Text2Scene
Praktikum

Raumerzeunging
+
Optimierung

Alexandru Barsan
Dayana Khadush
Jan Lycka

Abgabedatum: 18.03.2021

Text Technology Lab - Goethe Universität
Prof. Dr. Alexander Mehler
Alexander Henlein

Erklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

Frankfurt am Main, 18.03.2021

_____

Ort, Datum

Alexandru Barsan, Dayana Khadush, Jan Lycka

_____

Unterschrift

Inhaltsverzeichnis

1 Allgemein

Für eine $3D$ Indoor-Scene Generierung existieren etliche Modele, die auf unterschiedlichen Ideen basieren. Genrell besteht die Szenenerstellung aus 3 Schritten:

- *Offline Learning*: Lernen oder beschreiben von *Priors*. Dieser Schritt ist eine Extraction von Informationen verschiedener Art: über verschiedene Objekte und ihren Kategorien, Relationen zwischen Kategorien und die Häufigkeit des Vorkommens in Räumen. Diese Priors können von einem Model zu anderem unterschiedlich sein - "Hardkodiert", statistic-basiert oder von neuronalen Netzwerke gelernt.
- *Input Preprocessing*: aus dem Input werden wichtige Informationen entnommen und in das richtige Aufbau gebracht - interne Repräsentationen. Diese informationen kann die Raumgeometrie (die Höhe, die Breite und Länge, die Positionen von Fenster und Türen) sein, seine Kategorie usw. Darüber hinaus, es können auch Informationen über Objekte oder Objektkategorien sein, die in der Szene sein sollten.
- *Optimization:* Objektke werden ausgesucht und auf ausgewählte Positionen hinzugefügt.

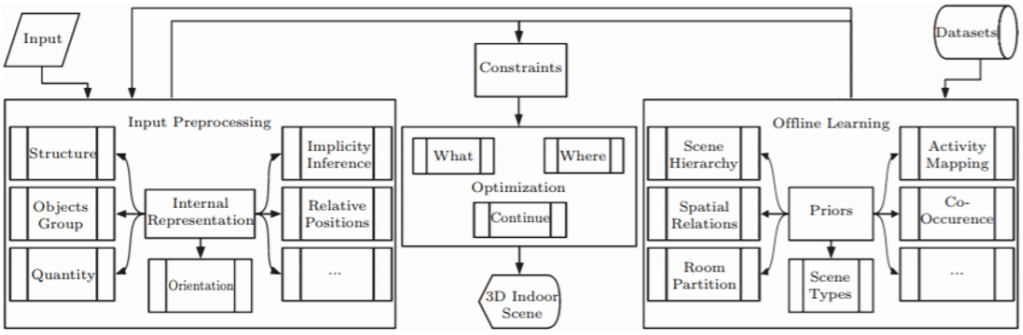

Abbildung 1.1: Architektur von Modellen für $3D$ Indoor-Scene Generierung. Zhang SH, Zhang SK, Liang Y et al. A survey of 3D indoor scene synthesis. **[1]**

Machine Learning algorithmen bieten eine starke Grundlage für das Lernen von sogenannten "Priorsünd für die Szenenoptimierung. Die Datensätze, die für die Prior-subtraktion verwendet werden können RGB Bilder und $3D$ Modele von Räumen oder Graphen-repräsentationen sein. Die Graphen können beispielsweise informationen über Aktivitäten liefern, die mit jeweiligen Objekten in einer Verbindung stehen. Die "co-occurrenceünd die Relationen zwischen unterschiedlichen Elementen können auch sehr gut in Baum-Graphen dargestellt werden. Bilder und $3D$-Modelle werden häufig in Modellen für Szenensynthese benutzt, die aus

eneinem *Convolutional Neural Network* bestehen. Ein solches Model ist **Deep-Synth**. Das Optimierungs-teil besteht sogar aus 3 CNNs. Die Szenensynthese und das Lernen von Priors erfolgt in $2D$. Die Generierung erfolgt iterativ mithilfe von 3 vortrainierten CNNs:

- *Continue* entscheidet ob neue Objekte in die Szene hinzugefügt werden.
- *Choose category* entschidet welches Objekt es sein sollte und wo muss das Objekt platziert werden.
- *Place object instance* definiert eine Konkrete Position und Rotation von dem Objekt und fügt das in die Szene.

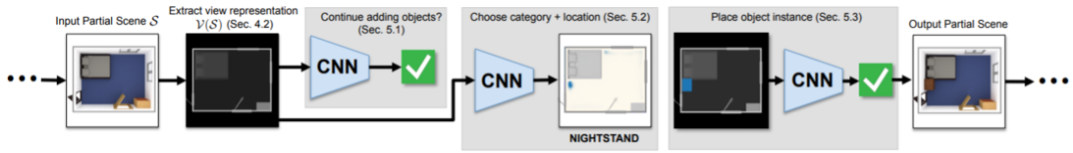

Abbildung 1.2: Deep-Synth architecture
[2]

Das Deep-Synth ist auf dem SUNCG Datensatz aufgebaut. Das heißt, das Lernen von Priors erfolgt mithilfe von Szenen aus SUNCG und die Objekte in den Szenen sind SUNCG-objekte. Dennoch ist es möglich die Model-Architectur für ein Model mit Objekten aus anderen Datensätze zu übernehmen. Dennoch ermöglicht das Model nur eine Generierung von Szenen, in denen keine Objekte auf anderen Objekten stehen oder liegen. Der Grund dafür ist die Repräsentationen, die verwendet werden - *top-down view* . Das model kann also keinen Fernseher auf eine TV-bank stellen.

2 Aufgabe 1

Deep-Synth ist ein Iteratives Model. Also in jedem Schritt haben wir eine Teilszene - ein Raum mit Objekte. Dies ermöglicht uns nicht nur Synthese von neuen Szenen, sondern auch das ”Möblieren”von vorhandenen Räume. Das bringt uns zu der Aufgabenstellung: Gegeben eine Teilszene in UIMA-format versuchen die Szene mit Deep-Synth zu verfvollständigen.

Die Aufgabe besteht daher aus folgenden Schritte:

- Deep-synth startbereit machen. Den Zugrif auf SUNCG Datasatz bekommen. Deep-synth trainieren;
- Unity-tool (VAnnotatoR) startbereit machen;
- Das UIMA Format in Deep-Synth Format konvertieren;
  - XMI zu Json konvertieren. Dabei sollte die Struktur von UIMA und von Deep-synth verstanden, interpretiert und angepasst sein;
  - ShapeNet-Objekte aus UIMA mit einem Objekt aus SUNCG austauschen. Dabei sind Größen, Formen und Kategorien zu beachten (z.B. Esstisch und Arbeitstisch);
  - UIMA Raum mit einem aus SUNCG ersetzen. Die Fenster und Türen sind zu beachten;
  - Die Positionen, Rotationen und Skalierungen von Objekten in UIMA Format in das von SUNCG konvertieren.
- Die Szene vervollständigen;
- Die Szene zurück in UIMA Format wandeln.

2.1 XMLToJSON

Um die in XML Datei beschrieben Szene ins JSON-Format zu konvertieren wurde eine neue Klasse “XMLToJSON” in das Tool hinzugefügt. Die Umwandlung basiert auf den Beispieldaten: *Example.xml* und *ACL21_Example.xml*.

Die Umwandlung ist folgendes strukturiert:

1. **Initialisierung des JSON Dokuments**
   Am Anfang erstellen JSONObject, das am Ende ins JSON Datei geschrieben wird.
2. **Einlesen und Parsen des XML Documents**:
   Das XML Document wird eingelesen und mit DocumentBuilder geparst, indem man die root speichert. Danach werden alle Objekte der Szene, die als *“isospace:SpatialEntity”* zu finden sind, in der *root* gesucht und in der *NodeList nList* gespeichert.

3. **Bestimmen von SUNCG Objekte**:
   Wir haben eine neue Datei res.scv erstellt, die alle benötigten Informationen über SUNCG Objekte aus Models.csv und ModelCategoryMapping.v2.csv erstellt. Die Methode *ShapeNetIDtoSUNCGID* wählt zufällig ein Objekt aus einer Liste allen gefundenen Objekten der gleichen Kategorie und ungefähr gleichen Größen.

4. **Erstellen *Maps* mit Parametern von SUNCG und ShapeNet Objekte**:
   Mit Methoden IdSUNCG, IdShapeNet in *RoomJsonImporter* erstellen Kollektionen von Ids von Objekten aus den Datensätze und ihre Beschreibungen, die dann für das austauschen von Modellen in Szenenen in UIMA und Deep-Synth benutzt werden. Für SUNCG sind es *fine_grained_class,coarse_grained_class* and *nyuv2_40class*. Für ShapeNet: *wnlemmas, name, category, tags*.

   Gegeben eine "wnlemmas Beschreibung, wir wollen alle Modele in SUNCG mit dem gleichen "fine_grained_class"finden. Wenn solchen Objekte nicht gibt, suchen wir unter den "coarse_grained_classüsw. Um die Größen von Objekte zu vergleichen, brauchen wir die Skalierung von dem Objekt, die bereits aus XML abgelesen wurde. Jetzt müssen nur die Größen von Objekte gelesen werden. Dafür haben wir die Methoden DimsSUNCG und DimsShapeNet, die Maps mit älignedDims"liefern. Alle Maps sind als statische Variablen gespeichert.

5. **for (Node in NodeList)**:

   5.1. **Extrahierung der Objekt, Position, Rotation und Skalierung IDs**:
   Erstens werden werden die IDs für das Objekt aus der XML Szene gespeichert in Variablen gespeichert.

   5.2. **Extrahierung der eigentliche x, y, z, w Koordinaten für Position, Rotation und Skalierung**:
   Wenn ein Objekt aus der Szene auch ins ShapeNet zu finden ist (und das Object ist kein Lebewesen, Spielzeug oder Besteck), dann werden die entsprechende Werte fur die IDs aus 5.1 mit Hifle der Methoden getPosition(), getRotation() und getScale() in double [ ] gespeichert.

   5.3. **Berechnung der x, z Koordinaten für Position**:
   Diese werden bezüglich auf die Roomgeometrie und dessen Parameters berechnet.

   5.4. **Berechnung der Rotation und Skalierung + Position $\Longrightarrow$ TransformationsMatrix**:
   Die informationen über die Position, Skalierung und Rotation von einem Objekt in Deep-Synth Szene sind als TransformationsMatrix gespeichert. Dabei ist zu beachten, dass Deep-Synth unterstützt keine Rotationen um **x** oder **z** Axen. Ausserdem ist auch wichtig, dass die Koordinaten von Objekte sind Global gegeben. Das bedeutet, dass die Parameter von dem Raum aus der Szene auch definiert sein sollten. Also, ein Objekt ist durch x und z Koordinaten und ein Winkel um die y-axis definiert. Diese Werte zusammen mit der Skalierung von dem Raum

ergeben die Gewünschte Transformationsmatrix (siehe das Bild 2.1, zpad = *"To separate the floor from empty spaces, floors will have a height of 0.5m."*).

```
def get_transformation(self):
    """
    Get the transformation matrix
    Used to render the object
    and to save in json files
    """
    x,y,r = self.x, self.y, self.r
    xscale = self.room.synthesizer.pgen.xscale
    yscale = self.room.synthesizer.pgen.yscale
    zscale = self.room.synthesizer.pgen.zscale
    zpad = self.room.synthesizer.pgen.zpad

    sin, cos = math.sin(r), math.cos(r)

    t = np.asarray([[cos, 0, -sin, 0], \
                    [0, 1, 0, 0], \
                    [sin, 0, cos, 0], \
                    [0, zpad, 0, 1]])
    t_scale = np.asarray([[xscale, 0, 0, 0], \
                          [0, zscale, 0, 0], \
                          [0, 0, xscale, 0], \
                          [0, 0, 0, 1]])
    t_shift = np.asarray([[1, 0, 0, 0], \
                          [0, 1, 0, 0], \
                          [0, 0, 1, 0], \
                          [x, 0, y, 1]])

    return np.dot(np.dot(t,t_scale), t_shift)
```

Abbildung 2.1: DeepSynth - Berechnung TransformationsMatrix
[3]

Das Problem liegt daran, dass die Objekte in der Uima-Szene dürfen beliebig gedreht werden. Dabei ist die Position als x, y, z Koordinaten gespeichert und liefert nicht die Tatsächliche Position eines Objects, so dass das Objekt dann Rotiert werden kann. Das kann man auch beim Rendern sehen: für je ein Objekt haben wir *id_center* Elternobjekt und sein Kind: *id*. Die Position des id$_center sind Positionen aus UIMA. Di tool$).

Viele Objekte in den Datensätze sind haben nicht die standarte Rotation: $up = (0, 1, 0)$ und $front = (0, 0, 1)$. Diese Vektoren zusammen mit Objektgrößen liefern uns das Kind-Objekt in Unity.

Mit diese verschieden Rotationen haben wir versucht letzendlich die TransformationsMatrix zu berechnen. Es wurde auch versucht die Methode von DeepSynthin unser Tool zu benützen aber man mit keine Erflog, da wir keine Werte für dieProjektion in unsere Szene haben. Vielleicht könnte man die Projektion irgend-wie berchnen um die Werte für xscale, zscale und yscale zu kriegen, damit mandann eigentlich nur die Werte in die Formel angegeben von DeepSynth ersetzum die richtige TransformationsMatrix zu kriegen. Dann sollte das SceneSynthModell eine *"logische"* Szene auch generieren.

Normalerweise um eine solche Matrix zusammenzustellen muss man die Position, Rotation und Skalierung Matrices multiplizieren wie in Abb 2.2. Es wurde auch versucht auf dieser Weise diese zu berechen aber mit kein Erflog.

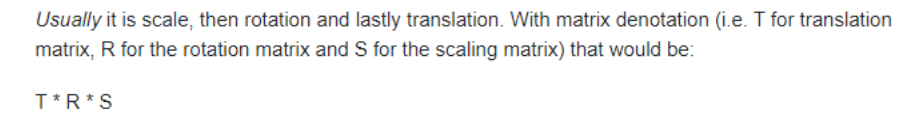

Abbildung 2.2: Übliche Methode zur Berechnung der Transformationsmatrix **[4]**

So dass verschiedene Variationen um diese von x, y, z für die Position und Skalierung und x, y, z, w Koordinaten für die Rotation angegeben im Quaternion zu berechnen versucht haben. Wie es früher offensichtlich gemacht wurde haben wir im Moment keine konkrete Lösung für dieses und foglich haben wir keine Szenen mit DeepSynth generiert welche in XML beschrieben wurden.

5.5. **Das Erstellen von einen JSONArray mit dem entsprechender min und max BBox**:
Die Bounding Boxes sind Koordinate von zwei Ecken des Raumes, die seine Position und Größe definieren.

5.6. **Die Erstellung des Raumes mit dem entsprechenden ID, Nodes, roomType, modelId und Bounding Boxes**:
Die Größe von dem Raum in der UIMA-Szene sind die BoundingBox Größen von dem Raumobjekt, das mit Suche nach spatialEntities mit ids erfolgt, das das Wort *room* beinhaltet. Zu beachten ist, dass Deep-Synth hat keine konkrete Information über mögliche Räume. Genau gesagt, SUNCG hat keine Modelle von Räume. Beim Generieren von neuen Szenen werden zusammen mit dem befehl zum Starten auch ßtartänd ënd"parameter gegeben, welche die Grenzen von Indexen von den Räumen aus SUNCG definieren, die als Grundlage für neue Szene benutzt werden. D.h. wenn $start = 0$, und wir wollen ein *Bedroom*, dann werden aus "deep-synth/data/bedroom/0.pkl"die Informationen über den Raum genommen: floor, welche in Form eines Tensors gespeichert sind.

Es werden mehrere JSONArrays und JSONObjects erstellt und kumuliert, damit der letztendliche JSONObject der DeepSynth Format überstimmt.
Der Raum wird in JSONObject roomObject erstellt. Weiterhin werden der Raum und die Objekten in diesem zu ein anderes Objekt gespeichert, welches zu einem Array addiert werden, welches danach zum initialem JSONObject "json" addiert werden, welches danach zum JSON Dokument geschrieben wird.

2.2 Optimaler Ablauf:

- Man erhalte eine UIMA (Shapenet) Datei in XML Format **Example.xml**
- Man konvertiere diese in eine JSON Datei **Example.json**, die dann mit DeepSynth kompatibel ist
- Dies wird gespeichert in **%deep-synth%/shapenet/Example.json**
- Man führe deep-synth aus, mit dem Kommando: python batch_synth_shapenet.py –save-dir newtest_bar –data-dir living –model-dir res_1_living –location-epoch 300 –rotation-epoch 300 –start 0 –end 1 –shapenet-dir shapenet
- In **batch_synth_shapenet.py** wird **Example.json** eingelesen und dessen Raumgröße ermittelt. Ein ähnlich großer Raum wird dass aus SUNCG ausgesucht **Raum.json**.
- Diese Informationen werden an **scene_synth_shapenet.py** weitergegeben, was im Vergleich zur standarden **synth_shapenet.py**, eine leere Szene mit **Raum.json** als vorlage direkt mit Nodes aus **Example.json** belegt. Danach werden es, wie gewöhnlich, weitere SUNCG Objekte eingefügt, solange es der continue_predictor erlaubt.
- Eines der Ergebnisse wird mit dem Stichwort Final versehen z.B. **0_0_8_final.json**
- Das Importer-tool (Schritt12.java) sucht nach einer Datei mit so einem Stichwort und liest **0_0_8_final.json** dann ein, als Interne Repräsentation der Szene. Es ersetzt alle Objekte mit ihren ShapeNet Pendants.
- Nach dessen Durchlauf erhalte man eine Shapenet XML Datei **0_0_8_final.xml**, die standardmäßig mit der internen Speicherfunktion produziert wird
- **Verbesserungsidee:** Man behalte alle Objekte aus Example.xml und fügt diese in **0_0_8_final.xml** ein, damit zB. eine rote Couch nicht durch eine blaue ersetzt wird, kraft der Konvertierung con XML – JSON- und wieder XML. Es ist besser Originale zu behalten als sie hin und her zu konvertieren. Uns geling es nicht, aufgrund von gelinde gesagt technischen limitierungen.

3 Aufgabe 2

3.1 Optimaler Ablauf:

- Man führt das Tool aus mit der Klasse GenerateNewRoomWithDeepSynth.java und dem argument „-roomtype **x**“ wo **x** ist bedroom, office oder living. Z.B. „-roomtype office“
- Dies baut ein Kommando zusammen, zB python batch_synth.py –save-dir aufgabe_2_tmp_data –data-dir bedroom –model-dir res_1_bedroom –start 0 –end 1 –rotation-epoch 180
- Dies führt das deepsynth-Python-Skript mit JEP aus und wartet, bis es zu Ende läuft.
- Es werden weitere SUNCG Objekte eingefügt, solange es der continue_predictor erlaubt. der Ergebnisse wird mit dem Stichwort Final versehen z.B. **0_0_8_final.json**
- Das Importer-tool (GenerateNewRoomWithDeepSynth.java) sucht nach einer Datei mit so einem Stichwort und liest **0_0_8_final.json** dann ein, als Interne Repräsentation der Szene. Es ersetzt alle Objekte mit ihren ShapeNet Pendants.
- Nach dessen Durchlauf erhalte man eine Shapenet XML Datei **0_0_8_final.xml**, die standardmäßig mit der internen Speicherfunktion produziert wird

3.2 Probleme:

Man brauche im Aktuellen Stand zugleich zugriff auf CUDA Grafikkarten und auf IntelliJ. Auf Rawindra kann man IntelliJ nicht installieren, da nur Kommandozeile und Lokal hat keiner von unserer Gruppe CUDA zu verfügung. D.h. wir müssen den Vorgang aufspalten und den CUDA-Teil Lokal nur simulieren. Wir laufen deep-synth manuell und fügen dann das Ergebnis in in %importer%/deepsynth/aufgabe_2_tmp_data**0_0_8_final.json** und erst dann führen wir das Tool (GenerateNewRoomWithDeepSynth.java) aus mit der Flagge -demo, sodass es mit CUDA nicht gerechnet wird und mit der Flagge -nodel, sodass es die Eingabe Für unser tool nicht gelöscht wird, da es erwartet wird, dass diese von deep-synth angelegt wird und deshalb wird sie ansonsten Sauberkeitshalber gelöscht.

Die Einrichtung von Deep-Synth hat auch die veränderung von create_data gefordert und auch eine Versionsveränderung gewisser deprecateden Module. Daher Ist mit drinne in Github repo auf https://github.com/janlycka/suncg_shapenet_tool das Verzeichnis Namens deep-synth-stand-april-21, wo alle Dateien gespeichert sind, zum Zeitpunkt unserer Abgabe. Von Interesse sind create_data.py und requirements2018.txt. Dies wurde in einer Issue angegangen https://github.com/brownvc/deepsynth/issues/8 .

4 Literatur

[1] Zhang S., Zhang S., Liang Y. et al. **A survey of 3D indoor scene synthesis**
https://www.researchgate.net/publication/333135099_A_Survey_of_3D_Indoor_Scene_Synthesis

[2] Kai Wang, Manolis S., Angel X. C., Daniel R. **Deep Convolutional Priors for Indoor Scene Synthesis**
https://kwang-ether.github.io/pdf/deepsynth.pdf

[3] Zhang S., Zhang S., Liang Y. et al. **A survey of 3D indoor scene synthesis - DeepSynth Algorithm row-773**
https://github.com/brownvc/deep-synth/blob/master/deep-synth/scene_synth.py

[4] Game Development Stackexchange
https://gamedev.stackexchange.com/questions/16719/what-is-the-correct-order-to-multiply-scale-rotation-and-translation-matrices-f