Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

3D grafikus szoftver alkalmazása neurális hálót tanító adathalmaz generálásához

TDK dolgozat

Készítette:

Mess Gergely István, Simon Csongor, Dominguez Zoltán

Konzulens:

Dr. Csorba Kristóf

2020

Tartalomjegyzék

Kivonat

Több iparágban, például a gyógyszer- és vegyiparban a pelletek, tabletták minőségellenőrzése leggyakrabban úgy zajlik, hogy a gyártás végén véletlenszerűen kiválasztott termék minőségéből extrapolálnak a teljes sarzs minőségére. Ez szinte mindig egy kevés termék veszteséggel jár. Egy esetleges rossz minőségű termék ellenőrzése a teljes sarzs hibásnak ítélésével járhat, ami feleslegesen nagy anyagi veszteséget jelent. Így tehát a lehető legjobb anyag-veszteség mentes köztes minőség-ellenőrzés nagyon értékes ilyen iparágakban.

Mára már a számítógépek számítási kapacitása oly mértékűre nőtt, hogy akár valósidőben képesek vagyunk neurális hálókkal videóból információt kinyerni. Ezek az információk lehetnek pelletek vagy tabletták mennyisége vagy alaktani jellemzői, mint például az átmérőjük. A tanítás nehézsége azonban nem a neurális hálók algoritmusa, hanem az őket tanító adathalmazok előállítása. Megfelelő minőségű és mennyiségű adat előállítása rengeteg szakértői munkaórát is igénybe vehet. Azonban ismerve a formai jellemzőiket 3d grafikus szoftverrel lehet az alakjukat és megfelelő beállításokat használva textúrájukat is élethűen modellezni.

Dolgozatunkban egy ilyen módszer lehetőségeit vizsgáljuk. Különböző méretű és alakú 3d modellek fizikai szimulációjával modellezzük például a valóságban szárító tálcára vagy futószalagra kiszórt pelleteket. 3d grafikus motornak a Blender-t használjuk, melynek interfésze lehetőséget ad arra, hogy létrehozzunk egy szoftvert tanító adat generálásra, majd a generált adatokra tanított háló teljesítményét teszteljük valós adatokon.

Abstract

The quality assurance and measurement of different products for example pellets and drug pills usually conducted by taking out a small portion of the batch randomly and extrapolating from their properties to the whole batch's properties. These methods result in a small loss most of the time. Furthermore, it is a huge monetary loss if the batch's quality doesn't reach the desired standard. So, intermediate quality assurance methods that doesn't involve raw material or product loss are highly beneficial.

In this day and age computational power reached a point where we can process videos in real time with neural networks. With such methods we can evaluate certain properties of pellets like its diameter or shape. The difficulty however is not the neural network but getting sufficient training data both in quality and quantity. Manual labelling takes a lot of time to produce even for experts of the field. With the knowledge of the shape and size of the pellets we could use a 3d graphics software to model them and their texture with the proper settings.

In this paper we would like to explore the potential of such method. We use physical simulation of 3d models with different size and shape to simulate pellets on a surface for example a cooling rack. We use Blender as a 3d engine and its programming interface to write a software that generates training data. We then evaluate the performance of a neural network trained on this dataset with real life photos.

1. fejezet

Bevezetés

Az iparban található több iparágban jellemző az a minőségellenőrzési technika, mikor egy gyártási fázis végén, a gyártott termékekből véletlenszerűen mintavételeznek, a vett minta jellemzőiből pedig következtetnek a teljes sarzs, vagyis az összes termék jellemzőire. Ezzel a módszerrel a probléma, hogy minták paramétereinek minőségbiztosítási határérték alatti értéke esetén a teljes sarzsot rossz minőségűnek bélyegezhetik.

Bár a mintavételezés matematikai és méréstechnikai szempontok figyelembevételével hatékonyan becsüli a vizsgált jellemzőket, de nem vitatható az a tény, hogy több minta felhasználásával hatékonyabb becslés adható. Több minta vizsgálatához gyorsabb és hatékonyabb mérési módszerekre van szükség, különben a vizsgálat költsége és időigénye nem éri meg azt, hogy a minőségbiztosítás fejlettebb legyen.

Egy hatékony módszer a folyamatba iktatott köztes ellenőrzés, mely a jellemzőket a folyamat közben méri, legoptimálisabb esetben különböző aktuátorokkal be is tud avatkozni a folyamatba. Ezzel a módszerrel a jellemzők nem csak mérhetők, hanem szabályozhatóvá válnak. A dolgozatunkban a folyamat közben történő mérés lehetőségeit kutatjuk fel, különös tekintettel a videóképpel végezhető mérésekre.

A videóképen valós időben végzett adatfolyam feldolgozására az elmúlt években nyílt lehetőség, a számítási kapacitás növekedésének köszönhetően. Mára ilyen adatfolyamok feldolgozásához nem kell szuperszámítógéppel rendelkezni, így az ipar számára is gyorsan megtérülő lehet a technológia, amennyiben hatékonyan képes a jellemzők mérésére.

A videóképpel történő mérés előnye továbbá, hogy kontaktmentes, tehát a terméket fizikailag nem kell hatásnak kitenni, elég távolról egy kamerával megnézni. Ezzel a tulajdonsággal a gyártósorba integrálás folyamata könnyűvé és mobilissé válik, bővebb termékhalmaz lesz elérhető. A videófolyamon történő mérés visszavezethető a képen történő mérésre, ezért a továbbiakban állóképen végzett méréseket említünk.

A munkánk során speciálisan műanyag pelletekkel foglalkozunk, mely egy vegyipari intermedier, bizonyos műanyag termékek gyártási folyamata során használt köztes termék. Ilyen speciális pelletek álltak rendelkezésünkre a Budapesti Műszaki és Gazdaságtudományi Egyetem PharmaTech laborjában található kutatói célra felállított gyártósornak köszönhetően. Ebből a gyártósorból került ki az összes dolgozatban szereplő pellet. Az eljárásunk minden bizonnyal egyéb pelletekre is alkalmazható, de egyéb pellet típusok tesztelésére nem volt lehetőségünk.

Első megközelítésben a pelleteknek a pozíciójának és a határoló dobozának meghatározását tűztük ki célul a képeken, illetve a videófolyamokon. A határoló doboz méretéből következtetni lehet a pelletek méretére, illetve formájára is. A termékek pozícióiból következtetni lehet futószalagon haladó termékek esetén az anyagáramra.

Egy folyamatirányító rendszerbe ezeket az információkat visszacsatolva elképzelhető, hogy a pelletek túl kis mérete esetén a folyamat nagyobb pelletek gyártásába kezd. A minőségbiztosítási paraméterektől függően a kis pelletek eldobhatók, vagy az átlag korrigálható a gyors beavatkozásnak köszönhetően, így a teljes sarzs minősége helyreállítható. Az anyagáram tulajdonsággal korreláló hibák gyakorlatilag azonnal kiszűrhetővé válnak, azáltal, hogy a folyamatra nem jellemző anyagáram mennyiséget mér a rendszer.

1.1 Adattömegek szükségessége

Napjainkban az adat egy nagyon értékes erőforrás, mivel kellő mennyiségű adat alapján ügyesen konstruált rendszerek hasznos információkat tudnak kinyerni, illetve akár erre alapozva döntést tudnak hozni. Például az előzetesen érkezett email tömegek alapján modellek képesek spamet detektálni, de napjainkban már mobiltelefonokon is különböző alkalmazások használnak arcfelismerést, vagy fényképezőprogramok felismerhetik a fotózott tárgyakat és élőlényeket, és ehhez igazíthatják a beállításaikat.

A modellek bonyolultságának és méretének növekedésével egyre több adat válik szükségessé. Ennek egyik fő oka, hogy a modell jó általánosító képességének eléréséhez az egyik legkézenfekvőbb és néha egyetlen módszer a nagy mennyiségű tanító adathalmaz. A nagy mennyiség biztosítja azt, hogy a modell nem lokális jellegzetességeket, hanem sokkal inkább a globális jellemzőket tanulja meg.

Adatok előállításának vagy gyűjtésének a nehézsége az, hogy a legtöbb modell felügyelt módon tanul, azaz szüksége van az adatokhoz tartozó megfelelő címkékre. A címkék mondják meg a modellnek, hogy az adatokhoz milyen érték vagy csoport tartozik. Például egy arcdetektálást végző modellnek nem elég különböző képeket biztosítani, meg kell mondani a képekről, hogy van-e rajtuk emberi arc vagy sem.

A képek felcímkézését nehéz automatizált módon végezni, hiszen, ha ez könnyű lenne, akkor valójában a feladat is megoldódott ezáltal. A képek felcímkézését manuális annotációnak is nevezi a szakirodalom, a későbbiekben így is hivatkozunk rá.

1.2 A manuális annotáció problémái

A kézi felcímkézés a pelletek esetében azt jelenti, hogy a pellet halmazt tartalmazó képeken a pozíció és a határoló téglalapot az annotálást végző embernek kell meghatározni. A folyamat több nehézséggel is rendelkezik.

Egyrészt nehézséget okoz az annotáló embernek a vizuális telítődés. A pelletek sűrű elhelyezkedése miatt a határoló dobozok elfedik a teljes képet, így némely pelletek eltűnhetnek a túltelítődött képen. A meglévő határoló dobozok vizualizációról történő elvétele sem eredményes, mert ekkor annak a veszélye áll fenn, hogy egy pelletet többször detektálunk.

Másrészt problémát okoz az emberi fáradás. A monoton munkafolyamat miatt a hibák száma nagyon megugorhat egy idő után, mivel az ember bizonyítottan nem képes folyamatos koncentrációra több órán keresztül.

Harmadrészt az annotációhoz szükséges idő és skálázhatóság sem optimális. Az emberi annotáció méréseink alapján képenként nagyjából 45 percet vesz igénybe. A gyorsaság a pontosság rontása nélkül nem növelhető, a tapasztalat sem segít a feladat nagy mértékű gyorsításában. Így a skálázás több emberi erőforrás bevetésével lehetséges. Például tízezer kép annotációja egy 10 fős csapatnak a fentebb számolt sebességgel több mint 31 napjába telne, ami 8 órás munkanappal számolva több mint 93 munkanap.

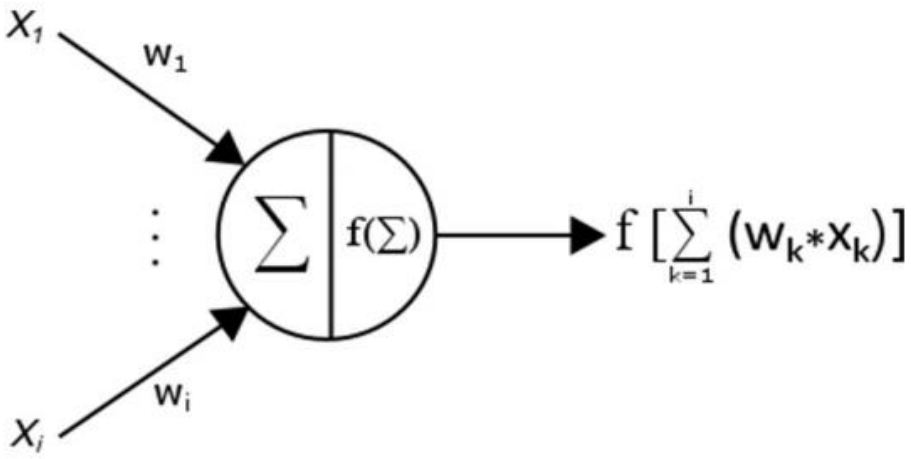
Ezzel szemben, ha képeket és hozzá tartozó annotációkat képesek vagyunk automatikusan generált módon előállítani, akkor ennél jelentősen gyorsabb módszer is rendelkezésünkre áll. A generálás sebessége ráadásul nem emberi tényezőtől, hanem számítógép erőforrás kapacitástól függ, ami jóval könnyebben skálázható.

2. fejezet

Kapcsolódó irodalom

Ahogy a bevezetőben már említettük, a gépi tanulást használó rendszerek bizonyos problémákra nagyon hatékony megoldást nyújtanak. A gépi tanulást használó modellek előnye, hogy nem kell a megoldás pontos és minden esetet lefedő menetét belekódolni a rendszerbe. Ehelyett a modell matematikai modellek segítségével a tanító adatok alapján képes megtanulni egy megoldási sémát, amire a definiált hibafüggvény minimális lesz.

A gépi tanulást használó modellek egyik speciális formái a neurális hálózatok [1]. A neurális hálózatok neuronokból állnak, melyek a bemenő adatokból egy aktivációs függvény segítségével kiszámítják a kimenetet. A neuronok egymással összekapcsolhatók, így alkotva egy teljes hálózatot. A neuronok bemeneteit súlyozza a hálózat, a tanítási folyamat ezen súlyok módosítása a hibafüggvény minimalizálása érdekében.



1. ábra - A neurális hálózatok építőelem a neuron. [2]

Napjainkban a komplexebb hálózatok több millió, de akár több milliárd súlyból, illetve neuronból állnak, ezáltal rendkívül összetett feladatokat is képesek megoldani.

2.1 Szintetikus tanuló adat előállítás

Szintetikus adatokról akkor beszélünk, ha ezek nem valós szenzorok eredményeként, hanem mesterségesen állították elő valamilyen módon. Képek esetében beszélhetünk olyanról, amiben a kép bizonyos részei lettek előállítva mesterségesen, vagy teljesen mesterséges képekről, mint a mi esetünkben is. Próbálkozás szintetikus tanító halmaz előállításra képfeldolgozási feladatokra egyáltalán nem újdonság.

A dolgozatunk témájául szolgáló gyógyszergyártási felhasználás egy speciális eset, de más köznapibb felhasználással foglalkozó munkák megvizsgálása is jó iránymutatásul szolgál. David T. Hoffmann et al. [3] többszemélyes póz becslő algoritmus tanítását vizsgálták részlegesen szintetikus képek segítségével, viszonyítási pontként egy tisztán valós képeken tanított háló szolgált. A csak a szintetikus képeken tanított hálóval nem értek el jobb eredményt, viszont amikor a valós képek halmazát dúsították a szintetikus képekkel akkor a haló jobb teljesítményt ért el.

Mivel a feladatuk a miénkhez képest jóval komplexebb ezért arról nem nyújt információt, hogy a mi általunk tanított háló teljesen szintetikus halmazzal milyen eredményt ér majd el, de következtetésként levonható, hogy a feladat komplexitásától nagyban függ a szintetikus halmaz alkalmazhatósága. Az viszont jó jel, hogy a vegyes halmazzal jó eredményt értek el úgy, hogy a szintetikus emberek a képeken nem voltak valósághűek.

Chaitanya Mitash et al. [4] objektum detektáló modellt tanítottak fel szintetikus képekkel, a halmaz által tanított modellek teljesítményét azzal növelték, hogy az objektumokat fizikai szimulációval helyezték el, ami a mi feladatunk esetében is megfelelő a pelletekre. Matthieu Grard et al. [5] szintetikus mélység képeket generáltak kontúr detektáló algoritmus tanításához szemétválogatási feladatra.

A mi feladatunkhoz hasonló körülmény, hogy a képeken sok objektum található, amik átfedésben vannak egymással. A mesterséges kép generálás során a valós kamerákkal történő mélységi képalkotási folyamathoz hasonlóan két egymástól kicsit elmozdított kamera állásból készítettek RGB képeket ahelyett, hogy a 3D-s szoftverrel mélység képeket állítottak volna elő. Ezt követően az így elkészült színes képeket használták fel mélység képek elkészítésére, ezzel másolva azt, ahogyan a valós képek estében jártak volna el, így elérve a megegyező képi hatásokat. Ebből levonható az a következtetés,

hogy nem feltétlenül kell a szintetikus képnek annak lennie, amit közvetlenül használnak, hanem lehet a valós folyamat egy bizonyos pontjához szükséges kép is.

Az esetükben a tisztán szintetikus halmazon tanított modell is jó eredményeket ért el, de a generált képeken jobban teljesített a valós képekhez képest, feltehetően a képek közti valósághűségben való különbségek miatt.

Love Lidberg sértő szimbólumokkal ellátott zászlók észlelésére alkalmas neurális haló tanításához készített részben szintetikus képeket [6]. Az ő esetében a legjobb eredményt az a modell érte el amelyiket olyan halmazon tanított, ami vegyesen tartalmazott valós és szintetikus képeket, azonban a tisztán generált képeken tanított háló is 80%-os pontosságot ért el.

Az előző két munkán látszik, hogy amennyiben nem túlságosan komplex feladatra szeretnénk a neurális hálót megtanítani, akkor elég lehet a szintetikus képekből álló halmaz a tanításhoz.

Xingchao et al. [7] olyan neurális hálók teljesítmény változását vizsgálták meg, melyeket objektum detektálásra tanítottak szintetikus adatokkal. Olyan eseteket vizsgáltak, amikor bizonyos alacsony szintű részletek nincsenek jelen, például a textúra, póz, valós háttér, vagy a forma.

Munkájukból azt tudták levonni konklúzióként, hogy az előre feltanított halók kevésbe érzékenyek az ilyen részletek hiányára. Azonban, az előre nem feltanított modellek esetében ezek hiánya teljesítmény vesztést okoz. Ez a mi esetünkben nem jelent gondot, mert a munkánkhoz a YOLO keretrendszer egy előre feltanított hálóját használjuk, így abban az esetben is képesek leszünk megfelelően tovább tanítani a modellt, ha bizonyos részeletek hiányoznak is a szintetikus képeinkről.

2.2 Blender

Fontos megemlíteni még a használt 3D szoftver kérdését, több előzőleg említett munka, valamint egyéb munkák [8] is a Blendert [9] használták, mint 3D tervező szoftvert. Ennek az oka a szoftver nyíltsága és az ingyenessége, valamint egyszerű szkriptelhetősége. A munkánkhoz mi is a Blendert választottuk az előzőleg felsorolt indokok miatt. Természetesen a modellezés lehetséges más szoftverekkel is, például az Autodesk által készített szoftverekkel vagy akár egy saját 3D-s szoftverrel is.

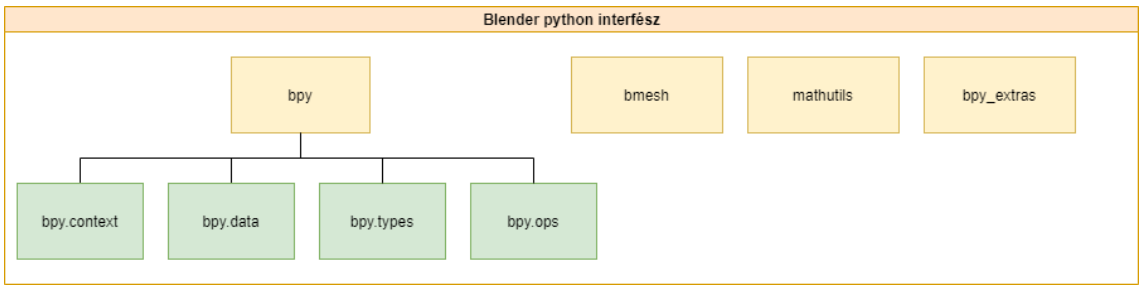
A Blender a BlenderFoundation és a Blender Institute szervezetek által készített ingyenes és nyílt forráskódú 3D alkotó környezet, ami támogatja a számítógépes képalkotással foglalkozó iparágak (játékfejlesztés, filmkészítés stb.) által használt folyamat rendszert, modellezést, csontozást, animációt, szimulációt, renderelést, kompozitálást, mozgáskövetést, videószerkesztést és a 2D-s animációt is.

A Blender központi elemeit C/C++ nyelven írták, így a mélyebb funkcionális bővítés ezeken a nyelveken keresztül lehetséges. A fejlesztéséhez történő csatlakozás mikéntjéről részletes dokumentáció található a Blender oldalán. Az olyan programozási feladatokhoz, amik nem igényelnek nagymértékű funkció bővítést a szoftver egy Python programozói interfészt szolgáltat [10], amivel elérhetők a vizuális felületen keresztül használt funkciók, valamint a speciális programfájlokban tárolt adatok.

A letölthető kiegészítő szoftver elemek ezen az interfészen keresztül készíthetők el. A Blender az elkészített Python szkripteket egy saját Python futtató környezetben futtatja, ami elkülönül a globális rendszer futtató környezettől, így a Python verzió és a telepített modulok is különbözők lehetnek.

2.2.1 A Python interfész

A Blender által kínált Python programozói interfész több külön álló modulból épül fel, a Blenderhez tartozó fő modul a „bpy”, ami további almodulokra bomlik szét.



2. ábra A Blender Python interfészének főelemei

Az 2. ábra szemlélteti a Python interfész által szolgáltatott modulokat, amik az esetek legnagyobb részében felhasználásra kerülnek. A bpy modulon belüli type modul tartalmazza a Blender által definiált típusokat.

A programon belüli adatokat a szoftver maga tartja karban, ezek eléréséhez pedig a bpy.data modul szolgáltat függvényeket. A Blender által definiált típusokból csak a kínált

függvényekkel lehetséges új objektum létrehozása, annak érdekében, hogy a rendszer képes legyen a benne lévő adatokat felügyelni. Az adatok alaposztályaként a bpy.types.ID osztály szolgál, ebből az osztályból leszármazó osztályok objektumaihoz saját tulajdonságokat is hozzárendelhetünk így általános módon extra adatokat szolgáltathatunk az objektumainknak, amennyiben szükséges.

A bpy.context modul a Blender jelenleg aktív kontextus elemiért felel. Az ebből a modulból elérhető információk a legtöbb esetben csak olvashatóak. Blenderben a kontextusok különböző felületi paneleknek felelnek meg, mint például a 3D szerkesztő nézet, a shader szerkesztő, kódszerkesztő, fájl elérő ablakok.

A felületi elemeken elérhető parancsokat a bpy.ops kínálja fel. Ennek a modulnak a segítségével amikor egy bizonyos lépés sorozatot szeretnénk automatizálni, akkor a manuálisan a gombok által használt műveleteket lehet meghívni. Komplexebb feladat esetén viszont kerülendő ezeknek a használata, mert az operációk kontextus függőek és egyéb nem kívánt mellékhatásokat okozhatnak.

A mathutils külön álló modul a Blenderhez használt matematikai elemekből és tulajdonságokból épül fel, mint a vektorok, mátrixok, kvaterniók.

2.3 A BlenderProc keretrendszer

Maximilian Denninger et al. a Blender által kínált Python interfészre építve készítettek egy nyílt keretrendszert szintetikus adathalmazok előállítására képfeldolgozó algoritmusok tanításához [11]. A keretrendszer az adathalmaz generálási feladatokra egy csővezeték rendszer felépítését biztosítja, amiknek a leírása yaml [12] kiterjesztésű fájlokban lehetséges. A csővezeték rendszer részegységei a modulok, amik jól elkülönült feladatokat látnak el, például objektum betöltő modul, RGB kép előállító modul, annotáció író modul.

A keretrendszer bővítése egyszerűen megtehető interfészek megvalósításával és a modul ősosztályból leszármazva új modulok létrehozásával. A konfigurációs fájlban értékeket definiálhatunk, amiket a modul kiolvashat és felhasználhat a futása közben. Egyes modulok az őket megelőző modulok eredményeit felhasználhatják úgy, hogy egy globális kulcsot rendelnek a modul kimenethez, amivel később el lehet érni az eredményt, vagy pedig a Blender által kínált adat tulajdonságokat használják az eléréshez.

A keretrendszerbe hivatalosan több adathalmazzal történő együttműködés is be van építve, mint például a Replica adathalmaz, ami 18 fotórealisztikusan rekonstruált belső szoba szcenáriót tartalmaz [13], valamint a SUNCG adathalmaza, ami szintúgy belső tér rekonstrukciókat foglal magába [14]. A keretrendszer felhasználták a BOP kihíváshoz [15] [16] nagy méretű szintetikus adathalmaz generálásra megadott objektum halmazból, ez az adathalmaz különböző felületeken elhelyezett mindennapi tárgyakat tartalmazott 6D-s póz becslő algoritmusok felhasználására.

A keretrendszer könnyű bővíthetősége miatt és mivel láthatóan jó eredményeket értek már el vele a múltban ezért a saját munkákhoz is ezt a keretrendszert fogjuk használni a kiinduláshoz, majd bővítjük a feladatunkhoz szükséges funkcionalitásokkal. Ezzel továbbá azt a hibát is elkerüljük, hogy amikor mások szintetikus adathalmaz generálási feladatban kezdenek, újra és újra létrehozzák azokat az alapvető alkotó elemeket, amik nagy általánosságban szükségesek egy generálási folyamathoz, például a kép előállítás, annotációk létrehozása, objektumok kezelése betöltése. Az általunk létrehozott funkció bővítéseket továbbá az open-source projekthez adjuk, így másnak is lehetősége lesz felhasználni azokat a saját munkájához.

2.4 Transfer learning

A transfer learning [17] technikájának használata a modell felépítésében és tanításánál is több előnnyel is jár. A módszer azért működik jól, mert hálózatok architektúrájának tervezése és felépítése rendkívül időigényes és nehéz folyamat. A transfer learning használatával egy jól bevált modell architektúráját át lehet emelni, és a kimeneti réteg megváltoztatásával egy más feladat megoldására lehet átképezni a modellt.

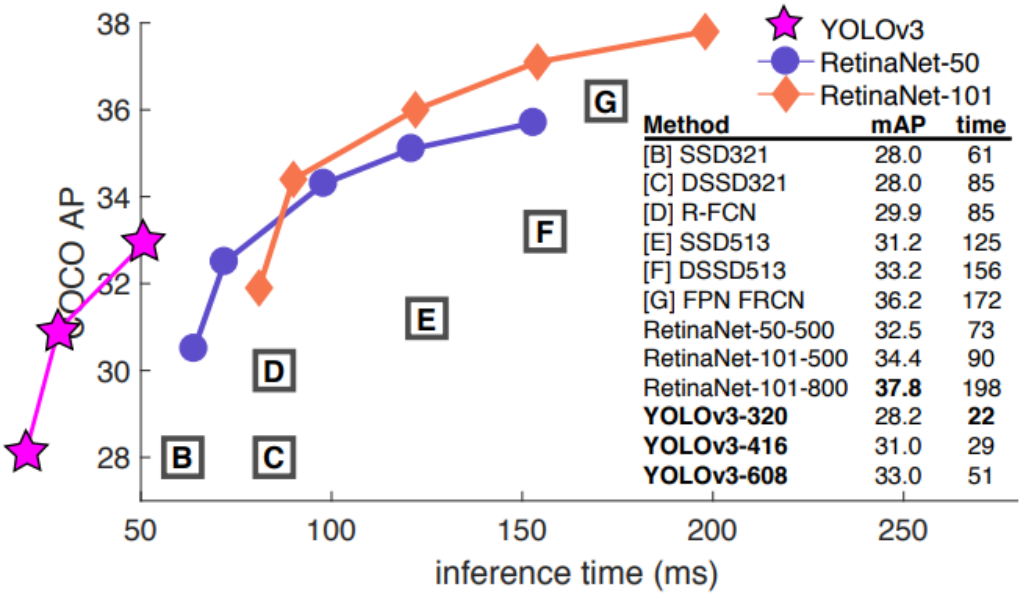
Egy másik előnye a technikának, hogy a modellen egy tanítási folyamat lezajlott egy jóval nagyobb számítógép erőforrást és egy lényegesen nagyobb adathalmazt használva. Ezekből következtethetünk arra, hogy a modell jól képes általánosítani, nincs túltanítva egy adott speciális problémára. A saját speciális esetünkre természetesen meg kell tanítani a modellt, de ez gyakran a kimeneti réteg módosításával is megtehető eredményesen.

2.5 Kész képfeldolgozó architektúrák

Képekkel foglalkozó modellből bőséges mennyiségű létezik, amelyekből rengeteg interneten is megtalálható és használható. Ilyen modellek például a R-CNN, Fast R-CNN, DPM vagy a YOLO [18]. Ezek a modellek konvolúciós hálózatok, melyek határoló kereteket határoznak meg.

2.5.1 A YOLO architektúra

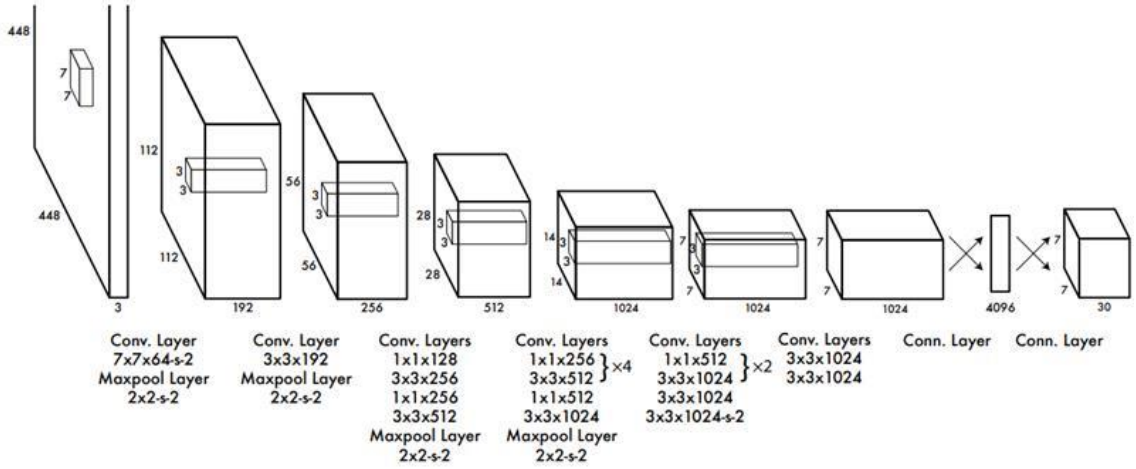
A modellek közül a YOLO nem a legpontosabb, viszont az összehasonlításokból [18] látható, hogy a teljesítményt is figyelembe véve egy gyors és a legtöbb alkalmazáshoz elég pontos modell, amely akár valós idejű detekcióra is képes. A YOLO architektúrának több változata is létezik, az architektúrát többen továbbfejlesztették, mivel az alternatívákkal összehasonlítva gyors detekciót tesz lehetővé.



3. ábra - A különböző architektúrák összehasonlítása [19]

A modell gyorsaságát a kompaktsága okozza. Míg a legtöbb képfeldolgozó tartalmaz külön konvolúciós hálót és klasszifikálót, addig a YOLO csupán egyetlen háló, konvolúciós és klasszifikáló rétegekkel. Ez lehetővé teszi a hatékony tanítást, hiszen csak egy hálót kell tanítani. Továbbá a YOLO detektálás során a teljes képet látja nem mozgó ablakokkal oldja meg a több objektum detektálását, hanem szegmensekre osztja a képet

és minden szegmensben 2 objektumot detektál. Ez egyben egy hibája is, hiszen, ha a szegmensen belül több, mint 2 objektum van akkor csak azt a kettőt detektálja, mely a legvalószínűbb. Még egy számunkra kifejezetten nagy előnye annak, hogy a teljes képet látja, hogy sokkal általánosabb jellemzőkre tanul rá. Nagyon jól szerepel valós képeken tanítva festményeken vagy rajzokon detektálva. Tesztek során a VOC 2007-es adathalmazon tanítva emberek felismerésére 59.2-es precizitást ért el, majd ugyanezekkel a súlyokkal a Picasso adathalmazon, mely festményekből áll közel azonos 53.3-as átlag precizitással detektált [18].

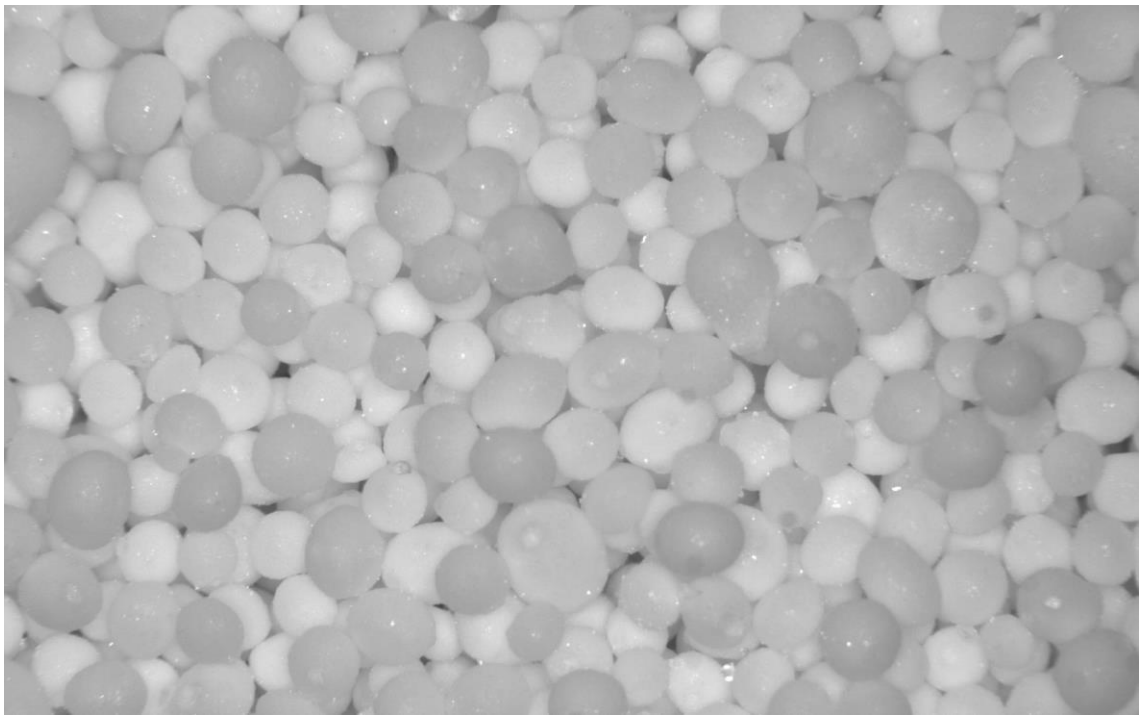


4. ábra – A YOLO architektúra

3. fejezet

A pellethalmaz modellezése

Ahogy az előző fejezetben említettük, a feladatunkhoz a BlenderProc keretrendszert fogjuk használni, illetve szükség esetén tovább bővíteni. A keretrendszer használatával fogjuk a pelleteket tartalmazó szintetikus képeket automatikusan generálni és a képfelismerő algoritmus tanításához szükséges annotációkat előállítani. Az előtt viszont, hogy megtudnánk alkotni a generáló rendszert, azt kell modelleznünk, hogy milyen elemekre lesz szükségünk, és milyen paramétereket, illetve változókat kell számításba venni. Ehhez először manuálisan rekonstruáltuk a lépéseket Blenderben, hogy olyan elrendezést és benyomást kapjunk, ami a válóságnak megfelelő.



5. ábra - Valós kép a pelletekről

3.1 Pellet objektumok létrehozása

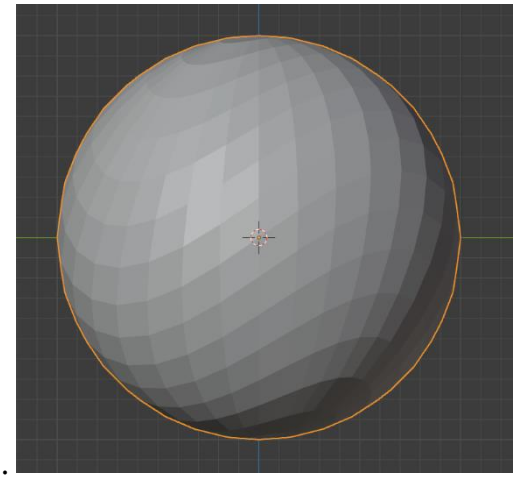
A pelletek modellezésének folyamata két eltérő módon történhet. Egyrészt elkészíthetünk manuálisan egy véges modell halmazt melyek elemeit fájlokban elmentjük és ott tároljuk.

Később ugyanezeket a modelleket használhatjuk fel a folyamat futásai során mintavételezés használatával. Másrészt procedurálisan generálhatjuk a modelleket és ezeket tárolhatjuk el vagy generálhatjuk újra minden egyes futásnál külön-külön.

A megoldáshoz a második opciót választottuk, mivel a pelletek esetében nincs túl nagy alakbeli különbség, így a procedurális generálás egy egyszerű feladat volt. Ezzel a módszerrel továbbá nagyobb varianciát is nyerünk a tanító halmaznak. Az első opciónak is van létjogosultsága, leginkább olyan feladatoknál, ahol a modellek alaki különbségeit nem, vagy csak nehezen lehetne procedurálisan előállítani. A mi esetünkben is használható lenne a módszer, melyben a modellek véletlenszerű pozicionálásával, illetve a mintavételezés természetéből adódóan is nyerhetnénk változatosságot a képeken. Azonban tanításhoz a lehető legnagyobb mértékű varianciát tartottuk fontosnak, mivel a változatosabb képek csökkentik a túltanulás esélyét.

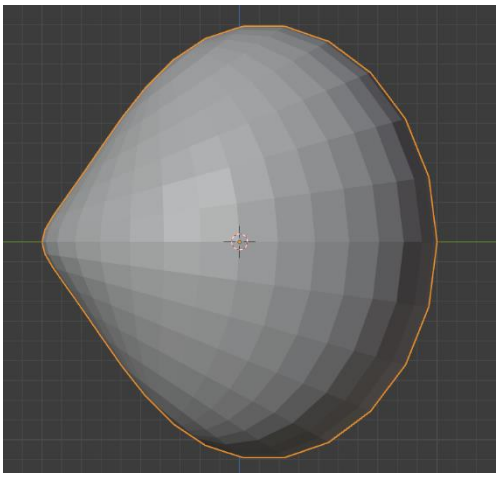
A képek központi elemei a pelletek, így a legfontosabb feladat azok tulajdonságainak meghatározása és modellezése volt. Az objektumok generálásához egy a programba beépített objektum manipulátort használtuk. Blenderben az objektum manipulátoroknak nagy szerepe van és széleskörben bevethetők. Az általunk használt manipulátor egyszerű deformációs műveleteket hajt végre az objektumokon egy kiválasztott tengelyen. Ennek segítségével biztosítva van a pelletek véletlenszerű formája, és alakbeli szórása, ami jól modellezi a valós pelletek tulajdonságait. A lehetséges műveletek a tekerés, hajlítás, csúcsosítás és a nyújtás, a pelletek modellezésnél mind a négyet hasznosítottuk.

A tekerés megadott szögben elforgatja az objektum vertexeit a definiált tengely körül, az ábrán látható módon.



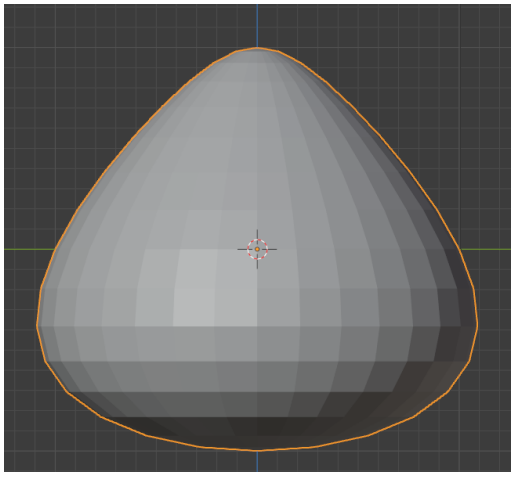
6. ábra - A tekerés manipuláció

A hajlításkor a tengely körül az objektumot egy szögben hajtja meg.



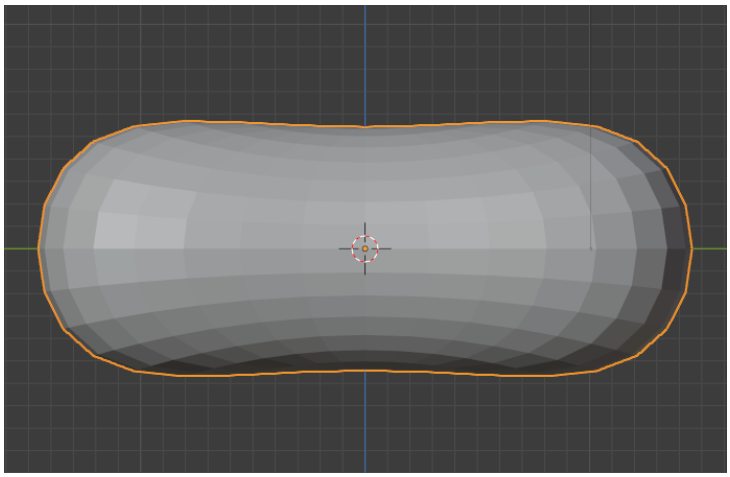
7. ábra - A hajtás manipuláció

A csúcsosításkor a manipuláció faktor értéke alapján a tengelyen áthelyeződik lényegében az objektum súlypontja, ezzel az egyik oldala hegyesebb formát ölt.



8. ábra – A csúcsosítás manipuláció

Nyújtáskor a manipulátor faktor értéke alapján elnyúlik az objektum a kijelölt tengelyen.



9. ábra – A nyújtás manipuláció

Fontos, hogy itt a tengelyeken az objektumok lokális tengelyét értjük és a nem a globális tengelyeket. Az objektumok lokális tengelyei az objektum origójából indulnak, ami az objektumok központját jelképezi. Egy manipulátor egy tengely körül képes egy műveletet végrehajtani, az effektusok kombinálásához viszont lehetséges akár több manipulátort is elhelyezni egy objektumon. A pelletek alapjául szolgáló objektum egy primitív gömb, amire manipulátorokat alkalmazunk.

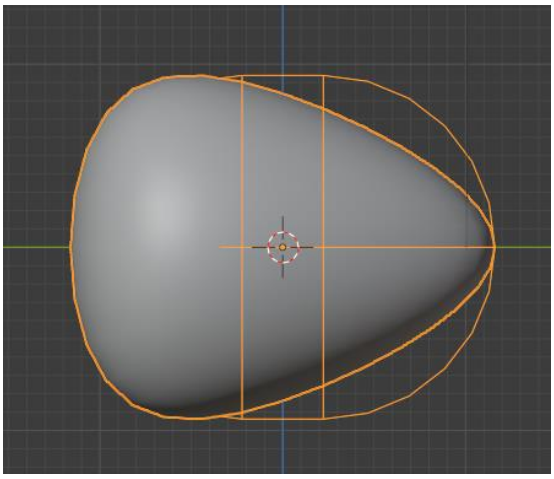
A gömb létrehozásakor fontos szempont, hogy hány darab vertexet tartalmaz, ami azért lényeges, mert minél több vertexből áll egy objektum, annál nagyobb lesz a sebességre való lassító hatása. Több száz objektum szerepel majd egy szcenárióban ezért a vertexek száma nagyban befolyásolja a képek elkészítésének sebességét.

Az objektumot felépítő pontok száma azonban túl kevés sem lehet, mert a manipulátor által végrehajtót változtatások is természetesebbnek hatnak minél több vertex áll rendelkezésre, valamint az élethűség szempontjából is fontos, hogy ne látszódjanak éles szögek a pelleteken. A sima felület létrehozását viszont segíti, hogy be lehet állítani az objektumoknál, hogy sima árnyalást szeretnénk így interpoláció segítségével kevés ponttal is elérhető a lekerekített felület hatása. Szerencsére a gömb létrehozás alapvető beállításaival megfelelő kinézetet lehet elérni, úgyhogy nem túlságosan nagy a vertex szám sem, így nem szükséges, hogy módosítsuk az alapértékeket. A generálás fejlesztői implementálását a Pellet generátor modul fejezetben részletezzük.

3.2 Fizikai szimuláció

Miután megvizsgáltuk a pellet objektumok előállításának a problémáját, a következő feladat az elhelyezésük. A megfelelő változatosság miatt ehhez szilárd test szimulációt alkalmazunk. A szimulációhoz szükséges egy olyan objektum, amire a ráejtjük a pelleteket, erre a célra manuálisan egy kocka objektumból készítettünk egy tartót, hogy a leeső pelleteket egy helyen tartsuk. Ez az objektum a szimulációban passzív objektumként fog részt venni, magyarán nem fog rá hatni a gravitáció, az a feladata, hogy a leeső pelletekkel interakcióba lépjen.

A szimulációban résztvevő objektumoknak van egy ütközési forma tulajdonsága, ez azért lényeges mert a szimuláció sebességét is befolyásolja, hogy a benne szereplő objektumoknak mekkora a vertex száma. Lehetséges azonban a komplex objektumok helyett a szimulációban hozzájuk rendelni egy egyszerű közelítő objektumot, amit a szimuláció használ a számításokhoz. Ennek olyan hátrányai lehetnek, hogy amennyiben a szimulációs objektum és az eredeti objektum között túl nagy a különbség, akkor nem realisztikus pozíciók keletkezhetnek. A Blender öt statikus objektum lehetőséget kínál fel, melyek a gúla, cilinder, kapszula, gömb és kocka. Ezek előnye, hogy nagyon kevés vertexből épülnek fel. A pelleteket legjobban a kapszula közelítette meg, azonban, ha csúcsosabb alakú a pellet akkor nagy tér jön létre a két objektum között, így amikor egymásra szórjuk a pelleteket akkor túl nagy távolságok keletkezhetnek.



10. ábra – A kapszula ütközéstest

Ezeken az alapobjektumokon kívül még két opció van, a test saját formája és a konvex burok. Természetesen az eredeti objektumot használva lehet elérni a legjobb eredményeket, de ez akár 3-5x lassabb szimulációt eredményezhet. A konvex burok egy közelítő objektumot hoz létre az eredeti objektum alapján, ez ahogy a neve is sugallja konvex testek esetén működik jól mert konkáv testeknél áthidalhat részleteket. Mivel a pelletek konvexek így ez tökéletesen megfelel nekünk plusz időt is nyerve azzal, hogy nem az eredeti testeket használjuk. A tartó objektum esetében viszont az eredeti objektum használatát választottuk, mivel a vertex száma kicsi ezért ez nem okoz gondot.

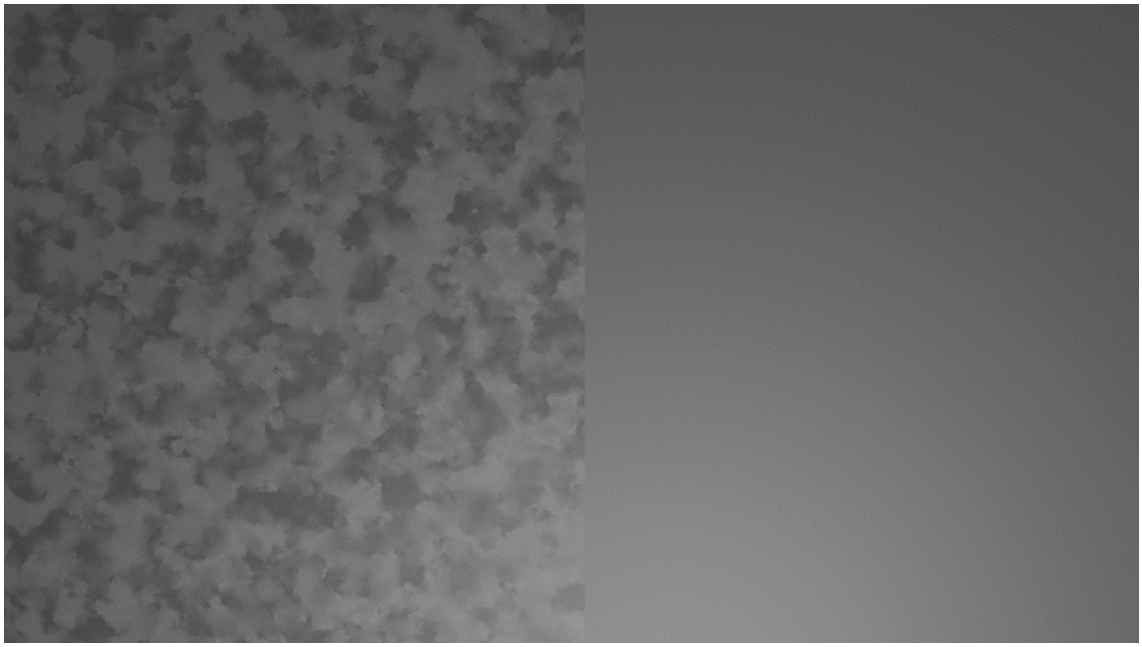
A szimuláció során még az is problémát okozhat, ha a pelleteket túl magasról ejtjük le, mert ezáltal a sebességük túl nagy értékű lesz, a szimulációs lépések közti időtartam pedig nem elég kicsi ahhoz, hogy amikor a tartó objektumhoz ér megfelelően kiszámítsa az ütközést, ami azt eredményezi, hogy átesik a tartó objektumon. Ennek a kiküszöbölésére lehetséges megadni egy távolság küszöbértéket, ami hozzászámítódik a testtől való távolsághoz az ütközés érzékelésnél. Ennél figyelni kell arra, ha túl nagy értéket adunk meg akkor lebegés effektus lesz tapasztalható. A tesztesetek alapján ennek az értéke kb. 0.02 egység körül optimális olyan helyzetekre amikor viszonylag magasból, 70 egység magasból ejtünk le pelleteket, ekkor még az érték elég kicsi ahhoz, hogy ne legyen érzékelhető a lebegés, de megakadályozza, hogy átessenek az objektumok.

3.3 Valósághű textúra előállítása

Az élethű képek előállításához még nagyon fontos, hogy megfelelően modellezzük a pelletek anyagtulajdonságait, valamint a fényviszonyokat. A mintaképet megvizsgálva a pelletek felületén az látszik, hogy a rájuk eső fényforrások sugarait szétszórják. Ez okozza azt, hogy matt hatást keltenek és a nem egy nagy egybefüggő fényvisszaverődés látunk, hanem sok kicsi világos pontot. Ezért egy olyan anyagot kell modellezni, ami nem egyenletesen veri vissza a fényt. Másik szembetűnő dolog az, hogy a kamera szenzorhoz közelebb lévő pelletek sötétebbek ezért, azt is modelleznünk kell a megoldásunkban, hogy a felsőbb pelletek sötétebbek legyenek, mint a többi.

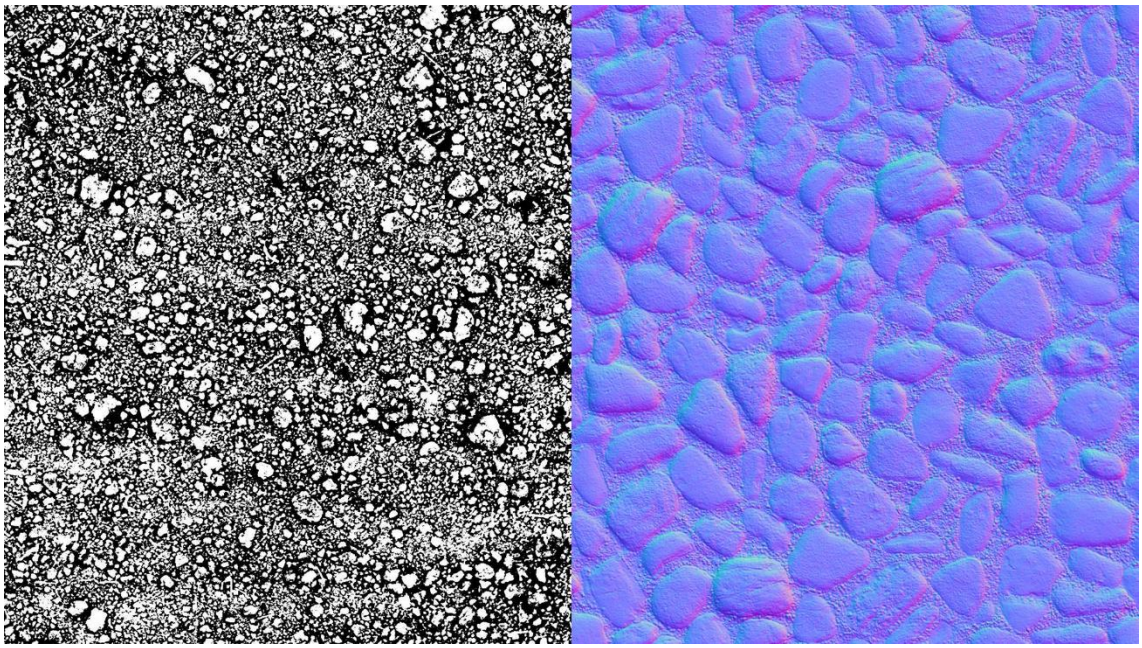
Kezdve a pelletek felületével, ahhoz, hogy mindegyik objektumra alkalmazható legyen, az anyag procedurális megvalósítását kell követni. Ennek az alapja egy zaj textúra, aminek a generálására a Blender beépítetten nyújt lehetőséget az anyag készítő rendszerében. A pelletek felületének megtörésére jó módszer, ha az anyag normál vektor

textúra bemenetének használunk egy 3D-s zajtextúrát. Ez azt eredményezi, ha egy nagyon zajos textúrát kötünk bele, hogy a felületen lévő normálvektorok, amik alapján a felület árnyalása számítódik össze vissza fognak állni, ezzel elérve azt, hogy megtörjük a felületet. Ez egy viszonylag költség hatékony módszer a 3D modellezésben részletek hozzáadására extra vertexek hozzáadása nélkül.



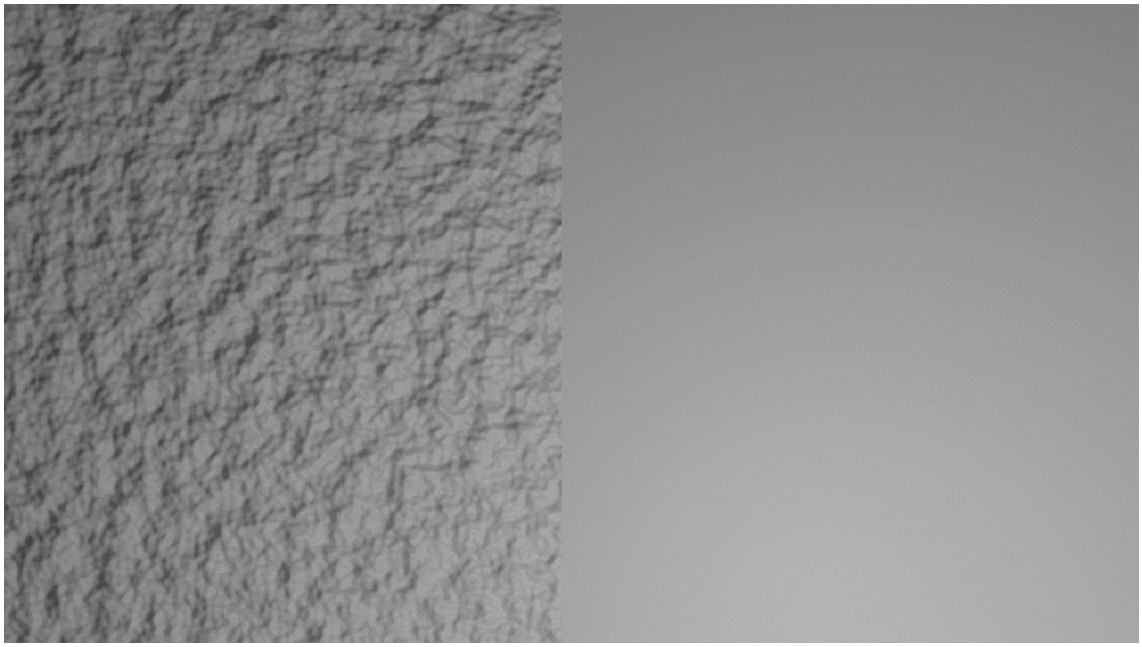
11. ábra - A felület zaj textúra normál vektorral és anélkül

Egy másik mód, ami hasonló, hogy a zaj textúrát angol szaknevén Bump textúrának használjuk, ami szintén a normálvektorok irányát befolyásolja. A két módszert egymás szinonimájaként is emlegetik, de van egy lényeges különbség köztük. A Bump textúrák intenzitásokat tárolnak, ami a relatív magasságot reprezentálják a kamera nézetétől így ezek tipikusan szürkeárnyalatos képek. A normál textúrák pedig irányokat tárolnak az RGB képek három szín csatornájával, Blenderben megfeleltetve a piros, zöld és kék csatornákat rendre az x, y és z tengelyeknek.



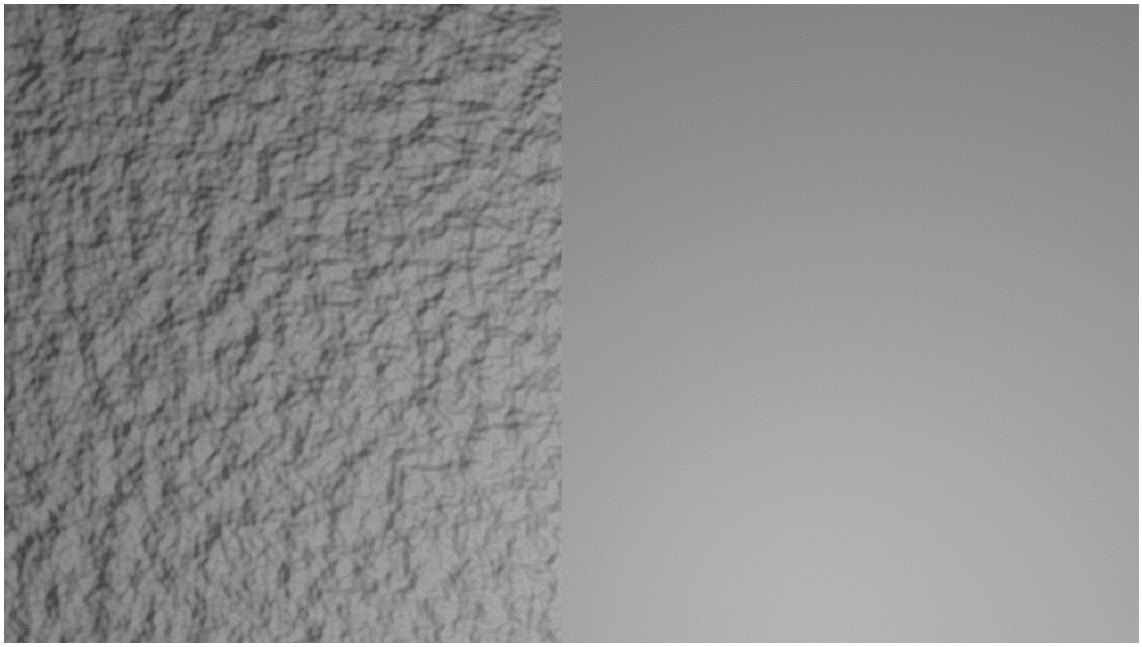
12. ábra - Bump és normál textúrák

Ha a Blender által kínált zaj textúra kimenetet átalakítjuk normál textúra bemenetté ahhoz a zaj RGB kimentét kell használnunk, de ezzel elvesztünk rengeteg részeletet a textúrából, azonban amikor Bump textúra bemenetté alakítjuk akkor elég a zaj intenzitás kimenetét használnunk és így megmarad a zaj textúra részletessége, ami a 13. ábra és a 12. ábra összehasonlításában is látszik.



13. ábra – A felület Bump zaj textúrával és anélkül

A felület megtörésére még alkalmas módszer lenne a zajt angol szaknevén Displacement textúrának használni, ami azt a hatást éri el, hogy a vertexeket a textúra alapján elmozgatja, ezzel valós geometriát hozva létre. Ez viszont azzal a költséggel jár, hogy sok vertexet igényel. Előnye viszont az előző két módszerhez képest, hogy mivel valós geometriát hoz létre, ezért a felületen lévő domborulatok képesek árnyékot vetni más objektumokra., míg mivel a normál és a Bump textúráknál csak az árnyalás változik, ez nem lehetséges, valamint lapos betekintési szögben az egész hatás szertefoszlik.

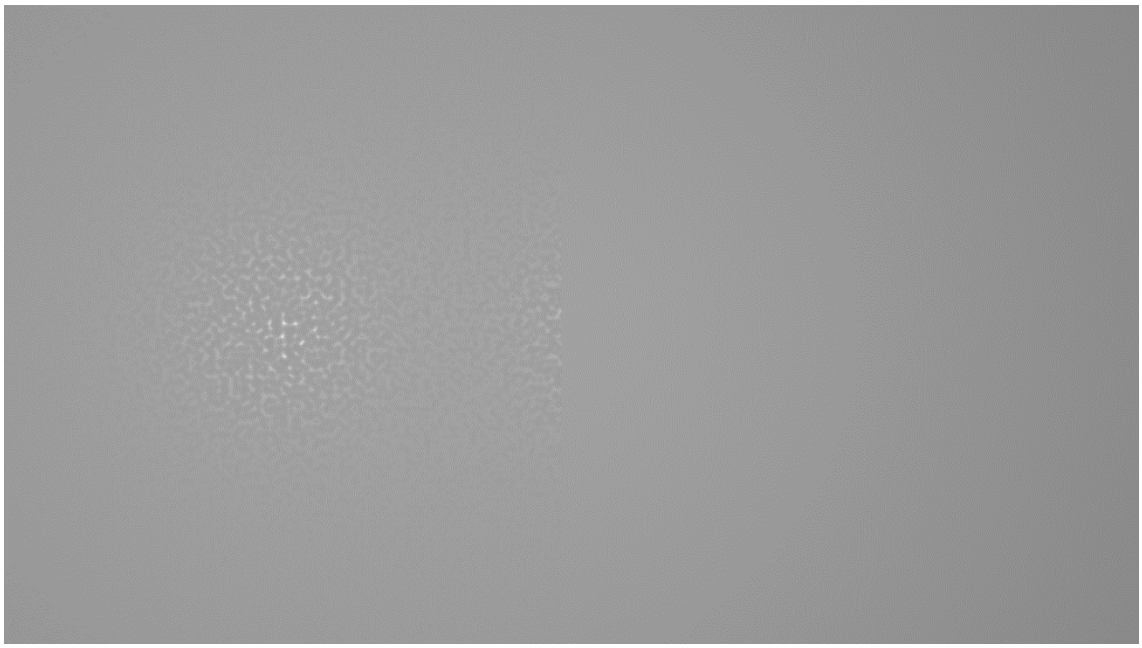


14. ábra- A felület Displacement zaj textúrával és anélkül

A képek alapján az látszik, hogy a zaj textúrát Bump textúraként lenne érdemes használni, de mi a végső anyagban végül mégis normál textúraként alkalmaztuk, mert azzal sikerült a mintaképen lévő apró fényvisszaverődéseket lemodellezni, míg Bump textúraként az apró részeletekkel túl erős volt a hatása és az egész felület matt lett.

Ahhoz viszont, hogy a képen látható fényes pontokat kapjunk, az objektumon még módosítanunk kell a felület fényszórási tulajdonságát, a durvaságát. A valóságban minél durvább egy felület, minél több kis barázda található rajta, annál jobban szétszórja a visszaverődő fénysugarakat, ezzel matt hatást érve el. A mi esetünkben azt szeretnénk látni, hogy kis pontokban verődik vissza a fény. Ehhez ugyanúgy használhatunk egy zaj textúrát, aminek az intenzitás kimenetét most egy színskálára kötjük, ami a 0 és 1 között értékekhez színeket rendel, a mi esetünkben fekete színt a 0 értékhez és fehér színt az 1-

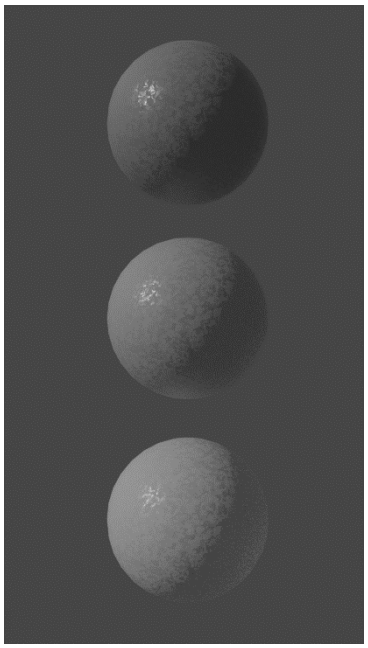
hez. A színskála kimenetét az anyag durvaságát befolyásoló bemenetére kötve a textúrán minél világosabb egy pont, annál jobban veri vissza a fényt, és a sötét részek kevésbé verik vissza, így egy olyan felület keletkezik, aminek a fényvisszaverő képessége nem konstans az egész felületre.



15. ábra A zaj textúra fényvisszaverődéssel és anélkül

A következő feladat annak a szimulálása, hogy a kamera szenzorhoz közeli pelletek sötétebb színűek. Erre szerencsére egy egyszerű módszer, ha vesszük a kamerától való távolságot, vagy pedig a pelletek z magasságát és ezeket az értékeket csatoljuk előzőleg látott módon egy színskálához, úgy, hogy a távolságot vagy a z értéket leosztjuk, hogy 0 és 1 közötti számot kapjunk. A mi megoldásunkban a z értéket vettük az objektumoknak, azért, mert a kamerától való távolsággal erős függőség keletkezik az anyag kinézete és a kamera pozíciók között, így különböző kamera állásokhoz módosítani kellene az anyag tulajdonságait. A színskála egy pár próba esettel beállítható a kívánt kimenethez, ahol leejtünk gömböket egymásra. Így a z érték alapján minden kamerában a felül lévő pelletek lesznek a sötétebb árnyalatúak.

Ha ezt a három effektet kombináljuk egy anyagban akkor a valós képekhez hasonló mintát kapunk.



16. ábra – A pellet anyag

3.4 Render motor és fények beállítása

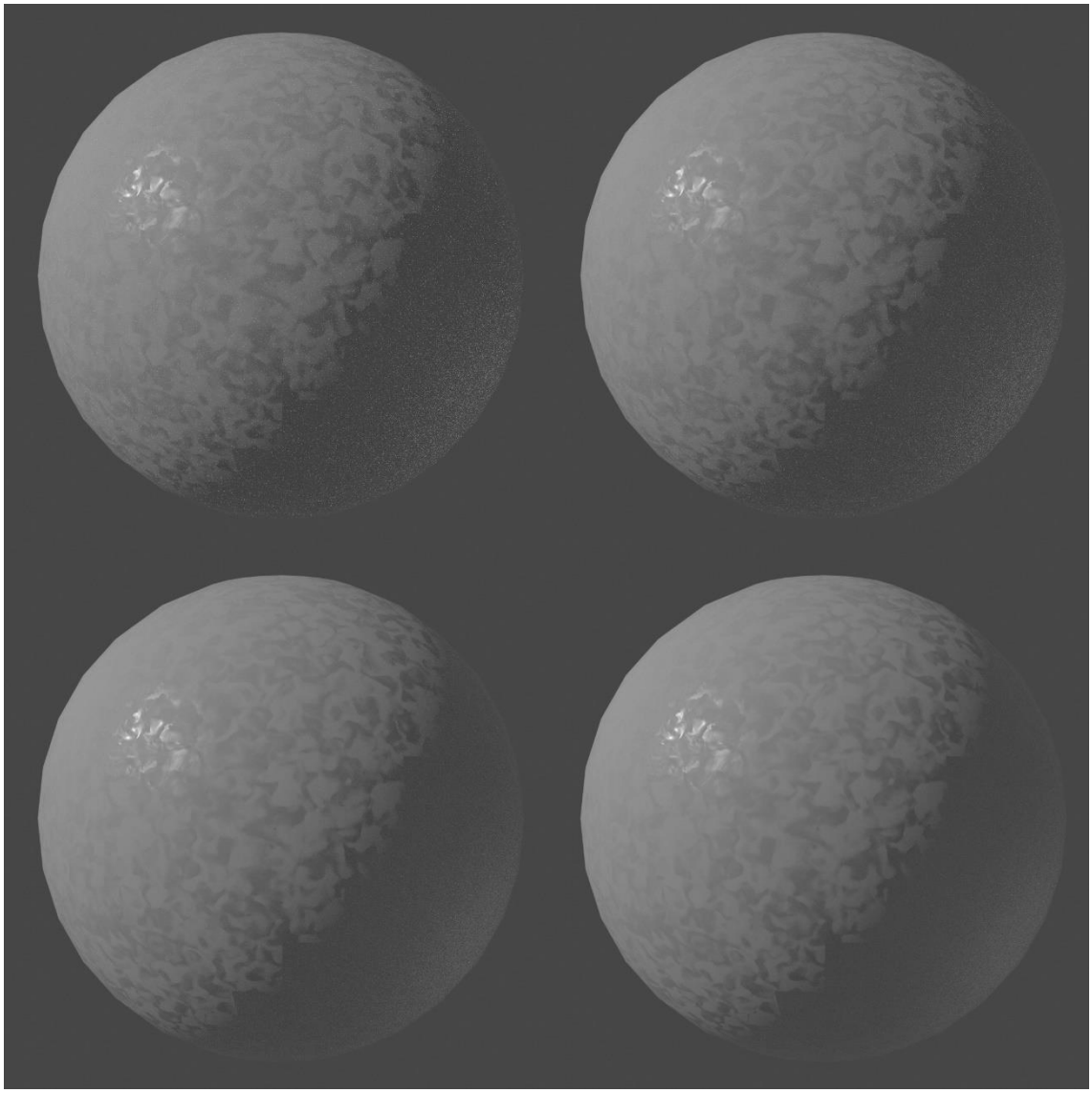
Az utolsó összetevő a képek ellőállításához a render beállítások és a fények. A mintaképen látszik, hogy nem egy irányból éri fényforrás, mivel egy laboratóriumi környezetben készült. Emiatt a modellezett szcenárióban is több fényre lesz szükségünk. Blenderben a nap típusú fényforrásnak nem számít a pozíciója az objektumokhoz képest, csak az iránya, így három nap lámpát használunk.

Abból a megfontolásból, hogy ne legyenek kivehetők az árnyékokból a különböző fényforrások, ezért kettő ezek közül nem idéz elő árnyékokat, csak fényt és visszaverődést biztosít. A mintaképen nem láthatók éles árnyékok, ezért úgy kell az árnyékot adó fény forrásunkat is beállítani, hogy lágy árnyékok keletkezzenek. Ez a nap lámpák esetében a szög tulajdonság állításával érhető el, ami a nap szögátmérőjét reprezentálja a földről nézve. Minél magasabb értéket adunk meg, annál puhább árnyékokat nyerünk.

A 3D-s adatokból történő 2D-s képek előállítását nevezzük render folyamatnak. Erre a folyamatra készültek a render motorok. Blenderben a 2.80-as verzió óta két fő render motor található, az Eevee [20], ami egy valós idejű render motor és a Cycles [21], ami pedig egy útkövető render motor. Mi a munkánkban a Cycles render motort alkalmaztuk mivel a render idő nem volt a legfontosabb szempont, mert a tanító adatokat nem szükséges folyamatosan újra előállítani. Továbbá ez a motor a fotórealisztikus képek

előállítására alkalmasabb, mint az Eevee, ami a sebességért cserébe bizonyos részleteket feláldoz, mint például a realisztikus fény szimulációt.

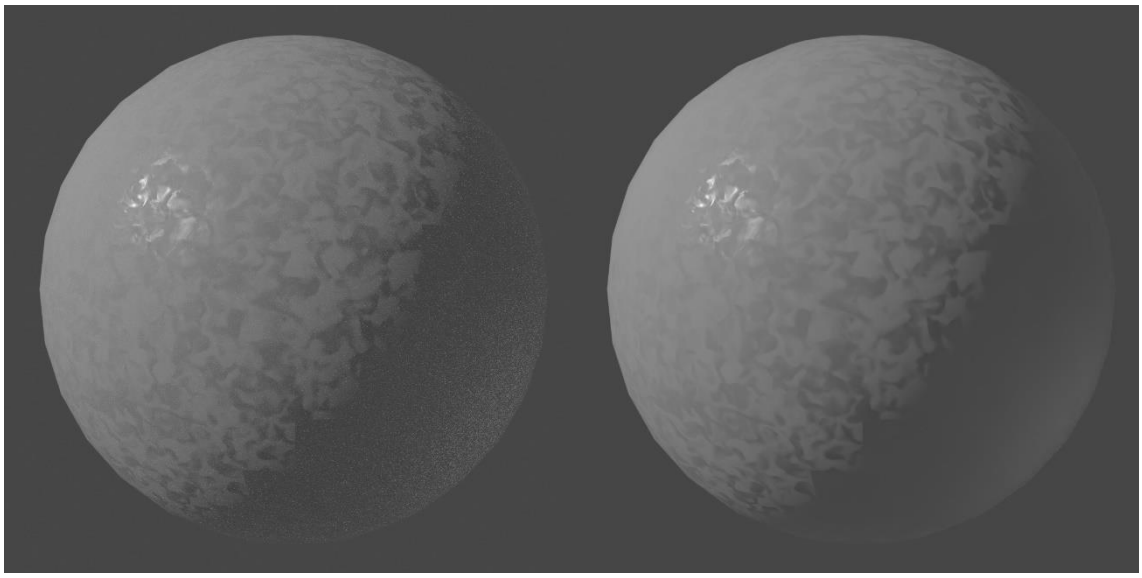
Cycles-ben a képek minőségét a felbontásukon kívül a mintavételezések száma befolyásolja nagymértékben, minél nagyobb a mintavételezési érték annál kevésbe lesz zajos a képünk.



17. ábra – Render kép 16 és 32 (felső sor), és 128 és 256-os (alsó sor) mintavételezési értékkel

A mintaképünket megvizsgálva látszik, hogy a valós képek minősége sem a legjobb, és bizonyos mértékben furcsa anomáliák is előfordulnak, így a mintavételezési szám alacsonyan tartása azt eredményezi, hogy a generált objektum jobban megközelíti az eredeti képet és még gyorsabb render időt is kapunk. A zajok eltüntetésére a

mintavételezési érték növelésén kívül még zajcsökkentő utóműveletekkel nyílik lehetőség, azonban a Blenderbe épített Intel zajcsökkentő modul hatása túl erős volt. Mivel a valós képeknek sincs jó minősége és nem kristály tiszták általában a pellet határok, így nem használtuk ezt a lehetőséget.



18. ábra - 32-es mintavételezés zajcsökkentés nélkül és azzal

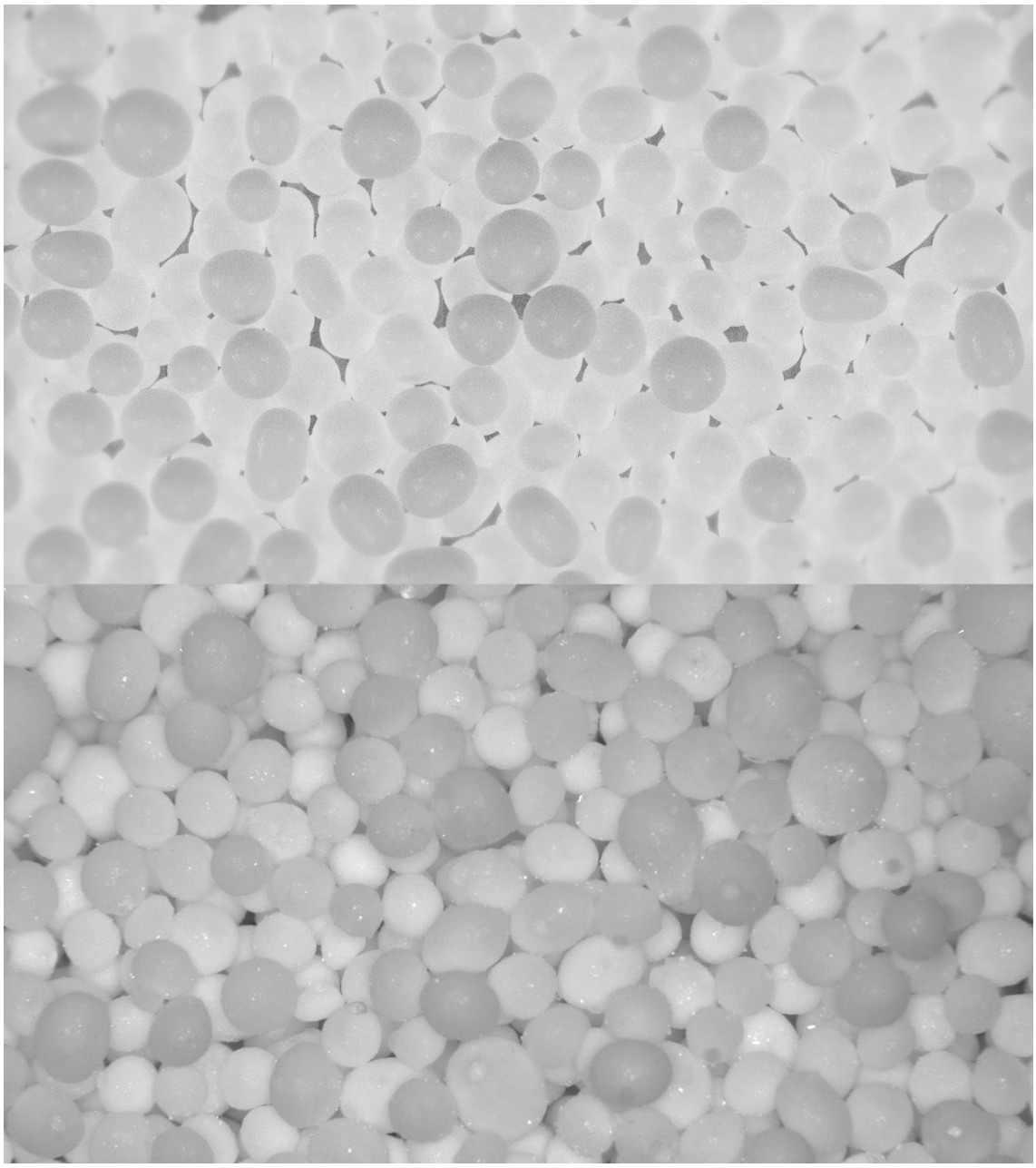
A megfelelő képhez a fény visszaverődések maximum számával kell még foglalkoznunk, a Blender alapértelmezett beállításai megfelelő képet eredményeznek, igaz befolyásolják a render sebességet, de az továbbra sem az elsődleges szempont. A maximum értékek feljebb vételével javul a kép, de nem szignifikáns mértékben, ezért nem éri meg módosítani. A generáló rendszer összeállításánál is az alapértelmezett értékekhez közelít használtunk.

3.5 Értékelés, összehasonlítás

Az elkészített szintetikus képet és a valós képet összehasonlítva természetesen látható a különbség, azonban a 2. fejezetben hivatkozott munkák alapján ahhoz, hogy sikeresen tanítsunk fel neurális hálókat objektum detektálási feladatokra, nem feltétlenül szükséges a valós képek tulajdonságainak a tökéletes rekreációja, így jó esély van ara, hogy képesek leszünk a szintetikus képekkel feltanítani egy hálót a feladatra.

Az előző részen leírt anyag igaz a mi feladatunkhoz specifikus, de nagy általánosságban el lehet mondani, hogy a procedurális anyagok nagyrésze egy zaj textúra rendszeren alapszik és különböző zaj típusok kombinálásával végtelen számú forma alakítható ki. Ezeket felhasználva a felül lévő módszerekhez hasonlóan elkészthető a számunkra szükséges kinézet. Egy objektumon akár több anyagot is fel lehet használni.

Komplexebb anyag készítési megoldás lehetne GAN (*generative adversarial networks*) használata, ahol két neurális hálózat tanul párhuzamosan (gerenátor és diszkriminátor). A generátor haló az anyag előállításával foglalkozik, az előállítás paramétereit módosítja, a diszkriminátor feladata pedig az, hogy megkülönböztesse a valós és a szintetikus anyagot. Ilyen hálózattal lehetséges lehetne képek alapján generálni akár anyagokat, ez a téma egy jövőbeli kutatás alapja lehet.



19. ábra - Szintetikus és valós kép

4. fejezet

Adathalmaz generálása

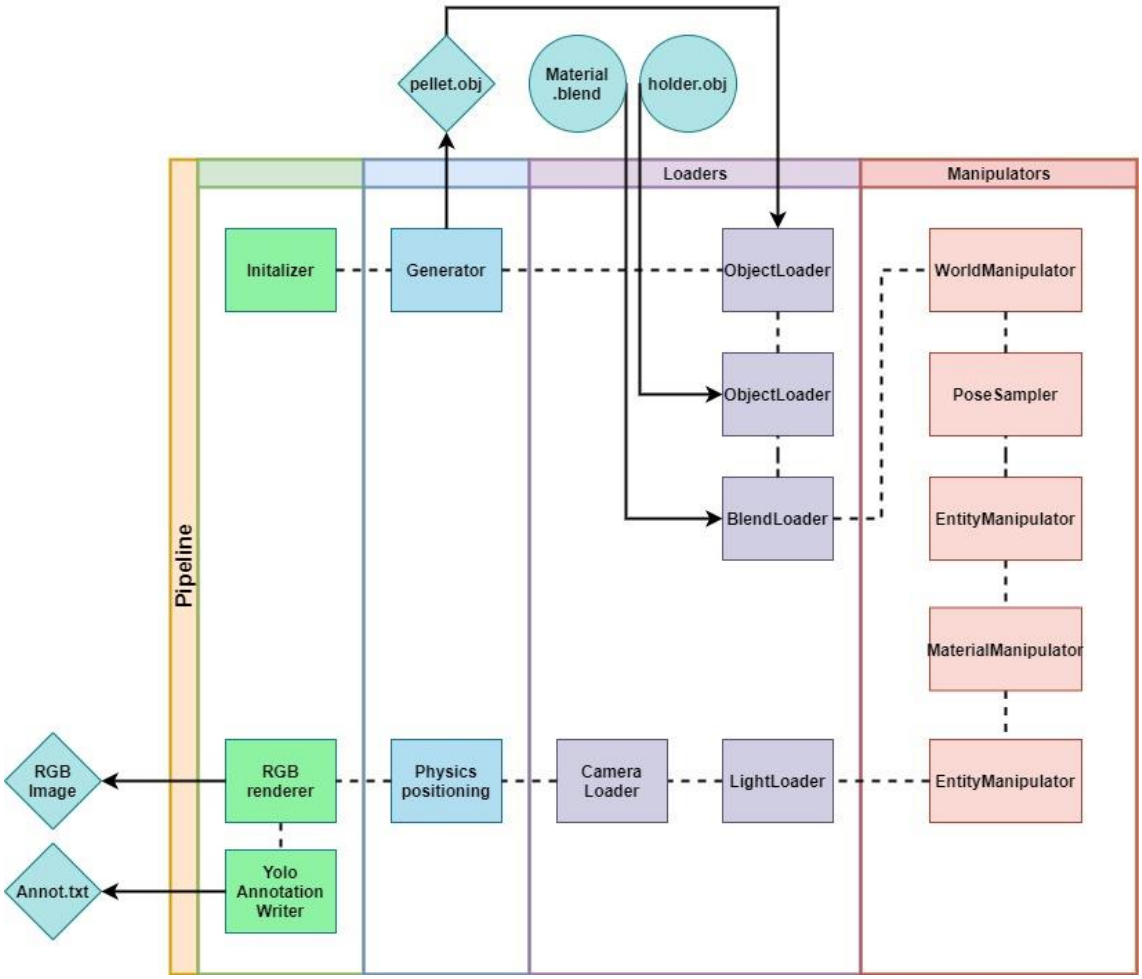
4.1 A generáló rendszer áttekintése

Ahogy már írtuk a BlenderProc keretrendszer részletezésénél, a szintetikus képeket előállító csővezeték rendszerek modulokból épülnek fel, amik elkülöníthető feladatokat látnak el, így ilyen részegységekben kell gondolkodni a rendszer összeállításához. Az első mindegyik csővezeték rendszerben a BlenderProc inicializáló modulja. Ez a modul felel az alapvető beállításokért, mint például a kimeneti mappák létrehozása és beállítása, a render folyamat során használt számítási eszköz detektálása (CPU, GPU) és beállítása. Továbbá ennek a modulnak lehet megadni globális konfigurációs értékeket, amik minden modul számára elérhetőek.

A BlenderProc rendszer futtatása egy futtató szkript segítségével történik, melynek az első paramétere a használandó konfigurációs fájl. További paraméterek igény szerint adhatók át neki, mely elemeket a yaml kiterjesztésű konfigurációs fájlokban lehet modulokhoz átvezetni. Az inicializáló modul után a pellet modellek generálására szolgáló modulnak kell következni, hogy ezt követően már minden szükséges adat rendelkezésre álljon a rendszernek.

Így a pellet generátor után olyan modulok következhetnek, amik betöltik az objektum modelleket, valamint az ezekhez tartozó anyagokat is. Ez után következnek azok a modulok, amik már a betöltött objektumokon dolgoznak, beállítják a kezdő pozícióikat, méreteiket, valamint rájuk helyezik a betöltött anyagokat. Az objektumok elrendezése után a következő feladat a fények és kamera pozíciók beállítása lenne, de ez akár meg is előzheti az objektum manipulációkat hiszen nem állnak függőségi viszonyban.

Miután a jelenet összes alkotó eleme a helyére került elvégezhetjük a pelletekre a fizikai szimulációt, szimuláció után pedig jöhet a kimenetek generálása, az RGB képek előállítása és a képekhez tartozó YOLO annotációs fájlok.



20. ábra A generáló rendszer felépítése

4.2 Pellet generátor modul

Mivel a BlenderProc előzőleg nem támogatott a miénkhöz hasonló feladatot ezért generátor modul még nem található benne, így ezt a modult nekünk kellett hozzá írni a rendszerhez. A modul két féle generálási módot támogat. Az elsőben a generált objektumokat fájlba elmenti így ezek később visszatölthetők. A másodikban a generált objektumok a rendszerben maradnak, nem kerülnek kimentésre és az objektumokra egy egyedi tulajdonság kerül, amivel a későbbi modulok azonosítani tudják őket.

A konfigurációs fájlban lehet megadni, hogy melyik generálási módot használjuk. A modul futási része a Pellet objektumok létrehozása fejezetben leírt manipulátorral történik. A konfigurációs fájlban megadható, hogy a manipulátor által végrehajtott műveletek mekkora eséllyel forduljanak elő, így változatosságot adva a modelleknek azzal, hogy nem mindegyik deformáció megy végbe az összes objektumra.

További beállítási lehetőség a deformációs faktor maximum értékének a megadása a műveletekhez, valamint az alap modellhez egy skálázási érték, amivel a kiinduló objektum méretét tudjuk állítani. Az objektum generálás lefut a konfigurációban megadott darabszámra, majd a generálás után az objektumokat kimenti fájlba a beállítástól függően.

A mi rendszerünk beállítása fájlba írja ki a generált objektumokat majd ezt tölti be a későbbi futások során.

4.3 Objektum és anyag betöltő modulok

Mivel az erőforrások betöltése általános feladat a keretrendszerben, ezért található hozzá implementáció. A konfigurációban meg kell adni a betöltendő fájl elérési útját, továbbá lehetőség van megadni egyedi tulajdonságokat és értékeiket, amiket a betöltődő objektumokhoz csatolunk. A mi rendszerünkben a fizikai szimulációt végrehajtó modulhoz szükséges tulajdonságot vettük fel, ami megadja, hogy melyik objektum játszik passzív, és aktív szerepet a szimulációban. További tulajdonság, amit még megadtunk a pellet objektumokhoz, a YOLO annotáció elkészítéséhez szükséges objektum osztály indexe vagy más néven címkéje.

Az anyagot a Blender egyedi projekt formátumából töltjük be, azért, mert az anyagokra nincs célspecifikus kiterjesztési formátum. Mivel minden 3D render motor más anyag leírási rendszerrel dolgozik, így még Blenderen belül is a két render motorban elkészített anyagok sem feleltethetők meg egymásnak teljesen. A konfigurációban az elérési úton kívül, mivel a projekt fájlok lényegében adattárolók, ezért azt az elérést is meg kell adni, hogy milyen erőforrásokat szeretnénk importálni a fájlból.

4.4 Világ objektum, és anyag manipulátor modulok

Az objektumok és a világ manipulációjára is vannak már beépített elemek a keretrendszerben így ezeket feltudjuk használni. A világ manipulátor modul segítségével a világ színét, valamint ennek a fényerősségét lehet állítani. A beállított színt látjuk amikor egy renderelt képnél nincs objektum a kamera előtt, valamint a fényerőség a

konstans globális megvilágítás erejét állítja, ami szintén a beállított színt használja. Ezt felhasználva világosabb kiindulást adunk a jeleneteknek.

Az objektum manipulátoroknál egy beágyazott szelekciós elemmel lehetséges kiválasztani, hogy a modulban megadott konfiguráció mely objektumokra érvényesüljön. A szelekció feltételének akármilyen beépített, vagy egyedi tulajdonságot és ezek tetszőleges kombinációját meg lehet adni.

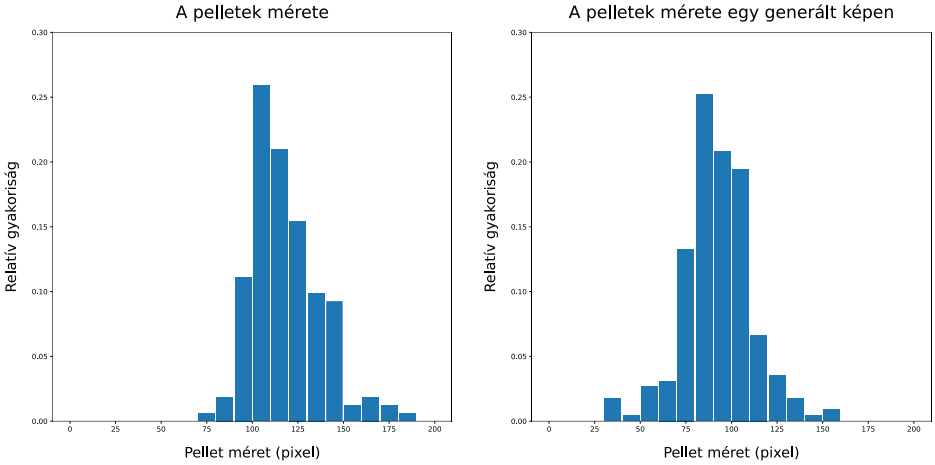
A manipulátor modult felhasználva felskálázzuk a tartó objektum méretét, amit direkt úgy készíttettünk el, hogy betöltéskor 1 egységnyi mérete legyen az összes tengelyen, így amikor érvényesítünk rá egy skálázási vektort, az objektum dimenziói megegyeznek a vektor értékeivel. Ezen kívül még állítható, hogy a tartó objektum megjelenjen-e render képeken, amivel azt lehet elérni, hogy csak a fizikai szimulációt segíti, de a fényhatásokban nem játszik szerepet, valamint a világ színe lesz a képek háttér színe, ezt az értéket állítva többféle kép variáció jöhet létre, ezzel is csökkentve a túltanítás esélyét.

További modulokkal ráhelyezzük a betöltött anyagokat a pelletekre és a tartóra. A tartóhoz készítettünk egy különálló anyagot, ami egy hullám textúrát tartalmaz és ez az anyag normál vektor bemenetére van kötve. Ezzel egy rácsozott megjelenést értünk el, ami néhány valós pelletekről készült képen látszik. Ennek az anyagnak egy anyag manipulálásra alkalmas modul segítségével állítható a hullám textúra skálázási értéke, valamint a normál vektorokra gyakorolt hatás faktora, viszont az érték viszont túl magasra állítása anomáliákat okozhat.

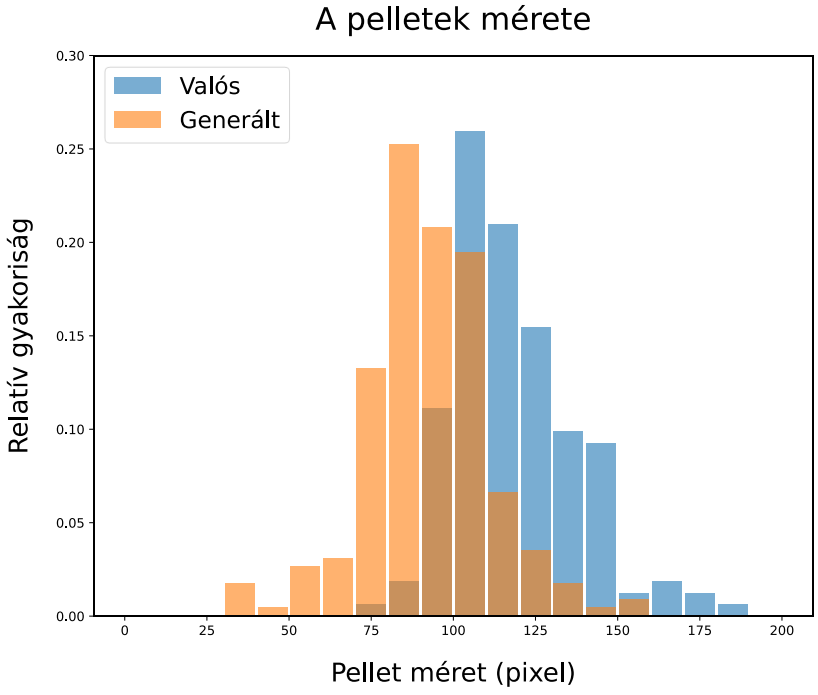
A pelletek kezdeti pozícióját egy pozíció mintavételezési modul segítségével adjuk meg, ami a kiválasztott objektumokra mintavételező elemek segítségével beállítja a lokációjukat és rotációjukat. A mi álltalunk használt mintavételezők egy 3D vektort adnak vissza a minimum és maximumként megadott vektorok között egyenletes eloszlással. A modul minden mintavételezés után megnézi, hogy a kiválasztott objektumok között nincs-e átfedés a már elhelyezettek között, ez viszont azzal jár, hogy ha kis helyett adunk meg és sok objektumunk van, a véletlen mintavételezés miatt nagy mértékben lassítják a folyamatot az újra próbálkozások. Mivel a lokáció függ attól is, hogy milyen méretet adtunk meg a tárolónak, valamint a pelletek száma és mérete is közrejátszik ezeknek az értékeknek a meghatározásában így ezek egymással korrelációban állnak.

Végül pedig a pelleteket skálázzuk egy általunk írt mintavételező segítségével, ami hasonlóan működik, mint az előzőekben leírt 3D vektor minta vételező, azzal a

különbséggel, hogy a visszaadott 3D-s vektor értékei megegyeznek és az egyenletes eloszláson kívül, normál eloszlással is működhet a véletlen generálás. Ez a mi megoldásunkban azért játszik nagy szerepet mert a valós képek alapján, a pelletek méreteinek az eloszlása normál eloszlást követ, ezt manuálisan annotált képek segítségével tudtuk meghatározni.



21. ábra - Pellet méret eloszlás pixelben, baloldalt valós képeken, jobboldalt generált képeken



22. ábra - Pellet méretek eloszlásainak összehasonlítása a generált és valós képeken

Az előzőleg említett mintavételezők használata bizonyos tulajdonságokra azért is jobb, mint beégetett értékek használata mert így automatikusan manuális konfiguráció változattás nélkül minden lefutás különböző lesz.

4.5 Fényforrás betöltő modul

A Render motor és fények fejezetben írt módon három fényforrást helyeztünk el a jelenetekben a megvilágosításhoz. A betöltő modulon kisebb bővítést kellet végezni annak érdekében, hogy állítani tudjuk, hogy melyik fényforrás hatására keletkezzenek árnyékok, valamint, hogy a szögátmérő tulajdonságnak lehessen értéket adni. A lámpák erősség és rotációs tulajdonságaira az objektumok tulajdonságaihoz hasonlóan mintavételezőket kötöttünk, hogy nagyobb variációt kapjunk a képek kinézetére. A rotációra alkalmazott mintavételező annyiban van korlátozva, hogy csak az y tengely mentén történik mintavételezés, a fények a z tengely mentén 0, 90, és 180 fokkal vannak elforgatva, így a pelletek előről, oldalról, és hatúról vannak megvilágítva. Az y mentén történő forgatás pedig a fénysugarak becsapódási szögét állítja.

4.6 Kamera pozíciókat betöltő modul

A BlenderProc lehetőséget ad arra, hogy a kamera pozícióinkat egy egyedi fájlból töltsük be, aminek a szerkezetét a konfigurációs fájlban definiáljuk, azonban lehetséges csak a konfigurációban megadni a kamera pozíciókat. Mivel a jelenlegi rendszerünk kis számú kamera pozícióval dolgozik, ez utóbbi nekünk megfelelő, mert így még nem befolyásolja a konfiguráció olvashatóságát. A folyamat lefutásának költsége miatt minél több képet szeretnénk készíteni futásonként, ez a mi beállításainkkal 6 kép, 3 részre felosztva szemből és oldalról a tárolón belüli területet, ahol a pelletek találhatók.

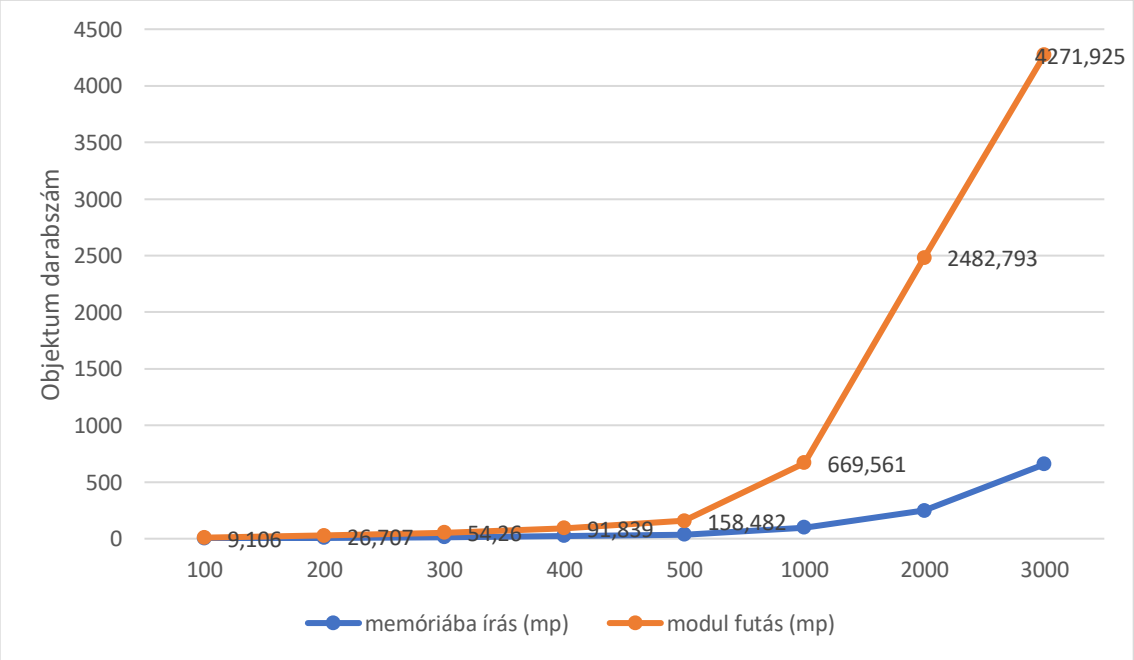
4.7 Fizikai szimuláció modul

A keretrendszerben már meglévő fizikai szimulációs modul konfigurációjában meg kell adni azt a minimum és maximum időt másodpercben amire a szimulációt szeretnénk futtatni. A másodperceket a modul átalakítja a Blender által használt képkockaszámra,

amivel a szimuláció dolgozik. Ha a minimum meghatározott idő eltelt, akkor megvizsgálja, hogy az objektumok mozgásban vannak-e és ezt az ellenőrzés másodpercenként megismétli, amennyiben megálltak leállítja a szimulációt még a maximum megadott idő előtt.

Viszont így is érdemes a maximum értékre megadni egy olyan időt amire az objektumok majdnem teljesen nyugodt állapotban vannak, mert sok objektum esetén a fizikai szimuláció instabilitása miatt úgynevezett rezgés fordulhat elő az objektumok között, valamint, ha a tároló objektumon kívülre esik egy objektum akkor az tovább zuhan így soha nem kerül nyugalmi állapotba. A Fizikai szimulációval foglalkozó fejezetben leírtak miatt még be kell állítani az objektumokhoz plusz ütközési határt, valamint, a felhasznált ütközési testet, ehhez bővíteni kellett a modul meglévő funkcionalitását, hogy a rendelkezésre álló test típusokat ráhelyezhessük egy szelekció által visszaadott objektum csoportra.

Mivel a generálást úgy lehet a legjobban gyorsítani, ha egy futásra minél több képet készítünk, ezért megvizsgáltuk, hogy az a módszer eredményesebb-e, amikor kevés objektummal dolgozunk, vagy pedig egyszerre sokkal készítünk egy nagyobb jelenetet, ahol több képet tudunk készíteni.



23. ábra - Fizikai szimuláció futási ideje különböző objektum mennyiségekre

Ahogy az ábra mutatja, az objektumok darabszámának növekedésével a fizikai szimulációs adatok memóriába történő írásának az időtartalma nem növekszik drasztikusan a darabszám százalékos növekedéséhez képest, azonban a modul tényleges futási ideje polinomiális jelleggel növekszik a darabszámhoz képest, így érdemes kevesebb objektummal dolgozni. Továbbá a teszt során érezhető volt az objektumok pozicionálására használt modul implementációs hátránya is, amikor az átfedés ellenőrzéskor végig megy a már elhelyezett objektumokon, így ennek a modulnak is polinomiális módon változik az ideje az objektumok számához képest.

További hátrányok, amik sok objektum esetén előjöhetnek, hogy növelik a szimuláció instabilitását, ami anomáliákhoz vezethet, valamint a szoftver nagyobb eséllyel áll le és ezzel elveszítve az addig elvégzett műveletek eredményét. Ezek miatt döntöttünk az mellett, hogy kevesebb objektummal dolgozunk és inkább többször futtatjuk a generátor rendszert egy nagy generálás helyett.

4.8 RGB render modul

A kimeneti képek előállításához tartozó modulban lehet beállítani a Render motor és fények fejezetben leírt render folyamathoz tartozó tulajdonságokat. Azért, hogy a képek minősége változó legyen a render mintavételezési számára egy mintavételező elemet alkalmaztunk, ami két megadott érték közötti számot generál. A fényvisszaverődések és a zajcsökkentő modul állításához bővítettük a meglévő implementációt, ez által a konfigurációban kikapcsoltuk az utólagos zajcsökkentést, valamint beállítottuk a számunkra megfelelő visszaverődési értékeket.

Továbbá, a valós képek szélein álltalában megfigyelhető egy bizonyos mértékű elmosódás, ennek a modellezésére a render modulhoz készítettünk egy egyedi függvényt, ami a Blender kompozíciós lehetőségeit kihasználva egy maszkkal ellátott homályosítás effektet helyez a kimenti képre. A konfigurációba kihelyeztük a homályosítás effekt erősségének állítását, valamint a maszk pozícióját, méretét, illetve a maszkra alkalmazott homályosítás erőségét is, hogy a maszkolás határai mennyire legyenek elmosva. A változatosság növelésére a felépített rendszerünkben a homályosítás erősségére, valamint a maszk méretére itt is mintavételezést alkalmaztunk.

4.9 YOLO annotációs modul

A BlenderProc-ban a YOLO rendszer által használt annotáció előállítására még nem volt implementáció, így az ehhez szükséges modult mi építettük fel. A modul a konfigurációban egy objektum szelektort vár, ami megadja azokat az objektumokat, amiket az annotációban fel szeretnénk tüntetni.

4.9.1 Transzformáció kamera koordinátákba

A befoglaló téglalapok meghatározásához át kell számolni az objektumok vertexeinek a lokális koordinátáit a kamera kép 2D-s koordináta rendszerébe.

Ez a folyamat az alábbi lépések mentén történik, a transzformációs folyamatokról részletesebben értekezik a Háromdimenziós grafika, animáció és játékfejlesztés könyv geometriai transzformációk fejezete [22].

1) Először vesszük a Blender által nyújtott függőségi gráf segítségével az objektum végső állapotát, ezzel megkapjuk a vertexeket az objektum lokális koordináta rendszerében.
2) Ezt követően az objektumot transzformáljuk egy mátrix segítségével, ami a lokális koordinátákat a világ koordináta rendszerbe alakítja át, ezt hívjuk transzformációs mátrixnak.
3) A világ koordinátákat transzformálva a kamera objektum saját transzformációs mátrixának az invertáltjával (ami igazából így a globális koordináta rendszerből alakítja át a pontokat lokális koordináta rendszerbe) a kamera objektum lokális koordináta rendszerében megkapjuk az objektum pontjainkat.
4) Miután ezzel megvagyunk, vesszük a képkocka négy határoló pontját és alkalmazzuk rájuk a perspektivikus torzítást.
5) Majd vesszük a felső két pontot, meghatározzuk belőlük a képkockának a minimum és maximum x koordináta értékeit, a bal szélső két pontból pedig az y koordináta szélsőértékeket.
6) Ezek segítségével kiszámoljuk az objektum pontjainak x és y koordinátáit a képkerethez méretéhez relatívan.

7) Ezután vesszük az x és y koordináták minimumát és maximumát lekorlátozva az értéküket 0 és 1 közé.
8) Ezek által kiszámoljuk a befoglaló téglalap bal felső sarkát, valamint a szélességét és magasságát.
9) Végül a kapott értékeket felszorozzuk a kép felbontásával így pixelekben megkapva a befoglaló téglalap tulajdonságait.

4.9.2 A véglegesített annotáció kiírása

A transzformációból visszakapott értékeket tovább szűrjük, azokat az objektumok nem kerülnek kiírásra, ahol a befoglaló téglalap kilógna a kép szélén. Ahol teljesül, hogy a befoglaló téglalap nincs a kép szélén, ott áttranszformáljuk a megkapott értékeket a YOLO által várt képhez relatív középponti x, y koordinátákra, magasságra és széleségre, ez azt jelenti, hogy itt vissza kell osztani a kép felbontásával. Azért más formában adjuk vissza a téglalapot a kiszámításkor, hogy máshol is felhasználható legyen a téglalapot kiszámoló segéd metódus a későbbiekben. Végezetül a fájlba beleírjuk az előállított YOLO-s adatokat és az objektumból kiolvassuk az egyedi tulajdonságként hozzáadott osztály indexét, amit a betöltéskor adtunk hozzá.

5. fejezet

A neurális hálózat tanítása

A YOLO bemenete egy egyenlő magasságú és szélességű kép, melynek mind a magassága mind a szélessége 32-nek a többszöröse. A tanítás során azonban meghatározott iterációnként a keretrendszer átméretezi a bemenetet, ezzel elősegítve, hogy általánosabb tulajdonságokra tanuljon rá a háló és ne legyen méretfüggő.

5.1 Hiperparaméterek optimalizálása

A hiperparaméterek optimalizálása során a legfőbb célunk az volt, hogy a lehető legjobban kihasználjuk a rendelkezésünkre álló hardvert. Adathalmazunk mindössze 600 generált képből állt, ami lényegesen kevesebb, mint a javasolt 2000 kép/osztály azonban, mivel elég egyszerű alakzatokról van szó így inkább a tanításra szántunk több időt. Továbbá a tanítást Google Colaboratory segítségével végeztük, ahol a felhasználható merevlemez is elég limitált.

Az általunk használt AlexeyAB nevű Github felhasználó által módosított YOLO keretrendszer 1000 iterációnként menti a súlyfüggvényeket. Egy iteráció a *batch* paraméterben meghatározott képet tartalmaz, melyet *subdivision* paraméter által meghatározott részre oszt fel a rendszer, tölt be a memóriába és dolgoz fel. Ezeken kívül még a hálón változtatni kellett az output-ot, hiszen a YOLOv3 80 osztályra van feltanítva, nekünk azonban csak 1 pellet osztályra van szükségünk. Ez azt eredményezi, hogy minden YOLO réteg előtti konvolúciós rétegben csökkenteni kellett a szűrők számát az alábbi képlet alapján:

$$filters = (classes + 5) * 3$$

A *batch* értékét 64-re állítottuk és először 16-os *subdivision*-t használtunk azonban ez túl sok memóriát igényelt, így meg kellett duplázni 32-re.

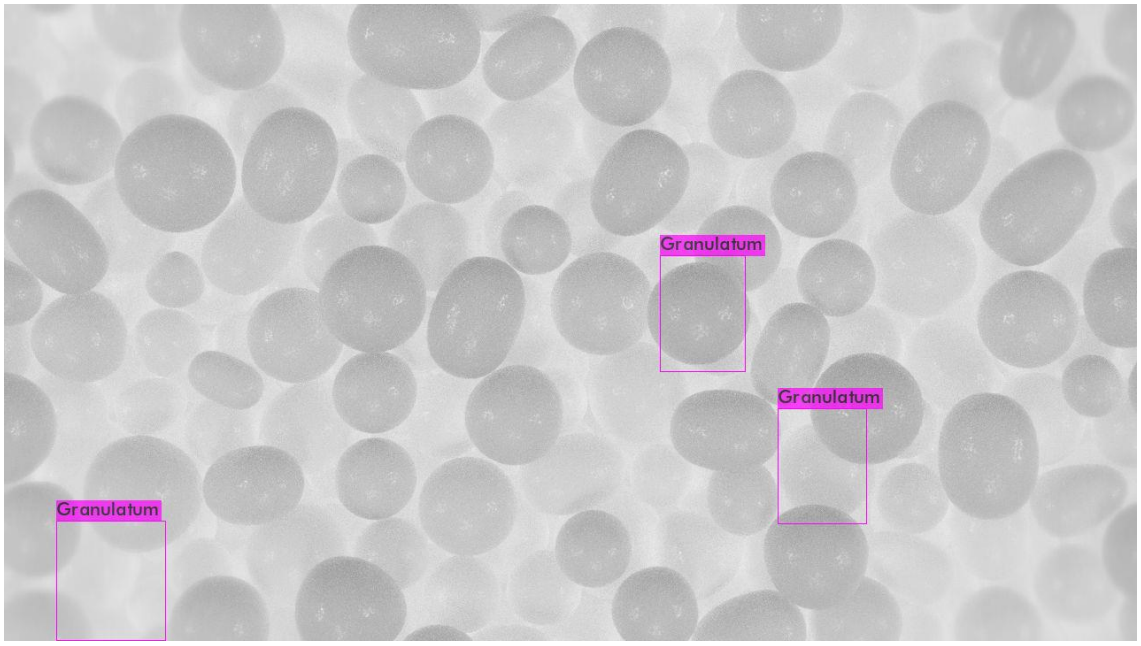
A módosított értékek listája tehát a következő:

- *classes* = 1

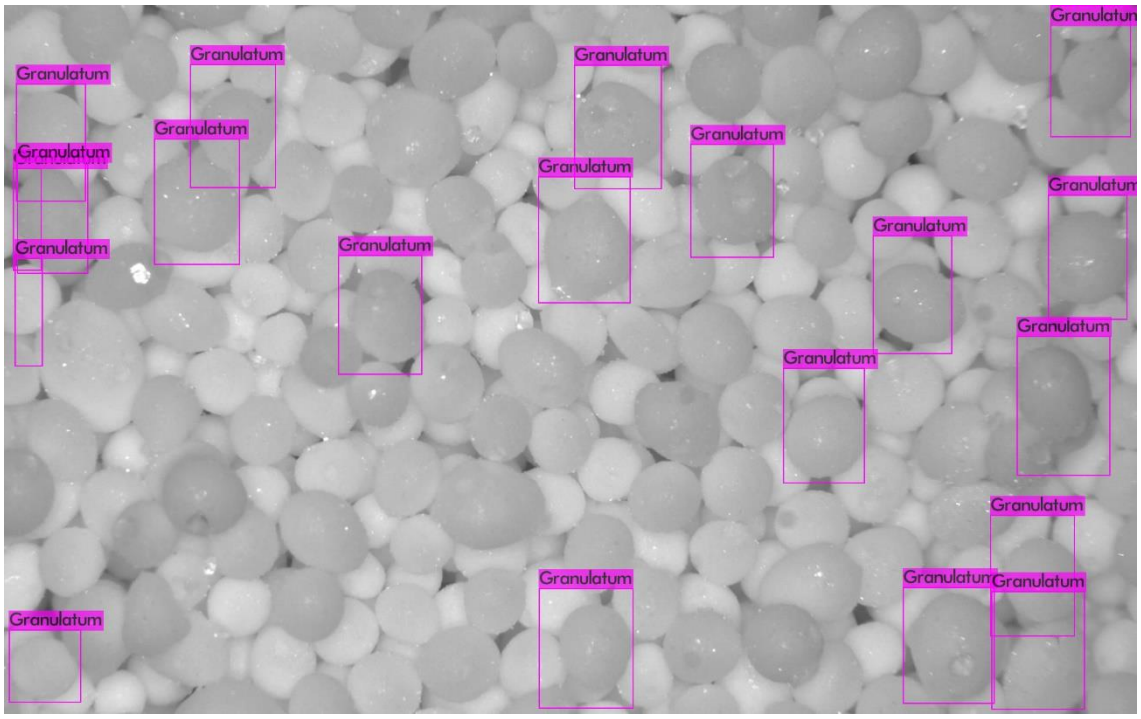
- *filters* = 18 (csak a YOLO réteg előtt konvolúciós rétegekben)
- *batch* = 64
- *subdivision* = 32

5.2 A tanítási folyamat

A tanítást 4000 iterációig futtattuk, ahol már elég jó eredményeket produkált a háló generált és valódi képeken egyaránt. A 24-25. ábrán látható az 1000 iterációig tanított háló által legalább 20% bizonyossággal detektált granulátumok.

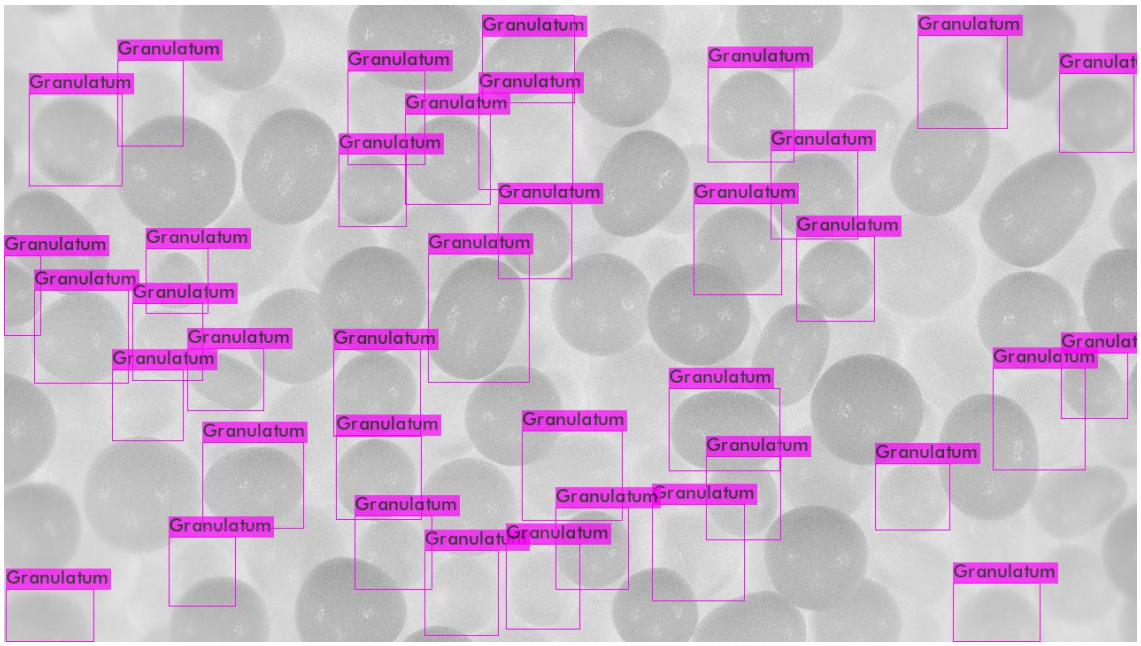


24. ábra - Detektálás legalább 20% bizonyossággal 1000 iteráció után generált képen

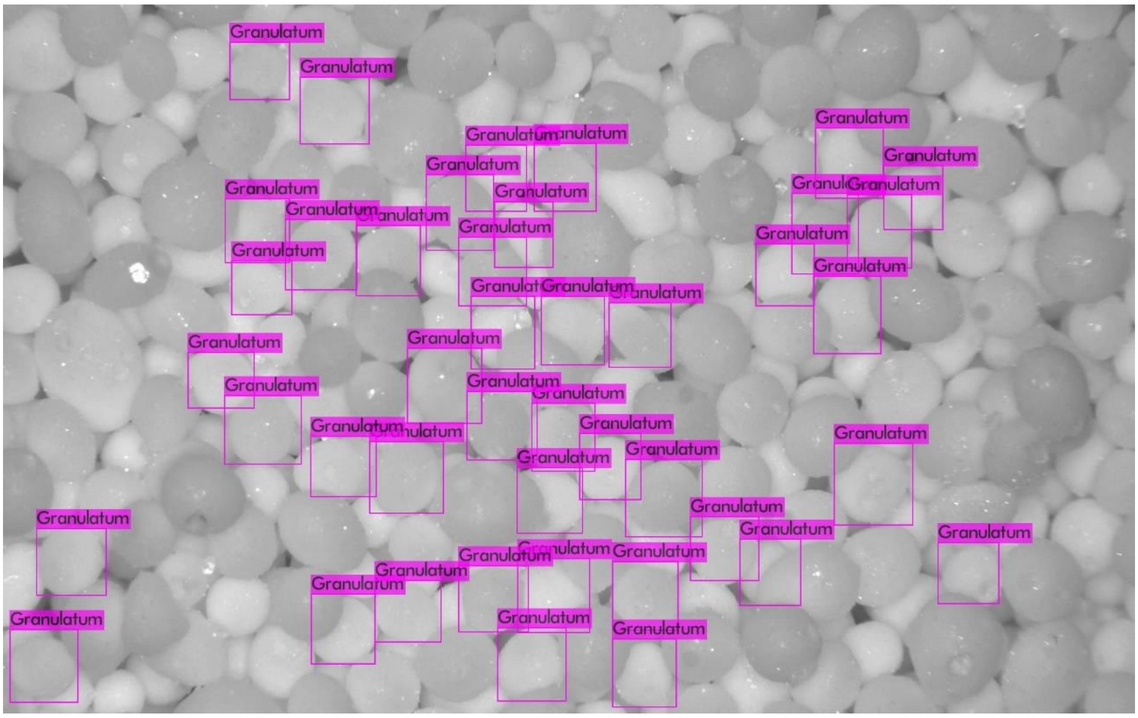


25. ábra - Detektálás legalább 20% bizonyossággal 1000 iteráció után valós képeken

A 26-27. ábrán pedig a 4000 iterációig tanított háló által legalább 80% bizonyossággal detektált granulátumok.



26. ábra - Detektálás legalább 80% bizonyossággal 4000 iteráció után generált képen



27. ábra - Detektálás legalább 80% bizonyossággal 4000 iteráció után valós képen

A 4000 iteráció után készített képeken a jobb demonstráció érdekében megemeltük a határértéket. Így jól látszik mennyivel pontosabb határoló kereteket generál a háló. Egy felmerülő probléma, hogy a háló detektálja a kép szélén lévő nem teljesen látható granulátumokat is. Ez azonban könnyen szűrhető a koordináták ismeretében.

6. fejezet

A modell teljesítményének mérése

6.1 Metrikák

Az eredményeink kiértékelése során az alábbi metrikákat használtuk:

- Intersection over Union (IoU):

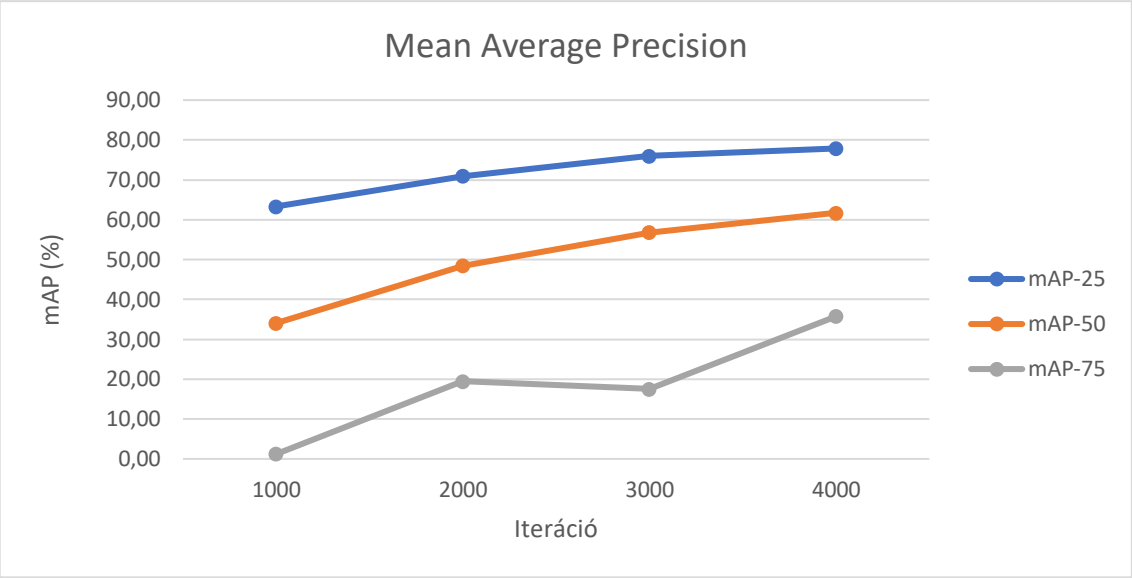
$$IoU = \frac{Metszet\ ter\ddot{u}lete}{Uni\acute{o}\ ter\ddot{u}lete}$$

- Mean Average Precision (mAP): átlag precizitás (precision) adott osztályra, melyből valós pozitívnak a legnagyobb általunk választott IoU határérték feletti határoló keret számít

$$Precizit\acute{a}s = \frac{Val\acute{o}s\ pozitív}{Val\acute{o}s\ pozitív + \acute{A}lpozitív}$$

6.1.1 Teljesítmény generált adatokra

Az alábbi eredményeket a YOLO beépített mAP funkciójával mértük az általunk használt tanító adathalmazon. Három különböző mérést végeztünk, három különböző IoU határértékkel, melyek rendre 25%, 50% és 75%.



28. ábra - Teljesítmény generált adatokon

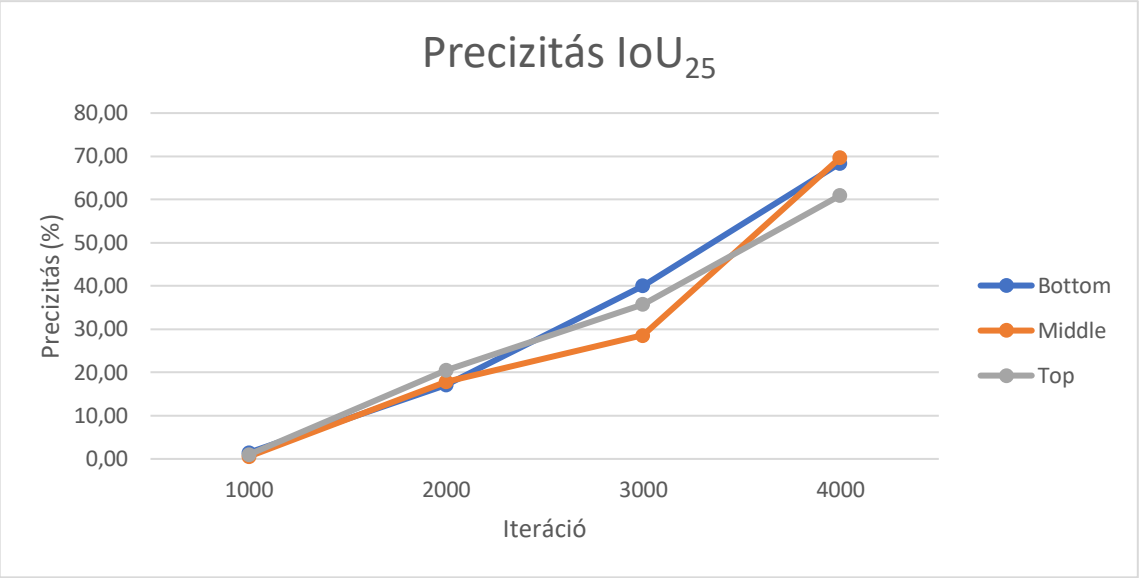
A diagrammon látható, hogy 25% és 50%-nál már csökken a meredekség azonban 75%-nál még nem. Tehát a teljesítmény még elméletben növelhető. Ez jó jel hiszen a méret meghatározására csak a magas bizonyossággal detektált határoló keretek alkalmasak. A sarzs átlag átmérőjének a pontos megbecsléséhez ehhez minél több, ideálisan a képen látható összes granulátum méretének meghatározására lenne szükség.

6.1.2 Validáció valódi adatokkal

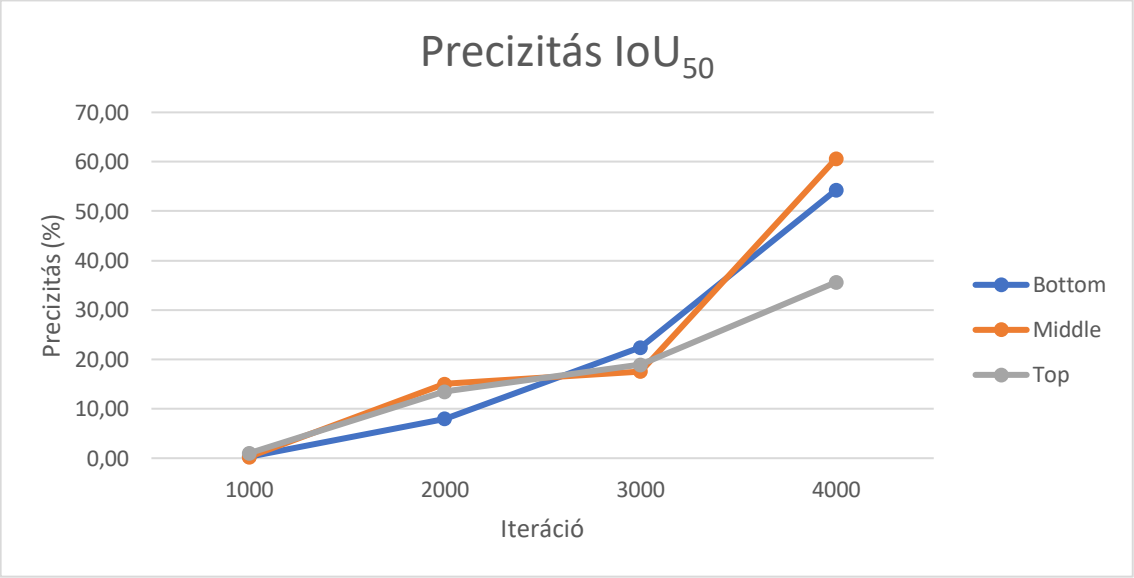
A valós adatokon való validációhoz ideálisan egy teljesen felannotált adathalmazra lenne szükségünk. Ez azonban a granulátumok sűrűsége miatt nehezen megoldható feladat. Úgyhogy úgy döntöttünk, hogy kiválasztunk véletlenszerűen képeket, melyeken annotálunk minél több granulátumot minél pontosabban három különböző mélységben:

- Top: legfelső réteg, ahol kizárólag takarásmentes granulátumok láthatóak (felső réteg)
- Middle: takarásban lévő, de még jól látszódó granulátumok (középső réteg)
- Bottom: több granulátum által takarásban levő granulátumok (alsó réteg)

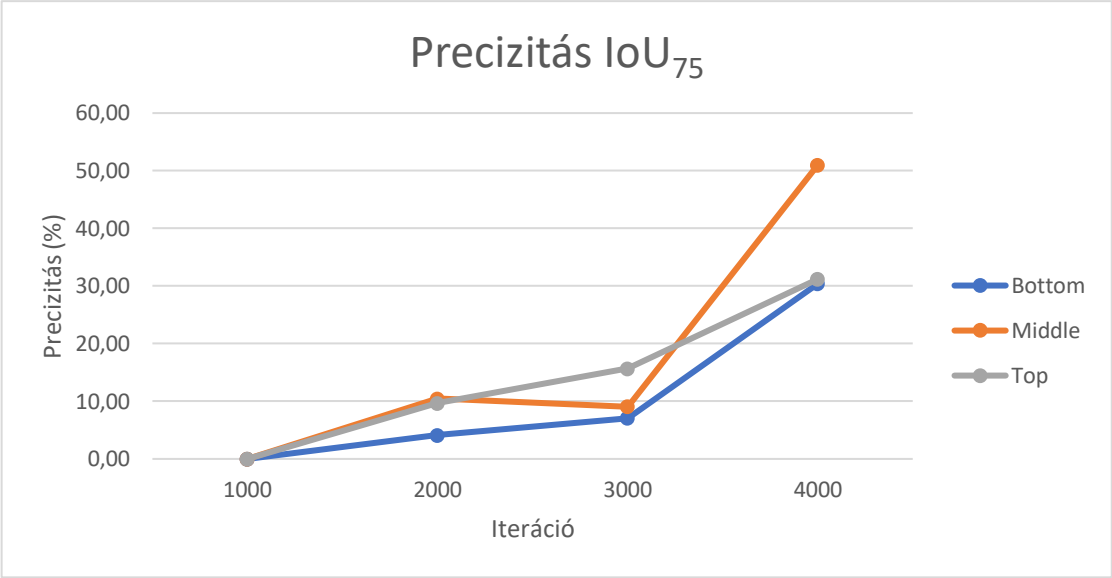
Azért választottuk meg így a validáló halmazt mert a képeken maximum 3 jól elkülöníthető réteg granulátum látható. Az ez alatt lévők már túl sötétek, valamint minél kevesebb látható egy granulátumból annál nehezebb meghatározni a határoló keretét. Kilenc képen összesen ~1200 granulátumot annotáltunk, majd megvizsgáltuk, hogy van-e egyezés egy adott IoU határérték felett. Az eredmények a 29-31. ábrán láthatóak.



29. ábra - Precizitás 25% IoU felett



30. ábra - Precizitás 50% IoU felett



31. ábra - Precizitás 75% IoU felett

Ami első ránézésre nagyon jól látszik, hogy a generált és a valódi képeken 4000 iteráció után hasonló eredményeket produkált a háló. Továbbá meglepően hasonló teljesítményt ért el a három különböző kategóriában és érdekes módon az alsóbb rétegekben rendre jobb eredményeket értünk el, mint a legfelsőben. Azonban koránt sem jelentős az eltérés bármilyen messzemenő következtetés levonásához.

Az annotált granulátumok számában sem volt jelentős különbség a rétegek között. Az alsó- (bottom) 339, a középső- (middle) 353 és a felső (top) rétegben 497 annotált granulátum volt.

A generált képekkel ellentétben a valós képeken a görbék meredeksége nem mutat csökkenő tendenciát, habár az eredmények szórása, valószínűleg a viszonylag kis mennyiségű adatoknak köszönhetően elég nagy. Jellegükből arra következtetünk, hogy a háló teljesítménye növelhető még tanítással a valós adatokon és még nem járunk a túltanulás közelében.

7. fejezet

Összefoglalás

A kutatásunkban műanyag pelleteket modelleztünk a Blender modellező szoftver és a BlenderProc keretrendszer felhasználásával. A műanyag pelletekről ömlesztett képeket generáltunk, a pelletek elhelyezkedését és határoló dobozait a képekhez tartozó fájlokba mentettük. Az így készült képek és annotációs adatok alapján a transfer learning módszerével egy előtanított konvolúciós neurális hálózat struktúrát tanítottunk tovább. A munkánk során a YOLOv3 modellt használtuk előtanított hálózatnak.

Kutatásunk során egyértelműen bizonyítottuk, hogy az eljárás működik az általunk vizsgált képekre és felhasználási területre. Határait azonban nem ismertük meg. A tanítás hosszán és az adathalmaz nagyságán nem volt lehetőségünk iterálni. Megfelelő erőforrások birtokában átfogó ismereteket nyerhetnénk a módszerünk korlátairól.

A kutatásunknak egy másik lehetséges folytatása egy komplexebb anyag készítési megoldás kidolgozása, például GAN használatával. Ezzel a módszerrel a generált képek sokkal jobban közelíthetők a valódi képekhez, a diszkriminátor hálózat egyes rétegei pedig a detektáló hálózat számára is hasznosak lehetnek.

A kutatás folytatásaként az annotáció generálást is tovább lehetne tovább finomítani. A jelenlegi annotáció készítő még csak azokra az objektumokra tud szűrni, amik beleesnek a kamera látóterébe, de nem képes érzékelni, hogy azok esetlegesen teljesen takarásban vannak, így fölösleges annotációk is kiírásra kerülnek. Ennek egy lehetséges megoldása, hogy a kamera objektumból sugárkövetés segítségével tovább szűkítjük az objektum halmazt a ténylegesen látható objektumok halmazára.

Egy további kutatási terület az általunk kiválasztott YOLO modellen kívül más modelleken is a tanítási folyamatot végig vezetni, és az eredményeket összehasonlítani az általunk elért eredményekkel.

Köszönetnyílvánítás

A FIEK_16-1-2016-0007 számú projekt a Nemzeti Kutatási Fejlesztési és Innovációs Alapból biztosított támogatással, a Felsőoktatási és Ipari Együttműködési Központ - Kutatási Infrastruktúra Fejlesztése (FIEK_16) pályázati program finanszírozásában valósult meg.

Irodalomjegyzék

[1] W. S. Sarle, *Neural Networks and Statistical Models,* 1994.

[2] B. Gyires-Tóth, Szerző, *BME Deep-Learning előadás.* [Performance]. 2020.

[3] D. Hoffmann, D. Tzionas, M. Black és S. Tang, *Learning to Train with Synthetic Humans,* 2019.

[4] C. Mitash, K. E. Bekris és A. Boularias, „A Self-supervised Learning System for Object Detection using Physics Simulation and Multi-view Pose Estimation," 9 3 2017.

[5] M. Grard, R. Brégier, F. Sella, E. Dellandréa és L. Chen, „Object segmentation in depth maps with one user click and a synthetically trained fully convolutional network," in *HFR - 10th International Workshop on Human-Friendy Robotics,* Napoli, 2017.

[6] L. Lidberg, „Object Detection using deep learning and synthetic data," 2018.

[7] X. Peng, B. Sun, K. Ali és K. Saenko, *Learning Deep Object Detectors from 3D Models,* 2015.

[8] N. Bhandari, *Procedural Synthetic Data for Self-Driving Cars,* MASSACHUSETTS: MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2018.

[9] B. Foundation, „blender.org," [Online]. Available: https://www.blender.org/. [Hozzáférés dátuma: 02 10 2020].

[10] B. Foundation, „docs.blender.org," [Online]. Available: https://docs.blender.org/api/current/index.html. [Hozzáférés dátuma: 02 10 2020].

[11] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi és H. Katam, *BlenderProc,* 2019.

[12] O. Ben-Kiki, C. Evans és I. d. Net, „yaml.org,” 01 10 209. [Online]. Available: https://yaml.org/spec/1.2/spec.html. [Hozzáférés dátuma: 27 10 2020].

[13] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. D. Nardi, M. Goesele, S. Lovegrove és R. Newcombe, „The Replica Dataset: A Digital Replica of Indoor Spaces,” 13 6 2019.

[14] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva és T. Funkhouser, *Semantic Scene Completion from a Single Depth Image*, 2016.

[15] T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas és C. Rother, „BOP: Benchmark for 6D Object Pose Estimation,” *European Conference on Computer Vision (ECCV),* 2018.

[16] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother és J. Matas, „BOP Challenge 2020 on 6D Object Localization,” *European Conference on Computer Vision Workshops (ECCVW),* 2020.

[17] L. Torrey, *Transfer Learning,* University of Wisconsin, 2010.

[18] S. D. R. G. A. F. Joseph Redmon, *You Only Look Once: Unified, Real-Time Object Detection,* University of Washington, 2016.

[19] A. F. Joseph Redmon, „YOLOv3: An Incremental Improvement,” University of Washington, 2018.

[20] B. Foundation, „docs.blender.org,” Blender Foundation, [Online]. Available: https://docs.blender.org/manual/en/latest/render/eevee/index.html. [Hozzáférés dátuma: 04 10 2020].

[21] B. Foundation, „docs.blender.org,” Blender Foundation, [Online]. Available: https://docs.blender.org/manual/en/latest/render/cycles/index.html. [Hozzáférés dátuma: 04 10 2020].

[22] S.-K. László, A. György és C. Ferenc, „Geometriai transzformációk,” in *Háromdimenziós grafika, animáció és játékfejlesztés*, Budapest, ComputerBooks / Leonardo SNS, 2005, p. 486.