BROWN UNIVERSITY

HONORS THESIS

Functional Chair Programs in ShapeAssembly

Author:
Bryce BLINN

Advisor and First Reader:
Daniel RITCHIE
Second Reader:
Srinath SRIDHAR

April 7, 2021

BROWN UNIVERSITY

Abstract

Department of Computer Science

Functional Chair Programs in ShapeAssembly

by Bryce BLINN

Object design, the process of creating a 3D shape, is one of the most fundamental parts of scene generation, as well as of the most labor-intensive. Deep generative models can assist with this process by allowing designers to quickly generate unique objects. Models to generate object representations such as voxelgrids and point clouds have been successful in generating widely variable unique objects. In the domain of mesh generation, however, many models fail to guarantee that generated mesh objects are physically valid and can exist in the real world.

ShapeAssembly is mesh-generative model that guarantees physical validity by converting mesh objects to a program representation. Each program "assembles" the object through operations defined in ShapeAssembly's language. The authors of ShapeAssembly then train a variational auto-encoder to generate new chair programs which can then be converted into physically valid meshes.

However, this model fails to guarantee functional validity of the objects it creates. An object is functionally valid if it is able to satisfy the function for which it was designed. In this thesis, I explore ways in which to modify the ShapeAssembly training paradigm to guarantee functionally valid objects, specifically chairs.

I define specific metrics of functionality on a given chair, then calculate differentiable loss functions over the chair based on these metrics that will increase the functionality of chairs generated by the ShapeAssembly variational auto-encoder. For every metric of functionality I consider, I evaluate my approach to increasing the functionality of the ShapeAssembly variational auto-encoder by comparing the chairs that it outputs to those of the ShapeAssembly chair dataset.

Contents

Chapter 1

Introduction

FIGURE 1.1: A figure of chair objects taken from the ShapeAssembly chair dataset.

Object design, the process of creating a 3D shape, is a major bottleneck in developing 3D scenes. For example a designer who wants to create a living room scene would have to to create chairs, tables, and cabinets from scratch, which can take a lot of time and effort. Incorporating generative models into the design process can accelerate this process. For example, tree and general vegetation generation can be modeled using L-systems, which use probabilistic recursion to recreate the fractal-like nature of plants [1]. Another popular model for object generation is the shape grammar. Similar to L-systems, a shape grammar recurses over smaller parts of a single shape following rules defined by a dataset or the designer, like the following shape grammar used to generate building façades [2].

Deep neural generative models have become popular for the generation of 3D objects, with different representations such as voxelgrids, point clouds, and meshes. Voxelgrid generation has been explored in papers such as 3D ShapeNets [3], which extends the 2D CNN image generation techniques to 3D objects. Point cloud generative models have had great improvements with the creation of PointNet [4], a discriminative network for point cloud classification and segmentation. The authors of the Generative PointNet model [5] incorporate the basic encoding structure of PointNet to derive a generative model that can generate point clouds representing objects such as chairs, bathtubs, tables, etc.

In mesh generation, one of the popular approaches is mesh reconstruction from a point cloud. The Atlas-Net model [6] takes a paper-mâiche—based approach to reconstructing a mesh from a point cloud; it generates an atlas 2D mesh parameterization for each input point cloud and then "wraps" the point cloud with the mesh parameterization to form a 3D object. This process, however, is prone to creating invalid non-manifold meshes which cannot exist in the real world. The Point2Mesh paper [7] uses an optimization process to reconstructing a surface mesh from a point cloud to circumvent non-manifoldness. However, this process can take as much as

three hours to generate meshes from higher resolution point clouds. Another popular approach to mesh generation is to compose a mesh out of smaller manifold meshes, thus guaranteeing manifoldness. The authors of StructureNet [8] follow this approach with a hierarchical composition procedure. However, StructureNet-generated objects can suffer from lack of physical feasibility; some of the objects generated by StructureNet either lack parts or have multiple parts occupying the same space.

ShapeAssembly [9] uses a hierarchical composition approach akin to that of StructureNet but avoids the problems with object feasibility by modifying the object representation which the model learns to generate. Any composite object can be represented as the product of a series of operations which create and assemble its parts. Moreover, it's possible to determine if an object is physically feasible by determining if the operations used to assemble it are physically feasible. For example, to assemble an object containing overlapping parts, one must have placed a new part inside of an already existing part, which is a physically impossible operation. Following from this logic, the authors of ShapeAssembly construct a domain specific language only defining physically feasible assembly operations. The ShapeAssembly deep generative model is trained to create objects represented as executable programs composed of these physically feasible operations, and thus the objects it generates are always feasible.

While these programs are always physically feasible, ShapeAssembly programs lack guarantees of functionality. For example, a bookshelf that cannot fit any books is not a functional bookshelf. A desk that is tilted to the left is non-functional as things that are placed on it will fall off.

Considering the functionality of all kinds of objects is a very difficult task as different objects function in different ways, so I limit my work to the domain of chairs and their functionality, such as those in Figure 1.1. I specifically address two aspects of chair functionality: person-independent functionality and person-specific functionality. Some chairs generated by the ShapeAssembly generative model are not able to stand on their own and are not functional independent regardless of their ability to be used by people. This occurs because either the chairs' parts are disconnected or because the chair itself is physically unstable. As well, some chairs generated by the ShapeAssembly model are not functional as an object in which to sit, as it would be impossible or extremely uncomfortable to sit in them. This happens when their geometry is such that even in the most comfortable pose, a person's body would not be supported by the chair.

In this thesis, I define specific metrics of person-independent and person-specific functionality. I then derive differentiable loss functions over the chair based on these metrics that will increase the functionality of chairs generated by the ShapeAssembly variational auto-encoder. In each section, I detail the results of training the ShapeAssembly variational auto-encoder with each loss metric I've defined and compare the functionality of the chairs it produces to that of the ShapeAssembly dataset. I show that these aspects of functionality can be quantified with numerical metrics, and I find that the loss metrics I create increase the functionality of these chairs.

Chapter 2

Background

As described in the introduction, ShapeAssembly is a domain specific language that defines executable programs that assemble a given object. As well as defining the ShapeAssembly domain specific language, the ShapeAssembly paper also details the model used to generate ShapeAssembly programs and its architecture. Here I will outline the information about the domain specific language and the generative model that is relevant to this thesis. For any other information not covered in this outline, one should consult the ShapeAssembly paper [9].

2.1 The ShapeAssembly Operation Structure

A object can be described as one or more programs in ShapeAssembly. These programs will have a hierarchical structure. Each ShapeAssembly program contains 4 different blocks of instructions: BBlock, CBlock, ABlock, and SBlock. BBlocks define the invisible bounding box over the entire object to which cuboids can be attached, CBlocks define cuboid parts that form the chair, ABlocks yield attachment protocols between cuboids/bounding boxes, and SBlocks yield symmetries within each object. In order to allow for hierarchy within each program, the cuboid parts defined in a CBlock need not be single cuboids; they can themselves be sub-programs.

ShapeAssembly defines specific functions within these blocks. Each program can be defined with the following operations: Cuboid, attach, squeeze, reflect, and translate. "Cuboid" is called within a BBlock or a CBlock and defines a new cuboid. Its arguments are its dimensions and a flag determining if it is aligned with its parent bounding box. "attach" is called within an ABlock and attaches its first cuboid argument to its second cuboid argument at the point on each cuboid given in the arguments. "squeeze" is also called within an ABlock and attaches the first cuboid argument between the second and third cuboid arguments at a face coordinate position specified in the arguments. The "translate" call is used within an SBlock and creates a translational symmetry group starting at the first argument and producing n more cuboids along the a axis that ends d distance away where n, a, and d are arguments to the function. The "reflect" call reflects the argument cuboid over the axis of the bounding volume defined in the arguments. Figure 2.1 displays a ShapeAssembly Program and its output.

2.2 The ShapeAssembly Generative Model

The ShapeAssembly paper also outlines a generative model for ShapeAssembly chair programs, a variational auto-encoder. Given a ShapeAssembly chair program, the auto-encoder can encode it into a vector in a latent space and then decode it back into the original chair program. From this, it is possible to generate a random vector

```
1.  def Chair():
2.    bbox = Cuboid(1, 1.5, .8, True)
3.    base = Base(.8, .5, .8, True)
4.    cube1 = Cuboid(.8, .1, .8, True)
5.    back = Back(.9, .8, .07, True)
6.    attach(base, bbox, .5, 0, .5, .5, 0, .5)
7.    attach(cube1, base, .5, 0, .5, .5, 1, .5)
8.    squeeze(back, bbox, cube1, top, .5, .05)

9.  def Base(l, w, h, aligned):
10.   bbox = Cuboid(l, w, h, aligned)
11.   cube0 = Cuboid(.2, .5, .2, True)
12    cube1 = Cuboid(.2, .5, .2, True)
13.   squeeze(cube0, bbox, bbox, top, .1, .1)
14.   squeeze(cube1, bbox, bbox, top, .1, .8)
15.   reflect(cube0, X)
      reflect(cube1, X)

17. def Back(l, w, h, aligned):
18.   bbox = Cuboid(l, w, h, aligned)
19.   cube0 = Cuboid(.9, .4, .07, True)
20.   cube1 = Cuboid(.1, .4, .05, True)
21.   attach(cube0, bbox, .5, 1, .5, .5, 1, .5)
22.   squeeze(cube1, bbox, cube0, bot, .3, .5)
23.   translate(cube1, X, 2, .5)
```

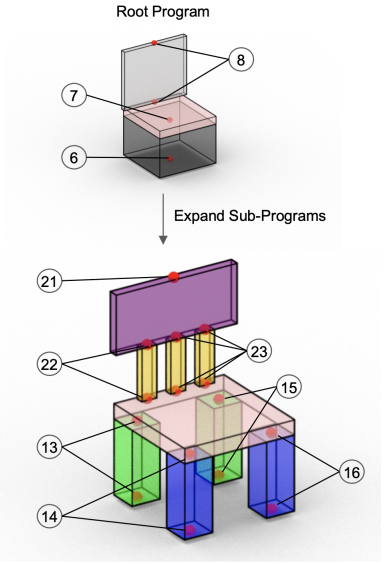


FIGURE 2.1: A figure of a ShapeAssembly program and its chair representation taken from the ShapeAssembly paper [9]. One can see that the subprograms "Base" and "Back" act as cuboids within the "Chair" program.

in the latent space and decode it into a new unseen chair, as can be seen in Figure 2.2.

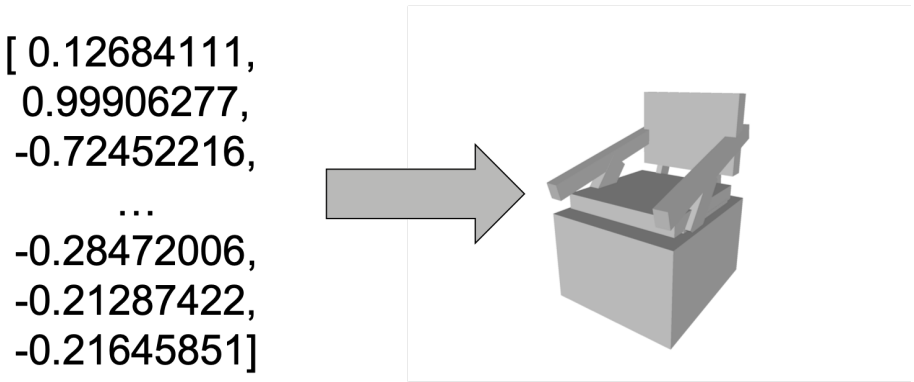


FIGURE 2.2: An example of a vector generated from a unit normal distribution and its decoded chair form.

The encoder first encodes an input program into a vector in the unit normal latent space using a Gated Recurrent Unit (GRU) and a random sampler. This vector is then decoded back into the program using the decoder. The decoder also uses a GRU network and decodes the vector into multiple 63-dimensional vectors where each vector contains the information to construct a command in the decoded program. Different slices of this vector are related to different commands; for example, the first 7 entries of the vector represent the probabilities that this command will be a given command type. The architecture of the ShapeAssembly variational auto-encoder can be seen in Figure 2.3.

It is important to note that the loss metrics used to train the auto-encoder in the

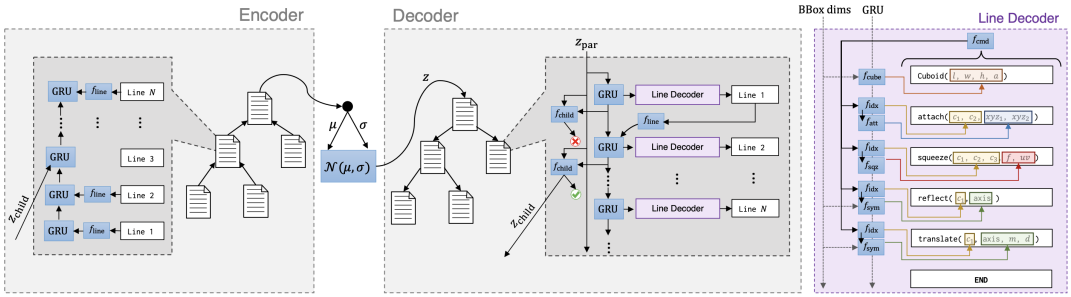


FIGURE 2.3: A figure of the ShapeAssembly variational auto-encoder taken from the ShapeAssembly paper [9].

ShapeAssembly paper are only supervised losses. These losses consist of the command loss, the cube parameter loss, the xyz parameter loss, the uv parameter loss, the symmetry parameter loss, the cuboid loss, the symmetry cuboid loss, the squeeze cuboid loss, the leaf loss, the bounding box loss, the axis loss, the face loss, the alignment loss, and the KL loss. The command loss trains the model to predict the correct command. The cuboid, symmetry cuboid, and squeeze cuboid, cube parameter, xyz parameter, uv parameter, symmetry parameter, axis, face, and alignment losses train the model to input the correct parameters in the various commands. The leaf and bounding box losses train the model to predict if a cuboid is a leaf or a bounding box. The KL loss is the Kullback–Leibler divergence metric and shapes the latent space of the encoder to be a unit normal distribution. It is also important to note that when the ShapeAssembly decoder decodes a latent space vector into a program, it will only decode semantically valid programs. During program generation time, if the program being built from the decoding becomes invalid, the decoder will stop building that program and begin a new program from a different random latent space vector.

Chapter 3

Enforcing Part Connectivity

3.1 Motivation

Concretely, connectivity within a chair can be described with graph terminology. If one considers the parts of a chair as nodes and the physical connections between parts as edges, a disconnected chair is a chair such that it would not be possible to traverse from each node to every other node in the graph. It is evident that a disconnected chair would be impossible to sit in as it could be easily separated into multiple pieces which have no connections holding them together. Such a chair is not functional independent of its purpose as a chair, as it cannot exist as a composite object. It would simply fall apart.

As described before, ShapeAssembly program representations allow for sub-programs to act as cuboids within their parent programs. When a sub-program is defined within a program, its bounding box acts as a cuboid within the parent program. Thus the bounding box of the sub-program satisfies ShapeAssembly's attachment requirements. The actual cuboids within the sub-program, however, are not required to attach to the points at which the bounding box is attached. This results in cuboids within a sub-program that might not be attached to any cuboids within the parent program. Only cuboids act as physical pieces of the chair, as the bounding boxes only exist to define a space in which cuboids can exist. Thus, if cuboids within a sub-program are not attached to cuboids within the parent program, the chair is disconnected.

While all of the chairs in the ShapeAssembly dataset are connected, this does not guarantee that the chairs produced by the ShapeAssembly generative model are also connected. Figure 3.1 displays disconnected chairs that are generated by a ShapeAssembly-trained generative model. The goal of this section is to determine if it is possible to use a unsupervised loss to increase the connectivity of the chairs outputted by the generative model.

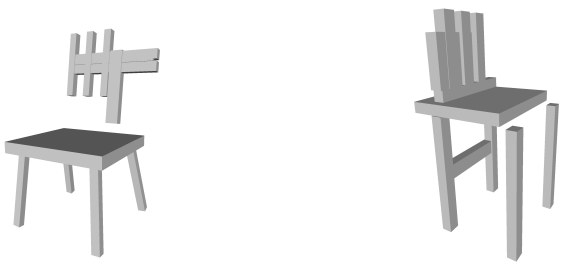


FIGURE 3.1: Chairs generated by the model that have disconnected parts

3.2 The Metric of Connectivity

The first challenge in increasing the connectivity of the generative model is to determine how to gauge connectivity. Computers cannot directly "look" at an object to spot gaps the way humans can. Two cuboids in a ShapeAssembly object are connected if they overlap at one or more points. While there exist algorithms to determine the intersections between convex polyhedra, I chose to implement a random point-sampling method to determine connectivity as it is very efficient.

3.2.1 The Sampling Algorithm

For each chair, I sample a fixed number of points, S. I determined empirically that sampling the cuboids evenly rather than by volume achieved the best results. So for each cuboid, I sample $N_i = \frac{S}{N}$ points where N is the number of cuboids in the chair. I chose $S = 20000$, a number of samples that allows for accurate connectivity calculations without being too computationally expensive. I also determined empirically that sampling the surface of each cuboid achieved better results than sampling the volume of each cuboid. As each cuboid has 6 faces, and each rectangle face is composed of two triangles, each cuboid is defined by 12 triangles. I divide the samples for one cuboid to be roughly even amongst all 12 triangles.

3.2.2 Two-Cuboid Connectivity

Given cuboid A and cuboid B, the connectivity between A and B is defined as follows. Let $\mathbf{S}_a$ and $\mathbf{S}_b$ denote the sets of point samples from cuboid A and cuboid B respectively, and let $d_H(X, Y)$ be the Hausdorff distance metric between subsets X and Y of some metric space:

$$\text{conn}(A, B) = d_H(\mathbf{S}_a, \mathbf{S}_b) \leq \alpha_{conn} \tag{3.1}$$

Because this sampling procedure is random, I must define a minimum distance below which two cuboids are defined as connected, α_{conn}. I assign α_{conn} to be 0.02 as determined by examining training data. For each chair in the training data, I determined what the maximum value for the connectivity threshold could be that would still define this chair as connected. The point sampling is random and the training data is not perfect, so picking the value for the threshold that classifies all chairs in the dataset as connected would not allow for a strong metric of connectivity. As can be seen in Figure 3.2, I chose a threshold such that most of the data would be classified as connected but increasing it would not yield much higher connectivity.

3.2.3 Chair Connectivity

To determine the connectivity of a chair, I first calculate the connectivity graph between cuboids in the chair. I then use a depth first search protocol to calculate the connected components of cuboids within each chair. If there is a single connected component, the chair is connected. Otherwise, the chair is disconnected.

3.3 The Loss Function

With the connectivity metric defined, it is now possible to design a differentiable loss metric to improve connectivity. If a chair is connected, then it has a single component and its loss is zero. If a chair is not connected, then it must have multiple

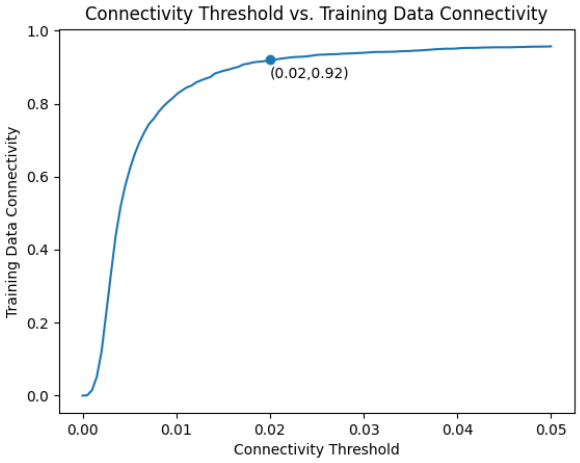


FIGURE 3.2: The graph of how many chairs in the dataset are considered connected based on the connectivity threshold

components. Whichever component contains the most cuboids is designated as the main component c_m. The loss for each other component c_j is calculated as follows. Let L represent the standard method of computing the length of a vector between two points in Euclidean space. For each sample $\mathbf{s}_j$ in component c_j,

$$\text{dist}_{c_m}(\mathbf{s}_j) = \min_{\mathbf{s}_m \in c_m} L(\mathbf{s}_j, \mathbf{s}_m) \tag{3.2}$$

This is simply the minimum distance between point $\mathbf{s}_j$ and all point samples in the main component c_m. Once the distance from $\mathbf{s}_j$ and c_m is calculated for every $\mathbf{s}_j$ in c_j, the distances are then sorted. The loss for c_j is then calculated as follows.

$$\text{loss}(c_j) = \sum_{j < \alpha_{limit} * N_j} (\text{dist}_{c_m}(\mathbf{s}_j) - \alpha_{conn}) \tag{3.3}$$

N_j is the number of point samples in the component c_j, and α_{limit} is a hyperparameter for the number of point samples in c_j to penalize, which is defined to be 0.02. It was empirically determined that penalizing all the points in the component would cause a "clumping" behavior in which cuboids would converge to a point so that each point on the cuboid would be minimally close to the main component. The total connectivity loss over the chair is calculated as

$$\text{loss}_{\text{conn}} = \sum_{c_j \neq c_m} \text{loss}(c_j) \tag{3.4}$$

3.4 The Results

To test this loss metric, I begin with the ShapeAssembly variational auto-encoder pre-trained with only supervised losses on the entire dataset for 120 epochs. I trained this model for 200 epochs with the connectivity loss and generated 100 chairs every 25 epochs. As can be seen in Figure 3.3, the pre-trained model achieves a connectivity of approximately 72%, and through training with the connectivity loss metric, the connectivity increases to around 95%, superseding the connectivity of the training set of 92%. Table 3.1 shows how the chairs produced by the generative model

have qualitatively changed over the course of training, displaying the decoding of the same vector before and after training.

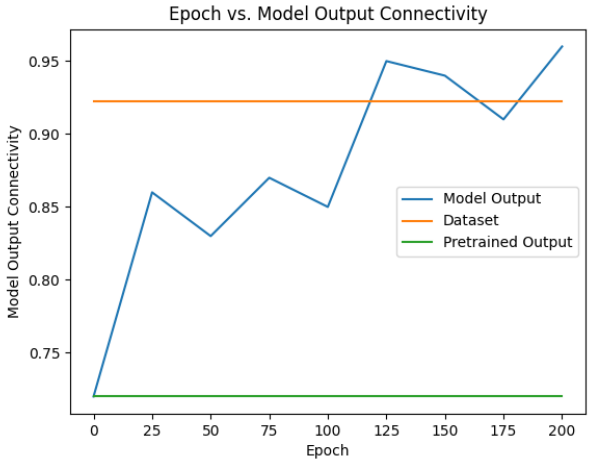


FIGURE 3.3: The results of training with the connectivity loss metric

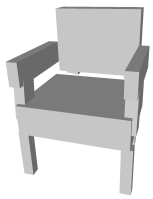
Before Connectivity Loss Training	After Connectivity Loss Training
	
	

TABLE 3.1: A random latent vector decoded into a chair with the pre-trained model and with the connectivity loss trained model after 125 epochs. The chair from the pre-trained model is disconnected, whereas the chair from the connectivity loss trained model is not. The left column is the chair decoded by the pre-trained model from two different views, and the right column is the chair decoded by the connectivity loss trained model from the same two views.

Chapter 4

Enforcing Chair Stability

4.1 Motivation

When limiting one's consideration to stationary chairs that do not move in any manner, unlike a rocking chair or a wheelchair, it is possible to define a specific notion of stability. A chair is stable if it can maintain one fixed position on the ground when placed on its base. It follows that the part of the chair that determines its stability is its base, which can be generally defined as the parts of the chair that touch the floor (or should touch the floor) when a chair is placed.

There are physical requirements on the base of a chair in order for it to be stable. The whole base must be able to touch the ground at one time, which means the base should be entirely co-planar. Moreover, the chair must be such that when placed on its base, its center of mass lies above the base. Without this restraint, the chair would topple over when placed on the ground. A chair that does not satisfy these constraints cannot stand stationary and does not function as a stationary object, which is a general expectation of chairs.

The ShapeAssembly generative model does not impose these requirements on the chairs it generates, which leads to some of the produced chairs being unstable. Figure 4.1 displays two chairs produced by the model that would not be stable if placed on the ground. The goal of this section is to determine if it is possible to use an unsupervised loss to increase the stability of the chairs outputted by the generative model.

FIGURE 4.1: Chairs generated by the model that are unstable. The chair on the right is also disconnected as well.

4.2 The Metric of Stability

As previously stated, the base of a chair is a determining factor of its stability. ShapeAssembly chair programs, however, have no indication of which cuboids in a

chair constitute its base. Thus in order to determine if the chair's base allows it to be stable, it must first be determined which parts of the chair constitute its base.

4.2.1 Determining the Base

The method of discerning the base used here was calculated heuristically. Consider the convex hull of a stable chair. In a perfect chair, the surface defined by the base should exactly line up with some co-planar faces of the convex hull. Here I assume that even in an unstable chair with base that is not co-planar, it's possible to assume that some of the points on the convex hull of the chair are part of the base and would make a stable base if co-planar. Therefore, I can restrict the search for the faces that make up the base to those of the convex hull of the chair. Figure 4.2 shows a chair, its convex hull, and its base.

FIGURE 4.2: The figure on the right is the convex hull of the chair in the left figure. The base of the chair is highlighted in a darker red.

At this point, I want to filter out some faces of the convex hull that I believe are unlikely to be a part of the base. The faces of the base of each the chairs in the ShapeAssembly chair program dataset should be on the ground and should face the ground. It was empirically determined that most chairs generated by the model trained on this dataset are approximately the same: their bases have faces that are generally toward the bottom of the chair and face the ground. From this I can filter out faces that I believe would not be in the base. Let min and max be the minimum and maximum y-coordinate points on the chair, respectively. Consider the face f in the convex hull of the chair and let $\mathbf{v}$ be its points and n_y be the y-component of its outward-facing normal.

$$\begin{aligned} \text{criterion}_1(f) &= \bigcap_{\mathbf{v} \in f} (\mathbf{v}_y - min < k \cdot (max - min)) \\ \text{criterion}_2(f) &= n_y < \theta \\ \text{is\_candidate}(f) &= \text{criterion}_1(f) \cap \text{criterion}_2(f) \end{aligned} \tag{4.1}$$

The boolean flags criterion_1 and criterion_2 are true for a face if it satisfies the criteria described before. In this equation k and θ are hyperparameters which allows for more rigid or more lenient filtering of candidate faces of the base. It was empirically determined that $k = 0.5$ and $\theta = \frac{\sqrt{2}}{2}$ allow for the best candidate faces.

From the remaining candidate faces, it was empirically determined that the best way to discern which faces are a part of the base is to measure the angle between each face and the largest remaining face. Let $\mathbf{n}_L$ be the largest face's normal, let $\langle \mathbf{x}, \mathbf{y} \rangle$ denote the Euclidian inner product of $\mathbf{x}$ and $\mathbf{y}$, and let f be an arbitrary candidate

face.

$$\text{is\_base}(f) = \langle \mathbf{n}_f, \mathbf{n}_L \rangle < \phi \tag{4.2}$$

In this equation, ϕ is a hyperparameter which allows for more rigid or more lenient filtering of faces in the base. It was empirically determined that $\phi = .9$ allows for the best base-faces. It follows that from the faces of the base, it is possible to determine which points of the chair constitute the base. Figure 4.3 displays the result of this process with each red area representing the base of the chair above it.

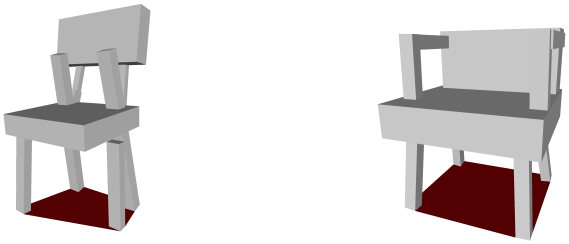


FIGURE 4.3: Chairs and their bases as determined by the heuristic

4.2.2 Flat Base Stability

I now know which points are in the base, but do not know how the base will lie on the ground. In order to know this, I need to know how to rotate the chair such that the plane that the base lies in is parallel with the negative y-direction. As the points determined to be in the base might not be co-planar, however, I first must find the plane that best fits these points. I use least squares regression to determine the equation for the plane that best fits the points in the base. I then construct a rotation matrix that will rotate this plane to the negative y-plane. Doing this will essentially rotate all the points in the chair such that the base plane is parallel to the negative y-plane.

Once the base has been rotated, it is possible to determine if the base is appropriately flat. Realistically, it's impossible to require that all points on the base be exactly co-planar as the points are able to span the entire field of real numbers, so a base is considered co-planar if it is approximately flat. Let $\mathbf{V}$ be the vertices of the base.

$$\text{is\_flat} = \bigcap_{\mathbf{v} \in \mathbf{V}} (\mathbf{v}_y - \min_y(\mathbf{V}) < \alpha_{base}) \tag{4.3}$$

α_{base} is the amount of deviation allowed from the base. Because this process is heuristically based, and because the chairs in the dataset are not perfect, picking a base deviation threshold of zero would classify almost all of the chairs in the training dataset as unstable, which should not be the case. As can be seen in Figure 4.4, I chose a base deviation threshold such that most of the data would be classified as having a co-planar base but increasing it would not yield much a higher proportion. The value of α_{base} used is 0.03.

4.2.3 Topple Stability

The second component of stability is discerning if the chair will topple when placed on its base. When the chair has been placed on the ground, it will topple if its center

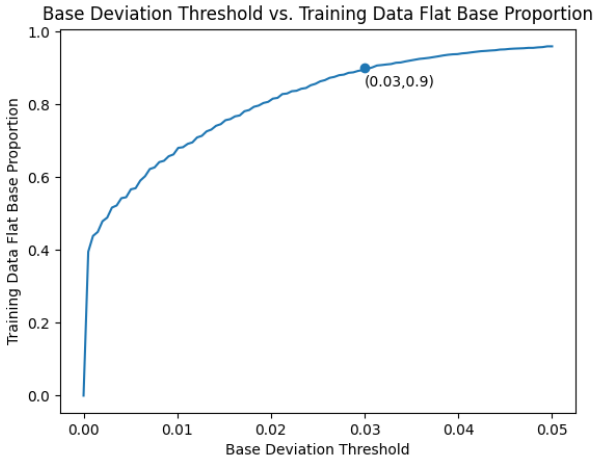


FIGURE 4.4: The graph of how many chairs in the dataset are considered to have co-planar bases based on the base deviation threshold

of mass does not lie over the 2-dimensional convex hull of its base. Using the previously calculated rotation matrix, it's possible to rotate the entire chair in the same way that the base was rotated.

As stated before, the center of mass must lie above the convex hull of the base. How far above the convex hull the center of mass lies is not important, so it is not necessary to consider the y-component of the convex hull of the base or the y-component of the center of mass. Thus, I project the convex hull of the base and the center of mass of the chair onto the xz-plane and treat the convex hull as a two-dimensional convex hull. Let s be the signed distance of the projected center of mass projected from the projected convex hull of the base. The chair is topple stable and the boolean flag is_topple_stable is true if and only if $s < 0$.

The chair is stable if is_flat is true and is_topple_stable is true.

4.3 The Loss Function

With the metric of stability determined, it is now possible to construct a differentiable loss function to improve stability. If a chair has a flat base, then its flat-base loss is zero. If a chair does not have a flat base, the loss is as follows. Let $\mathbf{V}$ be the vertices of the base.

$$\text{loss}_{\text{flat}} = \sum_{\mathbf{v} \in \mathbf{V}} \max(v_y - \min_y(\mathbf{V}) - \alpha_{base},\ 0) \tag{4.4}$$

If the chair will not topple when placed, then its topple loss is zero. If it will topple when placed, then the loss is as follows. $\text{loss}_{\text{topple}}$ is s as defined in Section 4.23, which is the signed distance of the projected center of mass from the projected convex hull of the base.

The total stability loss of the chair is calculated as

$$\text{loss}_{\text{stab}} = \text{loss}_{\text{flat}} + \text{loss}_{\text{topple}} \tag{4.5}$$

4.4 The Results

To test this loss metric, I begin with the ShapeAssembly variational auto-encoder pre-trained with only supervised losses on the entire dataset for 120 epochs. I trained this model for 200 epochs with the stability loss and the connectivity loss and generated 100 chairs every 25 epochs. When testing this model with the stability loss but without the connectivity loss, the model would sometimes learn to produce semantically invalid chairs which cannot be decoded. I believe this behavior might have been due to the "base flatness" component of the loss, as the loss tends to "pull" the legs that induce non-planarity away from the chair and must be balanced by the connectivity loss which pulls them toward the chair.

As can be seen in Figure 4.5 below the pre-trained model achieves a stability of approximately 72%. The model output hovers around this value until epoch 125, at which it starts to increase. I believe this is because around epoch 125, the connectivity is very high, and thus the connectivity loss will be lower. This could allow the stability loss to more effectively train the model as the two losses might work against each other when chair legs both float and are non-planar within the base. Through training with the stability loss metric, the stability increases to around 90%, roughly equal to the the stability of the training set. Table 4.1 shows how the chairs produced by the generative model have qualitatively changed over the course of training, displaying the decoding of the same vector before and after training.

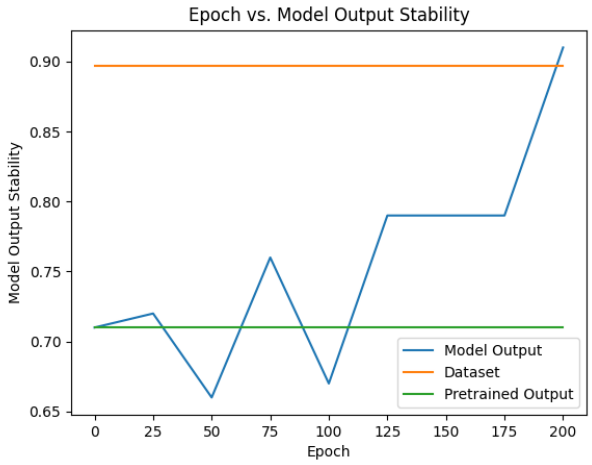


FIGURE 4.5: The results of training with the stability loss metric

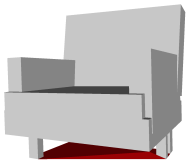
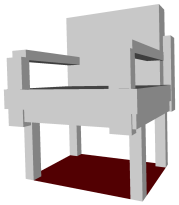
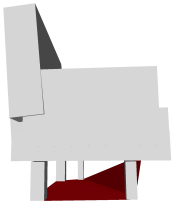
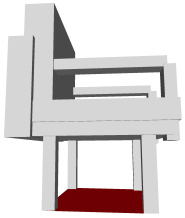
Before Stability Loss Training	After Stability Loss Training
	
	

TABLE 4.1: A random latent vector decoded into a chair with the pre-trained model and with the stability loss trained model after 125 epochs. The chair from the pre-trained model is unstable as one of the legs is in a bad position, whereas the chair from the stability loss trained model has the leg in a more stable position. The left column is the chair decoded by the pre-trained model from two different views, and the right column is the chair decoded by the stability loss trained model from the same two views.

Chapter 5

Increasing Chair Comfort

5.1 Motivation

Beyond creating chairs that can function independent of human interaction, the generative model should be able to create chairs that function in regards to human-related definitions of functionality. This leads to the next aspect of functionality: comfort. As chair programs from the ShapeAssembly dataset are modeled after existing chairs, they should be generally comfortable. It is completely possible, however, for the chairs produced by the generative model to be uncomfortable. The geometry of the chairs produced by the generative model need not fit a human body or support comfortable poses. The motivation of this section is to discern if it is possible to determine how comfortable a chair would be when sat in and create an unsupervised loss function to increase the comfort of the chair.

5.2 The Metric of Comfort

It is not immediately obvious how to calculate the metric of comfort. Comfort is subjective and depends on body proportions and preference. A person who likes to lay down might find a reclining chair more comfortable than a person who likes to sit up. A chair designed for people of average height might not support someone whose height drastically differs from the average. Moreover, one could sit in a chair in infinitely many different poses, and some of these poses will inherently be uncomfortable.

I propose that there is a general way to gauge a person's basic comfort in a chair. A comfortable chair should support certain areas of the body: the back and the glutes. While it's not possible to conclude that increasing the amount of contact between the back/glutes and the chair will make someone more comfortable, if a person sits in a chair and makes little-to-no contact to the chair with their back or glutes then they will not be very comfortable. Thus the amount of contact between the back/glutes and the chair can be used as a basic measurement of comfort. This poses the question, how does one compute how much back/glutes-to-chair contact a person might make with a chair? The answer is to use the pose that maximizes this contact, denoted as the optimal pose. From this, it's possible to define the next objective: a method of determining this optimal pose given a human body.

5.2.1 The Human Body Model

Before defining the optimal pose algorithm, it's necessary to first understand the human body model used in this process. The model used in this project is the SMPL model [10], a realistic 3D model of the human body that is based on skinning and blend shapes learned from thousands of 3D body scans. Figure 5.1 displays many

different SMPL models with different poses, shapes, and sexes. A SMPL model takes as input a biological sex, body shape parameters, and pose parameters and outputs a mesh with positions of various joints in the body. Ideally, I would derive a comfort loss metric to work over all biological sexes and body shape parameters. But, as an initial investigation, I started by fixing the sex and the body parameters.

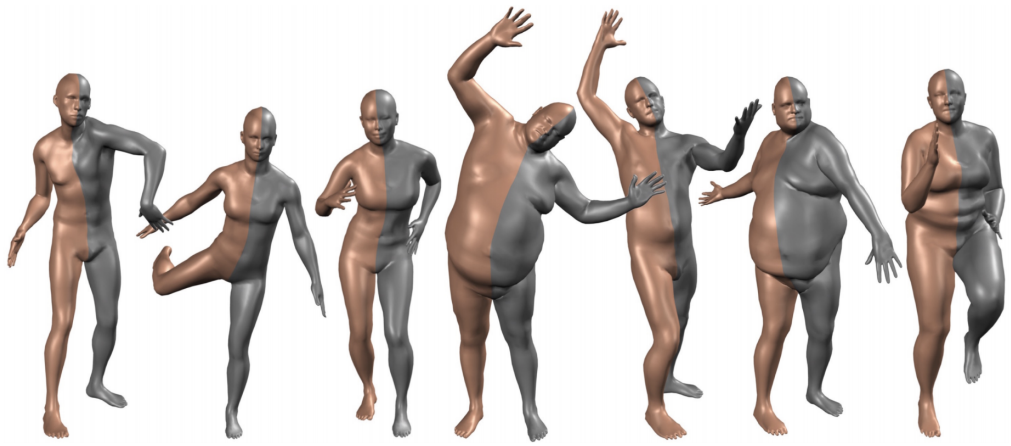


FIGURE 5.1: A figure containing SMPL body models with various sexes, poses, and body parameters taken directly from the SMPL paper [10].

The SMPL model contains 25x3 pose parameters, one for global orientation, one for body global translation, and 23 angle-axis vectors defining the rotation of the body's joints from a base pose. A real life human body has a restricted range of rotation for all joints, but the SMPL model does not restrict rotation in any way. Therefore, it is possible for a SMPL body model to take on a pose that is humanly impossible. Thus the optimal pose algorithm must take into account invalid poses.

5.2.2 The Optimal Pose Algorithm

The approach used here is based on that of the paper "Resolving 3d human pose ambiguities with 3d scene constraints" [11]. The authors use an optimization procedure to settle a body model into a location in a scene based on an initial estimate derived from an image. This algorithm follows a similar approach, however, it does not have any sort of image or initial estimate of where the body will be. To compensate for this lack of prior knowledge on where to place the body, I use a physics-based loss component.

The goal of this algorithm is to initialize a body in a neutral sitting position over a chair and then use an optimization process over different loss components to lower the body into the chair into a physically valid and comfortable pose. The loss over which the body's pose is optimized consists of three components: the physics component, the human body component, and the comfort component.

The Physics Component

The physics component is used to optimize the body model's pose focuses on approximating a physical simulation. The first part of this component is the gravity loss. As the body is instantiated over the chair, the gravity-based loss is used to

lower it into the chair. However, it is not immediately obvious how to define this loss. As the gradient of the gravitational loss should represent the force on the body, the gravitational loss itself should represent the gravitational potential energy of the body. However, penalizing just the center of mass of the body by its gravitational potential energy will not encourage any change in body's pose. On the other hand, treating each vertex in the body as a small mass will cause bending in the body's bones. The solution is to penalize the joints of the body, as those are points around which bones rotate. However, the joints are not evenly distributed throughout the body, so areas with more joints will experience higher gravitational pull than areas with less. To fix this, I weight each joint using a Voronoi diagram-based approach. Let $\mathbf{J}$ be the joints of the body, let $\mathbf{j}_i$ be an arbitrary joint, and let $\text{vol}(\mathbf{j}_i)$ measure the volume of the section of the body whose closest joint is $\mathbf{j}_i$.

$$w(\mathbf{j}_i) = \frac{\text{vol}(\mathbf{j}_i)}{\sum_{\mathbf{j}\in\mathbf{J}} \text{vol}(\mathbf{j})} \tag{5.1}$$

With the joint weights, it is now possible to define the gravity loss. Let y_{min} be the lowest y-coordinate of the chair.

$$\text{loss}_{\text{gravity}} = \sum_{\mathbf{j}\in\mathbf{J}} (j_y - y_{\text{min}}) \cdot w(\mathbf{j}) \tag{5.2}$$

The second part of the physics component is the penetration component, which deals with the body penetrating the chair. This loss directly penalizes any vertex of the body mesh for penetrating the chair as it is physically impossible to occupy the same space as another object. In order to determine if a vertex is penetrating the chair, it is only necessary to calculate the signed distance of the vertex over the chair. It follows that the signed distance of a vertex over the chair is the minimum signed distance of the vertex over all of the cuboids of the chair. Now it is possible to define the penetration loss. Let $\mathbf{V}$ be the vertices of the body mesh, let $\text{sd}_C(\mathbf{v})$ denote the signed distance of vertex $\mathbf{v}$ over chair C, and let $|x|$ denote the absolute value of x.

$$\text{loss}_{\text{pen}} = \sum_{\mathbf{v}\in\mathbf{V}} |\min(\text{sd}_C(\mathbf{v}),\ 0)| \tag{5.3}$$

The last part of the physics component is the self-penetration component. As the body is a non-convex deformable mesh, it is possible for the body to intersect itself, which is physically impossible. Thus there must be a loss component to prevent this from occurring. The authors of "Resolving 3d human pose ambiguities with 3d scene constraints" [11] also use a self-penetration loss function derived from the papers "Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees" [12] and "Capturing Hands in Action using Discriminative Salient Points and Physics Simulation" [13], using a bounding volume hierarchy to efficiently calculate a loss metric for body mesh self-intersection. I use this loss function as the self-penetration loss, $\text{loss}_{\text{self-pen}}$.

The physical loss component is calculated as a weighted sum over these three loss parts.

$$\text{loss}_{\text{physical}} = \alpha_{\text{gravity}}\text{loss}_{\text{gravity}} + \alpha_{\text{pen}}\text{loss}_{\text{pen}} + \alpha_{\text{self-pen}}\text{loss}_{\text{self-pen}} \tag{5.4}$$

The constants $\alpha_{\text{gravity}}, \alpha_{\text{pen}}, \alpha_{\text{self-pen}}$ were determined empirically as $\alpha_{\text{gravity}} = 20$, $\alpha_{\text{pen}} = 2.5$, $\alpha_{\text{self-pen}} = 10$.

The Human Body Component

The human body component of the optimization loss is used to restrict the body model's poses to a smaller set of poses. As stated beforehand, the SMPL model does not take into account human body constraints. Thus, the optimization procedure must contain a loss component restricting the pose of the body model to only humanly possible poses. Therefore, the first part of the human body component is the invalid pose loss, which penalizes the body for being in humanly impossible poses. This loss calculation is derived from the pose validity classification algorithm detailed in "Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction" [14]. Their formulation uses pose priors determined from a large dataset of human body poses to determine if a given pose is humanly possible. Their algorithm is implemented in MATLAB and takes as input the joints of the body and classifies a pose as valid or invalid. The original algorithm takes as input the vectors representing the major bones in the body. These bones can be seen in Figure 5.2.

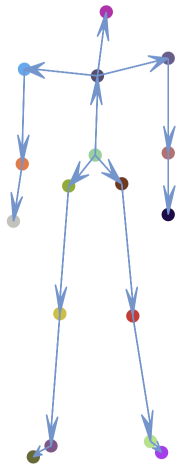


FIGURE 5.2: A figure of the different bone vectors used to compute pose validity. This image is taken directly from the paper "Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction" [14].

Each bone vector is then rotated from the global coordinate space to its local coordinate space, which is the direction it points in relative to the direction in which its parent bone vector points. Bone vectors that do not have parent bone vectors are left in global coordinate space. Then, each bone vector is re-parameterized from Cartesian coordinates (x, y, z) to 3D polar coordinates (ϕ, θ, r). The authors of this algorithm, through extensive motion capture, were able to model the extent of human poses as binary functions of these polar coordinates. For each bone, its ϕ and θ angles and the ϕ and θ angles of its parent bone (if it has a parent bone) are used to determine if it is in a valid pose. The authors created a discrete grid of values for each value of (ϕ, θ) coordinate where the value at each coordinate is zero if the pose is invalid and one if the pose is valid. From this discrete binary classification grid, I derived a discrete grid of values representing the signed distance from the boundary between zero and one in the initial grid, where negative values represent having a valid pose. An example of one bone's binary classification grid and resulting signed distance grid can be seen in Figure 5.3.

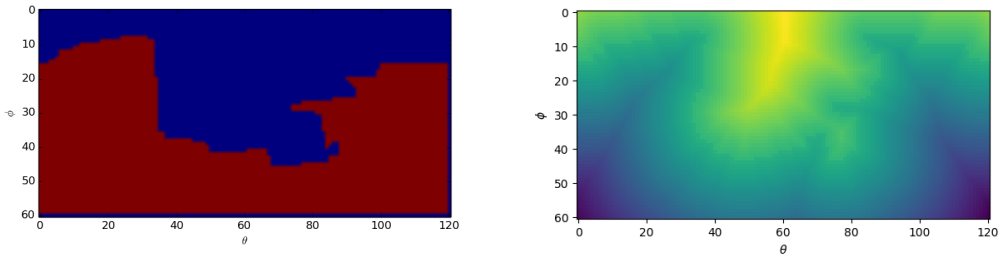


FIGURE 5.3: The binary classification grid for a bone as derived by the authors of "Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction" [14], and the grid representing the signed distance function of each coordinate from the boundary between values on the binary classification grid. On the left figure blue represents zero and red represents one. On the right figure, yellow represents higher values whereas dark blue represents lower values.

For any angle pair (ϕ, θ) the value of that pair in grid G is defined as the bilinear interpolation of that point on G. The angle pair is differentiable with respect to the pose and the bilinear interpolation is differentiable with respect to the angle, so any value computed is differentiable with respect to the pose. If a bone is invalid, then bilinear interpolation using its angle pair (ϕ, θ) will yield a positive value. Optimizing over this value will push the angle pair to have a lower value and eventually have a negative value and thus a valid pose. Thus I define the loss over a single bone to be the value of the bilinear interpolation function of the bone's angle coordinates over this grid if the value is positive, otherwise the loss is zero. The total invalid pose loss $\text{loss}_{\text{invalid}}$ is the sum of the loss of each bone vector.

The second part of the human body component is the symmetry loss. People generally sit in chairs such that their bodies are laterally symmetrical. In order to minimize the gravitational loss component, however, the optimization procedure will cause the body model to bend and rotate such that it is not laterally symmetrical. As described earlier, the SMPL model's pose parameters contain 23x3 angle-axis vectors defining the rotation of the body's joints from a base pose. Each of these joints can be split into one of two groups: the group of pair joints, and the group of solo joints. For two pair joints to be laterally symmetric, their angle-axis vectors must have the same x-values and opposite y and z-values. For a solo joint to be laterally symmetric, its angle-axis vector must have 0 for y and z. For the other two remaining pose parameters, the global orientation and the translation, lateral symmetry means that the orientation should have no rotation along the y and z-axes and the translation should have no component along the x-axis. From this, it is possible to define the symmetry loss. Let $\mathbf{A}$ denote all angle-axis vectors, $\mathbf{p}_a, \mathbf{p}_b$ be the angle-axis vectors of two pair joints, let $\mathbf{p}$ be the angle-axis vector of a solo joint,

and let $\mathbf{v}$ denote the vector $(1,\ -1,\ -1)$.

$$\begin{aligned}
\text{loss}_{\text{pair}} &= \sum_{\mathbf{p}_a, \mathbf{p}_a} \sum_{i \in x,y,z} |p_{a,i} - p_{b,i} \cdot v_i| \\
\text{loss}_{\text{solo}} &= \sum_{\mathbf{p}} (|p_y| + |p_z|) \\
\text{loss}_{\text{sym}} &= \text{loss}_{\text{pair}} + \text{loss}_{\text{solo}} + |p_{transl,x}| + |p_{orient,y}| + |p_{orient,z}|
\end{aligned} \tag{5.5}$$

The last part of the human body component is the neck and back loss. With the imposition of a symmetry loss, the optimization procedure will compress the neck and back of the human body model downwards by rotating the joints over the x-axis in order to minimize gravitational loss. While these poses are humanly possible, they are not particularly comfortable and will yield a non-optimal pose. The solution to this is to penalize the deviation of the x-axis component of each neck and back joint's angle-axis vector. However, rather than penalize the deviation from zero, I penalize the deviation from the initial value, as in this case it is not zero for some of the back joints. Thus the back and neck loss, $\text{loss}_{\text{back+neck}}$, is equal to the sum of the absolute difference between the initial x-rotation of the neck and back joints, and the current x-rotation of the neck and back joints.

$$\text{loss}_{\text{human}} = \alpha_{\text{invalid}} \text{loss}_{\text{invalid}} + \alpha_{\text{sym}} \text{loss}_{\text{sym}} + \alpha_{\text{back+neck}} \text{loss}_{\text{back+neck}} \tag{5.6}$$

The constants $\alpha_{\text{invalid}}, \alpha_{\text{sym}}, \alpha_{\text{back+neck}}$ were determined empirically as $\alpha_{\text{invalid}} = 1,\ \alpha_{\text{sym}} = 2.5,\ \alpha_{\text{back+neck}} = 10$.

The Comfort Component

The final component of the optimal pose algorithm is the comfort component. As the primary objective of the optimization procedure is to find a pose that optimizes back and glutes contact with the chair, this component, $\text{loss}_{\text{objective}}$ is simply the a function of the distance of vertices on the back and neck from the chair. Let $\mathbf{V}$ denote all vertices on the back and glutes, and let $\text{sd}_C(\mathbf{v})$ denote the distance of a vertex $\mathbf{v}$ from chair C.

$$\text{loss}_{\text{objective}} = \frac{1}{|\mathbf{V}|} \sum_{\mathbf{v} \in \mathbf{V}} max(\text{sd}_C(\mathbf{v}) - \alpha_{chair-thresh}, 0) \tag{5.7}$$

$\alpha_{chair-thresh}$ is a hyperparameter denoting the distance at which a vertex is close enough to the chair to be in contact and is set as $\alpha_{chair-thresh} = 0.001$

5.2.3 Initial Parameters and Hyperparameters

The initial parameters and hyperparameters used in this algorithm were determined empirically. The initial global orientation for the SMPL model used is $(-.75,\ 0,\ 0)$. The optimal initial translation is a function of the maximum and minimum dimensions of the chair, set as $(0,\ max_y + 0.2,\ 0.5min_z + 0.5max_z)$. The initial pose as well as the sex and body shape parameters were determined by finding motion capture data from a man in a sitting pose from the AMASS dataset [15], and can be seen in Figure 5.4. The learning rate used here is 0.001, and the number of iterations used is 300.

FIGURE 5.4: The body model in its initial pose.

5.2.4 The Results

In order to determine if the comfort loss would act as a valid metric over the chairs, I first sought to determine the distribution of comfort loss values over the chairs in the ShapeAssembly dataset. The following figure represents this distribution, where the height of each bar represents the percentage of chairs in the dataset that lie within that bar's lower and higher limits.

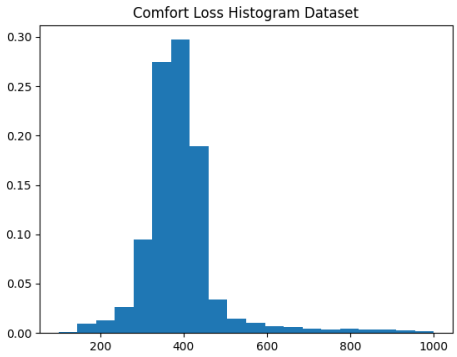


FIGURE 5.5: The normalized distribution of comfort loss values of the chairs in the dataset

The comfort loss follows a normal distribution, as can be shown in Figure 5.5. I qualitatively determined whether the simulation process properly sat the body model into the chair such that the back and glutes touched the chair if possible. In most chairs in which a person could sit "normally", the body model's final position is a normal sitting position. However, the body model does occasionally finish in a non-optimal position. Table 5.1 displays the body model in its optimal pose for multiple chairs, and Table 5.2 displays the optimized body model in the same chair with different loss components removed from the optimization process.

Low Loss Optimal Pose	High Loss Optimal Pose	Failed Optimal Pose
		

TABLE 5.1: Some results of the optimal pose algorithm over different chairs. On the left is a chair in which the body settles into an optimal pose with a low loss. In the middle is a chair in which the body settles into an optimal pose with a high loss. On the right is a chair in which the body does not settle into an optimal pose, as the glutes could touch the chair, but do not.

Full Optimization	No Symmetry Loss	No Human Body Component
		

TABLE 5.2: Resulting poses of the optimal pose algorithm over the same chair with certain loss components removed. On the left is the optimization with all loss components. In the middle is the optimization without the symmetry loss. On the right is the optimization without the human body component.

5.3 The Loss Function

Once the optimal sitting position has been determined, the comfort loss is simply equal to the comfort component loss. However, the derivation of the optimal sitting position is computationally expensive and takes upwards of a minute for each chair. Training a full neural network with such a loss function would take an extremely long time, so for this reason I replace this loss with a loss proxy.

5.3.1 The Loss Proxy

The loss proxy is itself a neural network that takes as input a chair and outputs the comfort loss value of that chair. As a neural network is just a function, the proxy will be differentiable. As it is a differentiable function, it can then be used to provide gradients to descend the comfort loss when training the generative model. What is not immediately obvious, however, is how to turn a ShapeAssembly chair program into a differentiable input for a loss proxy.

I determined that I could take a 3D grid of samples over the signed distance function of the chair to use as the input, which I could operate on with a 3D convolutional neural network. The signed distance function of the chair is sampled over

its bounding box at intervals to provide a $N \times N \times N$ grid, where N is the resolution of the sampling. The optimal value for N was empirically determined to be 64. Figure 5.6 displays a chair and a version of its sampled grid modified to allow one to see the sdf over the chair.

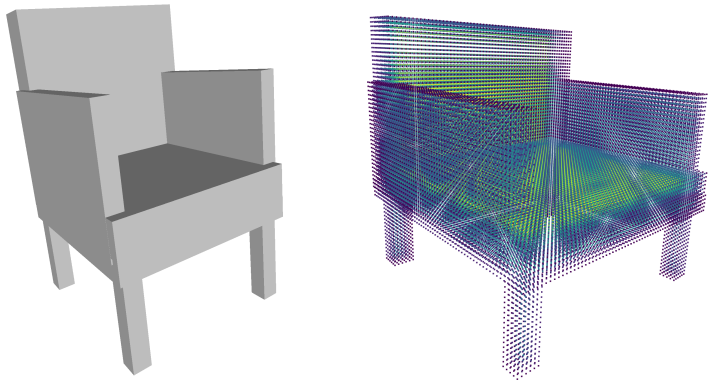


FIGURE 5.6: The figure on the left is the chair in a mesh representation, and the figure on the right is the chair in its grid representation. Each point in the grid representation is a point at which the signed distance function of the chair is sampled. Note that grid on the right is not the full grid used as input, it only contains samples that are inside the chair in order to help visualize the sdf within the chair.

The label for each chair is the comfort loss value determined through the optimal pose algorithm.

In order to train the proxy model it was necessary to derive a loss value for the loss proxy. I initially considered setting the loss to be the direct difference between the value the loss proxy predicts and the actual comfort loss value of the chair. However, this did not work as the model's outputs would collapse to the mean value of the training set.

I instead used a contrastive training paradigm. Contrastive training does not focus on the absolute accuracy of the model's predictions, but rather the comparative accuracy of the model's predictions. Given two chairs, the model should be able to differentiate which chair has a higher comfort loss value. Let l_0 and l_1 be the model's predictions for chair 1 and chair 2, respectively. Note that l_0 and l_1 can be any real value, they need not be probabilities that sum to one. Also let c_0 and c_1 be the true comfort loss values for chair 1 and chair 2, respectively. Let σ be the sigmoid function.

Algorithm 1 Loss function

if $c_0 > c_1$ **then**
  $\text{loss} = -\log(\sigma(l_0)) - \log(1 - \sigma(l_1))$
else
  $\text{loss} = -\log(1 - \sigma(l_0)) - \log(\sigma(l_1))$
end if

A model that achieves perfect contrastive accuracy would be able to differentiate between a chair that produces a higher loss and a chair that produces a lower loss, and would therefore be able to provide a gradient over a chair's parameters to lower its comfort loss.

5.4 The Results

To test the efficacy of the comfort loss proxy, I first trained the generative model, starting with the pre-trained model trained for 120 epochs. I sampled chairs from the pre-trained model and compared them with chairs generated by the model trained with the comfort loss proxy, and plotted their comfort loss distributions following the same procedure. In Figure 5.7 is possible to see that the comfort loss significantly shifts the distribution of comfort loss in the chairs generated by the model. Table 5.3 shows how the chairs produced by the generative model have qualitatively changed over the course of training, displaying the decoding of the same vector before and after training.

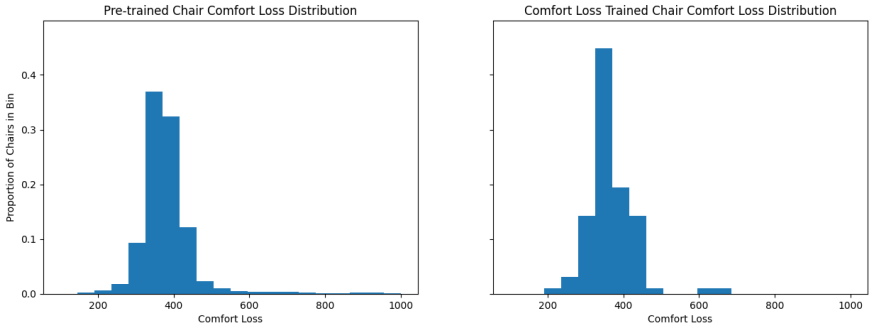


FIGURE 5.7: The normalized distribution of comfort loss values of the chairs generated by the pre-trained and comfort loss trained models. The x-axis is the comfort loss, and the y-axis is the proportion of chairs within each bin.

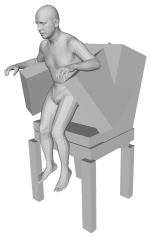
Before Comfort Loss Training	After Comfort Loss Training
	

TABLE 5.3: A random latent vector decoded into a chair with the pre-trained model and with the comfort loss trained model after 125 epochs. The chair from the pre-trained model has a comfort loss above 500, whereas the chair from the comfort loss trained model has a comfort loss below 500. The left figure is the chair decoded by the pre-trained model, and the right figure is the chair decoded by the comfort loss trained model.

Chapter 6

Conclusion

This paper investigated methods to improve the functionality of chairs generated by the ShapeAssembly generative model. The specific facets of functionality which I attempted to improve centered around person-independent and person-specific notions of functionality. As the metrics of functionality defined in this paper were unsupervised, the model was evaluated by comparing its performance on these metrics compared to that of the ShapeAssembly dataset.

6.1 Limitations

The connectivity metric and the comfort metric rely on the ability to calculate the signed distance of points from a given chair. This works particularly well with ShapeAssembly chairs, as the chair is assembled of cuboids which are convex and only have six faces each. However the signed distance function might not be as useful for a mesh that has many faces or is not easily separated into convex parts. One might consider using a some other distance metric such as the Chamfer distance for such a mesh.

The person-specific metric of functionality used in this paper does not extend beyond chairs. Moreover, there currently does not exist any objective metric of the functionality of an arbitrary object, as most objects are designed to function best under specific conditions and on specific tasks. However, training a model to generate a different class of object with person-specific functionality would only require one to use a metric of functionality specific to that class of object.

6.2 Future Work

Much of the further work to be considered focuses on the generalizability of the comfort metric and any person-specific metric of functionality over chairs. The current loss proxy model is trained with fixed body shape parameters and gender, so it is yet to be determined if the model could train such that it properly approximates the comfort loss when taking these parameters as an additional input.

As well, it has yet to be determined how well the loss proxy extends to other loss metrics. One such loss metric which might be useful to explore in the future would be a pose-based loss metric, i.e. a metric that determines how well a chair fits a pose based on the chair's optimal pose. This loss proxy could then be used to train the generative model to create chairs that fit a certain input pose.

Another possible extension of this research would be to modify the optimal pose algorithm to work on any kind of chair mesh. It would likely be necessary as well to modify the comfort loss model to operate on point clouds instead of grid samplings of the signed distance function of the chair, as some chair meshes might have high

resolution features which cannot be captured well with finite grid samplings. If the loss proxy network is capable of training with these modifications, one could then consider using the proxy model to train any kind of chair mesh generative model.

Bibliography

[1] P. Prusinkiewicz and A. Lindenmayer, "The algorithmic beauty of plants," *The Virtual Laboratory*, p. 1–2, 1996.

[2] P. Müller, G. Zeng, P. Wonka, and L. V. Gool, "Image-based procedural modeling of facades," *ACM SIGGRAPH 2007 papers on - SIGGRAPH 07*, 2007.

[3] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," 2015.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017.

[5] J. Xie, Y. Xu, Z. Zheng, S.-C. Zhu, and Y. N. Wu, "Generative pointnet: Energy-based learning on unordered point sets for 3d generation, reconstruction and classification," 2020.

[6] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "Atlasnet: A papier-mâché approach to learning 3d surface generation," 2018.

[7] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Point2mesh," *ACM Transactions on Graphics*, vol. 39, no. 4, Jul 2020. [Online]. Available: http://dx.doi.org/10.1145/3386569.3392415

[8] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas, "Structurenet: Hierarchical graph networks for 3d shape generation," 2019.

[9] R. K. Jones, T. Barton, X. Xu, K. Wang, E. Jiang, P. Guerrero, N. J. Mitra, and D. Ritchie, "Shapeassembly," *ACM Transactions on Graphics*, vol. 39, no. 6, p. 1–20, Nov 2020. [Online]. Available: http://dx.doi.org/10.1145/3414685.3417812

[10] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model," *ACM Trans. Graph.*, vol. 34, no. 6, Oct. 2015. [Online]. Available: https://doi.org/10.1145/2816795.2818013

[11] M. Hassan, V. Choutas, D. Tzionas, and M. J. Black, "Resolving 3d human pose ambiguities with 3d scene constraints," 2019.

[12] T. Karras, "Maximizing parallelism in the construction of bvhs, octrees, and k-d trees," in *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*. Eurographics Association, 2012, pp. 33–37. [Online]. Available: https://doi.org/10.2312/EGGH/HPG12/033-037

[13] D. Tzionas, L. Ballan, A. Srikantha, P. Aponte, M. Pollefeys, and J. Gall, "Capturing hands in action using discriminative salient points and physics simulation," *International Journal of Computer Vision (IJCV)*, vol. 118, no. 2, pp. 172–193, Jun. 2016. [Online]. Available: https://doi.org/10.1007/s11263-016-0895-4

[14] I. Akhter and M. J. Black, "Pose-conditioned joint angle limits for 3D human pose reconstruction," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) 2015*, Jun. 2015.

[15] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, "AMASS: Archive of motion capture as surface shapes," in *International Conference on Computer Vision*, Oct. 2019, pp. 5442–5451.