# Thumbnail images for preview: Arranging 3D models for viewing

A.Z Ito-Low

Master (MComp) of computer science with honours
The University of Bath
2022-2023

# Thumbnail images for preview: Arranging 3D models for viewing

Submitted by: Alexander Ito-Low

## Copyright



## Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

**Abstract**

The automatic generation of a thumbnail image from a collection of objects is a desirable method for the visualisation and advertisement of products. However, currently there is no framework that addresses this problem directly. Previously, Yang and Huang (2013) have presented their own process, TrayGen that effectively generates tray designs from a collection of objects.

Our primary contribution is the formal definition and implementation of a framework that generates a thumbnail image such that the visibility of object features is maximised. To achieve this, we base our approach on Yang and Huang's framework where we formulate guidelines for the view selection and the arrangement of objects that are specific to our problem. We then look at object arrangement as an optimisation problem.

We propose that a Poisson disk sampling method is an applicable approach to arrange objects in this setting. Through the qualitative evaluation of three data sets containing objects from different categories, we demonstrate that our framework is capable of generating images that exhibit object features. We also find that the Poisson disk-based approach provides desirable characteristics for object placement when applied in this setting.

**Keywords:** visualisation, viewpoint selection, object arrangement, optimisation

# Contents


**1 Introduction** **1**

**2 Literature and Technology Survey** **4**
2.1 The packing problem . . . . . 4
2.1.1 Optimisation . . . . . 4
2.1.2 Geometric representation . . . . . 5
2.2 Instances of the packing problem . . . . . 5
2.2.1 Packing in logistics . . . . . 5
2.2.2 Layout synthesis of interior scenes . . . . . 6
2.3 The viewpoint selection problem . . . . . 8

**3 TrayGen approach to packaging** **11**

**4 Problem overview** **14**
4.1 Implementation considerations . . . . . 15

**5 View selection and arrangement** **17**
5.1 Best view selection . . . . . 17
5.1.1 Camera calibration and mesh alignment . . . . . 17
5.1.2 Object viewpoint selection . . . . . 19
5.2 Object arrangement . . . . . 21
5.2.1 Arrangement guidelines . . . . . 21
5.2.2 Optimisation to arrange objects . . . . . 21
5.2.3 Poisson disk sampling for object arrangement . . . . . 24
5.3 Chapter summary . . . . . 25

**6 Results** **26**
6.1 Rationale . . . . . 26
6.2 Experiment setup . . . . . 27
6.3 Analysis of the results . . . . . 27
6.3.1 Performance . . . . . 27
6.3.2 Quality . . . . . 28
6.4 Discussion . . . . . 32

**7 Conclusions** **33**
7.1 Contributions . . . . . 33
7.2 Limitations and future work . . . . . 33
7.3 Final remarks . . . . . 34

**Bibliography** **35**

**A Resulting images for arrangement approaches** **39**
A.1 Full sized figures for best arrangements . . . . . 39

**B Downloadable code** **42**
B.1 File: main.cpp . . . . . 43
B.2 File: mesh.h . . . . . 44
B.3 File: mesh.cpp . . . . . 45
B.4 File: scene.h . . . . . 47
B.5 File: scene.cpp . . . . . 48

# List of Figures


1.1 Overview of problem addressed in this dissertation. Given multiple unorganised objects (left), our framework aims to arrange them in a visually appealing manner in the form of a thumbnail image (right). . . . . 1
1.2 Example of packing problem in the logistics domain. In this problem, objects are placed in a container with the goal of minimising space (Huang et al., 2022). . . . . 2
1.3 Pipeline for the packing problem in interior design. In this problem, the inputs are usually constraints and rules surrounding object relationships (Zhang et al., 2019). . . . . 2

2.1 Results of arrangement algorithms to generate desk arrangements. Position based dynamics (left) (Weiss et al., 2018). Simulated annealing (right) (Liu et al., 2021). . . . . 6
2.2 Interior synthesis can also be applied to generate floor plans (Wu et al., 2019). . . . . 7
2.3 Different viewpoints of the Stanford Bunny (Turk and Levoy, 1994). . . . 8
2.4 Automatic arrangement applied to city landscape generation using the genetic algorithm, as described in (Zhang and Yang, 2022). . . . . 9
2.5 Automatic arrangement to generate bas-relief layouts, as described in (Mao et al., 2021). . . . . 9
2.6 Automatic arrangement to generate tray layouts as described in (Yang and Huang, 2013). . . . . 10

3.1 Overview of TrayGen framework (Yang and Huang, 2013). . . . . 11
3.2 Aligned contours (left). Resulting contours (right) (Yang and Huang, 2013). 11
3.3 Objective terms impact the optimisation result (Yang and Huang, 2013). 12

4.1 UML diagram of class structure. . . . . 15

5.1 Default camera settings. . . . . 17
5.2 Centered camera settings. The whole of the book is now visible. . . . . 18
5.3 Object alignment and scaling. . . . . 19
5.4 Selected viewpoints of the objects. . . . . 20
5.5 Visibility ratios for different viewpoints of the guitar mesh. . . . . 20
5.6 Overlapping area (blue region) of two AABBs with equations. . . . . 22
5.7 Trade-off of different terms on the optimisation result. . . . . 23
5.8 Poisson sampling on a 2D grid (left). Averaged distribution over multiple runs (right) (Bridson, 2007). . . . . 24

6.1 Graphical interpretation of the timings of arrangement approaches. . . . 28
6.2 Initial layouts of the data sets after best view selection. . . . . 29
6.3 Layout results for each data set using the different arrangement approaches. 29
6.4 Average number of occlusions for each arrangement approach. . . . . 30
6.5 Optimisation curves for the best and worst results of each data set, with minimum objective values in brackets. . . . . 31

A.1 Layout results for each data set using the optimisation approach. . . . . 40
A.2 Layout results for each data set using the Poisson disk sampling approach. 41

# List of Tables

6.1 Performance statistics with inclusion of occlusion calculation. . . . . 28
6.2 Timings of arrangement approaches. . . . . 28

# Acknowledgements

I am thankful to my supervisor, Dr. Yongliang Yang for his support throughout the year with this project. His guidance about how to approach this project helped immensely.

# Chapter 1

# Introduction

How products are advertised has an influence on customer purchasing behaviour (Adhikary, 2014). Visual merchandising in particular is important, the displaying of objects in a way that catches the customer's attention (Prasad and Vetrivel, 2016). Visual merchandising is applicable to both real store environments where the design of the store front is important and online store environments where there is an emphasis on the generation of visually pleasing images available to the consumer for online preview. It is therefore important to generate displays of objects in a way that appeals to the customer.

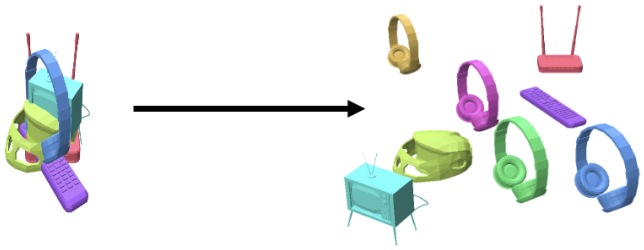


Figure 1.1: Overview of problem addressed in this dissertation. Given multiple unorganised objects (left), our framework aims to arrange them in a visually appealing manner in the form of a thumbnail image (right).

In this dissertation we focus on the generation of a thumbnail image of objects placed in a way that allows the viewer to easily receive visual cues about product functionality. For example, given multiple 3-dimensional objects (see Figure 1.1), how do we generate an image that arranges them in a way that provides the viewer with a representative view of those objects. While it is possible to manually achieve this, it can be time-consuming to explore the possible use of space to find the optimum solution. Therefore, an investigation into finding a computerised method to solve this problem could be desirable.

While this problem shares similarities with packing problems in logistics and interior design, its approach differs. In the logistics domain (see Figure 1.2), the goal of object arrangement is usually to minimise the container size to reduce costs. In interior design (see Figure 1.3), although it focuses more on object display, there are more constraints present on how objects should be placed with respect to each other. Our problem places an emphasis on the exhibition of objects.

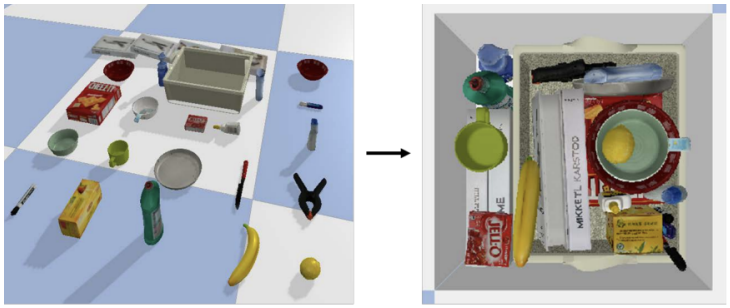



Figure 1.2: Example of packing problem in the logistics domain. In this problem, objects are placed in a container with the goal of minimising space (Huang et al., 2022).

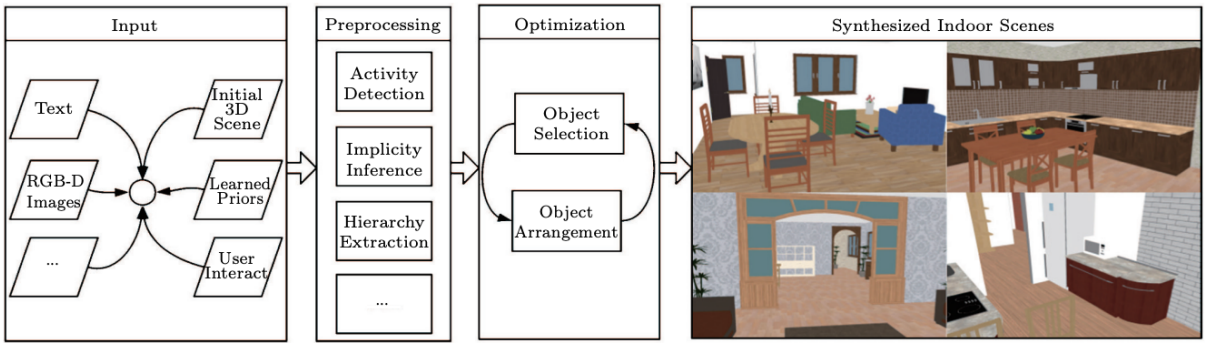



Figure 1.3: Pipeline for the packing problem in interior design. In this problem, the inputs are usually constraints and rules surrounding object relationships (Zhang et al., 2019).

Yang and Huang (2013) investigated the effective packing of objects for display in which they propose design guidelines about object placement based on shape perception and visual merchandising. This approach led to a computational framework that could automate designs for tray generation. Similarly, other works, for example Mao et al. (2021), and Lienhard et al. (2014) also place an emphasis on viewpoint selection and base their approaches purely on the geometric properties of objects. However, these differ to our research because they do not consider a framework that can generate a thumbnail image from the arrangement of objects in a 3D space.

Poisson disk sampling is a technique which has been researched as an approach to reduce aliasing in ray-tracing (Cook, 1986). Kopf et al. (2006) have suggested that Poisson disk sampling can be applied to object placement scenarios where there is a need to avoid object collisions. However, little is known about the effectiveness of this approach when applied in our problem domain.

This dissertation therefore makes two contributions. Our primary contribution is to build upon the work of Yang and Huang (2013) (Chapter 3), making an extension to their framework TrayGen such that a method to generate a representative thumbnail image is formally defined (Chapter 4) and implemented (Chapter 5). As our main research agenda lies in the construction of the overall pipeline, in our implementation we use a simpler viewpoint selection algorithm - the visibility ratio - to find the best viewpoint of the objects. Our research mainly lies in the arrangement phase where we propose adapting Yang and Huang's arrangement approach (extended pattern matching) to suit our problem. We also propose an arrangement approach that is based on Poisson disk sampling.

Our secondary contribution is to investigate the effectiveness of our framework (Chapter 6). We suggest that both of the arrangement algorithms have the ability to produce good thumbnail images. To compare the two arrangement approaches against each other and to determine their effectiveness in thumbnail image generation we test our framework on three different object data sets from different categories. The results suggest that both of the arrangement algorithms have the ability to produce quality layouts. The Poisson disk sampling approach is suggested to generate layouts with fewer occlusions and is faster when compared to the extended pattern matching approach.

# Chapter 2

# Literature and Technology Survey

We introduce the research areas related to our problem, namely packing problems and viewpoint selection. These areas help provide context to our research and provide the reader with a background so they can better understand where our research is positioned.

## 2.1 The packing problem

The placement of objects is a well-studied problem in operational research and in interior layout design. Also referred to as component layout, spatial arrangement, and the packaging problem, it focuses on the exploration of the optimal placement of objects subject to a set of constraints (Cagan, Shimada and Yin, 2002). A general approach to the packing problem involves the definitions of a geometric representation for objects, a geometric representation for the container space, an objective function for the evaluation of object placement, and a set of constraints. An understanding of optimisation and the geometric representation of objects is necessary to enrich our knowledge about the packing problem. We briefly introduce these concepts and then study packing problems with respect to different domains to gain an understanding of where our work is positioned in relation to current research.

### 2.1.1 Optimisation

Optimisation is defined as the search for an optimal solution amongst all of the possible solutions (Velho et al., 2011). This optimal solution may either be a minimum or a maximum. The criteria that we wish to optimise is encoded into a function called the objective function. The objective function whose optimum is a minimum is formulated as a function $f: S \rightarrow \mathbb{R}$ that is defined as:

$$min_{x \in S} f(x) \tag{2.1}$$

where $S$ is the solution space.

The solution is said to be the global optimum where it is the best possible solution in the solution space and is a local optimum where it is the best solution with respect to a subset of solutions. In the packing problem, an observation is that the environmental

constraints and desirable properties of object placement are usually encoded into the objective function. Another observation is that the solution space will be finite when the container space is finite and infinite otherwise. The consideration of local and global optima will influence the run time of the search for object placement. We will later see some examples of packing optimisation implementations (see section 2.2.2).

### 2.1.2 Geometric representation

A common geometric representation of models is the boundary representation: models represented as a collection of vertices, edges and faces (Cagan, Shimada and Yin, 2002). This provides a simple description of models and is used widely. In the packing problem, a common constraint is that objects should not collide with one another. Collision detection is computationally expensive in practice, therefore more complex structures for models are needed (Lin and Gottschalk, 1998).

A widely researched structure in literature is the hierarchical spatial data structure that recursively partitions a data space such that data is sorted spatially (Samet, 1984). As a result, this reduces the computational time of a search because only part of the data needs to be searched. For example, an octree is the representation of the 3D scene as an axis-aligned bounding box that is recursively divided eight times at the center of each box until there is only one object per bounding box (Samet, 1984; Akenine-Mo et al., 2018).

Another representation is an OBB tree that tightly fits a collection of polygons in a scene with a bounding box that has an arbitrary orientation then groups the collection objects into a tree hierarchy (Gottschalk, Lin and Manocha, 1996). This tree hierarchy will have less nodes than octrees due to the tighter packing of objects and subsequently leads to a faster search (Cagan, Shimada and Yin, 2002). Therefore, in this research context, we may want to consider collision detection algorithms that make use of hierarchical spatial structures or those that use bounding volumes.

## 2.2 Instances of the packing problem

### 2.2.1 Packing in logistics

In operations research, the packing problem aims to determine an optimal assignment of objects to bins (Ali et al., 2022). This optimal assignment is evaluated against a function that includes loading and transportation criteria. The problem is divided into two sub-problems: offline, where information about objects to be packed is provided beforehand; and online where objects are expected to be packed in real time. Recent research has focused on the online problem. Hu et al. (2020) investigated the real time packing of objects, while Huang et al. (2022) study the packing of irregular objects. Both approaches rely on the use of reinforcement learning to optimise box space utilisation and stability of the packed objects. Research in this domain focuses on logistic constraints as opposed to how objects are visually presented in a scene.

### 2.2.2 Layout synthesis of interior scenes

The synthesis of interior scenes has been a popular research topic in recent years. The objective of interior scene synthesis is the automatic generation of plausible scenes (Zhang et al., 2019). Compared to bin packing where spatial relationships between objects are rarely considered, the consideration of object relationships is central to the synthesis of interior scenes. Constraints are based around how objects are placed in the scene with respect to each other.

Weiss et al. (2018) implemented a physics-based continuous layout relying on position based dynamics to arrange objects. In this approach, objects are treated as particles and their positions and orientations are updated after each iteration to satisfy object constraints. Similarly, Zhang et al. (2021a) optimised object layout using position based dynamics. Both of these approaches use gradient based methods to arrange objects. In contrast, Liu et al. (2021) apply simulated annealing to optimise desk layout. Despite the speed of gradient-based methods, they do not consider the whole search space as the optimal layout is dependent on the starting position of objects (Cagan, Shimada and Yin, 2002). The results from Weiss et al. (2018) and Liu et al. (2021) for desk layouts are shown in Figure 2.1. Although simulated annealing can escape local optima, computation may be more expensive as the complexity of the objective function increases (Cagan, Shimada and Yin, 2002). Thus these optimisation approaches may not be appropriate for our layout problem.

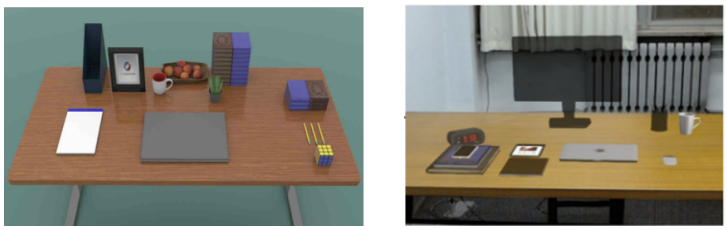


Figure 2.1: Results of arrangement algorithms to generate desk arrangements. Position based dynamics (left) (Weiss et al., 2018). Simulated annealing (right) (Liu et al., 2021).

Data driven and deep learning approaches have become more common in interior layout synthesis. Wu et al. (2019) trained a series of convolutional neural networks on annotated floor plans to predict the placement of walls and rooms given a boundary of a floor plan (see Figure 2.2). Li et al. (2019) also use a neural network architecture but they use a recursive network structure to accommodate for learning the underlying structure of a 3D scene more accurately. Meanwhile, Zhang et al. (2021b) learn contextual priors based off of inter-object relationships that are dependent on the position of the cursor specified by the user. Zhang, Xie and Zhang (2021) generate a graphical structure that encapsulates object-to-object relationships through the use of instance-based priors that are used to sort objects into coherent groups and then a heuristic rule-based approach is used to arrange objects.

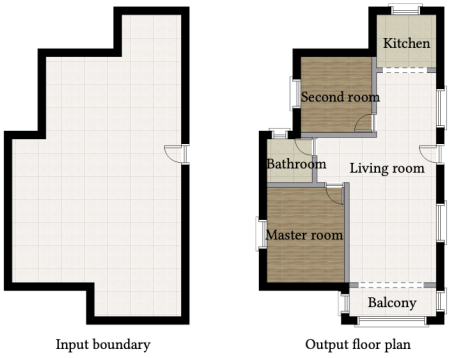



Figure 2.2: Interior synthesis can also be applied to generate floor plans (Wu et al., 2019).

From the study of data-driven and deep learning approaches, we find that these methods try to infer the spatial relationship between objects through the collection of lots of data. While data-driven and deep learning approaches have the ability to create more complex scenes for object display, we limit the scope of our research to focus primarily on the organisation of objects simply from their geometric features (Yang and Huang, 2013).

Interior scene synthesis also includes other methods that are not reliant on lots of data and consider a wider search space when optimising object layout. For example, Xie, Xu and Wang (2013) reshuffle furniture from a ground truth scene to create new scenes. The metric to measure scene quality is based on the distribution of the original scene with respect to an object. Although inter-object relations are still considered, data collection is not expensive when compared to data-driven and deep learning approaches.

Kán and Kaufmann (2018) used greedy cost minimisation to arrange furniture in an interior scene. The objective function encodes aesthetic and functional constraints corresponding to specific object categories. In the greedy cost minimisation approach it is able to search a wider search space because a set of transformations are applied to objects with a probability. For the arrangement of objects in our research, we are interested in an algorithm that can search a wide search space and an objective function that considers desirable object placement traits and placement constraints.

The study of scene synthesis has revealed desirable characteristics of the methods to optimise object placement. We desire an objective function that encodes desirable characteristics of the scene with respect to objects. For the exploration of the search space, we desire an algorithm with good search space coverage. In all of the implementations studied, they either require labelled data sets to encode constraints or they create spatial relationships through the construction of a structured representation of the scene. Similar to Yang and Huang (2013) our method does not require the use of annotated data sets to organise objects. Instead, we place a stronger emphasis on how well objects are exhibited in a scene (Yang and Huang, 2013). We therefore survey objective measures that quantify how well objects are exhibited in a scene and study cases of where object exhibition is applied.

## 2.3 The viewpoint selection problem

Figure 2.3: Different viewpoints of the Stanford Bunny (Turk and Levoy, 1994).

Viewpoint selection is the problem of selecting a good canonical view of an object in a scene (Blanz, Tarr and Bülthoff, 1999). Despite the bias associated with a good viewpoint, Blanz, Tarr and Bülthoff (1999) suggested that the geometry of an object contributes towards the preference of its viewpoint. In particular: the visibility of an object's features (see Figure 2.3) and the stability of its view with respect to small transformations. Research in viewpoint selection has yielded different measures for evaluating a view of an object. We briefly survey some of these techniques.

Vázquez et al. (2001) defined a measure called viewpoint entropy using information theory as its basis. This technique used the probability distribution of the relative area of the object's projected faces over a sphere of directions centered at the viewpoint to determine the amount of information captured. Gao, Wang and Han (2016) approximated the viewpoint quality using the observation that important object features can be viewed from the six principal viewing directions of the object's bounding box. This method approximates the object's visible faces and its projected area in each of its principal viewing directions and takes the weighted sum of these quantities to evaluate the viewpoint. These methods purely rely on the object's geometry. Therefore, they tend to be computationally efficient but are not fully representative of visual perception (Zhang and Fei, 2019).

Another class of techniques for viewpoint selection are those that are dependent on the visual attributes of objects. These methods often efficiently capture visual aspects of the object but ignore the geometric properties of the object as a whole (Zhang and Fei, 2019). Lee, Varshney and Jacobs (2005) measured viewpoint goodness though mesh saliency. Mesh saliency measures the regional importance of a 3D mesh by considering its curvature at different scales. This method, however, tends to be expensive in practice.

Plemenos, Sbert and Feixas (2004) introduced viewpoint complexity which is calculated using the sum of the number of visible polygons in the scene and the total visible area of the projected scene. In practice, a z-buffer is used to assign every surface with a distinct colour to approximate viewpoint complexity.

To obtain a more accurate model of the human visual system, researchers have resorted to the modelling of a metric that combines measures. Polonsky et al. (2005) proposed a set of guidelines referred to view descriptors that include geometric complexity of a 3D shape, descriptors based on view dependent features, and descriptors that assign values to regions with semantic meaning. Secord et al. (2011) created models that combined view attributes into a measure of viewpoint fitness. In this research, 14 view attributes were evaluated in a user study. Responses from the study were used to model the importance of each of the attributes. Secord et al. (2011) defined linear models of goodness which

combined k attributes into a single model. The results from this research suggest that surface visibility is the most important view attribute and it is recommended to use the single attribute, linear-3, and linear-5 models in practice.

Kucerova, Varhanikova and Cernekova (2012) investigated a geometric method, a visual attention method and an entropy-based method to determine which method could accurately predict good viewpoints. Results from this user study are aligned with Polonsky et al. (2005) and Secord et al. (2011) because they suggest that a combination of metrics that capture different object attributes are able to model the human visual system. The study of combined measures reveals that although there is no optimal solution for the viewpoint problem, a representative metric provides a reasonable approximation for finding good views. Interested readers can refer to Secord et al. (2011) and Zhang and Fei (2019) for a more comprehensive discussion of viewpoint selection methods.

Figure 2.4: Automatic arrangement applied to city landscape generation using the genetic algorithm, as described in (Zhang and Yang, 2022).

There are a few cases of where viewpoint selection has been used to display objects. Lienhard et al. (2014) used a weighted scheme to compute geometry based, aesthetic, and semantic view attributes for a model to determine its best viewing direction. Zhang and Yang (2022) proposed a novel 3D layout optimisation method that combined scene information with a genetic algorithm (see Figure 2.4). In both of these instances, the inclusion of viewpoint evaluation has led to results that are representative and that are able to account for human visual preference. These cases differ to our research because we consider the exhibition of multiple objects in the product domain as opposed to the architectural domain.

Figure 2.5: Automatic arrangement to generate bas-relief layouts, as described in (Mao et al., 2021).

Mao et al. (2021) proposed an automatic arrangement for bas-relief layout that introduced a set of evaluation indicators to account for the aesthetic aspect of object layout. Their visibility criteria can account for an individual model and multiple models. Their results are presented in Figure 2.5. The main difference between this and our research is that we consider how objects are visualised in a 3D space instead of a relief area.

Our research is based upon the work of Yang and Huang (2013). They introduced a framework that generates tray designs for objects such that the designs are compact and can exhibit object features to the consumer (see Figure 2.6). Our research extends this work by applying perception-based arrangement guidelines to generate a thumbnail image that displays 3D objects.

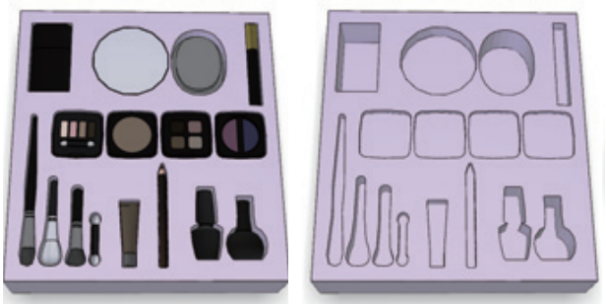


Figure 2.6: Automatic arrangement to generate tray layouts as described in (Yang and Huang, 2013).

# Chapter 3

# TrayGen approach to packaging

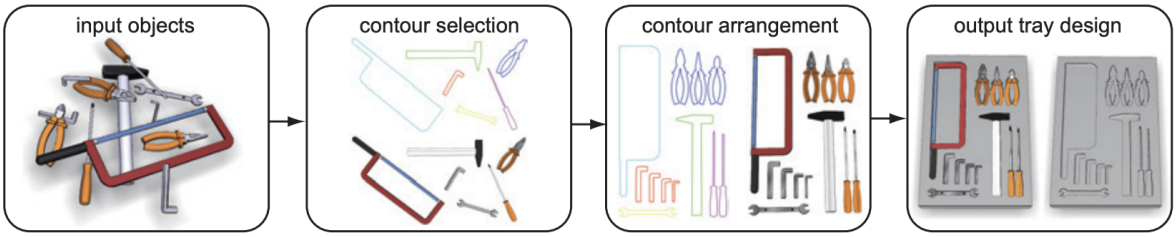



Figure 3.1: Overview of TrayGen framework (Yang and Huang, 2013).

Having established the context of our research, we now review Yang and Huang (2013) to understand perception-based arrangement guidelines for objects in the context of tray designs. In doing so, we can adapt this approach to generate a thumbnail image that provides a representative view of 3D objects.

Yang and Huang (2013) re-interpret the problem of tray design as the problem of computing the 2D projected contour of each object such that contours are designed to retain the meaning of their object and the arrangement of these contours should reflect their interconnections. As shown in Figure 3.1, a natural methodology that follows from this problem statement is a framework that contains two steps: Contour selection and contour arrangement.

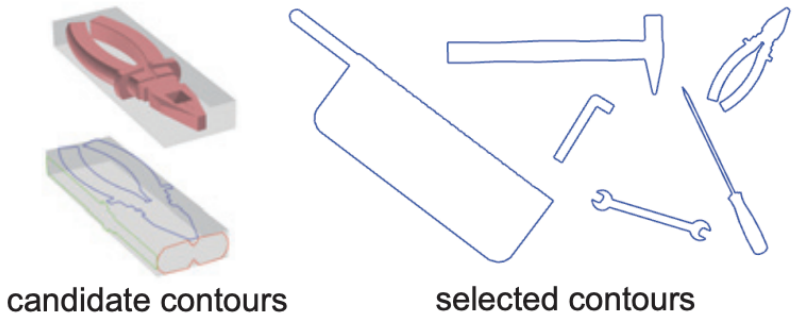



Figure 3.2: Aligned contours (left). Resulting contours (right) (Yang and Huang, 2013).

Contour selection consists of clustering, alignment and distinctive contour selection. Clustering involves grouping objects into groups according to their global geometric features. Once objects are clustered into groups, the alignment phase ensures that objects

in the same group are aligned consistently. Distinctive contour selection formulates viewpoint selection as an optimisation problem that maximises the saliency of a group and the view dependence dissimilarity. The resulting viewpoint of a group is one that highlights the differences between groups and ensures that salient features of a group are exposed. Figure 3.2 shows the output of the contour selection step.

The contour arrangement phase involves the alignment of contours with respect to the principal axes. Objects are then arranged regularly within their groups and an optimisation procedure is initiated to arrange all objects in the $xy$-plane. We elaborate on this optimisation procedure. To optimise the arrangement of all objects in the $xy$-plane, an equation that encapsulates the objectives for arrangement optimisation is proposed:

$$f(L) = f_{cp} + w_{cr} f_{cr} + w_{rp} f_{rp} + w_{cl} f_{cl} \tag{3.1}$$

where the values of the weights represent the trade-offs between the different optimisation terms (see Figure 3.3). The objectives for arrangement optimisation are the compactness of the object arrangement ($f_{cp}$), the total overlapping area between objects from different groups ($f_{cr}$), the relative position term ($f_{rp}$) that accounts for a group that can be assembled into a single object, and the closeness term ($f_{cl}$) that measures the squared distance between groups of objects that are explicitly marked to be close to one-another.

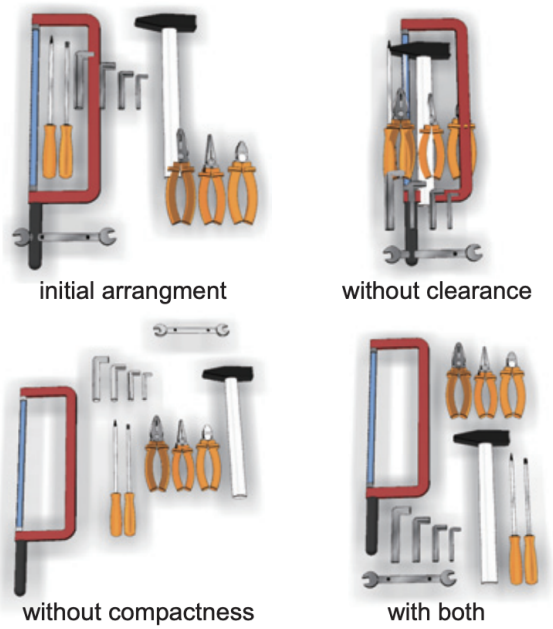



Figure 3.3: Objective terms impact the optimisation result (Yang and Huang, 2013).

The optimisation procedure starts with an initial placement of objects and the objective function is evaluated. A new layout is then generated by the translation of a randomly selected group with respect to a Gaussian term, a rotation of a randomly selected groups, and the swapping of two randomly selected groups. If the objective function decreases, this new layout is accepted, otherwise objects are returned to their previous layout at the previous time step. This process continues until a certain number of iterations is met or if the objective function does not decrease for a fixed number of iterations.

Their results indicate that TrayGen is able to generate plausible tray designs that highlight more object inter-relations when compared to manual creation and TrayGen is able to exhibit objects in an organised way. We therefore base our approach of arranging objects in a thumbnail image on this framework and adapt the details of this approach to suit our requirements.

# Chapter 4

# Problem overview

Yang and Huang (2013) have indicated that future work can utilise their perception-driven guidelines to visualise 3D model collections. Accordingly, we extend and adapt their framework.

More formally, in this dissertation we consider the problem of arranging objects in a 3D space such that a representative thumbnail image of the placed objects is generated. Specifically, we are interested in generating layouts such that the visibility of object features are maximised.

Our devised framework takes as input a set $S = \{S_1, S_2, \dots, S_n\}$ comprising of objects represented as triangle meshes in their natural orientation and explores individual object features and the layout of these features to derive a representative thumbnail image. The exploration of object features and layouts is based on Yang and Huang's perception-driven guidelines. Our framework is comprised of two stages:

1. **Best view selection**. This stage consists of the alignment and scaling of objects such that they are located in the same position in the world coordinates system and are at a consistent scale. Then the best viewpoint for each object is optimised through the use of surface visibility (Polonsky, 2005).

2. **Object arrangement**. What differentiates our framework from Yang and Huang's is the arrangement phase. Their framework deals with the arrangement of contours in the *xy*-plane, while our framework arranges 3D objects with respect to the 3D coordinates space. Our research therefore mainly focuses on this step.

   The output of the best view selection phase serves as the initialisation of the object arrangement. An optimisation procedure is then carried out on objects to create scenes where the visibility of object features are maximised.

   We have devised two methods of arranging objects: One uses extended pattern search introduced by Yin and Cagan (2000) and is derived from Yang and Huang (2013), while the other approach places objects based on a Poisson disk sampling scheme (Cook, 1986).

## 4.1 Implementation considerations

We choose C++ as the programming language to implement the proposed framework. This is due to its ease of integration with popular graphics libraries for the visualisation and processing of meshes.

Polyscope is chosen as the library to visualise meshes because of its simplicity and flexibility (Sharp et al., 2019). It works well in cases where the scene is dynamically updated because registered meshes can be overwritten. The programmer is also able to adjust some of the visualisation parameters. For example, the programmer is able to remove the ground plane of the scene and adjust the camera. libigl was used for reading the triangular meshes from a file format and storing mesh data (Jacobson et al., 2013). It provides many algorithms for geometric processing that can be used if necessary, however, libigl was only used to read triangular meshes. This method supports obj, off, stl, wrl, ply, mesh file formats.

For the geometric transformations of meshes, we use Eigen (Guennebaud, Jacob et al., 2010). Eigen is a C++ template for linear algebra that supports matrix and vector operations. libigl and Polyscope are both compatible with Eigen which makes it suitable. It is worth noting that camera calibration in Polyscope uses glm. A workaround used is to have Eigen to calculate vectors of interest and then initialise glm vectors to make use of these calculated vector components. Our implementation makes use of a mesh class and a scene class. The mesh class stores data attributes related to a single mesh and methods that transform the mesh in a scene.

While Polyscope does provide transformations for objects, we prefer to define a mesh class that is not tied to Polyscope such that there is a clear separation between the viewer and the meshes. It also allows us more freedom to customise the mesh class and to store mesh attributes.


Figure 4.1: UML diagram of class structure.

The scene class is responsible for reading in the meshes, visualising the meshes, and performing our framework on the meshes to generate a representative thumbnail image. A simplified UML diagram of our class structure is given in Figure 4.1. The UML diagram is not an exhaustive list of all of our methods as this dissertation comes with a fully functional implementation of our framework. Instead, we intend to provide the reader with a high-level understanding of our implementation such that those interested can re-create and build upon our work. We provide high-level functional requirements for the proposed implementation:

1. The implementation must use a programming language that offers easy integration with computer graphics libraries.
2. The implementation must separate the functionality of the scene and viewer from the mesh.
3. The scene class must be able to read in triangle mesh files.
4. The mesh class must be instantiated by the name of the mesh file.
5. The mesh class must store its geometric representation.
6. The mesh class must include methods to transform it.
7. The scene class must include a method for view selection.
8. The scene class must include a method for object arrangement.
9. The scene class could include alternative methods for view selection.
10. The scene class must output the final images of the meshes after object arrangement has been applied to the mesh.
11. The scene class could include testing methods for the verification and reporting of results.

Having reviewed our problem and discussed our implementation decisions, the subsequent chapter will explain our framework in greater detail.

# Chapter 5

# View selection and arrangement

## 5.1 Best view selection

Best view selection begins by loading the 3D meshes into the scene. In our implementation, we create a .txt file containing the file paths of the meshes. These meshes are initialized and are appended to an array. The array that holds the meshes is a class member of the scene which allows the scene to have access to all of the meshes. The viewer is initialised and its settings are tuned. We remove the ground plane and set the dimensions of the viewer to 256 x 256 pixels. To test our program, we visualise the scene by registering the array of meshes and taking a screenshot. An example is given in Figure 5.1. From the screenshot, we can see the default Polyscope camera settings do not adequately capture the rendered view. We are therefore required to calibrate the camera.

Figure 5.1: Default camera settings.

### 5.1.1 Camera calibration and mesh alignment

The approach used to calibrate the camera is based on the calculation of the scene's axis-aligned bounding box (AABB) that encloses all of the objects. Although Polyscope's default camera calibration works in a similar way, through manual calibration we are able to have access to the scene's AABB and have a finer control over the camera controls.

The computation of the scene AABB involves iterating through individual objects and comparing their AABB dimensions to the current AABB of the scene. Specifically, the

maximum corner of the scene's AABB is compared to the object's. If the object's maximum corner is greater than that of the scene, the scene's new maximum corner is updated to the value of the object's maximum corner. A similar comparison is applied to the scene's minimum corner, using the object's minimum corner and with the update condition being less than as opposed to greater than. The implementation is coded such that the scene's AABB will be set to the first object iterated and is then subsequently updated as needed as it iterates.

Once the computation of the scene's AABB is completed, the center of this AABB is calculated. To calibrate the camera, we firstly set the camera's target vector to look at the calculated center. Next, we set the camera's location with respect to this center. The camera's $x$-component is set to the $x$-component of the center, while the $y$ and $z$ components are altered. The $y$-component is set to the height of the scene's AABB. As for the $z$-component, we set it to the $z$-component of the center subtracted from a factor of the distance between one of the corners of the scene's AABB and the center. Code for the calibration of the camera is given in Listing 5.1 and the resulting visualisation for the earlier example is given in Figure 5.2.

Listing 5.1: Calibration of the camera

```
void Scene::centerCamera(vector<Mesh>meshes){
    compute_scene_bounding_box(meshes);
    Vector3d center = (MinCorner + MaxCorner)/2.0;
    double radius = (MaxCorner - center).norm();

    // Set camera location and target based on the center and radius
    glm::vec3 camera_location(center.x(),(MaxCorner[1]
       -MinCorner[1])+radius,center.z()-2.5*radius);

    glm::vec3 target(center.x(),center.y(),center.z());
    polyscope::view::lookAt(camera_location,target);
    return;
}
```

Figure 5.2: Centered camera settings. The whole of the book is now visible.

The next step is to align the objects in the primary axes and scale them to ensure that they are in the same world coordinates location as each other and the relative sizes of

the objects are reflected. Objects are scaled with respect to the distance between the maximum corner and minimum corner of the AABB. The objects are then aligned by subtracting all of the object's vertices from the minimum corner of the objects AABB. Figure 5.3 shows how the objects in our example are now aligned with each other.

Figure 5.3: Object alignment and scaling.

### 5.1.2 Object viewpoint selection

We need to specify some guidelines as to what constitutes a good viewpoint on the objects in the scene. Our problem requires object features to be maximised. We therefore want to maximise the visibility ratio of objects in the scene. Unlike Yang and Huang (2013), we do not consider similarity and saliency because we are more concerned with the construction of the overall pipeline that can generate representative thumbnail images. Future work can extend this pipeline to include more complicated viewpoint guidelines.

**Visibility ratio:** Polonsky et al. (2005) defines the visibility ratio as the ratio between the visible 3D surface area of an object and its total 3D surface area. In following this guideline, one can maximise the visible region of an object which ensures that it is well displayed and identifiable. According to Secord et al. (2011) the visibility ratio is suggested to be the most important view attribute when compared to 13 other view attributes and it is recommended to use this metric to approximate viewpoint fitness in practice.

Our implementation approximates the visibility ratio of an object using its image representation. To calculate the visibility ratio of an object, it is firstly rotated around the $y$-axis with respect to its center by multiples of 30°starting from 0°. The object is then registered into the Polyscope viewer at each point with a screenshot taken. Polyscope assigns a default colour to all registered objects. As a result, the visible surface area of the object is approximated as the total number of coloured pixels in the screenshot. The resulting visible surface area of the object and its corresponding rotation angle are stored and the process is repeated until the object has completed a rotation of 360°. The visibility ratios for all of the candidate rotation angles are then evaluated.

The visibility ratio of an object at a candidate rotation angle is approximated as its surface visibility divided by the sum of its surface visibility from all of its candidate rotation angles. The rotation angle with the highest visibility ratio is then chosen as the best viewpoint of the object and the object is then rotated by this angle.

In our implementation, once the object is rotated to its best viewpoint, the object's bounding box is recomputed and the object is re-aligned because its rotation may have misaligned it. The object is then removed from the viewer and the visibility ratio is then computed for the rest of the objects. Figure 5.4 shows the found viewpoints of the example objects.

Figure 5.4: Selected viewpoints of the objects.

Figure 5.5 shows an evaluation plot of the visibility ratio of a guitar mesh. There are distinct differences in the visibility ratio between the rotation angles. Rotation angles with low occlusions are shown to correspond to higher visibility ratios which gives an indication that the implementation for finding the best viewpoint of meshes works as expected.

One criticism of the implemented algorithm is that although it is able to maximise the surface area of an object, it is unable to capture more intricate details of the object. For example, the guitar is rendered facing backwards with its cords away from the camera.

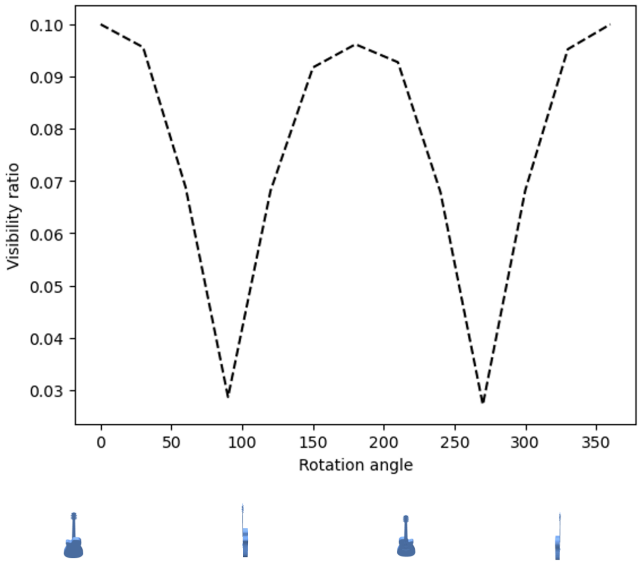



Figure 5.5: Visibility ratios for different viewpoints of the guitar mesh.

## 5.2 Object arrangement

Following from the selection of good viewpoints for objects in the scene, this section will describe our algorithms used to arrange the objects in the $xz$-plane. We discuss an optimisation approach to arranging objects which is based on that of Yang and Huang (2013) and then discuss an approach based on Poisson disk sampling.

### 5.2.1 Arrangement guidelines

We first follow Yang and Huang's guidelines for object arrangement, namely, compactness and clearance.

**Compactness:** Objects should have minimal large gaps between them to save space. This will create a more cohesive thumbnail image.

**Clearance:** Objects should avoid intersections with one-another. We come to the same conclusion as Yang and Huang in that the addition of a padding between objects creates less cluttered layouts.

We also introduce a new guideline that is specific to our problem.

**Occlusion:** Objects should avoid covering one another. This encourages a representative view of all of the objects in the scene such that the objects are well displayed.

### 5.2.2 Optimisation to arrange objects

We approach the arrangement of objects using an optimisation procedure derived from Yang and Huang (2013). We firstly describe the individual components of the objective function that measure the quality of an arrangement. We then describe the optimisation procedure. The notation used is the same as Yang and Huang (2013) to remain consistent and to acknowledge that this approach is derived from their research.

#### Objective function terms for arrangement optimisation

**Compactness.** The term $f_{cp}$ measures how tightly packed objects are in the scene:

$$f_{cp} = \frac{V_{bb}}{\sum_{i=1}^{N} V_i} \tag{5.1}$$

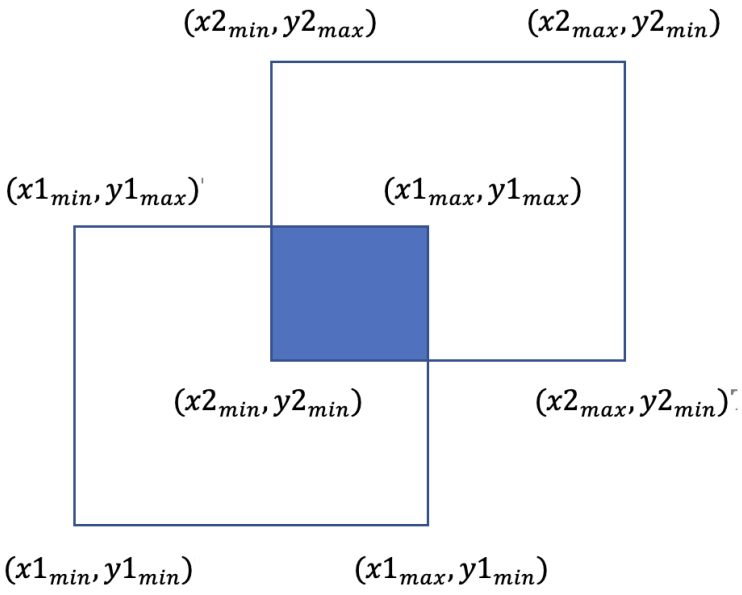
where $V_{bb}$ is the volume AABB that encloses all of the objects in the scene and $V_i$ is the volume of the AABB of object $S_i$.

**Clearance.** The term $f_{cr}$ denotes the sum of overlapping volumes between pairs of objects:

$$f_{cr} = \sum_{S_i, S_j} V_i \cap V_j \tag{5.2}$$

where $S_i$ and $S_j$ sum over all pairs of objects. We approximate the overlapping volume between pairs of objects using a calculation of their overlapping AABBs. The calculation follows from the 2D case of the overlapping area between two AABBs (see Figure 5.6).

In our implementation we add additional padding to all of the object AABBs because we reach same conclusion as Yang and Huang (2013) in that the avoidance of intersection between objects without extra separation is inadequate.




$$overlap\ x_{min} = \max(x1_{min}, x2_{min})$$
$$overlap\ y_{min} = \max(y1_{min}, y2_{min})$$
$$overlap\ x_{max} = \min(x1_{max}, x2_{max})$$
$$overlap\ y_{max} = \min(y1_{max}, y2_{max})$$

$$overlap\ area = (overlap\ x_{max} - overlap\ x_{min}) \times (overlap\ y_{max} - overlap\ y_{min})$$

Figure 5.6: Overlapping area (blue region) of two AABBs with equations.

**Occlusion.** The term $f_{oc}$ accounts for the sum of the overlapping area between pairs of objects:

$$f_{oc} = \sum_{S_i, S_j} A_i \cap A_j \tag{5.3}$$

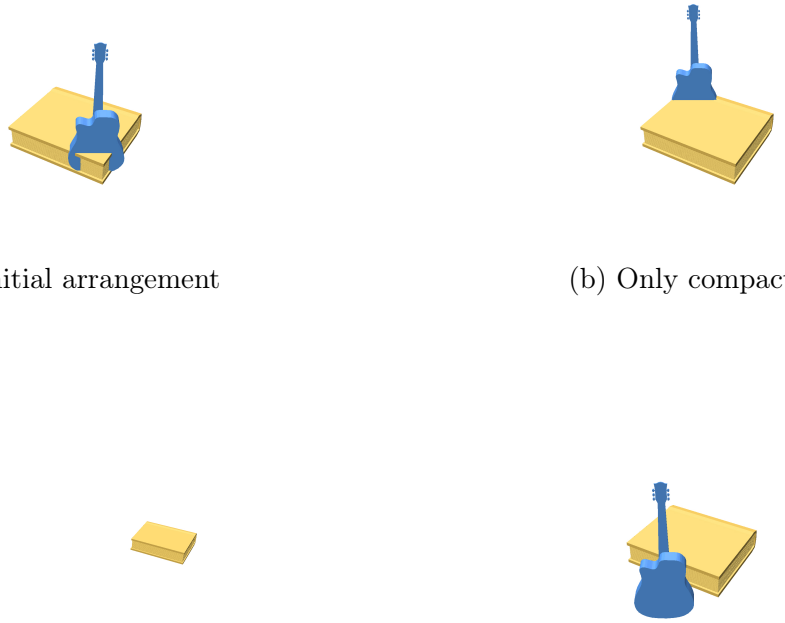
where $A_i$ represents the area of an object and $S_i$ and $S_j$ sum over all pairs of objects. To approximate the overlapping area, we count the number of coloured pixels shared by each pair of objects. Specifically, for a pair of meshes, a mesh is registered in Polyscope and a screenshot is taken, it is removed and the same procedure is undertaken for the second mesh. The overlapping number of coloured pixels shared by each screenshot approximates the overlapping area between a pair of objects. Polyscope does not provide access to its depth buffer, therefore the computation of $f_{oc}$ is generally expensive. In practice, we do not include the occlusion term in our objective function, instead we use it to rank the quality of an optimised arrangement (see below).

### Optimisation procedure

To find a good object arrangement, we define the following objective function that includes the previously described terms:

$$f(L) = w_{cr} f_{cr} + w_{cp} f_{cp} \tag{5.4}$$

where $w_{cr}$ and $w_{cp}$ are the weights of the terms. In our research, we find that $w_{cr} = 1$ and $w_{cp} = 0.025$ work well for our problem. Unless otherwise stated, all of the examples in this paper use these weights. Figure 5.7 shows the need to account for both compactness and clearance in the optimisation procedure.



(a) Initial arrangement

(b) Only compactness

(c) Only clearance

(d) Clearance and compactness

Figure 5.7: Trade-off of different terms on the optimisation result.

For optimisation, we use a similar extension of randomised pattern search as described in Yang and Huang (2013). The extended pattern search for 3D layout was first introduced in Yin and Cagan (2000) - its ease of use, independence from the calculation of derivatives, and ability to produce quality layouts make it an ideal algorithm to apply for object arrangement.

The optimisation procedure begins with an initial arrangement $L_0$. We assume that the initial arrangement is the placement of all objects at the origin such that objects are aligned in the world coordinates system. For the initial arrangement, we set $f(L_0) = \infty$. The optimisation algorithm will then iterate to try to improve upon the current arrangement of objects. At each iteration, one of the following exploratory moves is taken: (i) translation of a randomly selected object by values sampled from a Gaussian distribution representing the object's AABB depth and width; (ii) swapping the positions of two randomly selected

objects. In this optimisation approach, we assume that objects should be placed on the ground plane. Therefore, the translations and swapping of objects should only consider changes in the object's $x$ and $z$ coordinates.

The new arrangement of objects will be accepted if there is a decrease in the objective function, otherwise, the algorithm will go back to the previous iteration and try another exploratory move. We set the same stopping conditions as Yang and Huang (2013): the maximum number of iterations of the algorithm is 10000, while the algorithm will also terminate if the objective function does not decrease for 100 iterations.

To allow for the generation of multiple arrangement options for the user to choose from, we run 32 optimisations on the initial arrangement and rank them in ascending order according to a weighted sum of the occlusions present in the final arrangement and the number of white pixels in the final arrangement:

$$rank = w_{oc} f_{oc} + w_{white} whitePx \tag{5.5}$$

For all of the results in this paper, we set $w_{oc} = 1$ and $w_{white} = 0$. We also select the best object arrangement when we display the output of this approach.

### 5.2.3 Poisson disk sampling for object arrangement

An observation about the optimisation approach to arranging objects is that it aims to minimise objectives in the search for good arrangements. The problem of arranging objects according to guidelines can be rephrased as one that places objects such that they meet constraints. This results in an alternative algorithm. The idea of this approach is to sequentially place objects in the scene. If the placement of the current object does not meet the constraints, the placement of the object is repeated for a fixed number of times until it meets the constraints or until the maximum number of iterations is met. This is similar to interior scene synthesis methods because those methods also include constraints on object placement. However, this approach differs because the constraints make fewer assumptions about inter-object relationships.

This above algorithm is inspired by Poisson disk sampling. Poisson disk sampling is a technique that is applied to reduce aliasing in ray tracing (Cook, 1986). Illustrated in Figure 5.8, It works by the random generation of samples at non-uniform locations in an image (sampling space) with the requirement that samples are a certain distance apart from one-another. Given its success in sample placement, we investigate its utility to generate good object arrangements.

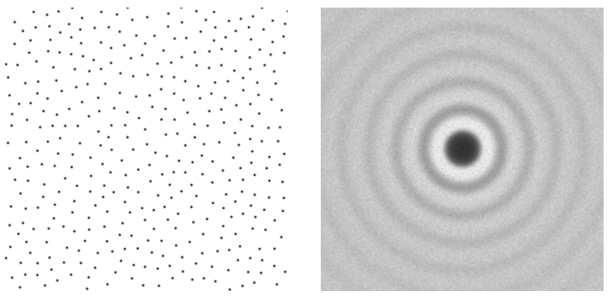


Figure 5.8: Poisson sampling on a 2D grid (left). Averaged distribution over multiple runs (right) (Bridson, 2007).

The algorithm begins with the same initial arrangement as the optimisation approach, denoted as $L_0$. At each iteration, the current object is appended to a list. This list is used to compute a sampling grid centered at the origin with bounds equivalent to the total bounds of the objects present in the list. These bounds are weighted to alter the shape of the sampling space. We set both the horizontal weight and vertical weights to 0.5 to create a smaller sampling space. The gradual increase of the size of the sampling grid as more meshes are placed was chosen such that the resulting arrangement is not too spread out, favouring compactness.

Having defined the sampling grid, a random point is sampled from this grid and the current object is then translated to the sampled position. The intersection of this object with the previously placed objects is then checked. If there is an intersection with any of the previously placed objects, the object translation is undone and the sampling procedure is repeated.

In the implementation, we set the maximum number of repetitions of the current object placement to 100 as we find that if one is not set, the algorithm may run infinitely. If after 100 iterations, the object still intersects with other objects, the algorithm will keep the object located at that position and will then proceed to the next iteration step. The algorithm terminates when all of the objects have been placed.

Similar to the optimisation approach, we assume that all of the objects should be placed on the ground plane. Therefore, the sampling grid should be 2D and transformations of an object should only consider its $x$ and $z$ coordinates. We also run this approach 32 times on the initial arrangement and rank them in ascending order according to Equation 5.5. We select the best object arrangement for all of the presented results using this approach.

## 5.3 Chapter summary

This chapter discussed the framework for the generation of a representative thumbnail image of a collection of objects. More specifically, we firstly calibrated the camera and ensured that the meshes were aligned with each other. We then formalised the notion of the best view in our implementation by considering the visibility ratio of a mesh from a viewpoint. Once this viewpoint was found, we introduced two approaches for arranging the objects in the scene. One was an optimisation approach derived from Yang and Huang (2013) while the other was an extension of Poisson disk sampling (Cook, 1986). The next chapter will examine the performance of our framework by investigating the suitability of the generated images. We will also compare the results generated by the two arrangement approaches.

# Chapter 6

# Results

We now evaluate our implemented framework. The resulting thumbnail images are screenshots of the outputs from the arrangement approaches discussed in the previous chapter. Each approach generates 32 thumbnail images of size 256 x 256 pixels. In this chapter, we firstly explain the rationale behind our evaluation, then outline the evaluation procedure, and finally discuss the results and limitations of the study. This chapter concludes with the future implications of our findings.

## 6.1 Rationale

The motivation behind our evaluation procedure is to investigate the performance of our framework and the quality of the results generated. In particular, we are interested in comparing the two arrangement algorithms to decide if one is better suited for the generation of a representative thumbnail image. Another research aim is to explore the efficiency of our framework.

Due to the stochastic nature of the arrangement approaches, the results generated are non-deterministic. This nature if not considered could invalidate the results. To mitigate this risk, we therefore treat each arrangement approach as the process to generate 32 thumbnail images of size 256 x 256 pixels.

Our evaluation considers the thumbnail images generated by each arrangement approach. A qualitative evaluation is undertaken because there is no ground truth image to compare the outputs of our framework against. Our evaluation also considers quantitative metrics to compare the two approaches and the overall framework: namely, the time taken to complete the individual stages of the framework and the average occlusion present in the images resulting from each arrangement approach.

Chapter 5 introduced occlusion as an arrangement guideline to encourage representative thumbnail images. Specifically, the less overlapping area there is in an image, the more is visible of the objects present in the image. We were also able to measure this quantity by summing the number of coloured pixels shared by each pair of objects. Given that the intersections between objects and the compactness of an arrangement can easily be observed from the output image, we quantify occlusion to provide a more scientific handling of the evaluation of the different arrangement approaches.

## 6.2 Experiment setup

For our evaluation, we follow a similar approach to Yang and Huang (2013). We define three different data sets (see Figure 6.2) that mimic products to be advertised. The first data set contains electronics, the second contains musical equipment, and the third contains toys. All of the assets were taken from Turbosquid (n.d.) and the data sets can be accessed from the Assets folder in the accompanying code.

For each of these data sets, we run our framework with the default parameters (see Chapter 5). For each run, we record the number of objects present in the scene, the total number of faces, the times taken for the individual stages of our framework, and the average number of occlusions present in the resulting images. The resulting images are also generated.

For brevity, we only include the best arrangement generated by each approach. This is defined as the arrangement with the least occlusion. For the optimisation approach, we also plot objective function curves for the result with the lowest occlusion score (the best result) and the result with the highest occlusion score (the worst result) to investigate the characteristics of their optimisation trajectories. The time taken to plot the objective function curves is discounted from the time of the optimisation approach.

To handle the measurement and reporting of program run time we referred to Harris-Birtill and Harris-Birtill (2021). They provide recommendations on how to formally report program run time in an experimental setting. Their recommendation includes a break down of the individual processes and how long each takes as well as any bottlenecks in the code. We suggest that the calculation of the average occlusions is an expensive calculation. To verify this claim, we re-run our framework without the occlusion calculation and report the run times of the arrangement approaches.

Another claim we make is that the average number of occlusions will be lower for the Poisson disk sampling based approach when compared to the optimisation approach. This is because in the former approach, we have control over the sampling space and if the space is large enough, we expect objects will be placed less compactly leading to a less occluded layout.

## 6.3 Analysis of the results

### 6.3.1 Performance

We performed our experiments on an Apple M1 pro (8-core) Macintosh system. The computation times for the individual stages of our framework are reported in Table 6.1. It shows an increase of mesh complexity corresponds to an increase in the execution time of view selection. This is expected because the computation of the best view requires the scene to be re-rendered for every mesh.

Table 6.2 shows the timings of the arrangement approaches under different settings. It is shown that the exclusion of the average occlusion calculation greatly improves the computational performance of the arrangement strategies.

In particular, Figure 6.1 suggests that the Poisson disk sampling based approach to arranging objects without the average occlusion calculation is the most efficient arrangement

Table 6.1: Performance statistics with inclusion of occlusion calculation.

| Data set | # items | # faces | view selection (s) | Optimisation (s) | Poisson (s) |
|---|---|---|---|---|---|
| Electronics | 8 | 48197 | 48.9485 | 2581.16 | 1499.43 |
| Music equipment | 11 | 457917 | 110.455 | 6017.55 | 2620.91 |
| Toys | 11 | 311904 | 92.5254 | 7094.28 | 2958.47 |

Table 6.2: Timings of arrangement approaches.

| Data set | Opt. w Occ. (s) | Poi. w Occ. (s) | Opt. w/ Occ. (s) | Poi. w/Occ. (s) |
|---|---|---|---|---|
| Electronics | 2581.16 | 1499.43 | 209.364 | 73.9861 |
| Music equipment | 6017.55 | 2620.91 | 1816.95 | 227.572 |
| Toys | 7094.28 | 2958.47 | 1006.37 | 183.818 |

strategy out of the examined approaches when applied to our data sets. The Poisson disk sampling approach sequentially places objects such that they satisfy constraints, while the optimisation approach explores the layout space through random search and evaluation. This clearly contributes to the efficiency of the Poisson disk sampling approach.

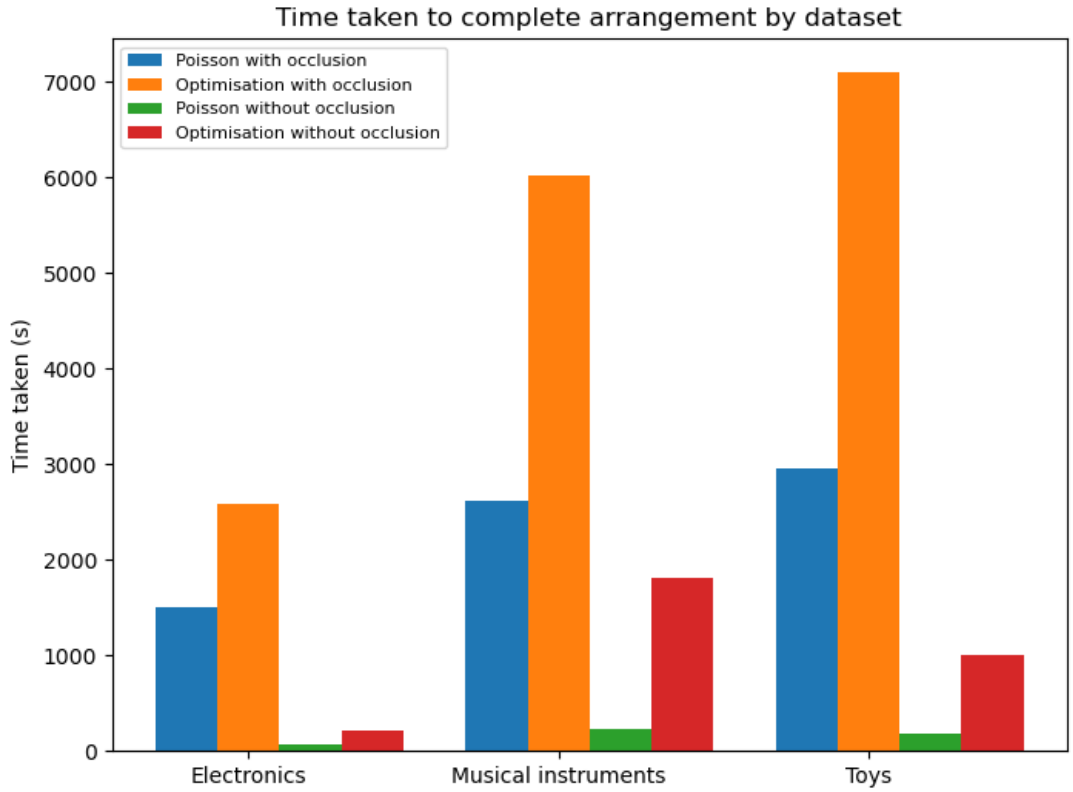
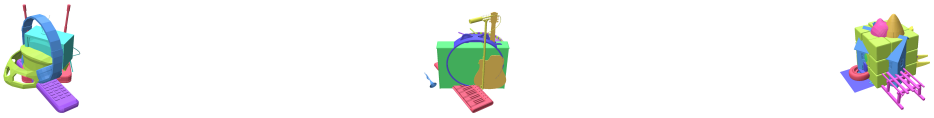



Figure 6.1: Graphical interpretation of the timings of arrangement approaches.

### 6.3.2 Quality

We now examine the visual results generated by each arrangement approach. Starting from an initial arrangement of each data set (see Figure 6.2) the results in Figure 6.3 demonstrate that the arrangement approaches are able to generate images that capture the visual features of the objects present in the scene. For the full sized images of the



(a) Electronics (b) Musical instruments (c) Toys

Figure 6.2: Initial layouts of the data sets after best view selection.

results, please refer to Appendix A. While the results do not exhibit severe occlusion, one observation is that the viewpoint selection algorithm is not always able to capture some of the objects with distinctive frontal views (for example the guitar).

(a) Electronics optimisation.

(b) Electronics Poisson.

(c) Musical instruments optimisation.

(d) Musical instruments Poisson.

(e) Toys optimisation.

(f) Toys Poisson.

Figure 6.3: Layout results for each data set using the different arrangement approaches.

A comparison of the images generated by each arrangement approach reveals that those from the optimisation approach appear more compact than those from the Poisson disk sampling approach. A more compact arrangement may be more desirable as the objects are more in view.

Figure 6.4 provides supplementary information about the characteristics of the arrangement approaches. These results suggest that the average occlusion in each of the data sets using the Poisson disk sampling approach is less than that of the optimisation approach. This meets our expectations because in the former approach, despite the stochastic nature of the sampling, one is able to alter the dimensions of the sampling space in a way that leads to less occlusions between meshes.

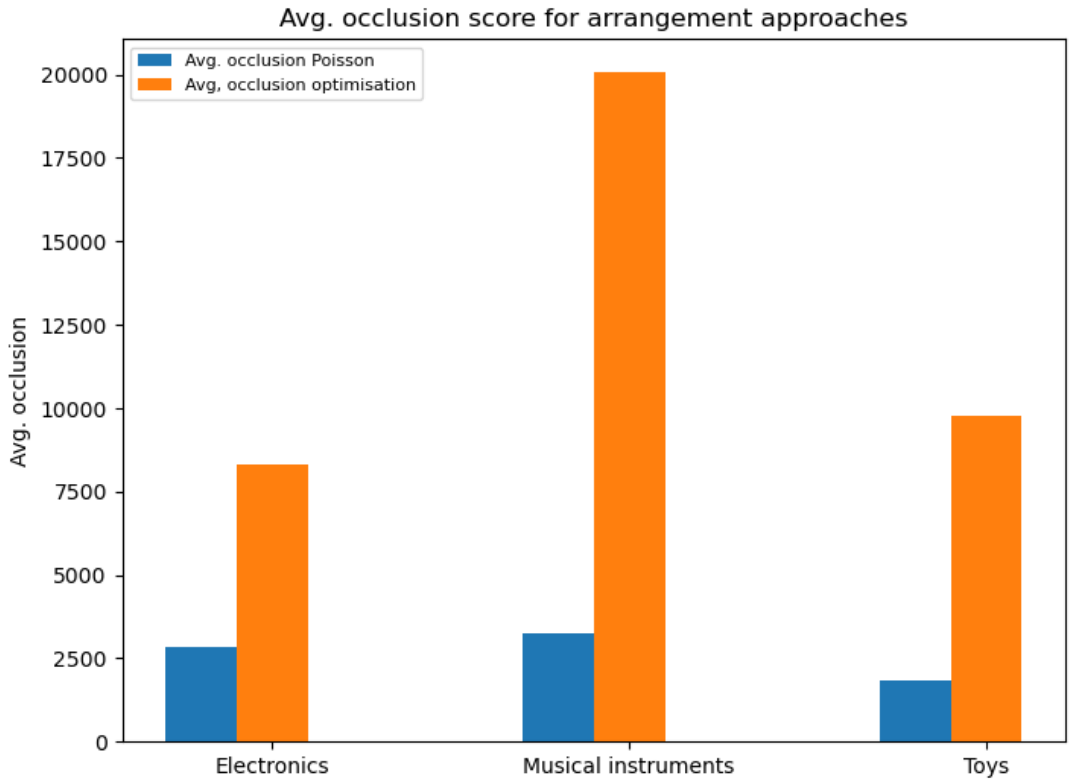
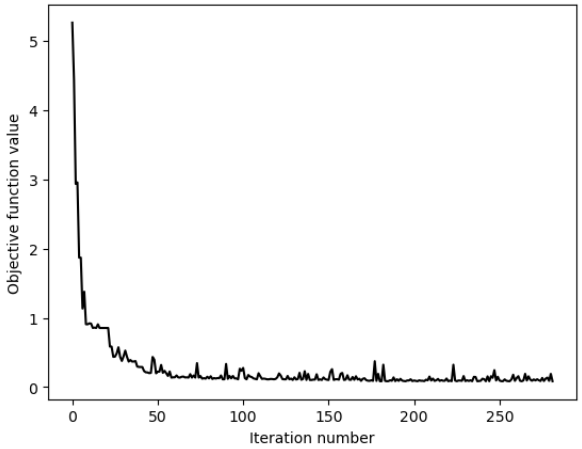


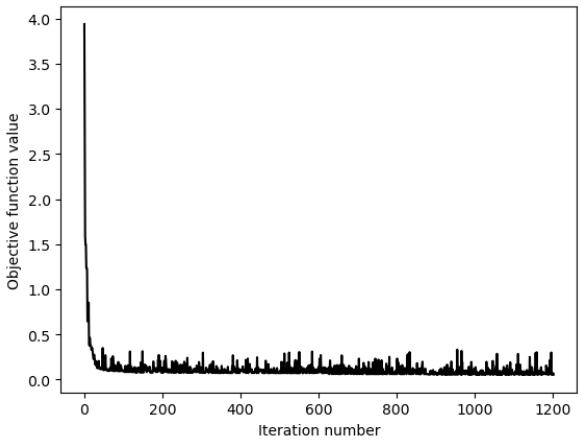

Figure 6.4: Average number of occlusions for each arrangement approach.

To investigate the characteristics of the different results obtained by the optimisation approach, we plot the optimisation trajectory of the best generated image and the worst generated image from each data set. We also include the minimum objective function value for the different trajectories. These are given in Figure 6.5.

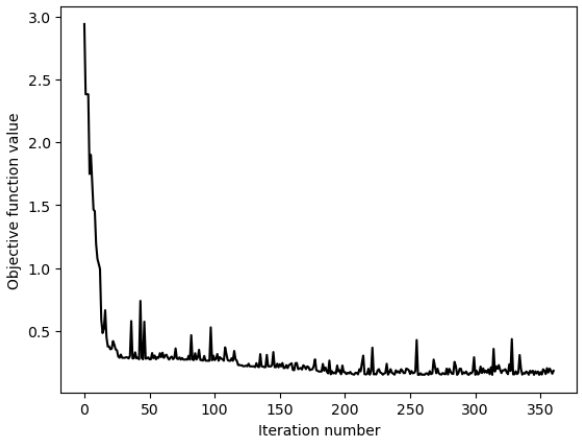
From these curves, we observe that the optimisation trajectory of the best results tends to fluctuate less than the worst results. This suggests that the best results are exploring the search space more effectively than the worst results. The objective values for the best results are higher than those of the worst results generated. This indicates that not all of the requirements are built into the objective function. This meets our expectations because we excluded occlusion from the objective function. Both of the curves decrease providing evidence that the optimisation is working as expected.



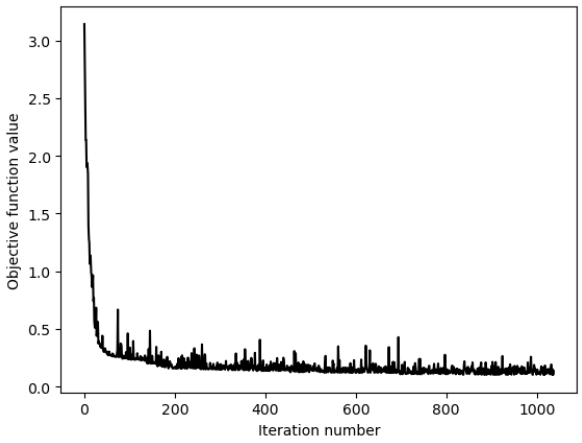

(a) Best arrangement electronics (0.0847014).



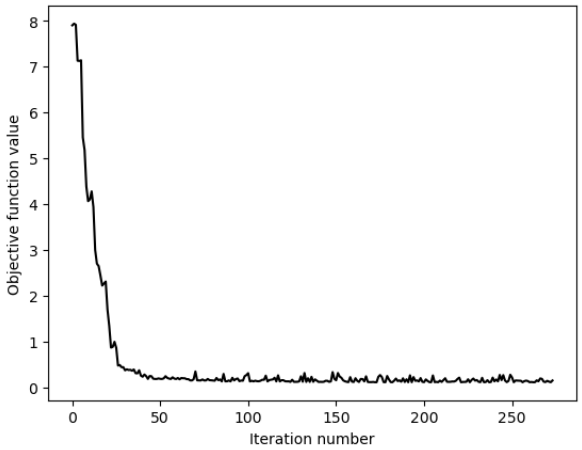

(b) Worst arrangement electronics (0.052192).



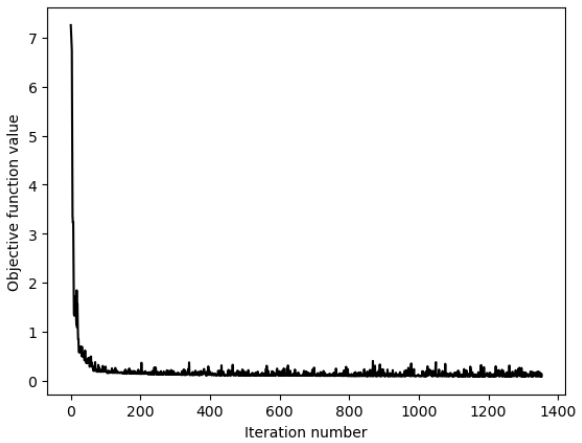

(c) Best arrangement instruments (0.15081).




(d) Worst arrangement instruments (0.10257).




(e) Best arrangement toys (0.113605).




(f) Worst arrangement toys (0.079543).

Figure 6.5: Optimisation curves for the best and worst results of each data set, with minimum objective values in brackets.

## 6.4 Discussion

The results of our evaluation suggest that the current formulation of our framework favours the Poisson disk sampling approach to arrangement to avoid occlusions between objects. However, the Poisson disk sampling approach is prone to generating less compact layouts which may make objects appear smaller. The optimisation approach on the other hand can generate compact layouts but is prone to occlusion. Our experiments suggest that this is due to the occlusion requirement being omitted from the objective function.

The occlusion calculation in our current implementation is a bottleneck in the procedure as omitting this calculation led to an increased efficiency of the arrangement approaches. Nevertheless, both strategies are able to generate plausible thumbnail images that display object features for the most part.

This evaluation has focused on the analysis of the characteristics of the different arrangement approaches, their efficiency, and the results they generated. As such, this evaluation did not consider a user study to verify the quality of the generated results. If this framework is to be used in practice or to be extended, future research should include a user study to further verify our framework and arrangement approaches.

# Chapter 7

# Conclusions

## 7.1 Contributions

Our primary contribution built upon Yang and Huang's framework for tray design and extended their approach to handle the generation of a representative thumbnail image of a collection of objects.

Specifically, we firstly formalised the notion of selecting a good viewpoint such that object features could be maximised through the use of the visibility ratio (Polonsky et al., 2005). To arrange objects, Yang and Huang's approach was adapted to optimise the visibility of object features and the volume occupied by objects in the scene. Another arrangement approach proposed was an extension of Poisson disk sampling to place objects such that no two objects intersected with one another.

Our secondary contribution was the evaluation of our framework and the individual arrangement approaches. The results demonstrate that our framework is able to generate plausible thumbnail images that provide a representative view of object features in most of the examined scenarios. Our results suggest that the Poisson-based approach is more efficient than the optimisation approach for object arrangement and while it generates arrangements that occupy more space, it generates arrangements that are not prone to severe occlusion. This indicates that the Poisson disk sampling technique is the more effective technique in this setting.

As a lesser contribution, this dissertation includes all of the source code of our implemented framework and includes all the results from our evaluation. Please see Appendix B.

## 7.2 Limitations and future work

Although our framework is able to generate representative thumbnail images, there exist a number of limitations that if addressed can improve our current work.

Our framework is simpler when compared to that used in Yang and Huang's method. As a result, although the images found in our approach are adequate, better results may be obtained if the complexity of the framework is improved upon. The parts of our framework that can be improved are the viewpoint selection stage, the inclusion of an alignment algorithm to deal with objects that have an arbitrary orientation, and most importantly,

the inclusion of techniques in the framework that address inter-relations among multiple objects (Yang and Huang, 2013).

The implementation of the optimisation procedure did not account for the occlusion requirement in the objective function. We acknowledged that this could have led to layouts with more occlusion. This is due to the limitations of using Polyscope as a visualisation tool for rendering meshes. Despite its simplicity and access to other geometry processing libraries, it is still quite constrained as a visualisation tool for our tasks. For example, it does not provide for texture support and it does not provide to access to its depth buffer. The lack of texture support detracts from the overall realism of the generated thumbnail images and the restricted access to the depth buffer means that Polyscope is not a suitable viewer for use in a commercial setting. Future work should therefore alter the objective function in the optimisation stage to account for a faster occlusion calculation using a different viewer that provides access to a depth buffer and supports mesh textures.

Furthermore, the Poisson sampling method can be improved upon to allow for more compact images. Our research indicated that the approach worked well for object placement. In our approach we had a sampling grid that expanded according to the previously placed meshes and the current mesh. Such a sampling grid can be refined to allow more compactness for object arrangement. Future research can explore this aspect.

In the future we would like to improve upon this work by addressing the above limitations. Once they have been addressed, a user study can be conducted to validate the effectiveness of the enhanced framework.

## 7.3 Final remarks

This dissertation has now reached completion. Having devised and evaluated a framework for the generation of a representative thumbnail image of a collection of objects, we hope that this will add to the literature surrounding visualisation and optimisation.

# Bibliography


Adhikary, A., 2014. Advertising: A fusion process between consumer and product. *Procedia economics and finance*, 11, pp.230–238.

Akenine-Mo, T., Haines, E., Hoffman, N. et al., 2018. *Real-time rendering*. AK Peters/CRC Press.

Ali, S., Ramos, A.G., Carravilla, M.A. and Oliveira, J.F., 2022. On-line three-dimensional packing problems: a review of off-line and on-line solution approaches. *Computers & industrial engineering* [Online], pp.108–122. Available from: `https://www.sciencedirect.com/science/article/pii/S0360835222001929`.

Blanz, V., Tarr, M.J. and Bülthoff, H.H., 1999. What object attributes determine canonical views? *Perception*, 28(5), pp.575–599.

Bridson, R., 2007. Fast poisson disk sampling in arbitrary dimensions. *Siggraph sketches*, 10(1), p.1.

Cagan, J., Shimada, K. and Yin, S., 2002. A survey of computational approaches to three-dimensional layout problems. *Computer-aided design* [Online], 34(8), pp.597–611. Available from: `https://www.sciencedirect.com/science/article/pii/S0010448501001099`.

Cook, R.L., 1986. Stochastic sampling in computer graphics. *Acm transactions on graphics (tog)*, 5(1), pp.51–72.

Gao, T., Wang, W. and Han, H., 2016. Efficient view selection by measuring proxy information. *Computer animation and virtual worlds*, 27(3-4), pp.351–357.

Gottschalk, S., Lin, M.C. and Manocha, D., 1996. Obbtree: A hierarchical structure for rapid interference detection [Online]. *Proceedings of the 23rd annual conference on computer graphics and interactive techniques*. pp.171–180. Available from: `http://gamma.cs.unc.edu/SSV/obb.pdf`.

Guennebaud, G., Jacob, B. et al., 2010. Eigen v3. http://eigen.tuxfamily.org.

Harris-Birtill, D. and Harris-Birtill, R., 2021. Understanding computation time: a critical discussion of time as a computational performance metric. *Time in variance*. Brill, pp.220–248.

Hu, R., Xu, J., Chen, B., Gong, M., Zhang, H. and Huang, H., 2020. Tap-net: transport-and-pack using reinforcement learning. *Acm transactions on graphics (tog)*, 39(6), pp.1–15.

Huang, S., Wang, Z., Zhou, J. and Lu, J., 2022. Planning irregular object packing via hierarchical reinforcement learning. *Ieee robotics and automation letters* [Online]. Available from: `https://ieeexplore.ieee.org/abstract/document/9954127`.

Jacobson, A., Panozzo, D., Schüller, C., Diamanti, O., Zhou, Q., Pietroni, N. et al., 2013. libigl: A simple c++ geometry processing library. *Google scholar*.

Kán, P. and Kaufmann, H., 2018. Automatic furniture arrangement using greedy cost minimization [Online]. *2018 ieee conference on virtual reality and 3d user interfaces (vr)*. IEEE, pp.491–498. Available from: `https://ieeexplore.ieee.org/document/8448291`.

Kopf, J., Cohen-Or, D., Deussen, O. and Lischinski, D., 2006. Recursive wang tiles for real-time blue noise. *Acm siggraph 2006 papers*. pp.509–518.

Kucerova, J., Varhanikova, I. and Cernekova, Z., 2012. Best view methods suitability for different types of objects. *Proceedings of the 28th spring conference on computer graphics*. pp.55–61.

Lee, C.H., Varshney, A. and Jacobs, D.W., 2005. Mesh saliency. *Acm siggraph 2005 papers*. pp.659–666.

Li, M., Patil, A.G., Xu, K., Chaudhuri, S., Khan, O., Shamir, A., Tu, C., Chen, B., Cohen-Or, D. and Zhang, H., 2019. Grains: Generative recursive autoencoders for indoor scenes. *Acm transactions on graphics (tog)* [Online], 38(2), pp.1–16. Available from: `https://dl.acm.org/doi/pdf/10.1145/3303766`.

Lienhard, S., Specht, M., Neubert, B., Pauly, M. and Müller, P., 2014. Thumbnail galleries for procedural models. *Computer graphics forum* [Online], 33(2), pp.361–370. Available from: `https://doi.org/https://doi.org/10.1111/cgf.12317`.

Lin, M. and Gottschalk, S., 1998. Collision detection between geometric models: A survey [Online]. *Proc. of ima conference on mathematics of surfaces*. vol. 1, pp.602–608. Available from: `http://gamma-web.iacs.umd.edu/papers/COLLISION/cms.pdf`.

Liu, J., Liane, W., Ning, B. and Mao, T., 2021. Work surface arrangement optimization driven by human activity [Online]. *2021 ieee virtual reality and 3d user interfaces (vr)*. IEEE, pp.270–278. Available from: `https://ieeexplore.ieee.org/abstract/document/9417801`.

Mao, J., Li, T., Zhang, F., Wang, M., Chang, J. and Lu, X., 2021. Bas-relief layout arrangement via automatic method optimization. *Computer animation and virtual worlds*, 32(3-4), p.e2012.

Plemenos, D., Sbert, M. and Feixas, M., 2004. On viewpoint complexity of 3d scenes. *International conference graphicon'2004*.

Polonsky, O., Patané, G., Biasotti, S., Gotsman, C. and Spagnuolo, M., 2005. What's in an image? towards the computation of the "best" view of an object. *The visual computer*, 21, pp.840–847.

Prasad, K.A. and Vetrivel, S., 2016. An empirical study on visual merchandising and its impact on consumer buying behaviour. *losr journal of business and management (losr-jbm)*, 18(11), pp.8–14.

Samet, H., 1984. The quadtree and related hierarchical data structures. *Acm computing surveys (csur)* [Online], 16(2), pp.187–260. Available from: https://dl.acm.org/doi/pdf/10.1145/356924.356930.

Secord, A., Lu, J., Finkelstein, A., Singh, M. and Nealen, A., 2011. Perceptual models of viewpoint preference. *Acm transactions on graphics (tog)*, 30(5), pp.1–12.

Sharp, N. et al., 2019. Polyscope. Www.polyscope.run.

Turbosquid, n.d. [Online]. Available from: http://www.turbosquid.com.

Turk, G. and Levoy, M., 1994. *Stanford bunny* [Online]. Stanford Computer Graphics Laboratory. Accessed on 18-02-2023. Available from: https://graphics.stanford.edu/data/3Dscanrep/.

Vázquez, P.P., Feixas, M., Sbert, M. and Heidrich, W., 2001. Viewpoint selection using viewpoint entropy. *Vmv*. Citeseer, vol. 1, pp.273–280.

Velho, L., Carvalho, P., Gomes, J. and Figueiredo, L. de, 2011. *Mathematical optimization in computer graphics and vision* [Online]. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Available from: https://dl.acm.org/doi/pdf/10.5555/2843511.

Weiss, T., Litteneker, A., Duncan, N., Nakada, M., Jiang, C., Yu, L.F. and Terzopoulos, D., 2018. Fast and scalable position-based layout synthesis. *Ieee transactions on visualization and computer graphics* [Online], 25(12), pp.3231–3243. Available from: https://ieeexplore.ieee.org/abstract/document/8443151.

Wu, W., Fu, X.M., Tang, R., Wang, Y., Qi, Y.H. and Liu, L., 2019. Data-driven interior plan generation for residential buildings. *Acm transactions on graphics (tog)* [Online], 38(6), pp.1–12. Available from: https://dl.acm.org/doi/epdf/10.1145/3355089.3356556.

Xie, H., Xu, W. and Wang, B., 2013. Reshuffle-based interior scene synthesis [Online]. *Proceedings of the 12th acm siggraph international conference on virtual-reality continuum and its applications in industry*. pp.191–198. Available from: https://dl.acm.org/doi/pdf/10.1145/2534329.2534352.

Yang, Y.L. and Huang, Q.X., 2013. Traygen: Arranging objects for exhibition and packaging [Online]. *Computer graphics forum*. Wiley Online Library, vol. 32, pp.187–195. Available from: https://onlinelibrary.wiley.com/doi/full/10.1111/cgf.12226.

Yin, S. and Cagan, J., 2000. An extended pattern search algorithm for three-dimensional component layout. *J. mech. des.*, 122(1), pp.102–108.

Zhang, S.H., Zhang, S.K., Liang, Y. and Hall, P., 2019. A survey of 3d indoor scene synthesis. *Journal of computer science and technology* [Online], 34(3), pp.594–608. Available from: https://link.springer.com/content/pdf/10.1007/s11390-019-1929-5.pdf?pdf=inline%20link.

Zhang, S.H., Zhang, S.K., Xie, W.Y., Luo, C.Y., Yang, Y.L. and Fu, H., 2021a. Fast 3d indoor scene synthesis by learning spatial relation priors of objects. *Ieee transactions on visualization and computer graphics* [Online], 28(9), pp.3082–3092. Available from: https://ieeexplore.ieee.org/abstract/document/9321177.

Zhang, S.K., Li, Y.X., He, Y., Yang, Y.L. and Zhang, S.H., 2021b. Mageadd: Real-time interaction simulation for scene synthesis [Online]. *Proceedings of the 29th acm international conference on multimedia*. pp.965–973. Available from: `https://dl.acm.org/doi/pdf/10.1145/3474085.3475194`.

Zhang, S.K., Xie, W.Y. and Zhang, S.H., 2021. Geometry-based layout generation with hyper-relations among objects. *Graphical models* [Online], 116, pp.101–104. Available from: `https://www.sciencedirect.com/science/article/pii/S1524070321000096`.

Zhang, Y. and Fei, G., 2019. Overview of 3d scene viewpoints evaluation method. *Virtual reality & intelligent hardware*, 1(4), pp.341–385.

Zhang, Y. and Yang, G., 2022. Optimization of the virtual scene layout based on the optimal 3d viewpoint. *Ieee access*, 10, pp.110426–110443.

# Appendix A

# Resulting images for arrangement approaches

## A.1 Full sized figures for best arrangements

Please see next page.

Figure A.1: Layout results for each data set using the optimisation approach.

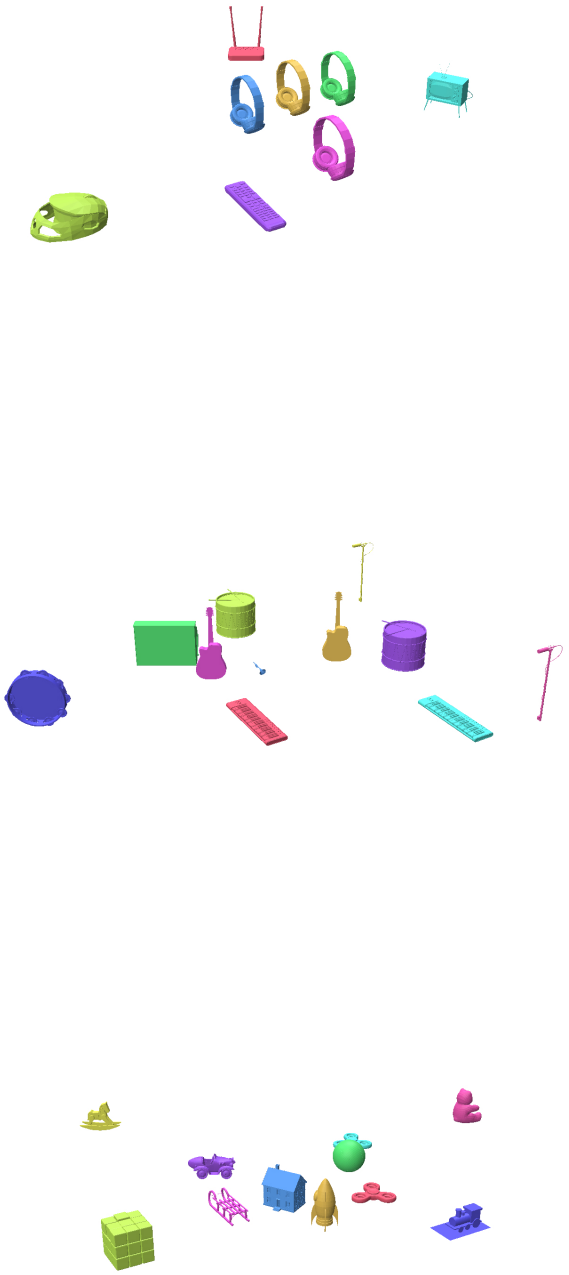


Figure A.2: Layout results for each data set using the Poisson disk sampling approach.

# Appendix B

# Downloadable code

Link to zipped code Please note that experimental results are in the code folder. Please refer to the ReadMe.

If the link does not work, we provide the implemented code below for all of the classes in our implementation. Note that this code will not run as it does not contain the dependency code.

## B.1 File: main.cpp

```
#include "polyscope/polyscope.h"
#include <iostream>
#include "scene.h"
#include <string>
#include <boost/filesystem.hpp>

using namespace std;

//Remove and re-create all of the necessary directories
void reset_folders(string folder_path){
    boost::filesystem::path directory_path(folder_path);
    boost::filesystem::remove_all(directory_path);
    boost::filesystem::create_directory(directory_path);
}

int main(){
    //Generate the folders for the results of our algorithm
    string final_images_poisson_path =
        "../../Final_images_poisson";
    string final_images_extended_pattern_path =
        "../../Final_images_pattern_search";
    string results_path = "../../csvs";
    reset_folders(final_images_poisson_path);
    reset_folders(final_images_extended_pattern_path);
    reset_folders(results_path);

    string file_name = "../../Assets/mesh_file.txt";

   //Initialise the scene with a file path to the mesh.txt file
    Scene scene(file_name);
    scene.visualizeContents();
}
```

## B.2 File: mesh.h

```
//This file includes code for a mesh class
#pragma once
#include <Eigen/Dense>
#include <string>

using namespace std;
using namespace Eigen;

class Mesh{
public:
    //Class attributes of a mesh
    MatrixXd V;
    MatrixXi F;
    Vector3d MaxCorner;
    Vector3d MinCorner;
    Vector3d translation;
    string file_name;
    double radius;

    Mesh(string input_file,double r);
    MatrixXd get_vertices();
    MatrixXi get_faces();
    void scale(double target_size);
    void center();
    Vector3d get_center();
    void translate(Vector3d translation);
    void translateObject();
    void inverse_translateObject();
    Vector3d get_inverse_translation();
    void rotateObject(double angle);
    void compute_bounding_box();
    string get_file_name();
};
```

## B.3 File: mesh.cpp

```
//This file includes code for a mesh class
#include <igl/read_triangle_mesh.h>
#include "mesh.h"
#include <chrono>
#include <random>

//Read the mesh file and assign padding for the bounding box
Mesh::Mesh(string input_file,double r){
    bool success = igl::read_triangle_mesh(input_file,V,F);
    if(success){
        file_name = input_file;
        //Find the bounding box
        MinCorner = V.colwise().minCoeff();
        MaxCorner = V.colwise().maxCoeff();
        translation << 0,0,0;
        radius = r;
    }

    else{
        std::cerr<<"Error: spelling mistake made or the file is
            empty"<<endl;
        std::exit(-1);
    }
}

//Get vertices
MatrixXd Mesh::get_vertices(){
    return V;
}

//Get faces
MatrixXi Mesh::get_faces(){
    return F;
}

//Scale by the distance between corners of the AABB
void Mesh::scale(double target_size){
    double diagonal = (MaxCorner - MinCorner).norm();
    double scale_factor = target_size/diagonal;
    V = (V.rowwise()-MinCorner.transpose())*scale_factor;
    compute_bounding_box();
    return;
}

//subtract all of the vertices by the minimum corner
void Mesh::center(){
    V = V.rowwise() - MinCorner.transpose();
}

Vector3d Mesh::get_center(){
    return V.colwise().mean();
}

void Mesh::translate(Vector3d translation){
    //Convert vertices to a homogeneous one
    MatrixXd vertices_homogeneous = V.rowwise().homogeneous();
    Affine3d transform = Affine3d::Identity();
    transform.translate(translation);
    //Set the new vertices and remove homogeneous coordinates
    V =
        ((transform.matrix()*vertices_homogeneous.transpose()).transpose().rowwis
    compute_bounding_box();
}

void Mesh::translateObject(){
    //Make static to ensure no repeated patterns
    static std::default_random_engine
        generator(std::chrono::system_clock::now().time_since_epoch().count());
    double lower_bound = 0;
    double upper_bound = 3;

    // Define the uniform distribution
    std::uniform_real_distribution<double>
        distribution(lower_bound, upper_bound);

    // Generate a random number using the distribution
    double std_dev1 = distribution(generator);
    double std_dev2 = distribution(generator);

    double width = MaxCorner[0] - MinCorner[0];
    double depth = MaxCorner[2] - MinCorner[2];

    //Sample translations in z direction
    std::normal_distribution<double>distribution1(0.0,std_dev1);

    //Sample translations in the x and z directions
    std::normal_distribution<double>distribution2(0.0,std_dev2);

    double x_translation = distribution2(generator)*width;
    double z_translation = distribution1(generator)*depth;

    //set the translation vector
```

```
    translation[0] = x_translation;
    translation[1] = 0;
    translation[2] = z_translation;
    translate(translation);
}

//Store translation of the mesh and invert it
void Mesh::inverse_translateObject(){
    Vector3d inverse_translation = get_inverse_translation();
    translate(inverse_translation);

}

Vector3d Mesh::get_inverse_translation(){
        return Vector3d(-translation[0], -translation[1],
            -translation[2]);
}

void Mesh::rotateObject(double angle){

    MatrixXd vertices_homogeneous = V.rowwise().homogeneous();

    //Compute center of the object
    Vector3d center = get_center();

    //Translate the object such that the centre is at the origin,
        rotate, and translate back.
    Affine3d transform = Affine3d::Identity();
    transform.pretranslate(-center);
    transform.prerotate(AngleAxisd(angle,Vector3d::UnitY()));
    transform.pretranslate(center);

    //Apply the transformation
    V =
        ((transform.matrix()*vertices_homogeneous.transpose()).transpose()).rowwis
    return;
}

void Mesh::compute_bounding_box(){
    MinCorner = V.colwise().minCoeff();
    MaxCorner = V.colwise().maxCoeff();
    MinCorner -= Eigen::Vector3d(radius, radius,
        radius).cast<double>();
    MaxCorner += Eigen::Vector3d(radius, radius,
        radius).cast<double>();
    return;
}

string Mesh::get_file_name(){
    return file_name;
}
```

## B.4 File: scene.h

```
#pragma once
#include "polyscope/polyscope.h"
#include <polyscope/surface_mesh.h>
#include <igl/read_triangle_mesh.h>
#include "../deps/polyscope/deps/stb/stb_image.h"
#include <iostream>
#include "polyscope/screenshot.h"
#include "mesh.h"
#include <fstream>
#include <string>
#include <cmath>

using namespace std;
using namespace Eigen;

class Scene{
public:

    Vector3d MaxCorner;
    Vector3d MinCorner;
    vector<Mesh>meshes;
    double time_taken;

    Scene(string input_file);
    void compute_scene_bounding_box(vector<Mesh>meshes);
    void centerCamera(vector<Mesh>meshes);
    void show_all(vector<Mesh>meshes);
    void remove_all(vector<Mesh>meshes);
    void show_mesh(Mesh &mesh);
    double compute_coloured_pixels(double degrees);
    void rotate_mesh(Mesh &mesh,double angle);
    void find_best_view();

    void swap_meshes(Mesh &mesh1,Mesh &mesh2);
    double compactness(vector<Mesh>meshes);
    double compute_occlusion(Mesh &mesh1,Mesh &mesh2,double
        w1,double w2);
    double compute_intersection(Mesh &mesh1,Mesh &mesh2);
    double clearance(vector<Mesh>meshes);
    double occlusionArea(vector<Mesh>meshes);

    double take_final_picture(string
        file_path,vector<Mesh>meshes,int index);
    double optimizeArrangement(int iterations, vector<Mesh>&
        meshes,int index,string file_path,double w1,double w2);
    Vector2d generate_point(Vector2d minPoint,Vector2d maxPoint);
    bool is_intersecting(vector<Mesh>meshes,int index);
    double optimizePoisson(vector<Mesh>& meshes,int index,double
        x_modifier,double z_modifier,string file_path);
    Vector2d calculate_bounds(vector<Mesh>meshes);
    double poissonArrangement(string file_path,int iterations);
    double extendedPatternSearchArrangement(string file_path,int
        iterations);
    void visualizeContents();
    void write_to_csv(vector<double>data,string filename);
};
```

## B.5 File: scene.cpp

```
//Change all your functions so they can produce and rank multiple
    pictures.
#include "scene.h"
#include "exception"
#include <cstdlib>
#include <chrono>
#include <cstdio>
#include <utility>

//Registers the meshes into the mesh list and centers and scales
    the meshes
Scene::Scene(string input_file){
    time_taken = 0;
    string title;
    string file_path = "../../Assets/";
    ifstream Read_file(input_file);
    int num_faces = 0;
    while(getline(Read_file,title)){
        Mesh read_mesh(file_path+title,0.05);
        num_faces += read_mesh.get_faces().rows();
        //Scale and center the meshes to ensure that they are on
            the same plane
        read_mesh.scale(1.0);
        meshes.push_back(read_mesh);
    }
    cout<<"Number of faces: "<<num_faces<<endl;
    Read_file.close();
    polyscope::init();
    polyscope::options::groundPlaneMode =
        polyscope::GroundPlaneMode::None;
    polyscope::options::alwaysRedraw = true;
    polyscope::view::windowWidth = 256;
    polyscope::view::windowHeight = 256;
}

//Centers the camera in polyscope by computing the scene bounding
    box
void Scene::compute_scene_bounding_box(vector<Mesh>meshes){
    // Find the overall minimum and maximum corners of the scene
    Vector3d overall_min =
        Vector3d::Constant(std::numeric_limits<double>::max());
    Vector3d overall_max =
        Vector3d::Constant(std::numeric_limits<double>::min());

    for(const auto& mesh:meshes){
        Vector3d mesh_min = mesh.MinCorner;
        Vector3d mesh_max = mesh.MaxCorner;

        overall_min = overall_min.cwiseMin(mesh_min);
        overall_max = overall_max.cwiseMax(mesh_max);
    }

    MinCorner = overall_min;
    MaxCorner = overall_max;
    return;
}

void Scene::centerCamera(vector<Mesh>meshes){
    compute_scene_bounding_box(meshes);
    Vector3d center = (MinCorner + MaxCorner)/2.0;
    double radius = (MaxCorner - center).norm();

    // Set the camera location and target based on the center and
        radius
    glm::vec3 camera_location(center.x(),(MaxCorner[1] -
        MinCorner[1])+radius,center.z()-2.5*radius);
    glm::vec3 target(center.x(),center.y(),center.z());
    polyscope::view::lookAt(camera_location,target);
    return;
}

//Register all meshes into polyscope
void Scene::show_all(vector<Mesh>meshes){
    for(int i = 0; i<meshes.size(); i++){
        MatrixXd vertices = meshes[i].get_vertices();
        MatrixXi faces = meshes[i].get_faces();
        string title = meshes[i].get_file_name();
        polyscope::registerSurfaceMesh(title,vertices,faces);
    }
    return;
}

//Remove all mehses from polyscope
void Scene::remove_all(vector<Mesh>meshes){
    for(int i = 0; i<meshes.size(); i++){
        string title = meshes[i].get_file_name();
        polyscope::removeStructure(title);
    }
    return;
}

//Register meshes into polyscope
```

```
void Scene::show_mesh(Mesh &mesh){
    MatrixXd vertices = mesh.get_vertices();
    MatrixXi faces = mesh.get_faces();
    string title = mesh.get_file_name();
    polyscope::registerSurfaceMesh(title,vertices,faces);
    return;
}

//Computes the number of coloured pixels in the image
double Scene::compute_coloured_pixels(double degrees){

    string file_name = "best_angle"+to_string(degrees)+".jpg";

    polyscope::screenshot(file_name,false);

    int width, height, channels;
    unsigned char* pixels = stbi_load(file_name.c_str(), &width,
        &height, &channels,0);
    double image_pixels = 0.0;

    for (int i = 0; i < width * height * channels; i += channels) {
        if (pixels[i] == 255 && pixels[i+1] == 255 && pixels[i+2]
            == 255) {
            continue;
        }
        else{
            image_pixels+=1.0;
        }
    }
    stbi_image_free(pixels);
    return image_pixels;
}

//rotates the meshes around their center in polyscope
void Scene::rotate_mesh(Mesh &mesh,double angle){
    double rotation = (angle*M_PI)/180.0;
    mesh.rotateObject(rotation);
    MatrixXd vertices = mesh.get_vertices();
    MatrixXi faces = mesh.get_faces();
    string title = mesh.get_file_name();
    vector<Mesh>mesh_list;
    mesh_list.push_back(mesh);
    centerCamera(mesh_list);
    polyscope::registerSurfaceMesh(title,vertices,faces);
    return;
}
```

```
//Finds the best view point of all the objects using visibility
    ratio
void Scene::find_best_view(){
    for(int i = 0; i<meshes.size(); i++){
        vector<double>angles;
        vector<double>pixels;

        string title = meshes[i].get_file_name();

        for(double angle = 0; angle<=360; angle+=30){

            double rotation;
            if(angle == 0){
                rotation = 0;
            }

            else{
                rotation = 30;
            }

            angles.push_back(angle);
            rotate_mesh(meshes[i],rotation);
            double image_pixels = compute_coloured_pixels(angle);
            pixels.push_back(image_pixels);
        }

        std::vector<double> probabilities(pixels.size());
        double sum = accumulate(pixels.begin(), pixels.end(), 0.0);
        transform(pixels.begin(), pixels.end(),
            probabilities.begin(),
        [sum](double x) { return x / sum; });


        auto maxElementIterator =
            max_element(probabilities.begin(),probabilities.end());
        int maxElementIndex = distance(probabilities.begin(),
            maxElementIterator);
        double best_angle = angles[maxElementIndex];

        rotate_mesh(meshes[i],best_angle);
        meshes[i].center();
        meshes[i].compute_bounding_box();
        polyscope::removeStructure(title);

        write_to_csv(probabilities,"../../csvs/best_view.csv");
    }
```

```
    centerCamera(meshes);
    return;
}

//swap meshes
void Scene::swap_meshes(Mesh &mesh1,Mesh &mesh2){
    Vector3d center1 = mesh1.get_center();
    Vector3d center2 = mesh2.get_center();
    Vector3d offset(center1[0] - center2[0], 0.0, center1[2] -
        center2[2]);
    mesh1.translate(-offset);
    mesh2.translate(offset);
    return;
}

//Compute the compactness term:
double Scene::compactness(vector<Mesh>meshes){

    double length = MaxCorner[0] - MinCorner[0];
    double width = MaxCorner[1] - MinCorner[1];
    double height = MaxCorner[2] - MinCorner[2];

    double scene_volume = length*width*height;
    double volumes = 0.0;

    for (const Mesh& mesh : meshes) {
        length = mesh.MaxCorner[0] - mesh.MinCorner[0];
        width = mesh.MaxCorner[1] - mesh.MinCorner[1];
        height = mesh.MaxCorner[2] - mesh.MinCorner[2];
        double volume = length * width * height;
        volumes += volume;
    }

    if(volumes == 0.0){
        throw std::runtime_error("There are no meshes tried to
            divide by 0");
    }

    return scene_volume/volumes;

}

//Calculate the occlusion area between 2 meshes
double Scene::compute_occlusion(Mesh &mesh1,Mesh &mesh2,double
    w1,double w2){

    show_mesh(mesh1);
    polyscope::screenshot("object1.jpg",false);

    int width1, height1, channels1;
    unsigned char* pixels1 = stbi_load("object1.jpg", &width1,
        &height1, &channels1,0);

    polyscope::removeStructure(mesh1.get_file_name());

    show_mesh(mesh2);
    polyscope::screenshot("object2.jpg",false);

    polyscope::removeStructure(mesh2.get_file_name());

    int width2, height2, channels2;
    unsigned char* pixels2 = stbi_load("object2.jpg", &width2,
        &height2, &channels2,0);

    double occluded_pixels = 0.0;
    for (int i = 0; i < width1 * height1 * channels1; i +=
        channels1) {
        bool contains_object_1 = (pixels1[i] != 255 ||
            pixels1[i+1] != 255|| pixels1[i+2] != 255);
        bool contains_object_2 = (pixels2[i] != 255 ||
            pixels2[i+1] != 255|| pixels2[i+2] != 255);
        bool is_white =
            not(contains_object_1)&&not(contains_object_2);

        if(is_white){
            occluded_pixels += w1;
        }

        if(contains_object_1 && contains_object_2){
            occluded_pixels += w2;
        }
    }
    stbi_image_free(pixels1);
    stbi_image_free(pixels2);
    return occluded_pixels;
}

//Calculate the intersection volume between 2 meshes
double Scene::compute_intersection(Mesh &mesh1,Mesh &mesh2){
    double xmin = std::max(mesh1.MinCorner[0], mesh2.MinCorner[0]);
    double ymin = std::max(mesh1.MinCorner[1], mesh2.MinCorner[1]);
    double zmin = std::max(mesh1.MinCorner[2], mesh2.MinCorner[2]);
    double xmax = std::min(mesh1.MaxCorner[0], mesh2.MaxCorner[0]);
    double ymax = std::min(mesh1.MaxCorner[1], mesh2.MaxCorner[1]);
```

```
    double zmax = std::min(mesh1.MaxCorner[2], mesh2.MaxCorner[2]);

    double length = std::max(0.0,xmax-xmin);
    double width = std::max(0.0,ymax-ymin);
    double height = std::max(0.0,zmax-zmin);

    double intersection_volume = length*width*height;
    return intersection_volume;
}

//Compute the clearance term
double Scene::clearance(vector<Mesh>meshes){
    double total_intersection_occlusion = 0.0;
    if(meshes.size() == 1){
        return total_intersection_occlusion;
    }

    for(int i = 0;i<meshes.size()-1; i++){
        for(int j = i+1;j<meshes.size();j++){
            double intersection_volume =
                compute_intersection(meshes[i],meshes[j]);
            total_intersection_occlusion += intersection_volume;
        }
    }
    return total_intersection_occlusion;
}

//Compute the area occlusion of the image
double Scene::occlusionArea(vector<Mesh>meshes){
    double occlusion = 0.0;
    if(meshes.size() == 1){
        return occlusion;
    }

    for(int i = 0;i<meshes.size()-1; i++){
        for(int j = i+1;j<meshes.size();j++){
            occlusion +=
                compute_occlusion(meshes[i],meshes[j],0,1);
        }
    }
    return occlusion;
}

//write a value of vectors out to a csv for graphs generated in
    python
void Scene::write_to_csv(vector<double>data,string filename){
    ofstream file(filename.c_str(), ofstream::app);

    for (int j = 0; j < data.size(); j++) {
        file << data[j];
        if (j != data.size() - 1) {
            file << ",";
        }
    }

    file << "\n";

    return;
}

double Scene::take_final_picture(string
    file_path,vector<Mesh>meshes,int index){
    centerCamera(meshes);

//    Uncomment if you want faster runs
    double occlusionScore = 0;

//    //Uncomment if you want automated rankings (this ranks by
    occlusion)
//    double occlusionScore = occlusionArea(meshes);
//
    show_all(meshes);
    string filename = file_path+"arrangement_" + to_string(index)
        + ".jpg";
    polyscope::screenshot(filename,false);
    remove_all(meshes);
    return occlusionScore;
}

//Arrange the objects
double Scene::optimizeArrangement(int iterations, vector<Mesh>&
    meshes,int index,string file_path,double w1,double w2){
    unsigned seed =
        std::chrono::high_resolution_clock::now().time_since_epoch().count();
    srand(seed);

    int decrements = 0;
    double L = std::numeric_limits<double>::max();
    vector<double>data;
    for(int i = 0; i<iterations; i++){

        if(decrements>100){
            break;
        }
```

```
    int index_1 = -1;
    int index_2 = -1;
    int choice = 1;

    if(meshes.size()>1){
        choice = rand()%2 + 1;
    }

    //translate a randomly selected object
    if(choice == 1){
        index_1 = rand()%meshes.size();
        meshes[index_1].translateObject();
    }

    //swap the position of 2 objects
    else{
        index_1 = rand()%meshes.size();
        index_2 = rand()%meshes.size();
        while (index_1 == index_2) {
            index_2 = rand() % meshes.size();
        }
        swap_meshes(meshes[index_1],meshes[index_2]);

    }
    centerCamera(meshes);
    double Lnew = w1*clearance(meshes) +
        w2*compactness(meshes);
    data.push_back(Lnew);

    if(Lnew<L){
        L = Lnew;
        decrements = 0;
    }

    else{
        decrements+=1;
        if(choice == 1){
            meshes[index_1].inverse_translateObject();
        }
        else{
            swap_meshes(meshes[index_1],meshes[index_2]);
        }
    }
}

//Write the optimisation values for each run of the
    optimisation. One can use this to generate graphs for
    experiments
    auto timer = std::chrono::high_resolution_clock::now();
    auto start = timer;
    write_to_csv(data,"../../csvs/optimisation.csv");
    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed_time =
        std::chrono::duration_cast<std::chrono::duration<double>>(end
        - start).count();
    time_taken += elapsed_time;

    double occlusion = take_final_picture(file_path,meshes,index);
    return occlusion;
}

//Generating a point in the area of the sampling grid
Vector2d Scene::generate_point(Vector2d minPoint,Vector2d
    maxPoint){
    static std::default_random_engine
        generator(std::chrono::system_clock::now().time_since_epoch().count());
    std::uniform_real_distribution<double> distx(minPoint[0],
        maxPoint[0]);
    std::uniform_real_distribution<double> distz(minPoint[1],
        maxPoint[1]);
    double x = distx(generator);
    double z = distz(generator);
    Vector2d generatedPt(x, z);
    return generatedPt;
}

//Check if the current mesh is intersecting with any of the
    previous meshes
bool Scene::is_intersecting(vector<Mesh>meshes,int index){
    for(int i = index-1;i>=0; i--){
        double intersection_volume =
            compute_intersection(meshes[index],meshes[i]);
        if(intersection_volume>0.0){
            return true;
        }
    }
    return false;
}

double Scene::optimizePoisson(vector<Mesh>& meshes,int
    index,double x_modifier,double z_modifier,string file_path){
    vector<Mesh>current_meshes;
    for(int i = 0; i<meshes.size(); i++){
```

```
        current_meshes.push_back(meshes[i]);
        //Center the grid at the origin
        Vector2d MinCorner(0,0);
        Vector2d MaxCorner(0,0);
        Vector2d bounds = calculate_bounds(current_meshes);
        MinCorner[0]-= bounds[0]*x_modifier;
        MinCorner[1]-= bounds[1]*z_modifier;
        MaxCorner[0]+= bounds[0]*x_modifier;
        MaxCorner[1]+= bounds[1]*z_modifier;

        bool outcome = false;
        int count = 0;
        //Try different arrangement of the object until a valid
            ones is found or the maximum number of iterations is
            reached
        do{
            Vector2d candidate_point =
                generate_point(MinCorner,MaxCorner);
            Vector3d
                offset(candidate_point[0],0,candidate_point[1]);
            meshes[i].translate(offset);
            outcome = is_intersecting(meshes,i);
            if(outcome){
                count+= 1;
                if(count != 100){
                    meshes[i].translate(-offset);
                }
            }
        }while (outcome and count<100);
    }

    double occlusion =  take_final_picture(file_path,meshes,index);

    return occlusion;
}

//Calculate the bounds of the sampling grid according to current
    meshes
Vector2d Scene::calculate_bounds(vector<Mesh>meshes){
    double x_dist = 0;
    double z_dist = 0;
    for(const auto& mesh:meshes){
        Vector3d mesh_min = mesh.MinCorner;
        Vector3d mesh_max = mesh.MaxCorner;

        x_dist += mesh_max[0] - mesh_min[0];
        z_dist += mesh_max[2] - mesh_min[2];
    }
    Vector2d bounds(x_dist,z_dist);
    return bounds;
}

//Full poisson procedure
double Scene::poissonArrangement(string file_path,int iterations){

    vector<pair<int,double>> scores;
    for(int i = 0; i<iterations; i++){
        vector<Mesh>mesh_list = meshes;
        double score =
            optimizePoisson(mesh_list,i,0.5,0.5,file_path);
        scores.push_back(make_pair(i, score));
    }

    //sort according to the occlusions (ascending order)
    sort(scores.begin(), scores.end(), [](const pair<int, int>& a,
        const pair<int, int>& b) {
        return a.second < b.second;
    });

    double avg_occlusions = 0;
    for (int i = 0; i < scores.size(); i++) {
        int index = scores[i].first;
        int score = scores[i].second;
        avg_occlusions += (double)score;
        string old_filename = file_path+"arrangement_" +
            to_string(index) + ".jpg";
        string new_filename = file_path+"ranking_" +
            to_string(i+1) + "_score_" + to_string(score) + ".jpg";
        rename(old_filename.c_str(), new_filename.c_str());
    }
    avg_occlusions = avg_occlusions/scores.size();
    return avg_occlusions;
}

//Full optimisation process
double Scene::extendedPatternSearchArrangement(string
    file_path,int iterations){
    vector<pair<int,double>> scores;
    for(int i = 0; i<iterations; i++){
        vector<Mesh>mesh_list = meshes;
        double score =
            optimizeArrangement(10000,mesh_list,i,file_path,1,0.025);
        scores.push_back(make_pair(i, score));
    }
```

```
    sort(scores.begin(), scores.end(), [](const pair<int, int>& a,
        const pair<int, int>& b) {
        return a.second < b.second;
    });

    double avg_occlusions = 0;
    for (int i = 0; i < scores.size(); i++) {
        int index = scores[i].first;
        int score = scores[i].second;
        avg_occlusions += (double)score;
        string old_filename = file_path+"arrangement_" +
            to_string(index) + ".jpg";
        string new_filename = file_path+"ranking_" +
            to_string(i+1) + "_score_" + to_string(score) + ".jpg";
        rename(old_filename.c_str(), new_filename.c_str());
    }
    avg_occlusions = avg_occlusions/scores.size();
    return avg_occlusions;
}

//Full pipeline
void Scene::visualizeContents(){
    auto timer = std::chrono::high_resolution_clock::now();

    centerCamera(meshes);
    show_all(meshes);
    polyscope::screenshot("Initial.jpg",false);
    remove_all(meshes);

    auto start1 = timer;

    find_best_view();

    show_all(meshes);
    polyscope::screenshot("bestview.jpg",false);
    remove_all(meshes);

    auto end1 = std::chrono::high_resolution_clock::now();
    auto elapsed_time1 =
        std::chrono::duration_cast<std::chrono::duration<double>>(end1
        - start1).count();
    std::cout << "Elapsed time for view finding: " <<
        elapsed_time1 << " seconds" << std::endl;

    show_all(meshes);
    polyscope::screenshot("best viewing.jpg",false);
    remove_all(meshes);

    auto start2 = timer;

    double occlusion1 =
        poissonArrangement("../../Final_images_poisson/",32);

    auto end2 = std::chrono::high_resolution_clock::now();
    auto elapsed_time2 =
        std::chrono::duration_cast<std::chrono::duration<double>>(end2
        - start2).count();
    std::cout << "Elapsed time for poisson arrangement: " <<
        elapsed_time2 << " seconds" << std::endl;
    std::cout << "The avg number of occlusions: " << occlusion1 <<
        " occlusions" << std::endl;


    auto start3 = timer;

    double occlusion2 =
        extendedPatternSearchArrangement("../../Final_images_pattern_search/",32);

    auto end3 = std::chrono::high_resolution_clock::now();
    auto elapsed_time3 =
        std::chrono::duration_cast<std::chrono::duration<double>>(end3
        - start3).count();
    double time_difference = elapsed_time3 - time_taken;
    std::cout << "Elapsed time for extended pattern arrangement: "
        << time_difference << " seconds" << std::endl;
    std::cout << "The avg number of occlusions: " << occlusion2 <<
        " occlusions" << std::endl;
}
```