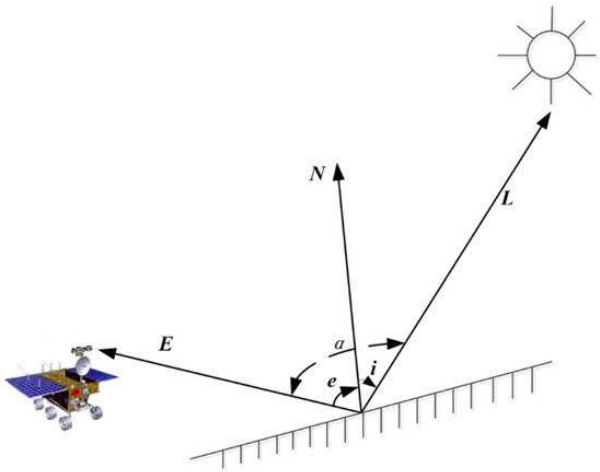
Photometric stereo:

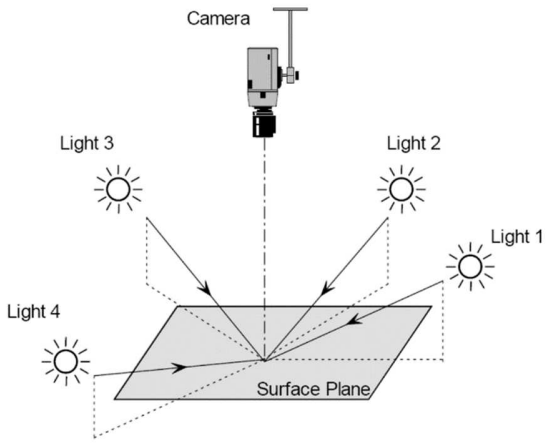
Vysvetlenie – čo to vlastne je – možno aj kompletná ukážka

Je to technika pre určovanie normálových (cize kolmých) vektorov povrchu pomocou nasvietenia objektu z viacerých uhlov. Základom je predpoklad, že úroveň svetla, ktoré sa od povrchu odráža závisí na orientácii tohto povrchu voči svetlu a voči pozorovateľovi, v našom prípade kamere.



https://www.mdpi.com/2072-4292/13/15/2975/htm

Čím viac snímok s rôznym uhlom nasvietenia máme, tým presnejšie vieme určiť vektory povrchu.



http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/CS6320-CV-S2012-Photometric-Stereo.pdf

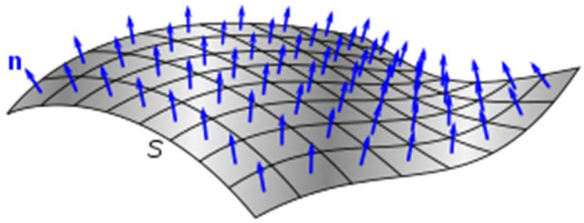
Je k tomu veľa teórie no nejdem zachádzať do podrobností.

TODO:

http://www.ian-hales.com/2019/06/26/introduction-to-photometric-stereo-1-the-basics/

==== prihodit nejaky pekny link ====

Keď každému pixelu na kamere priradíme vektor, dostaneme vektorovú mapu. Každý vektor má súradnice X Y Z a jeho dĺžka je 1 – teda každý pixel má hodnoty pre jednotkový kolmicu na povrch v danom bode.



https://en.wikipedia.org/wiki/Normal_(geometry)

Ak jednotlivé vektory zakódujeme do farby:

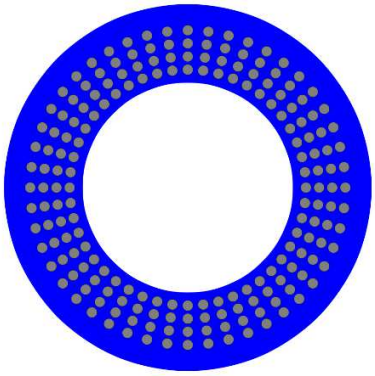
```
Kódovanie vektorovej mapy
  X: -1 to +1 :  Red:     0 to 255
  Y: -1 to +1 :  Green:   0 to 255
  Z:  0 to -1 :  Blue:  128 to 255
```

Potom vektorová mapa, ktorá sa používa aj v počítačovej grafike vyzerá asi takto:

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

kde normálová mapa (stredný obrázok) bola vygenerovaná z 3D scény (ľavý obrázok). Pravý obrázok ilustruje aplikovanie normálovej mapy na rovný povrch pri zobrazovaní grafiky.

Ako funguje naše svetlo, jeho naprogramovanie a spinanie



Naše svetlo, ktoré pre tento účel pre nás vyvinula externá firma, má možnosť individuálne spínať 192 LEDiek – 48 LED v štyroch sústredných kruhoch. No tým, že má zabudovanú elektroniku na riadenie, je jeho ovládanie veľmi jednoduché. Totiž svetlo v prevádzke ovládame len dvomi signálmi: NEXT a RESET.

Pri použití štandardného káblu je zapojenie svetla:

hnedý – 24V

modrý – GND

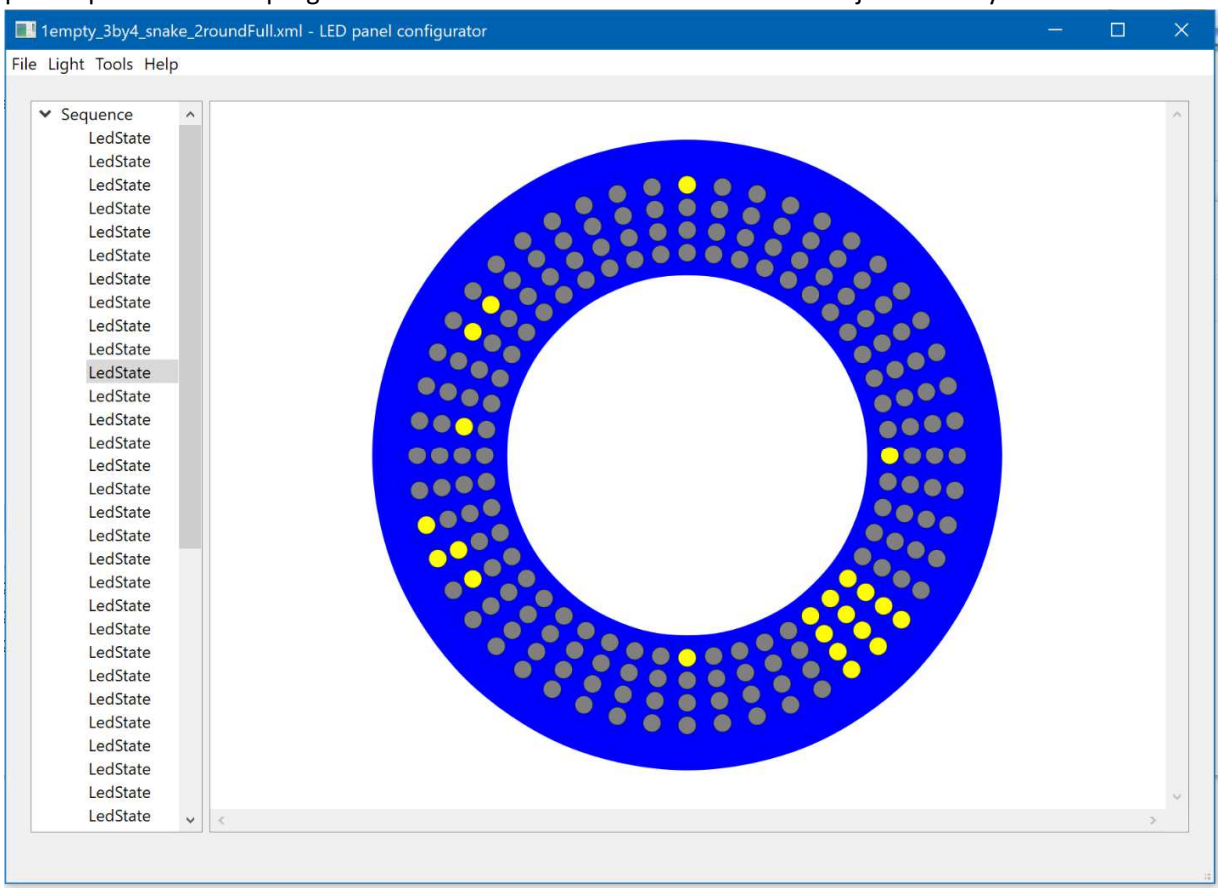
biely – reset

čierny – next

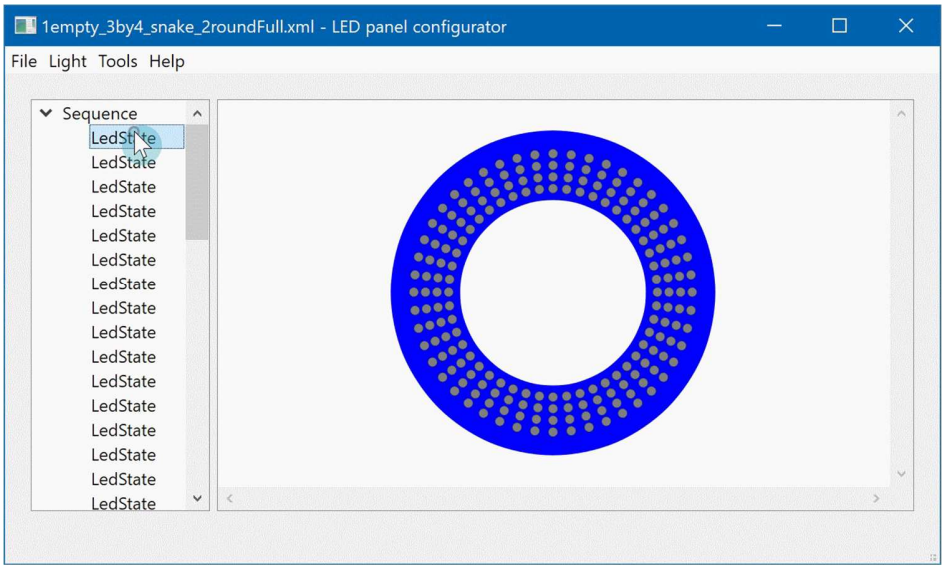
Signál NEXT znamená, že sa majú zopnúť všetky LEDky z nasledujúceho kroku (LedState) v nakonfigurovanej sekvencii. A RESET znamená, že sa má svetlo vrátiť na prvý krok sekvencie. Svetlu teda musíme najskôr nejako určiť, ktoré LEDky sa majú v jednotlivých krokoch zapnúť.

Na to slúži konfiguračný súbor, ktorý vytvárame aktuálne pomocou programu SegmentedLedPanel_v02.

Pre každý krok v sekvencii si vyklikáme, ktoré LEDky sa majú zapnúť. Takto nadefinovaný konfigurák potom priamo z tohto programu odošleme cez MiniUSB kábel do riadiacej elektroniky svetla.



Na začiatok sa mi celkom osvedčila táto konfigurácia, v ktorej robím 2 kolá, v každom kroku mám zapnuté 4x3 LEDky a vždy posúvam tento segment o 2 stĺpce, v poslednom kroku prvého kola posúvam segment o 3 LEDky, kedy mi vzniká posun všetkých segmentov voči prvému kolu o 1 LEDku:



Robím to tak preto, že ak mi stačí menej kvalitná normálová mapa tak urobím len 24 snímok (prvé kolo), kedy aj ušetrím čas na snímkovaní a keď potrebujem kvalitnejšiu mapu tak snímkujem 48 snímok. To že mám naraz zapnutých 12 LEDiek je výhoda pri snímkovaní, lebo mi stačí použiť nižšiu expozíciu.

A samozrejme netreba zabudnúť že prvý krok sekvencie nechávam prázdny, aby svetlo po dokončení snímkovania, a prijatí signálu RESET, mohlo oddýchnuť.

TODO:

=== prilozit vytvoreny konfig subor ===

=== upozornit na chyby programu a moznost exportovat CSV a editovat v exceli ===

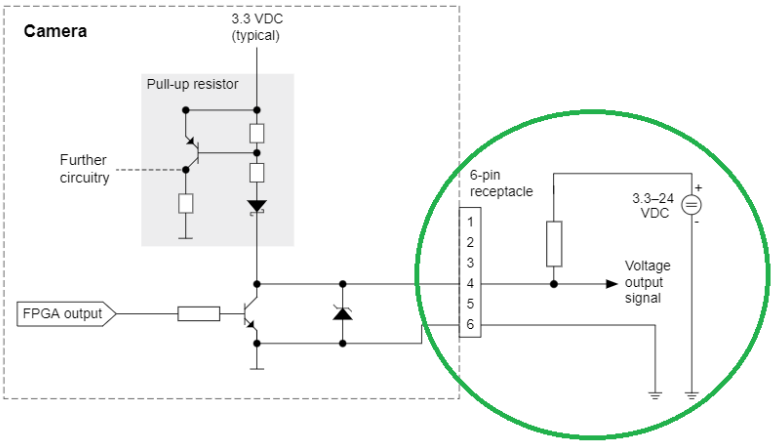
Ako nakonfigurovat kameru aby spravne spolupracovala so svetlom

Mame viacero moznosti ako ovladat snimkovanie a prepinanie svetla. No najrychlejsie snimkovanie dosiahnem vtedy ked prepneme kameru do free-run a ona si vždy po dokončení expozície sama prepne svetlo svojím binárnym výstupom. Vtedy vieme vyuzit maximálnu frekvenciu snímkovania kamery.

K tomu potrebujeme kameru ktorá má IO Output line, správne ju zapojiť a správne nastaviť jej interné parametre.

Ukážem na príklade s Basler ACE kamerou :

Schéma zapojenia výstupu z kamery:



- plati pre kamery basler ace2

Nás zaujíma tá oblasť v zelenom krúžku. Naše svetlo potrebuje na prepnutie signal nad 7V. V schéme je ukázané, že výstupný signál získame, ak použijeme pull-up rezistor na kladný pól zdroja. Naše svetlo má už tento rezistor integrovaný, takže nie je potrebné nič dopĺňať, stačí výstup 4 (v mojom pripade Žltý káblik) pripojiť priamo na NEXT signál svetla a pin 6 (v mojom príade biely káblik) kamery pripojit na zem.

TODO:

==== doplniť schemu svetla = ako si mozu prepnut na pouzitie bez pull up a s pull up ====

Inštalácia potrebných knižníc

Na to aby plugin fungoval su potrebne:

1) Cognex Designer

2) Microsoft Visual C++ Redistributable 2015-2019

3) NVIDIA Driver + CUDA - u mna funguje napriklad cuda_11.3.1_465.89_win10.exe stiahnutelne by to malo byt tu: https://developer.nvidia.com/cuda-downloads

4) Nainstalovat MTS_PhotometricStereo_Plugin.exe **- spustit ako administrator**

5) MTS_PhotometricStereo_lic.lic licencny subor na ceste C:\ProgramData\Cognex\Designer\Plugins\MTS

V Designeri potom aktivuj MTS_PhotometricStero plugin pre projektový súbor v ktorom ho chces využiť.

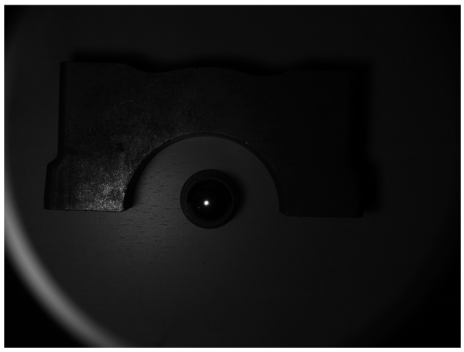
Kalibrácia pomocou guličky

Aby program vedel odkiaľ svieti svetlo – teda aby mal potrebné uhly pre určenie normálových vektorov – robíme kalibráciu pomocou guľe. Nie je to nič zložité.

Potrebujeme k tomu nejakú lesklú guličku. Celkom fajn sú napríklad guličky, čo sa používajú do väčších ložísk. Guličku umiestnime pred kameru približne do vzdialenosti v ktorej budeme neskôr snímkovať reálny objekt.

Celú mechaniku kamery a svetiel by sme už mali mať fixnú, aby sa voči sebe už nepohybovali (aj keď podľa skúseností, mierne pohyby nie sú veľkým problémom, no záleží na požadovanej presnosti). Ideálne tiež aby sme si nastavili a zaostrili objektív na požadovanú vzdialenosť.

Nastavíme si expozíciu kamery tak, aby sa nám pri zapnutí jednotlivých svetiel vytváral na guličke malý odlesk rozsvieteného svetla, napríklad takto:



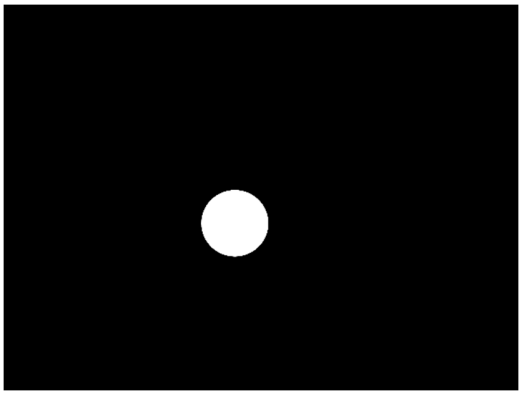
Obrázok 1 psShot_000.bmp

Na základe pozície odlesku svetla na guličke bude potom vediet program vyhodnotiť uhly jednotlivých svetiel.

Nasnímkujeme túto guličku postupne s každým svetlom, ktoré budeme používať a v poradí, v ktorom ich vždy budeme zapínať. Takže, ak ideme používať 48 svetiel, mali by sme mať 48 snímok s rôznou pozíciou svetlého bodu na guličke. Náš program je pripravený tak, že všetky tieto snímky by mali byt v jednom priečinku a je k nim potrebné vytvoriť textový súbor s názvom ***„lightsImgsList.txt“*** v ktorom sú zapísané všetky názvy snímok, každý na novom riadku a v poradí v akom boli snímkované. U mňa napríklad:

psShot_000.bmp
psShot_001.bmp
psShot_002.bmp
psShot_003.bmp
psShot_004.bmp
psShot_005.bmp
...
psShot_047.bmp

Teraz k tomu ešte program potrebuje vedieť, kde v obraze je kalibračná gulička, aby podľa toho vedel vypočítať uhly. Na to si v nejakom programe vytvoríme takýto obrázok – masku:



Obrázok 2 calibBallMask.png

Biele pixely na maske znamenajú guličku a čierne sú ignorované pozadie. Tento obrázok si uložím do pričinku so snímkami guličky pod názvom **calibBallMask.png**

Inicializácia PS

Na to, aby sme mohli využívať MTS_PhotometricStereo, je potrebné ho najskôr inicializovať. Inicializovať znamená v tomto prípade:

1) **Vytvoriť objekt MTS_PhotometricStereo**
2) **Načítať súbor s popisom pozícií svetiel "lights.yml", ktorý získame procesom kalibrácie pomocou guličky**
3) **Načítať normálovú mapu roviny, ak chceme robiť aj korekciu do roviny.**

1) Vytvoriť objekt MTS_PhotometricStereo

Najlepšie asi je vytvoriť si globálny tag v tag manažéri Designera typu MTS_PhotometricStereo.

Ja mám napríklad *$PhotometricStereo_cam1* .

Potom už len v skripte, napríklad v *OnStartup* zavoláme konštruktor: (Na to aby sme mohli použiť v skripte funkcie nášho pluginu, potrebujeme pridať MTS_PhotometricStereoForDesigner.dll do referencií skriptu.)

```
$PhotometricStereo_cam1 = new MTS_PhotometricStereoForDesinger.MTS_PhotometricStereo();
```

2) Načítať súbor s popisom pozícií svetiel "lights.yml"

*$PhotometricStereo_cam1.**setupLights**(string <cesta priečinku>);*

Kde *„cesta priečinku"* je cesta ku priečinku kde je uložený ***lights.yml*** (pozor! cesta pričinku, nie cesta súboru) alebo ***snímky kalibračnej guličky*** a súbory ***calibBallMask.png*** a ***lightsImgsList.txt*** na základe ktorých si pri zavolaní funkcie **setupLights** program sám vytvorí ***lights.yml***.

Teda u mňa je to takto:

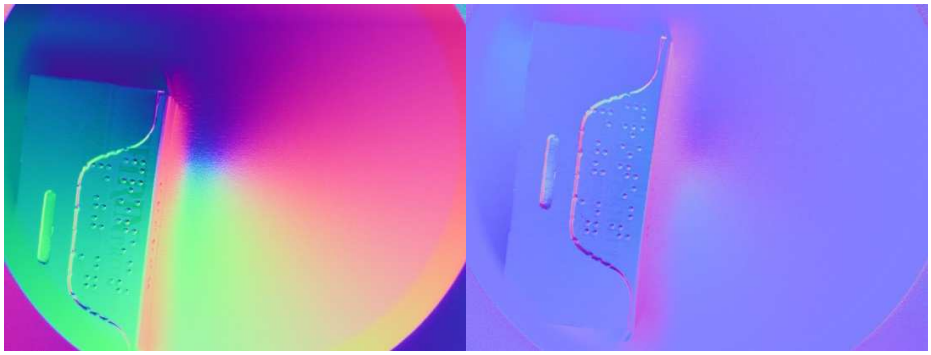
*$PhotometricStereo_cam1.**setupLights**(@"C:\MTS\PS_TemplateImages\5MR\calibBall")*

Funkcia vracia hodnotu 1 v prípade úspešneho načítania ***lights.yml***

3) **Načítať normálovú mapu roviny**

Tento krok nie je nutný, pretože program môže fungovať aj bez neho.

No toto je rozdiel medzi nenormalizovaným (v ľavo) a normalizovaným obrazom (v pravo):



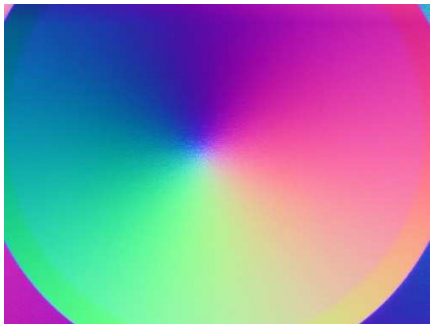
Na nenormalizovanom obraze je vidieť, že vektory sú veľmi ovplyvnené tým, kde v obraze sa nachádzajú a nesmerujú priamo k nám ale sú orientované na strany. Pretože aj rovina (väčšia časť obrázku) má rôzne farby. Obraz je tak trochu "kužeľovitý".

Na "vyrovnanom" (normalizovanom) obraze je vidieť, že vektory sú už konzistentnejšie, a smerujú viac k nám. Totiž fialová farba znamená vektor, ktorý smeruje kolmo na rovinu obrazu. Lepšie sa potom na tomto obraze vyhodnocuje, pretože nezáleží až tak veľmi na pozícii sledovaného objektu a mal by mať podobné farby.

Tá posledná veta bola zámerne napísaná so spojeniami "až tak veľmi" a "mal by mať", pretože ešte aj teraz sa trochu tie vektory môžu líšiť v závislosti na pozícii v obraze. Všimnite si že napríklad v strede vyrovnaného obrazu sa tie farby líšia a tým to nie je úplná rovina.

Veľa závisí od snímkovaných materiálov a tiež vzdialenosti od objektívu a tiež od naklonenia roviny objektu.

Vyrovnaný obraz totiž aktuálne získavame naivným prístupom. Pomocou MTS_PhotometricStereo nasímame a spracujeme obraz roviny, približne vo výške, v ktorej bude záujmová časť objektu, ideálne s náklonom roviny podobným aký bude mať objekt. Napríklad, pre získanie normalizovanej snímky vyššie som, použil biely papier (z toho aj názov funkcie), ktorý som položil na stôl, na ktorom je položená krabička, ktorú som neskôr snímkoval. Výsledok bol taktýto:



Túto normálovú mapu som potom použil pri inicializácii normalizácie vo funkcii

*$PhotometricStereo_cam1.**loadPaperNormals**();*

Takto:

*$PhotometricStereo_cam1.**loadPaperNormals**(@"C:\MTS\PS_TemplateImages\5MR\paper\cam1_2021-11-18_13-23-33.fpn");*

Ak má objekt inicializovanú túto mapu, potom pri volaní funkcie pre spracovanie snímok, môžeme dať ako argument *correctByPaper* hodnotu **true**:

```
$PhotometricStereo_cam1.processPhotometricStereo_GPU(true);
```

Čím hladší povrch normalizačnej roviny, a čím podobnejší lesk roviny s leskom snímaného objektu, tým lepšie výsledky dosiahnete.

Snímkovanie a spracovanie do vektorového obrazu

Keď už máme hotovú inicializáciu, môžeme začať spracovávať snímky do normálovej mapy.

Aby objekt MTS_PhotometricStereo vedel, ktoré snímky má spracovať, musíme mu ich naskôr poslať. No ešte pred tým mu pre istotu povieme, aby si vymazal všetky doterajšie snímky, teda aby začínal vždy s prázdnym zásobníkom. Teda postup je nasledovný:

*$PhotometricStereo_cam1.**clearShotImages**();*

A potom mu posielame jednotlivé snímky:

*$PhotometricStereo_cam1.**pushShotImage**(Cognex.VisionPro.ICogImage simCam1_Acquire_Image);*

*$PhotometricStereo_cam1.**pushShotImage**(Cognex.VisionPro.ICogImage simCam1_Acquire_Image);*

*$PhotometricStereo_cam1.**pushShotImage**(Cognex.VisionPro.ICogImage simCam1_Acquire_Image);*

……

Keď už sme mu poslali presne toľko snímok, koľko je vektorov pre svetlá v súbore *lights.yml* (v mojom prípade teraz 48) tak spustím funkciu pre skonštruovanie normálovej mapy:

MTS_PhotometricStereoForDesinger.MTS_PSImageData psImage =
*$PhotometricStereo_cam1.**processPhotometricStereo_GPU**(correctByPaper);*

Aktuálne máme normálovú mapu v internom formáte aby sa s ňou dalo efektívne pracovať ďalej. Takže, ak si ju chcem pozrieť, alebo s ňou ďalej pracovať ako s farebným obrazom, potrebujem si ju previezť na RGB snímku. Na to môžem použiť metódu objektu samotnej normálovej mapy, kde získam BMP verziu:

```
psImage.getRGBImage();
```

No aby som s tým mohol pracovať vo vision pro ďalej, zabalím si tieto BMP dáta do VisionPro obrazového objektu:

```
return new Cognex.VisionPro.CogImage24PlanarColor( psImage.getRGBImage());
```

S týmto už potom klasicky fungujeme vo VisionPro ako s farebným obrazom.

Odporúčam ukladať si tieto normálové mapy aj v originálnom formáte, pomocou metódy:

```
psImage.saveAsFPNFile(<cesta súboru> + ".fpn");
```

Načítanie z tohto súboru sa potom robí pomocou metódy **loadFromFPNFile(filePath)** teda napríklad takto:

```
MTS_PhotometricStereoForDesinger.MTS_PSImageData imData
= new MTS_PhotometricStereoForDesinger.MTS_PSImageData();
imData.loadFromFPNFile(@"C:\MTS\PS_TemplateImages\imgs\objectImage1.fpn");
```

Vyhodnocovanie pomocou normálovej mapy

TODO:

==== Dopísat pouzitie filtrov ===

==== spomenut neuronove siete ====