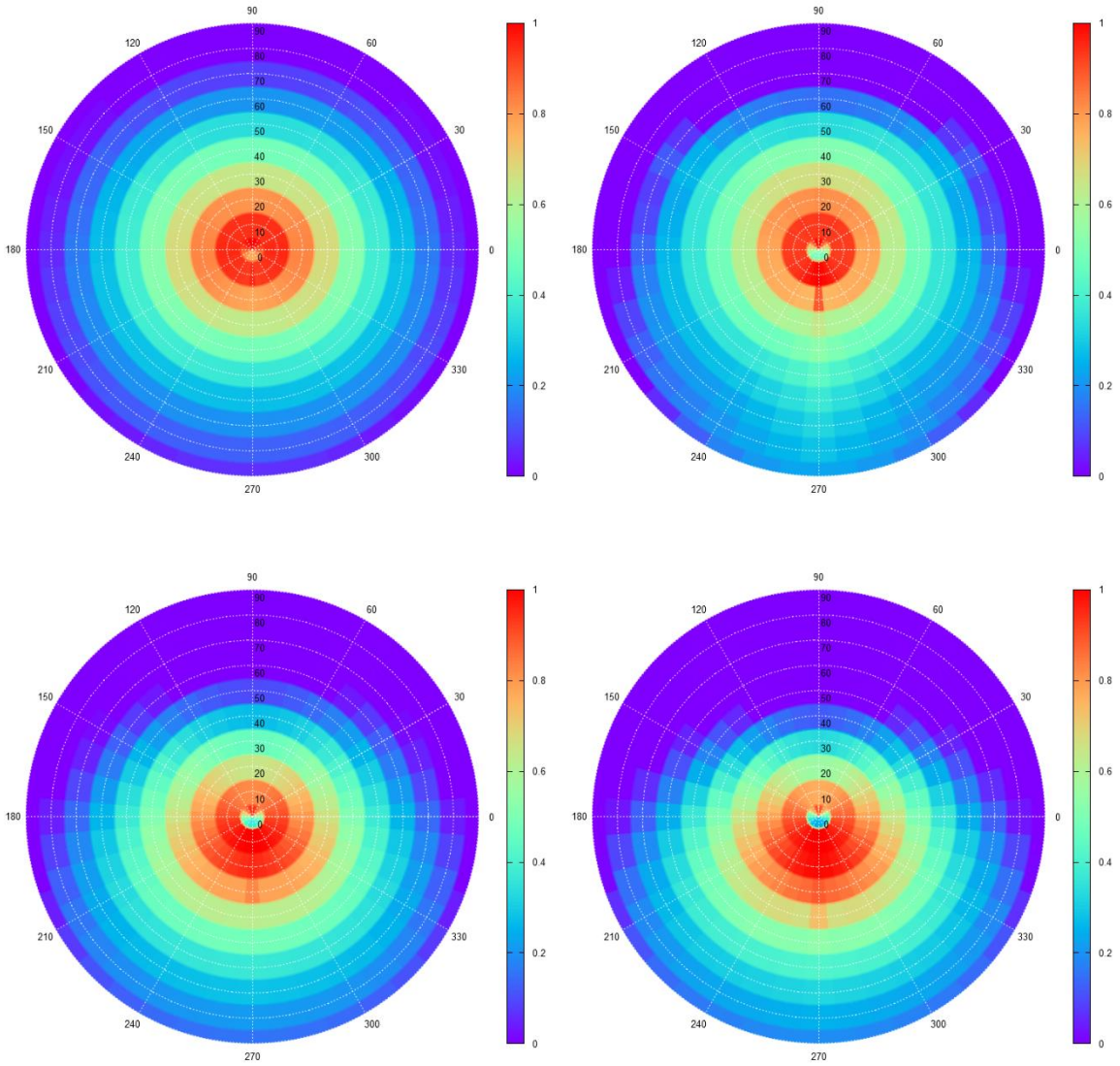
$$\mathrm{BSDF}_{\mathrm{model}}(\Theta_{\mathrm{view}}) = \mathrm{k_s}\,\frac{\cos^{\mathrm{n}}(\Theta_{\mathrm{view}}-\Theta_{\mathrm{peak}})}{\cos\Theta_{\mathrm{min}}} \tag{25}$$

ahol Θ_{view} a megfigyelési szög, $\Theta_{min} = \min(\Theta_{peak}, \Theta_{view})$, k_s az anyagi minőségtől függő spekuláris reflexiós / transzmissziós együttható, valamint az $n \in [0...\infty)$ egész szám egy modell paraméter, ami az ideálisan diffúz (Lambert-féle) felülettől való eltérés hatását szabályozza. A modell-számolásokat a Scilab numerikus matematikai program környezetben írt szkript segítségével valósítottam meg. A 13. ábrán a nulla azimutális szögbeállítás mellett egy adott mintára, transzmisszióban mért BSDF adatok, valamint a (25) összefüggés felhasználásával számított elméleti BSDF görbék vannak feltűntetve a megfigyelési (radiális) szög függvényében, négy különböző beesési szögre vonatkozóan, melyek értekei rendre 10°, 20°, 30° és 40°.



13. ábra. Diffúzor mintára kapott mért BSDF adatok összevetése a Lambert-Phong modell eredményével 0°-os azimutális szög mellett, különböző beesési szögekre. Zöld folytonos vonal: Lambert-Phong modell számolás eredménye, Piros pontok: mért érékek.

A 14. ábrán ugyanerre a diffúzor anyagra kapott BSDF mérési adatok hőtérképen ábrázolt értékei láthatók (összhangban a 13. ábra grafikonjaival).



14. ábra. Diffúzor mintára mért BSDF adatok hőtérképe, különböző beesési szögekre (10°, 20°, 30°, 40°). Az azimutális szög tartomány 0°...360°, a radiális 0°...90°.

A 15. ábrán látható összesítő táblázatban egy adott beesési szöghöz (10°), a (7)-(14) egyenletek alapján a direkt irányszögekből inverz művelettel számított félgömbi, azimutális és radiális szögek, valamint az ezekben a pontokban mért BSDF adatok numerikus értékei találhatóak. A táblázatban a pirossal jelölt értékek olyan hemiszférikus radiális szögekhez tartoznak, melyek a (18) egyenlet miatt nem állhatnak elő. Méréseim során több fajta diffúzor anyagra vettem fel BSDF hőtérképeket. A Lambert-Phong modell keretében számított BSDF értékek kvalitatíve jól egyeznek a mérési eredményekkel, ami alátámasztja, hogy az általam alkalmazott méréstechnikai módszer valós adatokat szolgáltat.

AOI = 10.0	Radial Direct	0.00	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.00
Azimuthal Direct											
0.0		0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
10.0		10,00	5,08	6,84	7,78	8,39	8,85	9,22	9,54	9,85	10,15
20.0		20,00	10,15	13,70	15,59	16,81	17,71	18,44	19,08	19,68	20,28
30.0		30,00	15,22	20,60	23,43	25,26	26,60	27,68	28,63	29,51	30,38
40.0		40,00	20,28	27,56	31,35	33,77	35,53	36,95	38,17	39,31	40,43
50.0		50,00	25,34	34,61	39,36	42,36	44,53	46,25	47,73	49,10	50,43
60.0		60,00	30,38	41,77	47,51	51,08	53,62	55,61	57,32	58,87	60,38
70.0		70,00	35,41	49,10	55,84	59,94	62,82	65,06	66,94	68,64	70,28
80.0	Azimuthal	80,00	40,43	56,65	64,40	69,00	72,17	74,60	76,62	78,43	80,15
90.0	Hemi Spherical	90,00	45,44	64,49	73,26	78,31	81,71	84,27	86,38	88,25	90,00
100.0		100,00	50,43	72,73	82,49	87,91	91,47	94,11	96,25	98,11	99,85
110.0		110,00	55,41	81,51	92,20	97,87	101,50	104,13	106,23	108,05	109,72
120.0		120,00	60,38	91,01	102,48	108,24	111,82	114,36	116,36	118,07	119,62
130.0		130,00	65,33	101,51	113,45	119,08	122,46	124,82	126,64	128,18	129,57
140.0		140,00	70,28	113,34	125,21	130,42	133,44	135,50	137,08	138,39	139,57
150.0		150,00	75,22	126,93	137,84	142,26	144,74	146,40	147,66	148,70	149,62
160.0		160,00	80,15	142,66	151,30	154,54	156,32	157,49	158,36	159,08	159,72
170.0		170,00	85,08	160,59	165,45	167,18	168,10	168,71	169,16	169,53	169,85
180.0		180,00	0,00	180,00	180,00	180,00	180,00	180,00	180,00	180,00	180,00

AOI = 10.0	Radial Direct	0.00	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.00
Azimuthal Direct											
0.0		10,00	20,00	30,00	40,00	50,00	60,00	70,00	80,00	90,00	100,00
10.0		10,00	19,92	29,90	39,88	49,87	59,87	69,86	79,86	89,85	99,85
20.0		10,00	19,69	29,59	39,53	49,49	59,47	69,45	79,43	89,41	99,39
30.0		10,00	19,31	29,08	38,95	48,87	58,81	68,77	78,73	88,69	98,65
40.0		10,00	18,78	28,37	38,15	48,02	57,92	67,84	77,77	87,71	97,64
50.0		10,00	18,11	27,47	37,15	46,95	56,80	66,69	76,59	86,50	96,41
60.0		10,00	17,30	26,40	35,96	45,69	55,49	65,34	75,21	85,09	94,98
70.0		10,00	16,36	25,16	34,60	44,26	54,02	63,83	73,68	83,54	93,40
80.0	Radial	10,00	15,29	23,78	33,09	42,69	52,42	62,21	72,03	81,88	91,73
90.0	Hemi Spherical	10,00	14,11	22,27	31,47	41,03	50,73	60,50	70,32	80,15	90,00
100.0		10,00	12,82	20,65	29,78	39,30	49,00	58,77	68,58	78,42	88,27
110.0		10,00	11,43	18,96	28,05	37,57	47,27	57,05	66,88	76,73	86,60
120.0		10,00	9,96	17,23	26,33	35,88	45,61	55,42	65,27	75,14	85,02
130.0		10,00	8,42	15,51	24,68	34,29	44,07	53,91	63,79	73,68	83,59
140.0		10,00	6,81	13,85	23,16	32,87	42,70	52,58	62,50	72,42	82,36
150.0		10,00	5,15	12,35	21,87	31,67	41,56	51,49	61,44	71,39	81,35
160.0		10,00	3,46	11,12	20,86	30,76	40,71	50,68	60,65	70,63	80,61
170.0		10,00	1,73	10,29	20,22	30,19	40,18	50,17	60,16	70,16	80,15
180.0		10,00	0,02	10,00	20,00	30,00	40,00	50,00	60,00	70,00	80,00

AOI = 10.0	Radial Direct	0.00	10.0	20.0	30.0	40.0	50.0	60.0	70.0	80.0	90.00
Azimuthal Direct											
0.0		8,082E-01	9,983E-01	8,626E-01	5,469E-01	2,458E-01	9,438E-02	3,045E-02	0,000E+00	0,000E+00	0,000E+00
10.0		9,546E-01	1,000E+00	8,596E-01	5,400E-01	2,415E-01	9,067E-02	2,972E-02	0,000E+00	0,000E+00	0,000E+00
20.0		9,447E-01	9,944E-01	8,475E-01	5,261E-01	2,359E-01	8,838E-02	2,924E-02	0,000E+00	0,000E+00	0,000E+00
30.0		9,253E-01	9,866E-01	8,324E-01	5,114E-01	2,276E-01	8,471E-02	2,803E-02	0,000E+00	0,000E+00	0,000E+00
40.0		8,976E-01	9,741E-01	8,099E-01	4,959E-01	2,181E-01	8,203E-02	2,700E-02	0,000E+00	0,000E+00	0,000E+00
50.0		8,652E-01	9,598E-01	7,870E-01	4,812E-01	2,147E-01	8,030E-02	2,613E-02	2,592E-04	0,000E+00	0,000E+00
60.0		8,307E-01	9,421E-01	7,646E-01	4,652E-01	2,078E-01	7,849E-02	2,549E-02	0,000E+00	0,000E+00	0,000E+00
70.0		7,914E-01	9,248E-01	7,382E-01	4,492E-01	2,017E-01	7,633E-02	2,402E-02	0,000E+00	0,000E+00	0,000E+00
80.0	**Measurements**	7,512E-01	9,063E-01	7,136E-01	4,337E-01	1,983E-01	7,551E-02	2,307E-02	0,000E+00	0,000E+00	0,000E+00
90.0		7,114E-01	8,864E-01	6,911E-01	4,225E-01	1,948E-01	7,313E-02	2,203E-02	0,000E+00	0,000E+00	0,000E+00
100.0		6,739E-01	8,657E-01	6,704E-01	4,121E-01	1,892E-01	7,175E-02	2,108E-02	0,000E+00	0,000E+00	0,000E+00
110.0		6,406E-01	8,462E-01	6,467E-01	4,000E-01	1,849E-01	6,842E-02	1,970E-02	0,000E+00	0,000E+00	0,000E+00
120.0		6,069E-01	8,276E-01	6,242E-01	3,819E-01	1,771E-01	6,510E-02	1,832E-02	0,000E+00	0,000E+00	0,000E+00
130.0		5,754E-01	8,086E-01	5,970E-01	3,641E-01	1,685E-01	6,199E-02	1,689E-02	0,000E+00	0,000E+00	0,000E+00
140.0		5,473E-01	7,888E-01	5,715E-01	3,430E-01	1,564E-01	5,827E-02	1,577E-02	0,000E+00	0,000E+00	0,000E+00
150.0		5,244E-01	7,724E-01	5,451E-01	3,218E-01	1,456E-01	5,417E-02	1,451E-02	0,000E+00	0,000E+00	0,000E+00
160.0		5,041E-01	7,551E-01	5,192E-01	2,989E-01	1,361E-01	5,097E-02	1,404E-02	0,000E+00	0,000E+00	0,000E+00
170.0		4,907E-01	7,413E-01	5,002E-01	2,851E-01	1,287E-01	4,855E-02	1,348E-02	0,000E+00	0,000E+00	0,000E+00
180.0		4,834E-01	7,581E-01	4,847E-01	2,717E-01	1,235E-01	4,665E-02	1,352E-02	0,000E+00	0,000E+00	0,000E+00

15. ábra. Diffúzor anyagra mért BSDF értékek 10°-os beesési szög mellett. Az azimutális szög tartomány 0°...180°, a radiális 0°...90°. A felső két táblázat a direkt megfigyelési irányból számított félgömbi koordinátákat tartalmazz (ezekbe a pozícióba kell beállni a két léptetőmotorral). A harmadik táblázat az ezekben a pontokban mért értékeket tartalmazza.

7. Összefoglalás

Dolgozatomban egy kutatás-fejlesztési célból megépített kétirány-szóráseloszlás függvény mérőberendezés általam fejlesztett vezérlő és adatfeldolgozó alkalmazását mutattam be. Ismertettem az alkalmazott mérési módszert és a kapcsolódó elméleti fizikai hátteret. Részletesen bemutattam a detektálási irányt beállító léptetőmotorok és a teljesítmény mérő műszer vezérléséhez, valamint a mért adatok hőtérképen való megjelenítéséhez készített C/C++ nyelven írt moduljaimat. A mérési adatokat összehasonlítottam modell számolások eredményével, amelyek arra engedtek következtetni, hogy a mérésvezérlő program megfelelően működik.

Irodalomjegyzék

1. Philip Dutré: Global Illumination Compendium, The Concise Guide to Global Illumination Algorithms, Computer Graphics, Department of Computer Science, Katholieke Universiteit Leuven, pp. 1-64, 2003.
2. Berthold K. P. Horn, Robert W. Sjoberg: Calculating the Reflectance Map, Massachusetts Institute of Tehcnology and Artificial Intelligence Laboratory, pp. 1-30, 1978.
3. John C. Stover: Optical Scattering – Measurement and Analysis, 2nd Edition, Bellingham, Washington, USA: SPIE – The International Society for Optical Engineering, 1995.
4. Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Lambert2.gif.
5. Wikipedia, https://en.wikipedia.org/wiki/Bidirectional_scattering_distribution_function
6. Arduino, https://www.arduino.cc/
7. Coherent, https://www.coherent.com/
8. ZEMAX LLC, https://www.zemax.com/
9. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery: Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press, 2007.
10. Breault Research Organization: Scattering in Advanced System Analysis Program, 6400 East Grant Road, Tucson USA, pp. 1-62, 2014.
11. László Neumann, Attila Neumann, László Szirmay-Kalos: Compact Metallic Reflectance Models, Computer Graphics Forum Vol. 18 No. 3, pp. 161-172, 1999.

Mellékletek

1. A fejlesztett *DTCoherent* nevű program forrásállományai

   Global.h, Global.cpp: BSDF állomány formátum definíciója
   CDTMath.h, CDTMath.cpp: matematikai segédfüggvények
   MotorControl.h, MotorControl.cpp: léptetőmotorok vezérlése
   FieldMaxInterface.h, FielMaxInterfeca.cpp: lézer teljesítménymérő vezérlése
   DTCoherentView.hm DTCoherentView.cpp: felhasználói felület és alkalmazás logika
   bsdf.cmd: gnuplor szkript a hőtérképek előállításához
   bsdfmodel.sce: scilab szkript a modellszámoláshoz

2. Telepítési útmutató (*installation.pdf*)
3. Az alkalmazás telepítő programja (*setup.exe*)