Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Synthetic 3D Scan Generation with Parameterized Models of Bags and Bundles

Bachelor Thesis

2024
Martin Ropjak

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Synthetic 3D Scan Generation with Parameterized Models of Bags and Bundles

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	doc. RNDr. Martin Madaras, PhD.
Consultant:	Mgr. Lukáš Gajdošech

Bratislava, 2024
Martin Ropjak

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program:

Študijný odbor:
Typ záverečnej práce:
Jazyk záverečnej práce:
Sekundárny jazyk:

Názov:

Anotácia:

Vedúci:
Katedra:
Vedúci katedry:

Dátum zadania:

Dátum schválenia:

garant študijného programu

....

študent vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:
Study programme:

Field of Study:
Type of Thesis:
Language of Thesis:
Secondary language:

Title:

Annotation:

Supervisor:
Department:
Head of department:

Assigned:

Approved:

Guarantor of Study Programme

....

Student Supervisor

Acknowledgments: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Neurónové siete môžu byť efektívne použité na úlohy spracovania obrazu so vstupnými 2D a 3D dátami. Pri spracovaní štruktúrovaných mračien bodov, trénovací dataset môže byť anotovaný manuálne, alebo synteticky vygenerovaný. Pre úlohy lokalizácie alebo klasifikácie objektov zabalených v sáčkoch a balíkoch, vieme použiť parametrizovaný a textúrovaný model sáčku, alebo fyzikálnu simuláciu balenia do balíčka na generovanie syntetických trénovacích dát. Vygenerovaný dataset vieme použit na odhad 6D pózy, alebo na klasifikáciu sáčkov, prípadne na odhad normál deformovaných balíčkov.

Kľúčové slová: syntetické dáta, parametrizované sáčky, lokalizácia, klasifikácia

Abstract

Neural networks can be effectively used for image processing tasks using 2D and 3D input data. When structured point clouds are processed, the training data can be annotated manually, or synthetically generated. For the localization or classification of objects bailed in bags and bundles, a parameterized and textured model of a bag or a physically-based bundling simulation can be used to generate synthetic training data. The generated dataset can be used for 6D pose estimation, classification of bags, or normal estimation of deformed bundles.

Keywords: synthetic data, parametrized bag, localization, classification

Contents

List of Figures

List of Tables

Introduction

The field of 3D computer vision and artificial intelligence has seen significant advancements in recent years driven by the availability of large-scale datasets and powerful machine learning techniques. One critical aspect of this progress is the availability of high-quality 3D scans for training and evaluation. However, obtaining real-world 3D scans can be challenging due to limitations in data acquisition, privacy concerns, and cost.

In this thesis, we address the problem of synthetic 3D scan generation, focusing on bags and bundles. Our main goal is to create a framework that generates realistic 3D scans of bags and bundles, which can be used for various applications, such as object recognition, pose estimation, and scene understanding. Consecutively we will evaluate the quality of the generated datasets using existing neural network for point cloud processing, using metrics such as normal estimation, classification, or 6D pose estimation.

Chapter 1

Theoretical overview

As the use of artificial intelligence continues to grow each year, so does the interest in automating tasks using neural networks. To tackle these tasks, various techniques are being used, such as classification and 6D pose estimation from structured point clouds. However, the availability of large and diverse 3D datasets presents a challenge.

> Since the beginning of the deep learning era, the performance on many computer vision tasks increased drastically. This is closely related to the availability of large real-world datasets as for image classification, object detection and segmentation, or autonomous driving.[8]

Creating datasets for such tasks can be complex and time-consuming. Techniques like manual modeling or manual scanning of objects often lead to limitations in dataset size and diversity of data. Consequently, training neural networks on real-world data alone becomes impractical.

To address this limitation, we have developed a pipeline for synthetic 3D data generation. Our pipeline produces an arbitrary number of 3D data samples, complemented by 2D data annotations. These annotations include point clouds, depth maps, RGB images, and normal maps. By leveraging synthetic data, we aim to enhance the training and evaluation of neural networks.

In this chapter, we explore existing research in the field of synthetic data generation and dive into the theoretical foundations of our approach.

1.1 Related work

Being able to calculate the 6D transformation from an object to the camera is crucial in various applications, including the manipulation of objects using robotic arms. This procedure is commonly referred to as 6D object pose estimation, which involves determining the object's location in 3D space and its orientation[?].

In this thesis, our primary focus will be on constructing a robust pipeline for generating synthetic datasets of 3D bags to improve 6D pose estimation for bin-picking tasks. We draw inspiration from related work that also employs similar techniques to create synthetic 3D datasets, as seen in [9]. However, our approach differs in that it is designed for 6D pose estimation of bags instead of boxes.

One of the largest publicly accessible datasets for object 6D pose estimation in general is the Fraunhofer IPA Bin-Picking dataset[8]. This dataset consists of real-world and simulated scenes with different objects and is fully annotated with 6D poses. The dataset is generated using physical simulation, where various objects are generated within a scene with random positions and orientations above a bin. Subsequently, these objects are dropped into the bin, generating new random scenes.[7].

Our pipeline draws inspiration from both works mentioned and utilizes physical simulation to generate random scenes. Pack models of various shapes and sizes are created above a bin at random positions with random orientations and then dropped into the bin.

1.2 Software Selection

Our choice of software for creating synthetic 3D models of bags focuses on the use of Blender as our primary software tool. We've chosen to adopt Blender for several compelling reasons.

Blender, as an open-source project, is driven by a dynamic community of contributors. A collaborative environment empowers individual creators to make both minor and substantial code changes, resulting in continuous improvements throughout its development. The availability of new features and responsive bug fixes makes Blender an ideal choice for our 3D modelling.

Blender is designed to support the entire 3D pipeline, offering capabilities in rigging, animation, simulation, rendering, compositing, motion tracking, video editing, and even game creation.

Blender provides a robust Python API, allowing us to automate tasks and access Blender's functionalities via Python code. By utilizing Python scripting, we can create custom workflows, automate repetitive processes, and customize Blender to our specific needs. Furthermore, the Blender API enables the development of specialized plug-ins, enabling us to customize the application.

1.3 Blender enviroment

1.3.1 Blender API modules

The Blender API serves as a powerful gateway for users to manipulate various aspects of Blender, such as scenes, meshes, particles, etc. Blender also features an embedded Python interpreter that loads on the launch of the Blender application. One limitation of the API is that only a single Blender file can be opened at a time. The Blender API consists of various Python modules, such as bpy, mathutils, and bmesh.

The bpy module functions as the top level module within Blender's API. It allows users to perform operations that mirror actions available in the GUI version of Blender. For instance, operations such as scaling, translation, applying modifiers, etc. Additionally, developers can reveal the Python code behind these GUI operations in preferences under the interface section. The bpy module also includes submodules such as app, context, data, msgbus, ops, path, props, types, and utils[4].

The mathutils module[2] equips users with mathematical tools for working with vectors, matrices, colors, and other essential mathematical constructs.

The bmesh module[1] provides access to Blender's internal mesh editing API. Its features grant users access to Blender's own mesh editing toolset. The user can utilize operations such as split, separate, collapse, and dissolve and is provided with geometry connectivity data. The bmesh module was designed to be nearly standalone, minimizing dependencies.

1.3.2 Geomtery nodes

This Blender tool leverages a node-based system for geometry manipulation, offering flexibility and efficiency. If a geometry node tree is linked to a modifier, it is referred to as a node group[3].

Within Blender, node groups are a pivotal feature of geometry nodes. They allow us to simplify node trees and hide away any reusable or complex subtrees. From a programmer's perspective, a node group can be treated similarly to a function. It encapsulates a set of nodes, allowing us to treat the entire group as a single node. This abstraction simplifies complex or reusable subtrees, enhancing readability and maintainability. Just like a function accepts input parameters and produces output, a node group can define its own input and output parameters. Node groups are not bound to a single node tree and can be reused across different node trees, promoting consistency and reducing redundancy.

Blender allows nesting of node groups within other node groups. However, node group can never contain itself, to prevent infinite recursion.

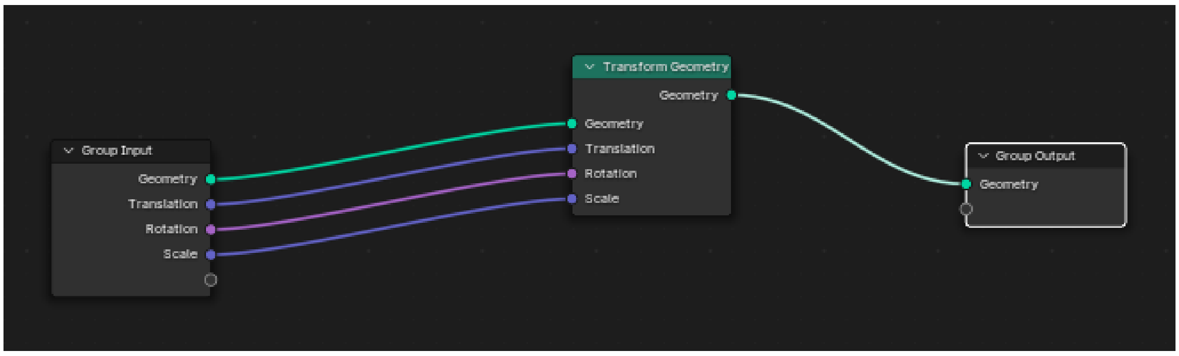

Figure 1.1: An example of a node tree with inputs including object geometry, scale matrix, rotation matrix, and translation matrix, all linked to a transform node. The node tree outputs the transformed geometry.

1.4 Textures

In the world of computer graphics, textures are essential and one of the fundamental techniques for enhancing the visual appeal of 3D objects. Textures serve as a powerful technique to represent surface properties without delving into material or geometry details. Instead of explicitly modeling every bump, scratch, or imperfection, we are able to approximate material properties by simply using an image, which will approximate a real-world material. By mapping texture onto our 3D model, we achieve a convincing visual representation.

This technique is effective because, unless we need to look at objects up close or require highly detailed realism, this approximation is sufficient. When we observe an object from a distance or in motion (such as in animations), our eyes perceive the overall impression rather than focusing on every detail. As long as the texture captures the fundamental nature of an object, our brains fill in the rest.

By leveraging textures, we avoid modeling intricate geometry and complex materials. When rendering a scene, the geometry of objects can be represented as rough polygonal models, and the material properties can be approximated using textures. This approach can significantly speed up rendering, making it possible to create visually appealing content even in real-time applications.

1.4.1 Procedural textures

There are various methods for generating textured images. Manual creation involves demanding work, where photorealistic textures can be developed by hand or by using photographs. However, each approach has its limitations. Handcrafting textures can be challenging, especially for materials that are difficult to replicate, requiring considerable artistic skill. On the other hand using a photo as a texture may introduce biases in lighting and curvature.

Imagine generating a texture that seamlessly repeats itself, like a pattern. Texture synthesis offers a solution, by creating textures from sample images to resemble the input. This method is particularly useful for generating textures with repeating patterns. One approach involves writing procedures to simulate material appearance. In procedural synthesis, one commonly employed function is known as Perlin noise[10].

Procedural textures are defined mathematically, providing precise control over their appearance. Leveraging this control, we can manipulate texture appearance using parameters provided by procedural texture.

1.4.2 Texture mapping

One of the challenges of texturing involves mapping textures to object surfaces. It's important to decide how and where we want to apply the texture and then find a way to map the texture with minimal distortion.

In 1974, Edwin Catmull made a significant contribution to computer graphics by generating the first textured images, where surfaces were represented as parametric patches. Each point on the 3D surface corresponds to a specific 2D point in parameter space, represented by the coordinates (u, v). This 2D-to-3D correspondence allows us to map a two-dimensional texture image seamlessly onto the 3D surface. By leveraging the (u, v) parameters associated with any point on the patch, we can compute the corresponding pixel location in the texture image[6].

However, there is a problem with this simple approach, as it will usually result in aliasing artifacts. Therefore, it is important to use more complex sampling and filtering methods.

1.4.3 Blender textures

Blender features a range of built-in procedural texture nodes. These nodes accept texture coordinates and various parameters as input, producing a color or value as output. Unlike traditional texture setups, no texture data-blocks are required; instead, node groups can be employed for reusing texture configurations.
Some useful Blender texture nodes are[5] :

1. **Image Texture Node** - Applies an image is as a texture.

2. **White Noise Texture Node** - Returns a random number based on an input *seed*. The *seed* can be a number, a 2D vector, a 3D vector, or a 4D vector.

3. **Noise Texture Node** - Evaluates a fractal Perlin noise at the input texture coordinates.

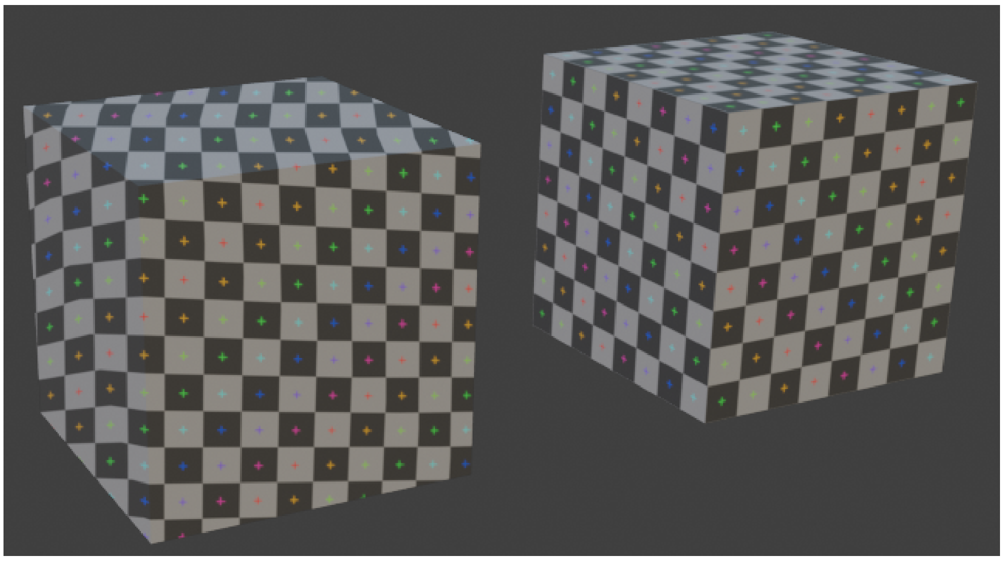

Figure 1.2: An example of UV mapping on a 3D cube in Blender. The left cube displays incorrect checkerboard mapping with distortion, while the right cube demonstrates correct checkerboard mapping.

4. **Voronoi Texture Node** - Evaluates a Worley Noise at the input texture coordinates.

Chapter 2

Concept

In this chapter we look at theoretical approach and design of our pipeline. We describe parameters of our parametric 3D models, textures and our approach of generating new 3D models.

2.1 Bags and parameters

2.1.1 Parametric three side seal pack

Package parameters:

- **X Size**: Dimension of the package in the X-axis
- **Y Size**: Dimension of the package in the Y-axis
- **Z Size**: Dimension of the package in the Z-axis
- **Left Margin Size**: Width of the left margin
- **Right Margin Size**: Width of the right margin
- **Top Margin Size**: Height of the top margin
- **Top Corners Angle**: Roundness angle of the top corners

2.1.2 Parametric doy pack

Package parameters:

- **X Size**: Dimension of the package in the X-axis
- **Y Size**: Dimension of the package in the Y-axis
- **Z Size**: Dimension of the package in the Z-axis

- **Left Margin Size**: Width of the left margin
- **Right Margin Size**: Width of the right margin
- **Lower Top Margin Size**: Height of the lower portion of the top margin
- **Seal Strap Size**: Height of the seal strap within the top margin
- **Seal Strap Protrude Size**: Width of the seal strap extending beyond the top margin
- **Tear Strap Size**: Height of the tear strap within the top margin
- **Tear Strap Protrude Size**: Width of the tear strap extending beyond the top margin
- **Tear Strap Indent Size**: Width of the indent in the tear strap of the top margin

2.1.3 Parametric pillow pack

Package parameters:

- **X Size**: Dimension of the package in the X-axis
- **Y Size**: Dimension of the package in the Y-axis
- **Z Size**: Dimension of the package in the Z-axis
- **Top Margin Size**: Height of the top margin
- **Bottom Margin Size**: Height of the bottom margin

2.1.4 Parametric four side seal pack

Package parameters:

- **X Size**: Dimension of the package in the X-axis
- **Y Size**: Dimension of the package in the Y-axis
- **Z Size**: Dimension of the package in the Z-axis
- **Left Margin Size**: Width of the left margin
- **Right Margin Size**: Width of the right margin
- **Top Margin Size**: Height of the top margin
- **Bottom Margin Size**: Height of the bottom margin

Chapter 3

Implementation

3.1 x

Conclusion

Bibliography

[1] Bmesh module. [Accessed on 27/4/2024] url:https://docs.blender.org/api/current/bmesh.html.

[2] Mathutils module. [Accessed on 27/4/2024] https://docs.blender.org/api/current/mathutils.html.

[3] Node groups. [Accessed on 26/4/2024] url:https://docs.blender.org/manual/en/latest/interface/controls/nodes/groups.html.

[4] Python console. [Accessed on 29/4/2024] url:https://docs.blender.org/manual/en/latest/editors/python_console.html.

[5] Texture nodes. [Accessed on 28/4/2024] url:https://docs.blender.org/manual/en/latest/modeling/geometry_nodes/texture/index.html.

[6] Darwyn Peachey Ken Perlin Steven Worley William R. Mark David S. Ebert, F. Kenton Musgrave and John C. Hart. *Texturing & Modeling: A Procedural Approach,Third Edition.* Morgan Kaufmann Publishers, 2003. ISBN:1–55860–848–6, url:https://flurry.dg.fmph.uniba.sk/webog/SuboryOG/bohdal/TexturingAndModelingAProceduralApproach.pdf.

[7] Fraunhofer IPA. Fraunhofer ipa bin-picking dataset. [Accessed on 30/4/2024] url:https://www.bin-picking.ai/en/dataset.html.

[8] Marco F. Huber Kilian Kleeberger, Christian Landgraf. Large-scale 6d object pose estimation dataset for industrial bin-picking. [Accessed on 29/4/2024] url:https://arxiv.org/abs/1912.12125v1.

[9] Peter Kravár and Lukáš Gajdošech. Novel synthetic data tool for data-driven cardboard box localization. [Accessed on 30/4/2024] url:https://arxiv.org/abs/2305.05215.

[10] Li-Yi Wei. A crash course on texturing. [Accessed on 29/4/2024] url:https://graphics.stanford.edu/~liyiwei/courses/Texturing/paper/paper.pdf.