

# Neurosymbolic 3D Models: Learning to Generate 3D Shape Programs

Daniel Ritchie



BROWN  
Computer Science

This guy!



**WHO AM I?**





# Brown University

- Located in Providence, Rhode Island
- #14 University in the US (US News)



A photograph of the Brown University Computer Science Department building. The building is a multi-story brick structure with a prominent central glass-enclosed section. Two large trees with green leaves are in the foreground, framing the building. A set of concrete steps leads up to the glass entrance. A semi-transparent dark grey banner is overlaid across the middle of the image, containing the department's name and a list of program details.

# Brown Computer Science Department

- 37 full-time faculty
- 2-year Masters program
- Fully-funded PhD program (5 years)
- #25 for CS Graduate Study (US News)



# Brown Visual Computing



# BVC

## Faculty



James Tompkin



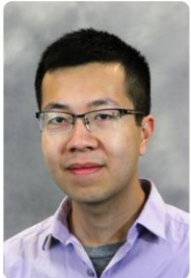
Daniel Ritchie



Srinath Sridhar  
*Starting Fall 2020*



Chen Sun  
*Starting Summer 2021*



Jeff Huang



David Laidlaw



Barbara Meier



John "Spike" Hughes



Andries "Andy" van Dam

<http://visual.cs.brown.edu/>

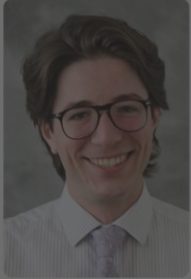
- Nine (9) faculty
- Active research in graphics, vision, HCI, visualization, ...
- Regularly publish in top visual computing venues (SIGGRAPH, CVPR, ICCV, ...)

# Brown Visual Computing



# BVC

## Faculty



James Tompkin



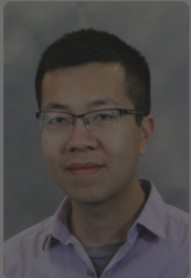
Daniel Ritchie



Srinath Sridhar  
*Starting Fall 2020*



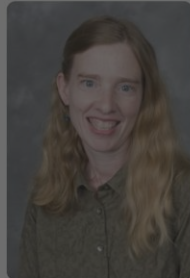
Chen Sun  
*Starting Summer 2021*



Jeff Huang



David Laidlaw



Barbara Meier



John "Spike" Hughes



Andries "Andy" van Dam

- Andy van Dam:  
co-founder of ACM SIGGRAPH  
(pre-cursor to SIGGRAPH)

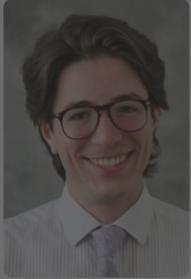
<http://visual.cs.brown.edu/>

# Brown Visual Computing

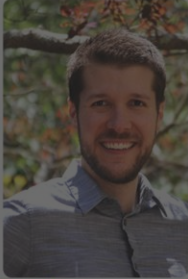


# BVC

## Faculty



James Tompkin



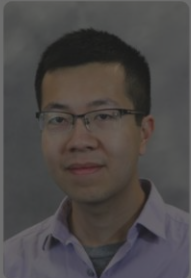
Daniel Ritchie



Srinath Sridhar  
Starting Fall 2020



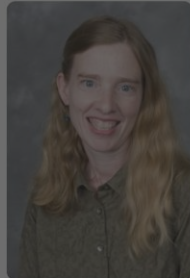
Chen Sun  
Starting Summer 2021



Jeff Huang



David Laidlaw



Barbara Meier



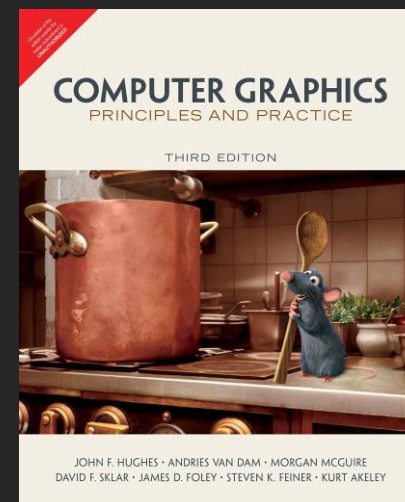
John "Spike" Hughes



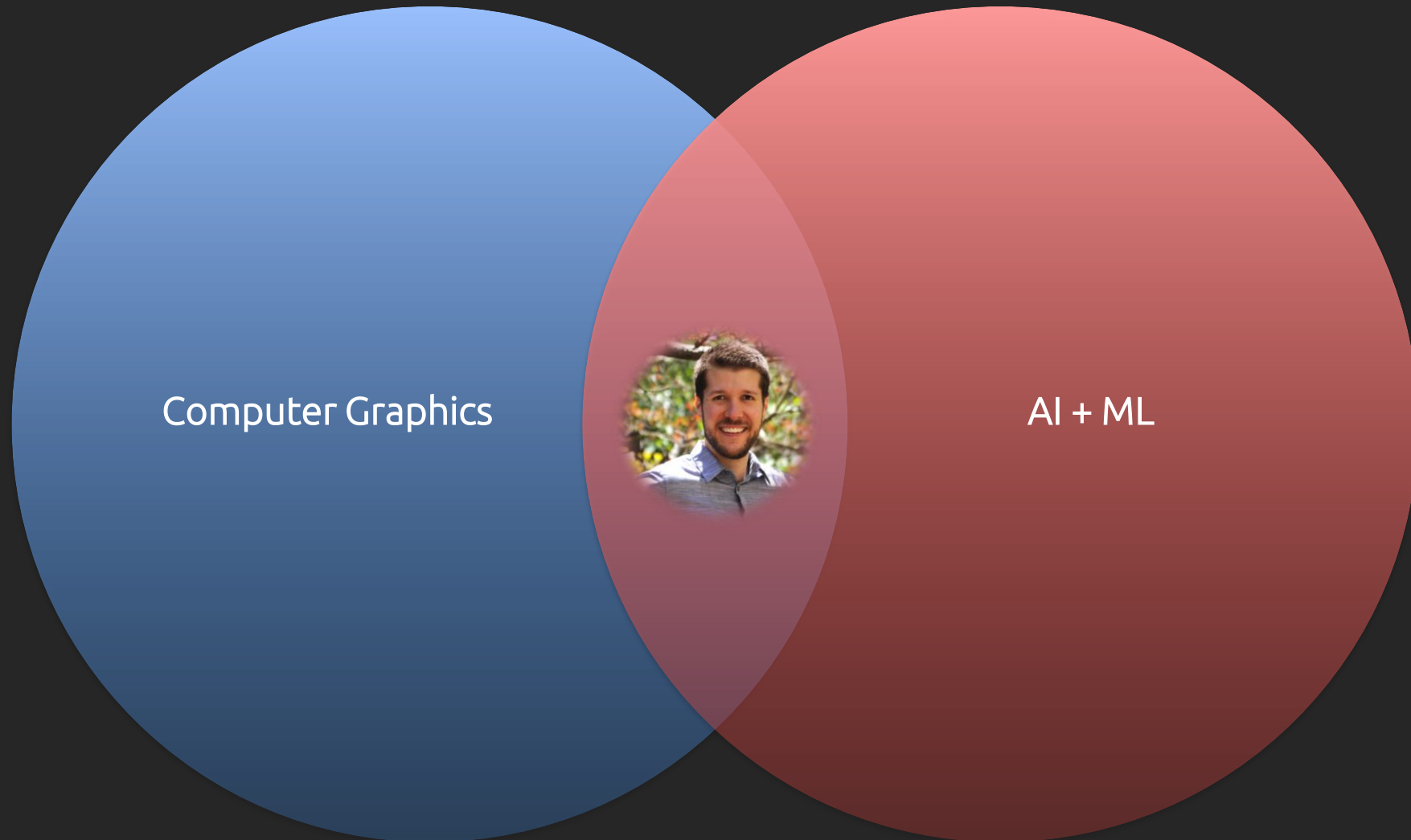
Andries "Andy" van Dam

<http://visual.cs.brown.edu/>

- Andy van Dam & Spike Hughes:  
Authors of  
“Computer Graphics:  
Principles and Practice”

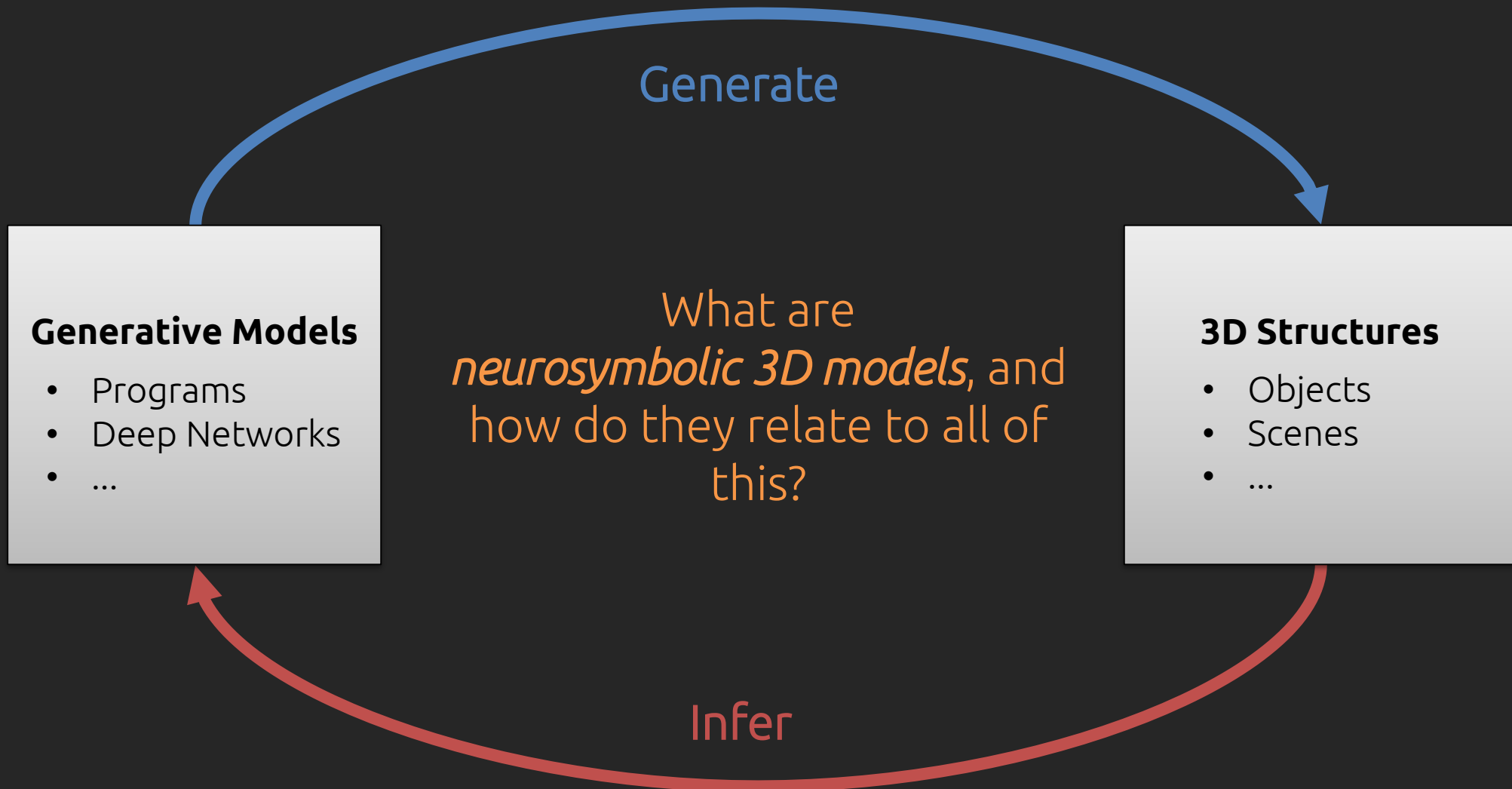


# My Research (Broadly)





# My Research (Specifically)



**FIRST, A LITTLE BACKGROUND  
& MOTIVATION...**



# Increasing Demand for 3D Content


Traditional driver: Entertainment (Games, VR, ...)



# Increasing Demand for 3D Content

E-Commerce (esp. furniture / interior design)

Furniture / Bedroom Furniture / Bedroom Sets / Queen Bedroom Sets / SKU: BYST7012



**Weldy Panel Configurable Bedroom Set**  
See More by [Brayden Studio](#)  
★★★★★ 35

**\$1,047.96**  
Open Box Outlet Price: [Available](#)  
[See All Special Offers & Savings \(4\)](#)

FREE Shipping  
Ships by Wed, Aug 29  
Ship To: [02906 - Providence](#)

**Weldy Panel Bed**  
\$322.99  
★★★★★ 5  
Size (2)  
Select Size  
[View Details](#)

**Weldy 2 Drawer Nightstand**  
\$149.99  
★★★★★ 8  
Select Quantity  
1  
[View Details](#)

**Weldy 6 Drawer Double Dresser**  
\$574.98  
★★★★★ 2  
Select Quantity  
1  
[View Details](#)

Price: \$1,047.96 Items: 3  
[Add to Cart](#)

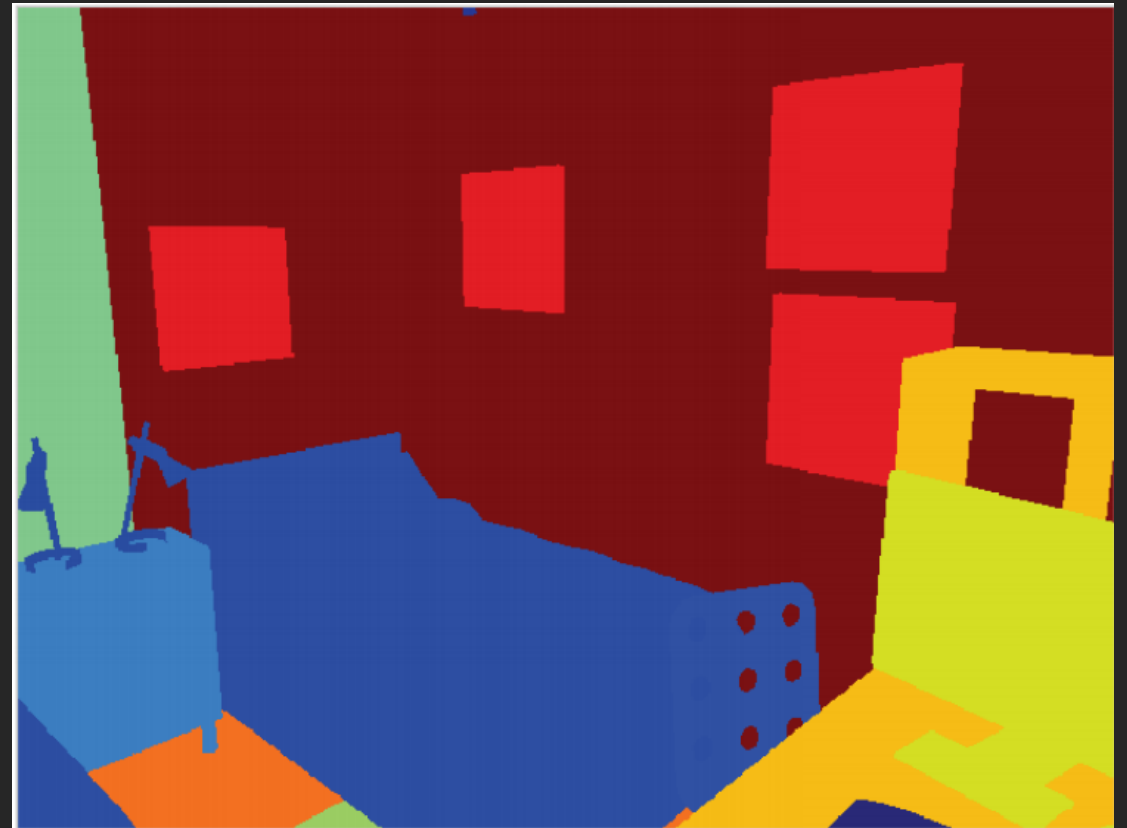


# Increasing Demand for 3D Content

New driver: Artificial Intelligence (“Graphics for AI”)



3D Scene



Semantic Segments

# Increasing Demand for 3D Content

New driver: Artificial Intelligence (“Graphics for AI”)



[aihabitat.org](https://aihabitat.org)

**Habitat: A Platform for Embodied AI Research**

**facebook** Artificial Intelligence



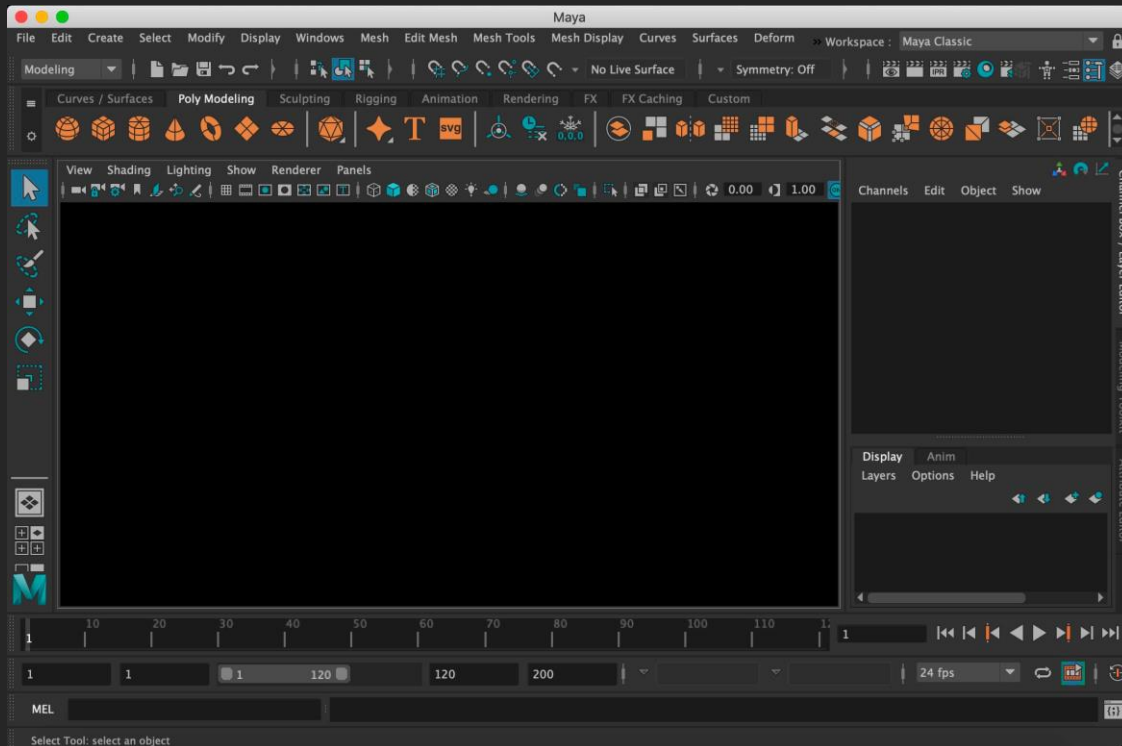
# Increasing Demand for 3D Content

New driver: Artificial Intelligence (“Graphics for AI”)

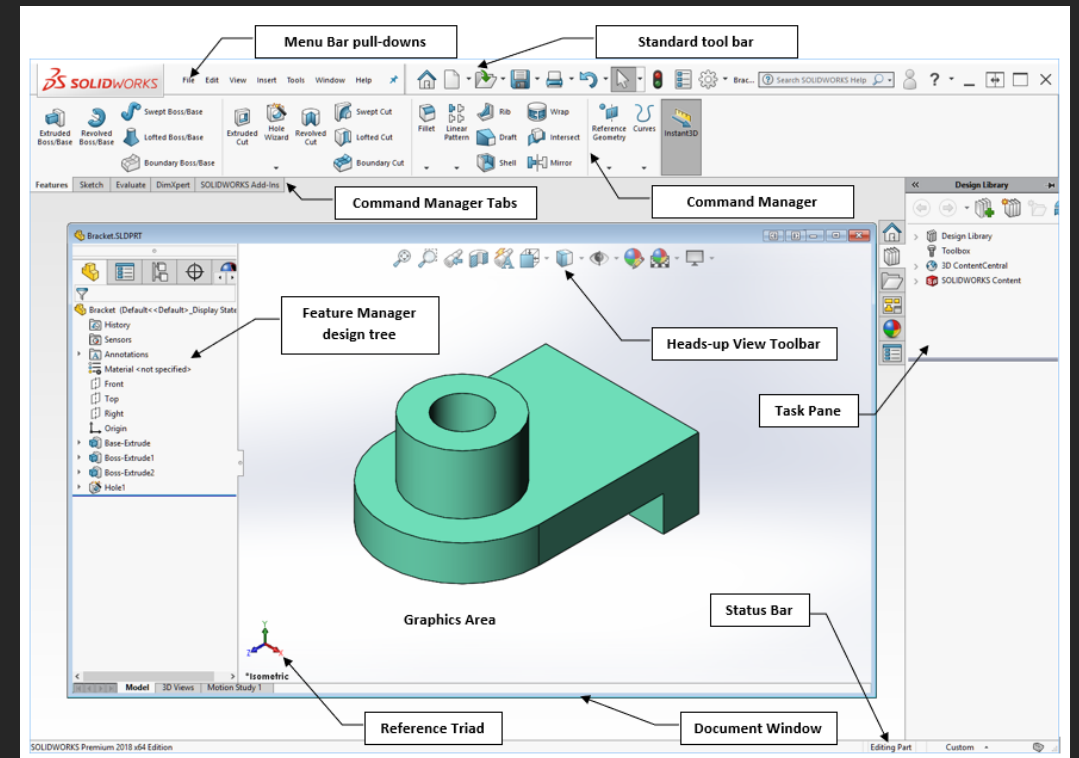


# Current Practice Can't Meet Demand

Manual 3D modeling: still slow, still hard to learn



Maya



Solidworks

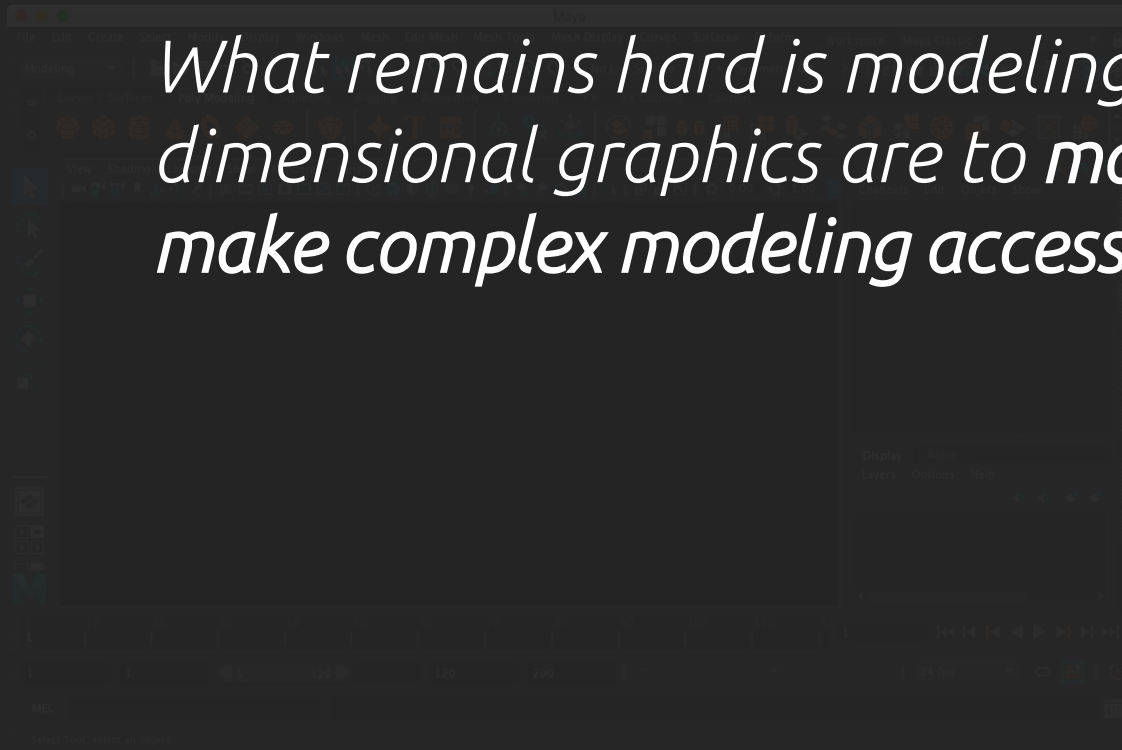
# Current Practice Can't Meet Demand

*"The difficulty of generating images has been overwhelmed by a five-thousand-fold improvement in price/performance of computing.*

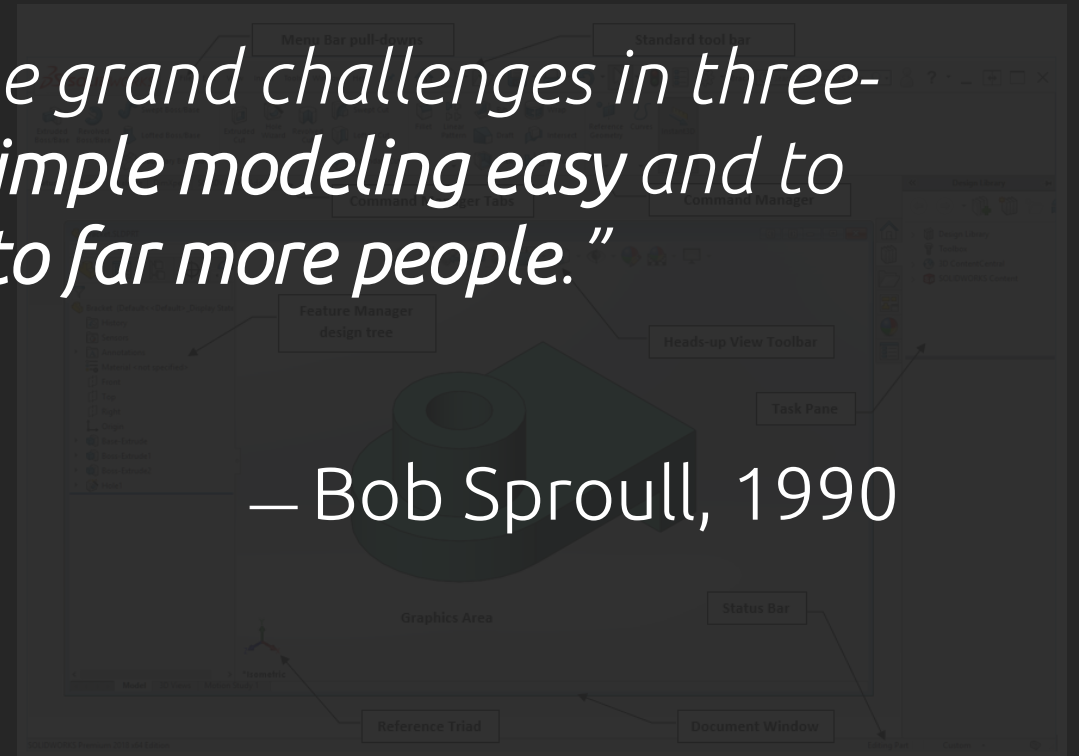
*Manual 3D modeling: still slow, still hard to learn*

*What remains hard is modeling...the grand challenges in three-dimensional graphics are to **make simple modeling easy** and to **make complex modeling accessible to far more people.**"*

— Bob Sproull, 1990



Maya



Solidworks



# Generative Models to the Rescue!?

For the purposes of this talk:

*Generative model: a procedure which can be executed to generate novel instances of some 3D object class*

# Benefits of Generative Models

3D content generation at scale



SpeedTree, Unreal Engine



CityEngine

# Benefits of Generative Models

Explore modeling possibilities





# Benefits of Generative Models

Strong prior for vision systems



StructureNet: Hierarchical Graph Networks for 3D Shape Generation, Mo et al. 2019

# Two Classes of Generative Model

## Procedural Models

*Pros:*

- High quality output by construction



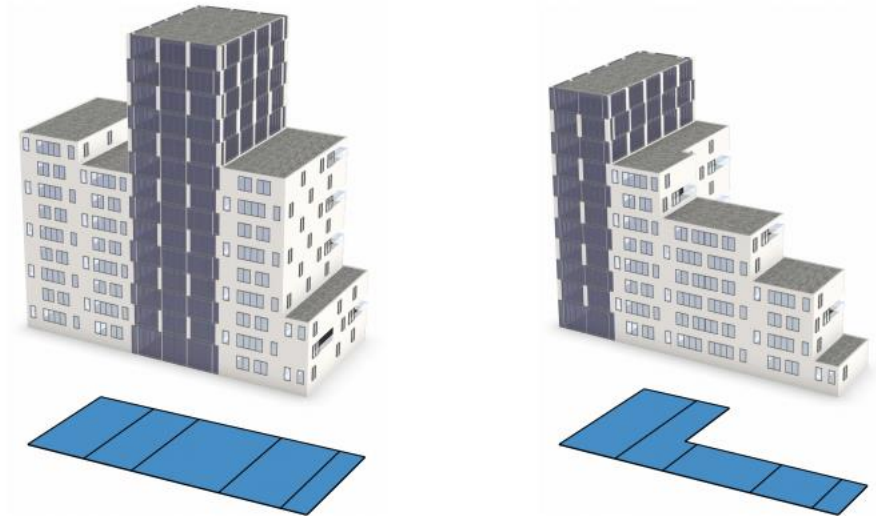
# Two Classes of Generative Model

## Procedural Models

### *Pros:*

- High quality output by construction
- Interpretable & editable

```
Parcel --> split("x") { rand(8, 16): Footprint | ~1: Parcel }  
Footprint --> event(IdentifyLargest) extrude(area()/6) Mass  
Mass --> case { get("isLargest"): Offices | else: Apartments }  
Offices --> ...  
Apartments --> ...  
  
event IdentifyLargest =  
  with(A = map(n:$nodes, area(n)), largest = index(A, max(A)),  
    foreach($nodes) { set("isLargest", $index == largest) } )
```





# Two Classes of Generative Model

## Procedural Models

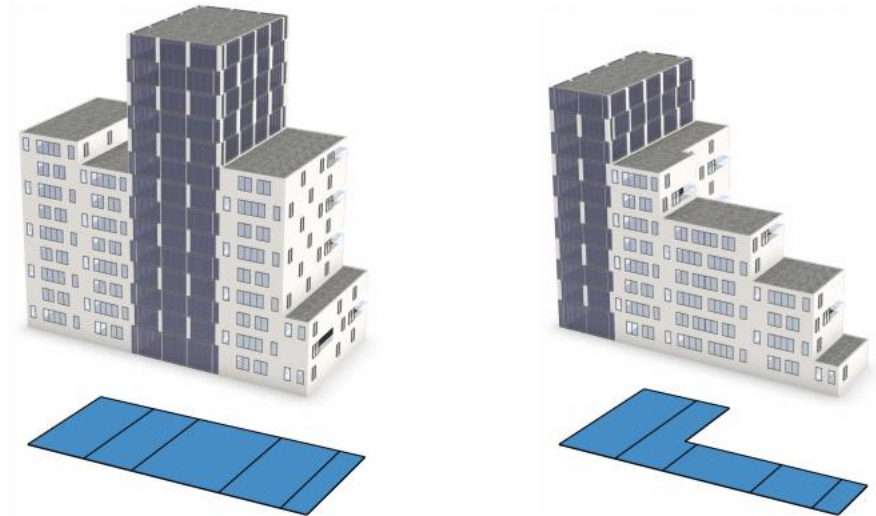
### *Pros:*

- High quality output by construction
- Interpretable & editable

### *Cons:*

- Difficult to author

```
Parcel --> split("x") { rand(8, 16): Footprint | ~1: Parcel }  
Footprint --> event(IdentifyLargest) extrude(area()/6) Mass  
Mass --> case { get("isLargest"): Offices | else: Apartments }  
Offices --> ...  
Apartments --> ...  
  
event IdentifyLargest =  
  with(A = map(n:$nodes, area(n)), largest = index(A, max(A)),  
    foreach($nodes) { set("isLargest", $index == largest) } )
```



# Two Classes of Generative Model

## Procedural Models

### *Pros:*

- High quality output by construction
- Interpretable & editable

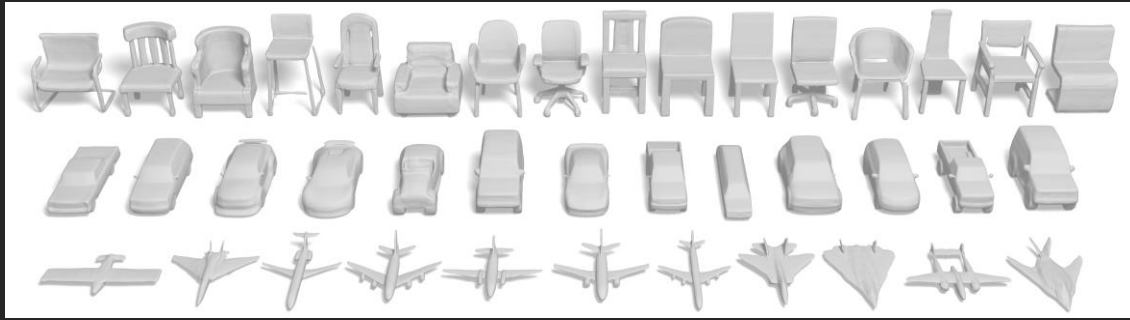
### *Cons:*

- Difficult to author
- Limited output variety



Learning to Generalize Kinematic Models to Novel Objects, Abbatematteo et al. 2019

# Two Classes of Generative Model



Learning Implicit Fields for Generative Shape Modeling , Chen & Zhang 2019

## Deep Generative Models

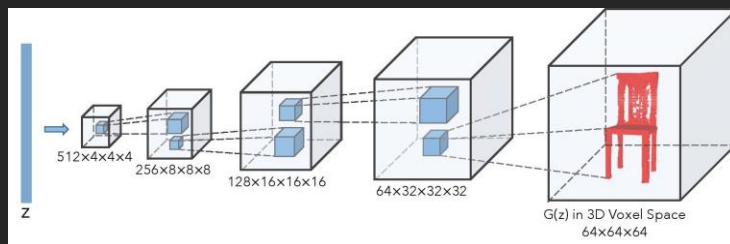
### Pros:

- Variety (any class of shape)
- Easy to author ("just add data")

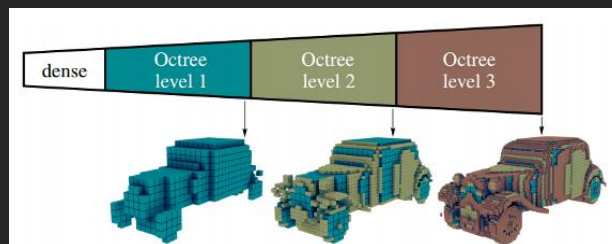


# Recent High-Profile Successes

## 3D-GAN



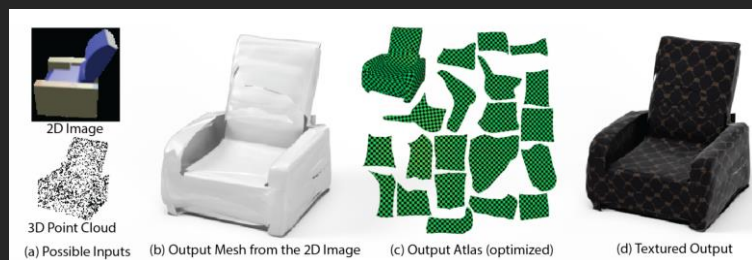
## Octree Generating Nets



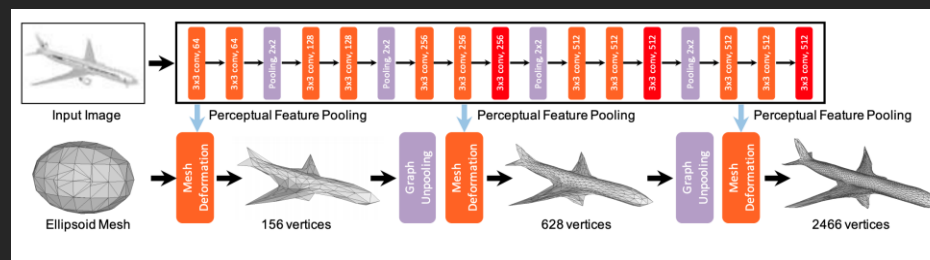
## PointFlow



## AtlasNet



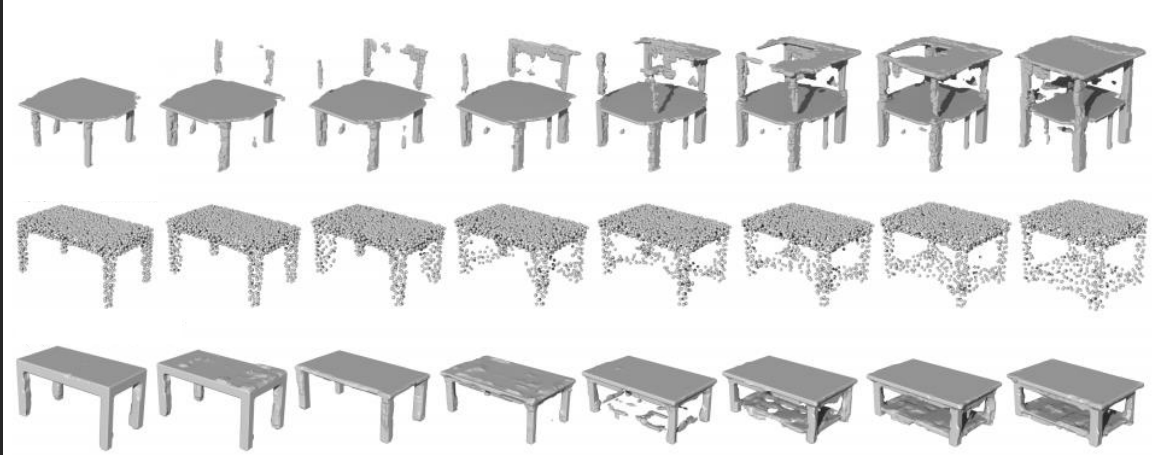
## Pixel2Mesh



## IM-Net



# Two Classes of Generative Model



## Deep Generative Models

### Pros:

- Variety (any class of shape)
- Easy to author ("just add data")

### Cons:

- Inconsistent output quality
- Inscrutable representation

# Two Classes of Generative Model

## Procedural Models

### *Pros:*

- High quality output by construction
- Interpretable & editable

### *Cons:*

- Difficult to author
- Limited output variety

## Deep Generative Models

### *Pros:*

- Variety (any class of shape)
- Easy to author ("just add data")

### *Cons:*

- Inconsistent output quality
- Inscrutable representation

*How can we get all of these...*



# Two Classes of Generative Model

## Procedural Models

### *Pros:*

- High quality output by construction
- Interpretable & editable

### *Cons:*

- Difficult to author
- Limited output variety

## Deep Generative Models

### *Pros:*

- Variety (any class of shape)
- Easy to author ("just add data")

### *Cons:*

- Inconsistent output quality
- Inscrutable representation

*How can we get all of these...*



*...with none of these?*



# Generative Models Capture Variation

Some modes can easily be expressed symbolically:

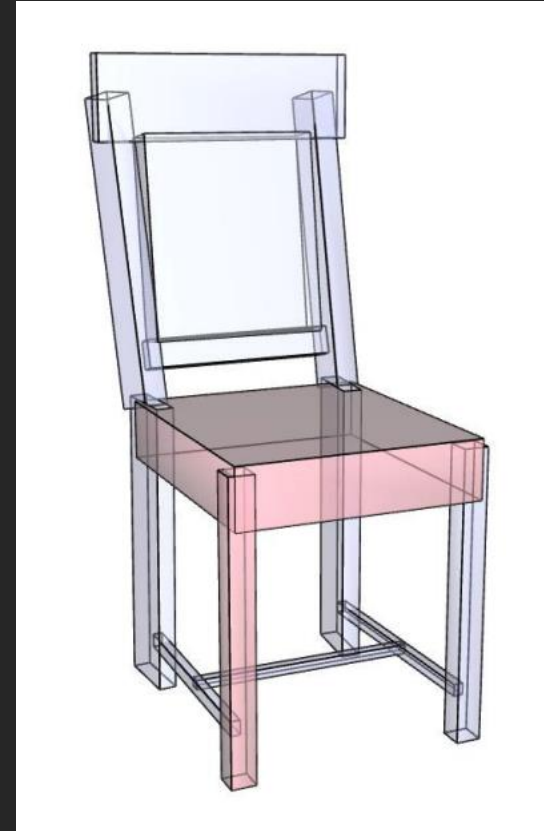
- Hierarchy



# Generative Models Capture Variaton

Some modes can easily be expressed symbolically:

- Hierarchy
- Connectivity

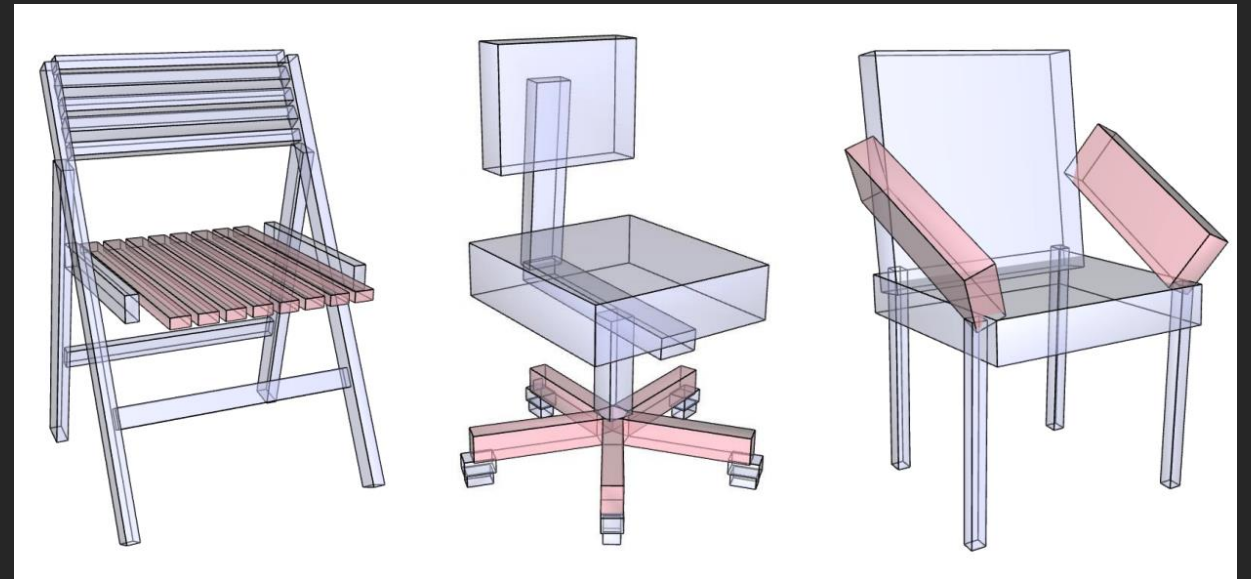




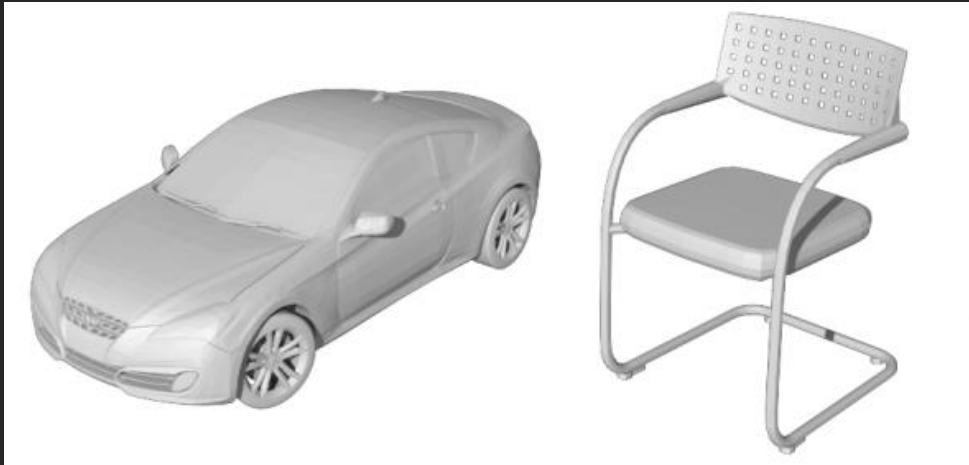
# Generative Models Capture Variaton

Some modes can easily be expressed symbolically:

- Hierarchy
- Connectivity
- Symmetry
- ...



# Generative Models Capture Variaton



Learning Implicit Fields for Generative Shape Modeling , Chen & Zhang 2019

Some modes are hard to express symbolically:

- Fine-detailed geometry

# Generative Models Capture Variaton



Learning Implicit Fields for Generative Shape Modeling , Chen & Zhang 2019

Some modes are hard to express symbolically:

- Fine-detailed geometry
- Complex inter-part correlations
- ...

# Generative Models Capture Variaton

Some modes can easily be expressed symbolically:

- Hierarchy
- Connectivity
- Symmetry
- ...

Some modes are hard to express symbolically:

**Design Philosophy:**  
Use symbols where possible  
Use neural nets for everything else

- ...

metry

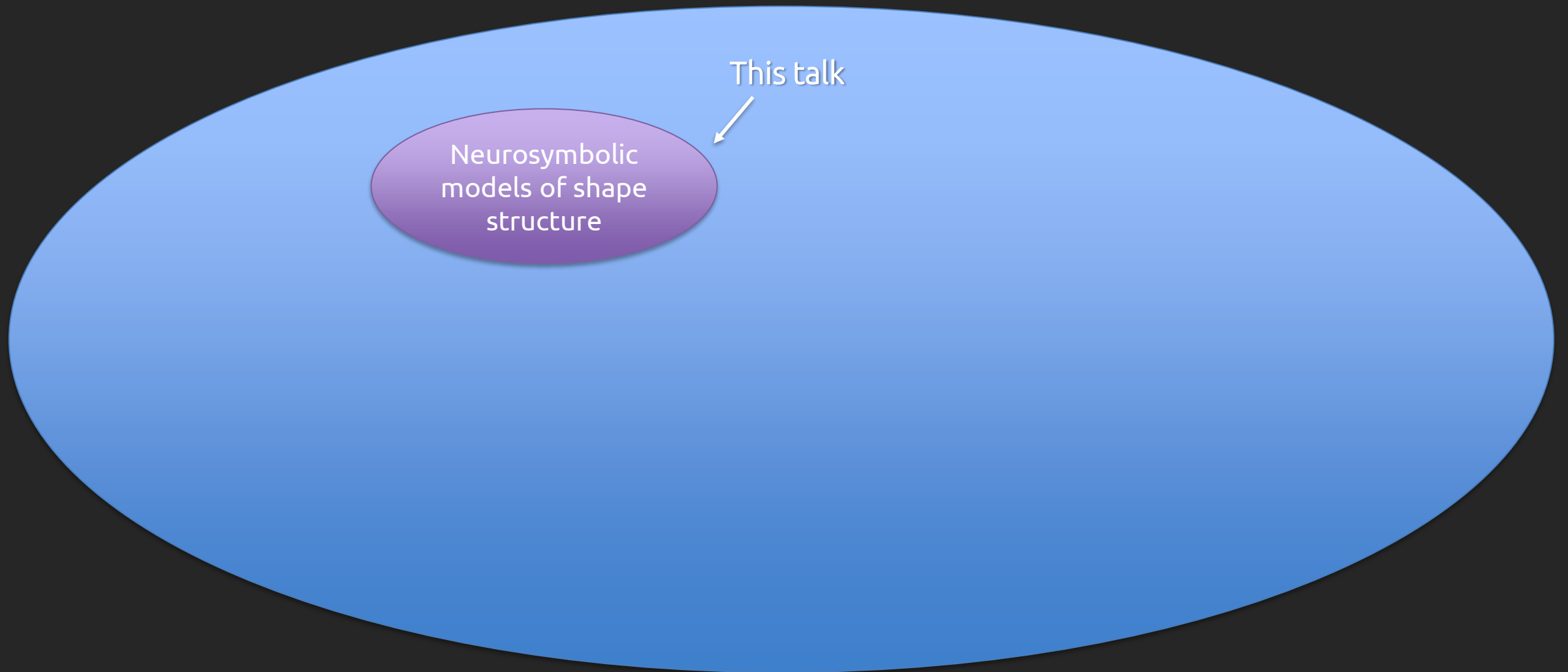
correlations



## *Neurosymbolic 3D Model:*

A generative model of a class of 3D objects which models some modes of variability via explicit symbols and others via a neural latent space

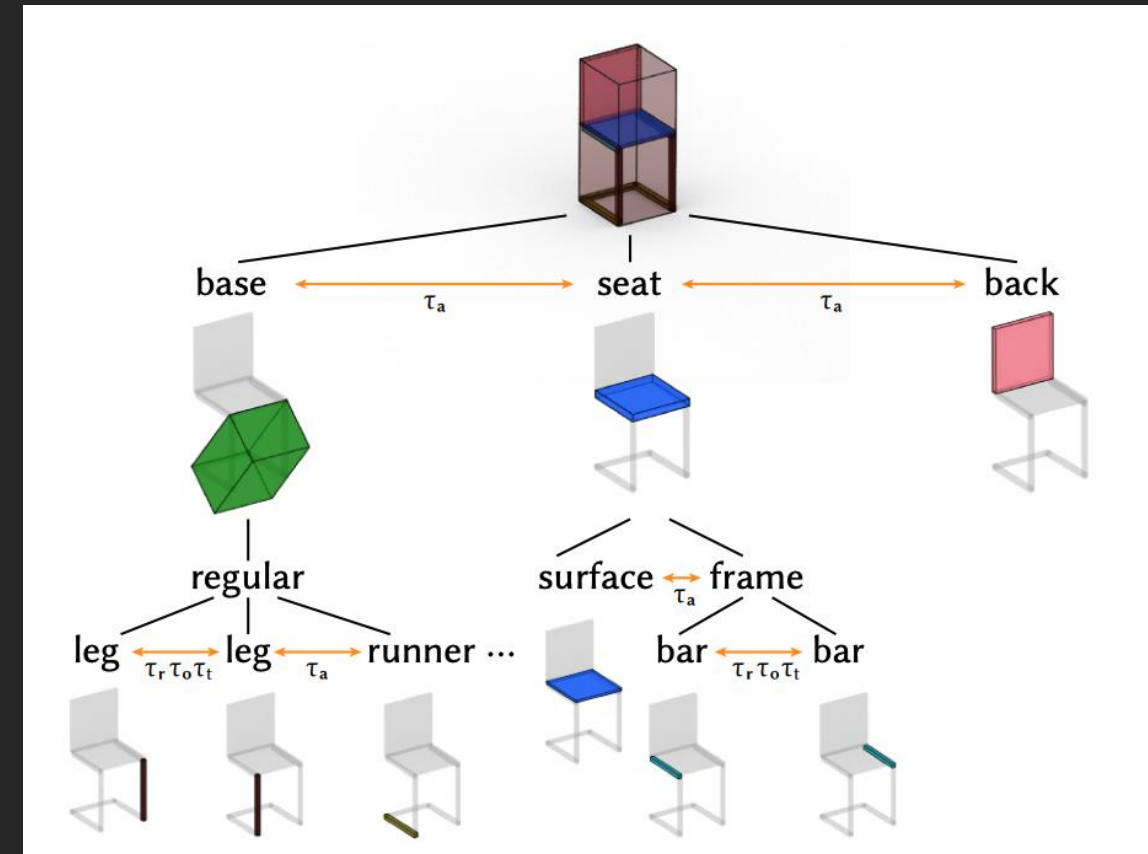
# Neurosymbolic 3D Model Design Space



# **NEUROSYMBOLIC MODELS OF SHAPE STRUCTURE**

# What Do I Mean by *Shape Structure*?

- Parts (as oriented bounding boxes)
- Relations
  - Hierarchy, connectivity, symmetry, ...
- Useful despite low geometric detail
  - Ex: robot motion planning → infer all parts + relations given point cloud observation
- Focus on *manufactured* objects
  - E.g. chairs, tables, airplanes...





# What Do I Mean by *Shape Structure*?

- Parts (as oriented bounding boxes)
- Relations
  - Hierarchy, connectivity, symmetry, ...
- Useful despite low geometric detail
  - Ex: robot motion planning → infer all parts + relations given point cloud observation
- Focus on *manufactured* objects
  - E.g. chairs, tables, airplanes...
  - Can extend to organic objects via e.g. generalized cylinder decomposition



Generalized Cylinder Decomposition, Zhou et al. 2015

# The “Holy Grail” of Structure Modeling

A single, interpretable procedural model that generates the structures of every object in a given shape class (e.g. chairs, airplanes)

But...

# Two Classes of Generative Model

## Procedural Models

### *Pros:*

- High quality output by construction
- Interpretable & editable

### *Cons:*

- Difficult to author
- Limited output variety

*Can a strategic use of neural nets eliminate these?*



# Eliminating Procedural Cons

**Problem:** Hard to author

**Solution:** Train a neural net to write them for us

**Problem:** Limited output variety

**Solution:** Latent space of neural net will capture the variability that the symbolic program does not



# ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis [SIGGRAPH Asia 2020]

R. KENNY JONES, Brown University  
THERESA BARTON, Brown University  
XIANGHAO XU, Brown University  
KAI WANG, Brown University  
ELLEN JIANG, Brown University  
PAUL GUERRERO, Adobe Research  
NILOY MITRA, University College London  
DANIEL RITCHIE, Brown University

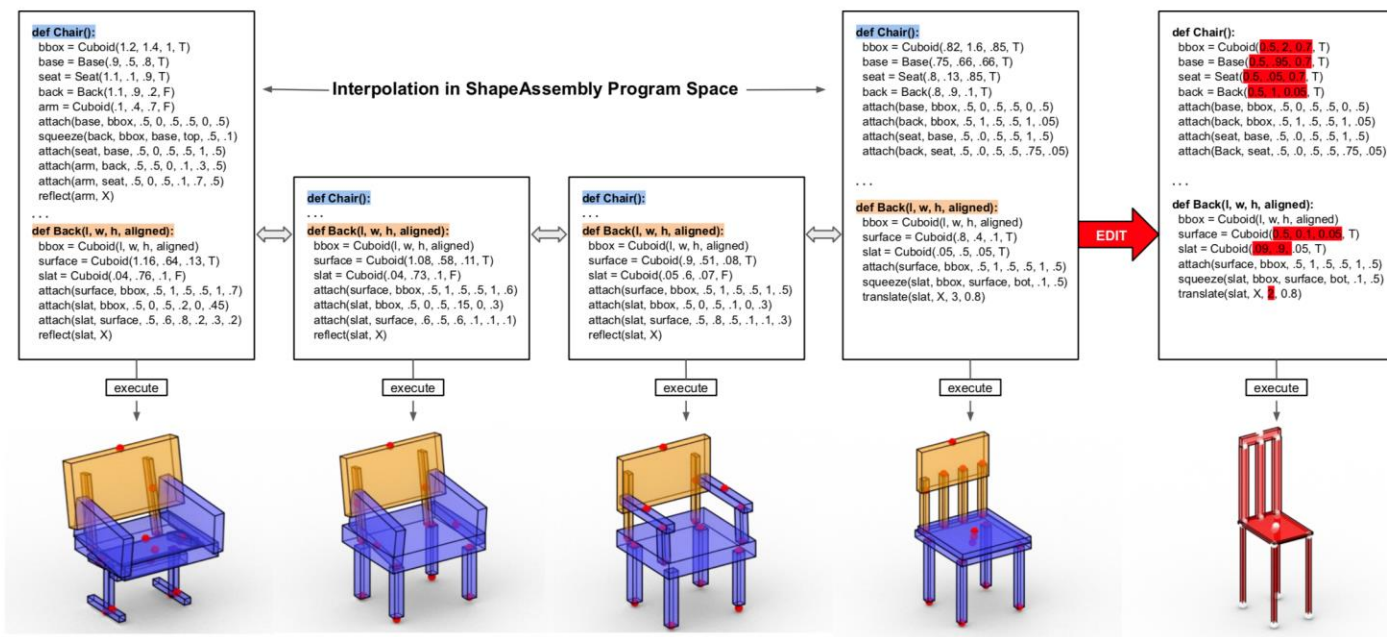


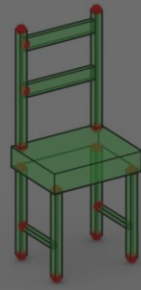
Fig. 1. We present a deep generative model which learns to write novel programs in SHAPEASSEMBLY, a domain-specific language for modeling 3D shape structures. Executing a SHAPEASSEMBLY program produces a shape composed of a hierarchical connected assembly of part proxies cuboids. Our method develops a well-formed latent space that supports interpolations between programs. Above, we show one such interpolation, and also visualize the geometry these programs produce when executed. In the last column, we manually edit the continuous parameters of a generated program, in order to produce a variant geometric structure with new topology.

# A Neurosymbolic 3D Modeling Pipeline

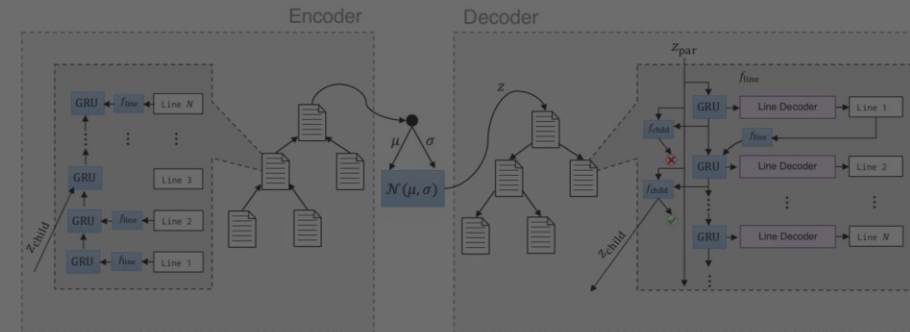
## ShapeAssembly DSL (Section 4)

Start  $\rightarrow$  BBlock; CBlock; ABlock; SBlock;  
 BBlock  $\rightarrow$  bbox = Cuboid( $l, h, w$ , True)  
 CBlock  $\rightarrow$   $c_n$  = Cuboid( $l, w, h, a$ ) ; CBlock | None  
 ABlock  $\rightarrow$  Attach ; ABlock | Squeeze ; ABlock | None  
 SBlock  $\rightarrow$  Reflect ; SBlock | Translate ; SBlock | None  
 Attach  $\rightarrow$  attach( $c_{n_1}, c_{n_2}, x_1, y_1, z_1, x_2, y_2, z_2$ )  
 Squeeze  $\rightarrow$  squeeze( $c_{n_1}, c_{n_2}, c_{n_3}, f, u, v$ )  
 Reflect  $\rightarrow$  reflect( $c_{n_1}, \text{axis}$ )  
 Translate  $\rightarrow$  translate( $c_{n_1}, \text{axis}, m, d$ )  
 $f \rightarrow$  right | left | top | bot | front | back  
 $\text{axis} \rightarrow$  X | Y | Z  
 $l, h, w \in \mathbb{R}^3$   
 $x, y, z, u, v, d \in [0, 1]^2$   
 $a \in [\text{True}, \text{False}]$   
 $n, m \in \mathbb{Z}^+$

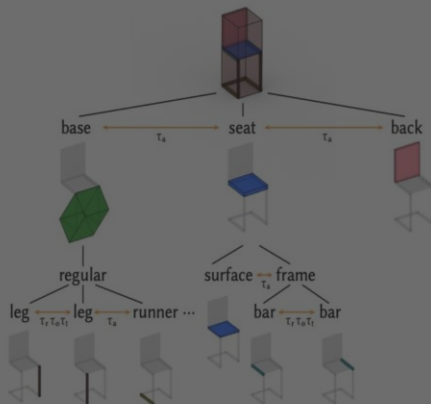
## Shapes to Training Programs (Section 5)



## Learning to Generate Programs (Section 6)



## Input Hierarchical Part Graphs



```

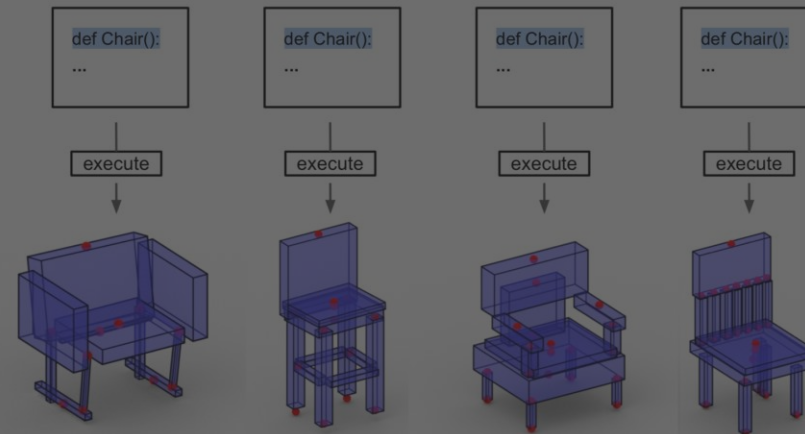
def Chair():
    bbox = Cuboid(.7, 1.7, .5, True)
    prog1 = Program1(.7, .6, .5, True)
    prog2 = Program2(.7, .9, .05, True)
    cube2 = Cuboid(.7, .15, .5, True)
    attach(prog1, bbox, .5, 0, .5, .5, 0, .5)
    attach(cube2, prog1, .5, 0, .5, .5, 1, .5)
    squeeze(prog2, bbox, cube2, top, .5, .1)

```

```

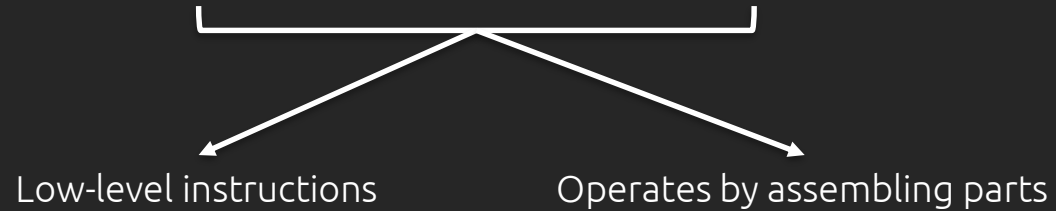
def Program1(l, w, h, aligned):
    bbox = Cuboid(.7, .6, .5, True)
    prog3 = Program3(.05, .6, .5, True)
    squeeze(prog3, bbox, bbox, top, 0, .5)
    reflect(prog3, X)

```



# ShapeAssembly

An “assembly language” for part-based shapes

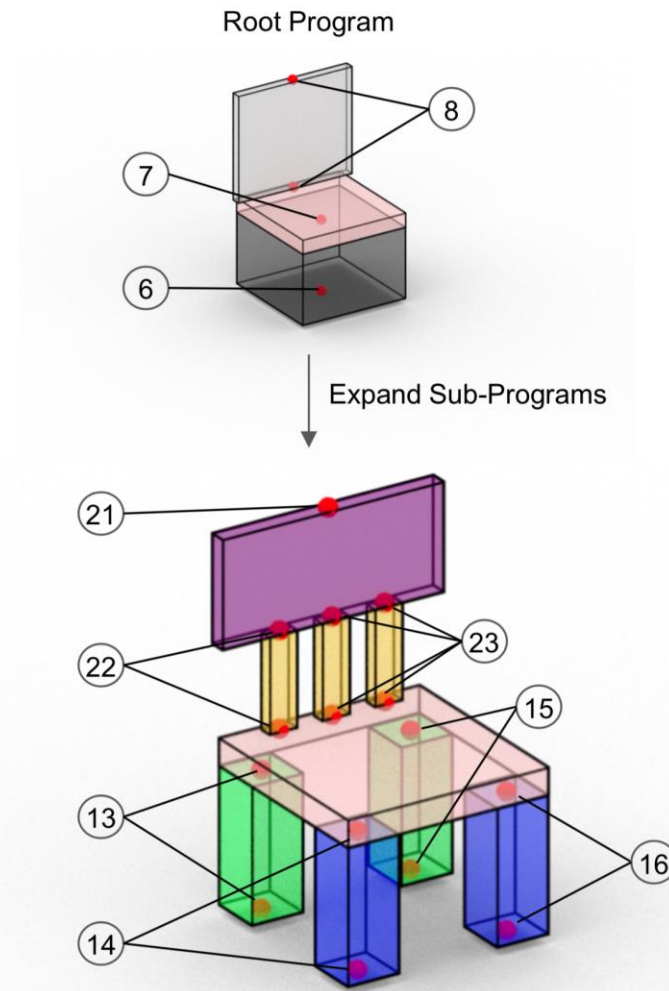


# Anatomy of a ShapeAssembly Program

```
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)

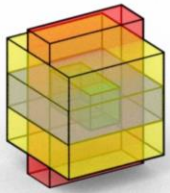
9. def Base(l, w, h, aligned):
10.    bbox = Cuboid(l, w, h, aligned)
11.    cube0 = Cuboid(.2, .5, .2, True)
12.    cube1 = Cuboid(.2, .5, .2, True)
13.    squeeze(cube0, bbox, bbox, top, .1, .1)
14.    squeeze(cube1, bbox, bbox, top, .1, .8)
15.    reflect(cube0, X)
16.    reflect(cube1, X)

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)
```



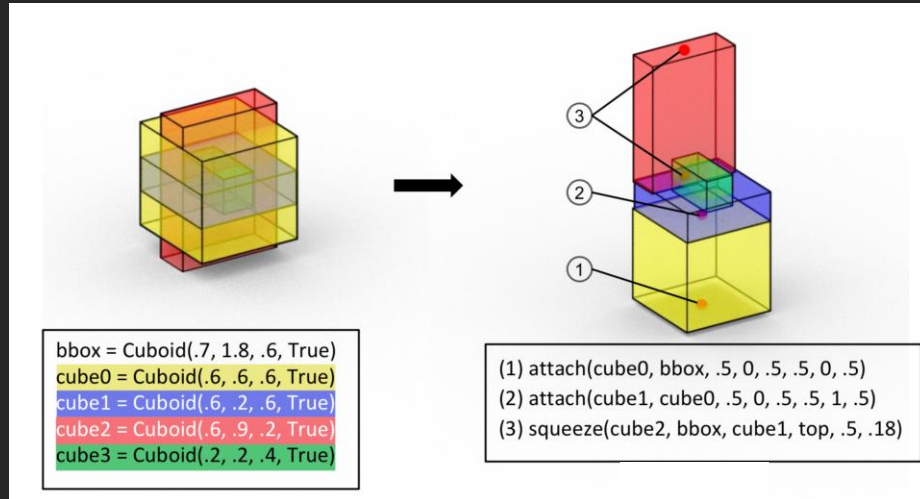


# Execution Semantics

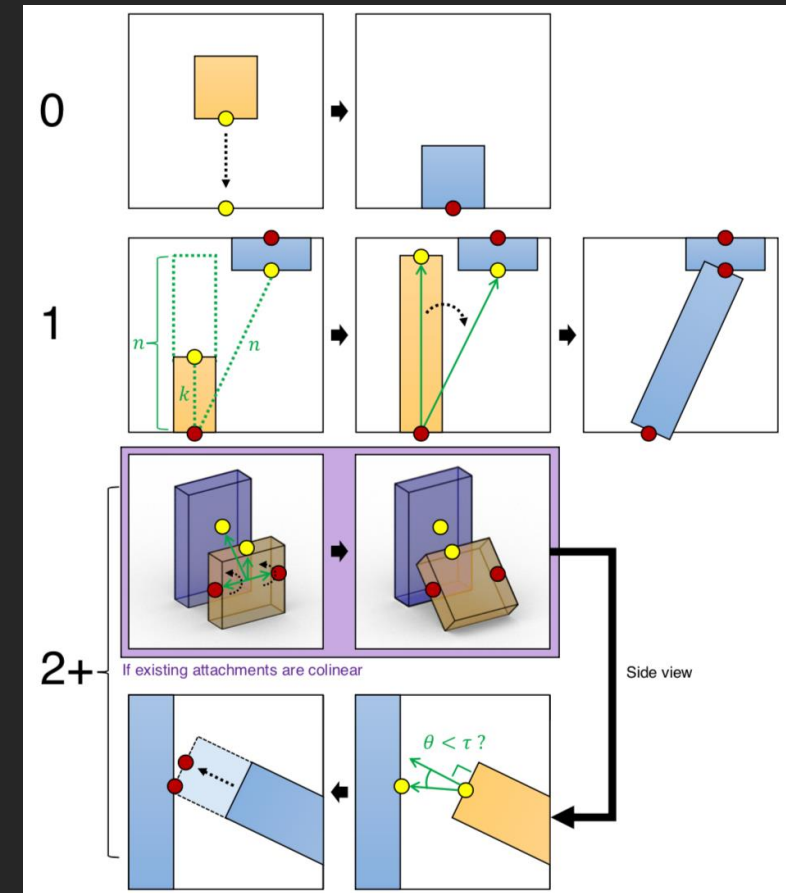


```
bbox = Cuboid(.7, 1.8, .6, True)  
cube0 = Cuboid(.6, .6, .6, True)  
cube1 = Cuboid(.6, .2, .6, True)  
cube2 = Cuboid(.6, .9, .2, True)  
cube3 = Cuboid(.2, .2, .4, True)
```

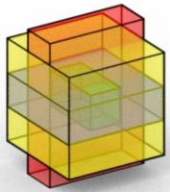
# Execution Semantics



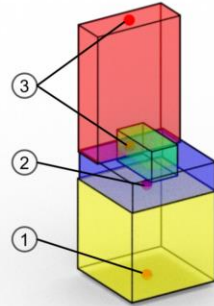
## Semantics of **attach**



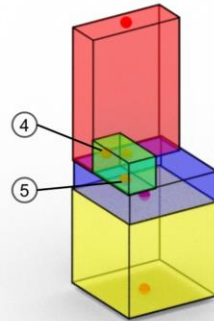
# Execution Semantics



```
bbox = Cuboid(.7, 1.8, .6, True)
cube0 = Cuboid(.6, .6, .6, True)
cube1 = Cuboid(.6, .2, .6, True)
cube2 = Cuboid(.6, .9, .2, True)
cube3 = Cuboid(.2, .2, .4, True)
```

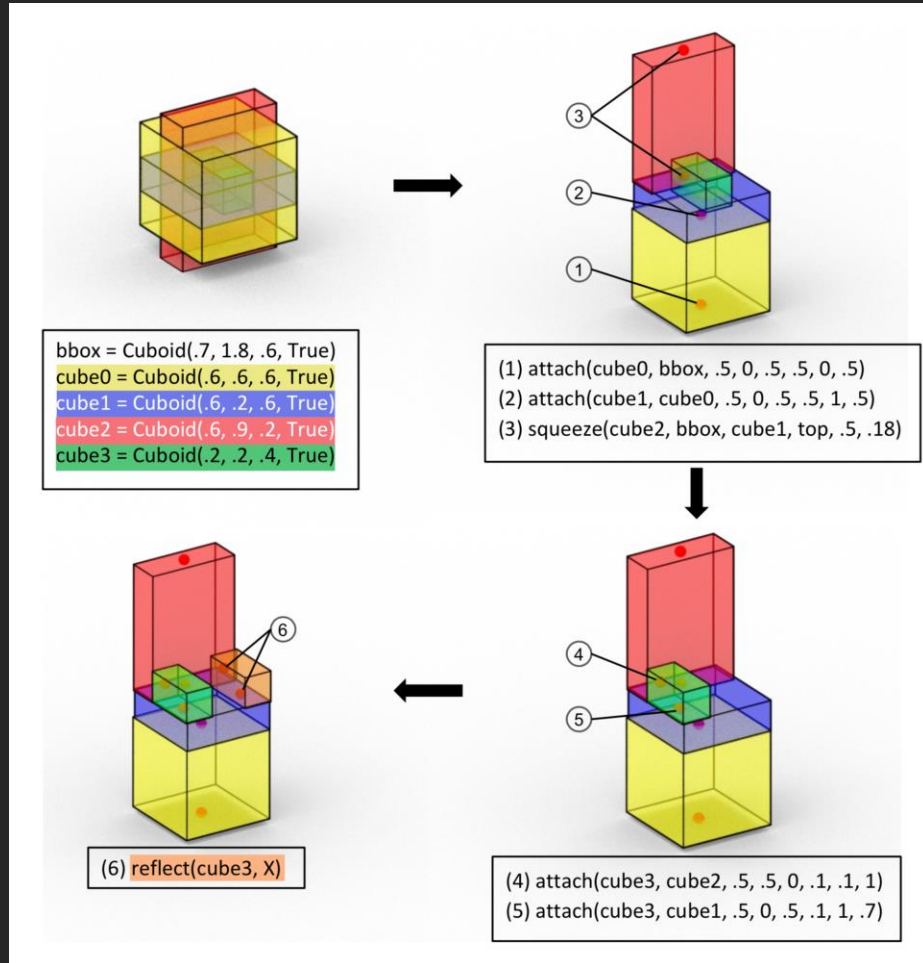


```
(1) attach(cube0, bbox, .5, 0, .5, .5, 0, .5)
(2) attach(cube1, cube0, .5, 0, .5, .5, 1, .5)
(3) squeeze(cube2, bbox, cube1, top, .5, .18)
```

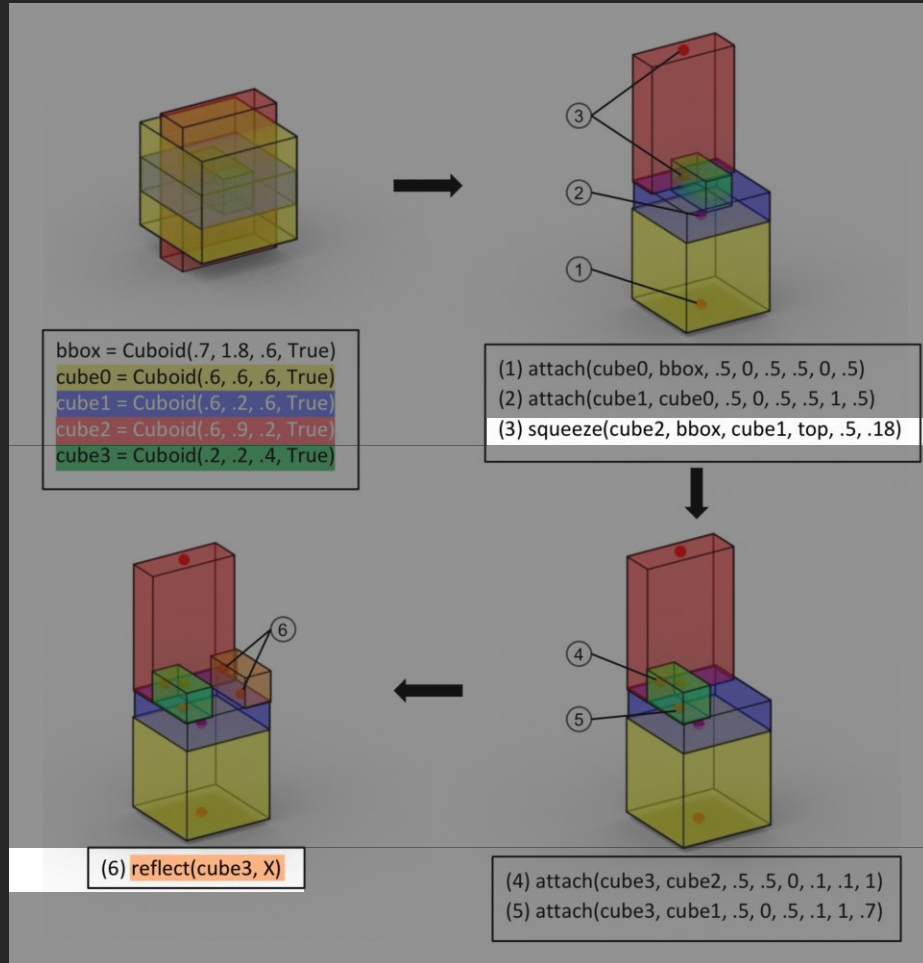


```
(4) attach(cube3, cube2, .5, .5, 0, .1, .1, 1)
(5) attach(cube3, cube1, .5, 0, .5, .1, 1, .7)
```

# Execution Semantics



# Execution Semantics

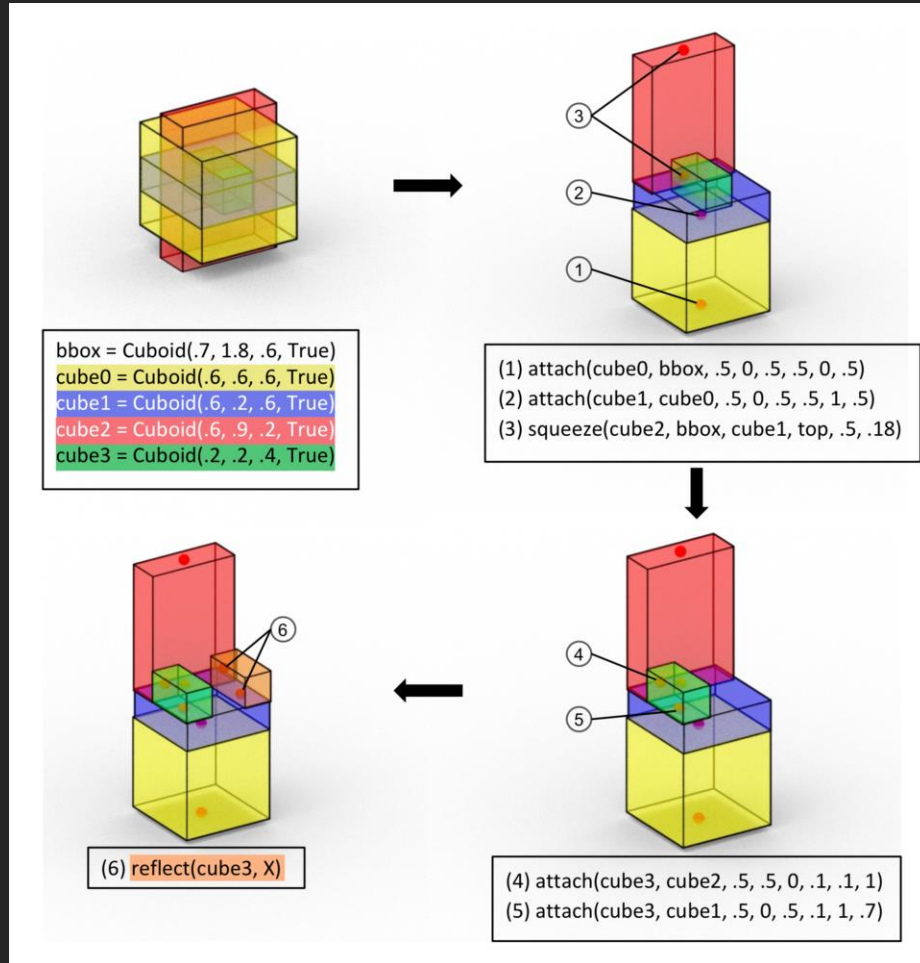


## Macros:

squeeze, reflect, translate  
expand into multiple Cuboid +  
attach statements



# Execution Semantics



Differentiable execution:

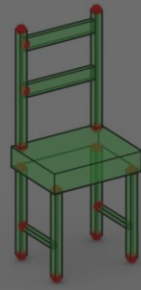
Output geometry is differentiable with respect to continuous parameters of input program

# A Neurosymbolic 3D Modeling Pipeline

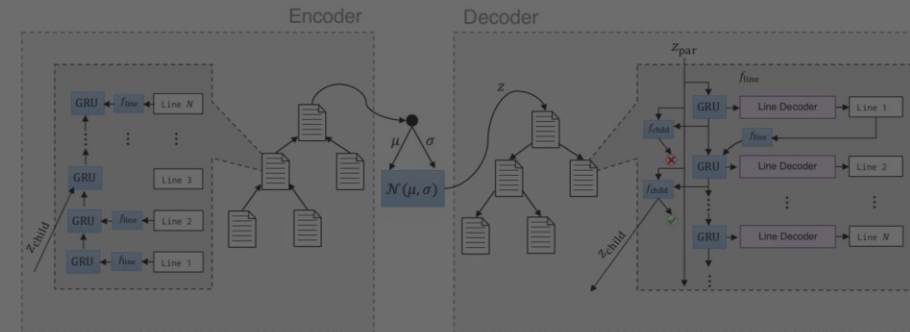
## ShapeAssembly DSL (Section 4)

Start  $\rightarrow$  BBlock; CBlock; ABlock; SBlock;  
BBlock  $\rightarrow$  bbox = Cuboid( $l, h, w$ , True)  
CBlock  $\rightarrow c_n$  = Cuboid( $l, w, h, a$ ) ; CBlock | None  
ABlock  $\rightarrow$  Attach ; ABlock | Squeeze ; ABlock | None  
SBlock  $\rightarrow$  Reflect ; SBlock | Translate ; SBlock | None  
Attach  $\rightarrow$  attach( $c_{n_1}, c_{n_2}, x_1, y_1, z_1, x_2, y_2, z_2$ )  
Squeeze  $\rightarrow$  squeeze( $c_{n_1}, c_{n_2}, c_{n_3}, f, u, v$ )  
Reflect  $\rightarrow$  reflect( $c_{n_1}, \text{axis}$ )  
Translate  $\rightarrow$  translate( $c_{n_1}, \text{axis}, m, d$ )  
 $f \rightarrow$  right | left | top | bot | front | back  
axis  $\rightarrow$  X | Y | Z  
 $l, h, w \in \mathbb{R}^3$   
 $x, y, z, u, v, d \in [0, 1]^2$   
 $a \in [\text{True}, \text{False}]$   
 $n, m \in \mathbb{Z}^+$

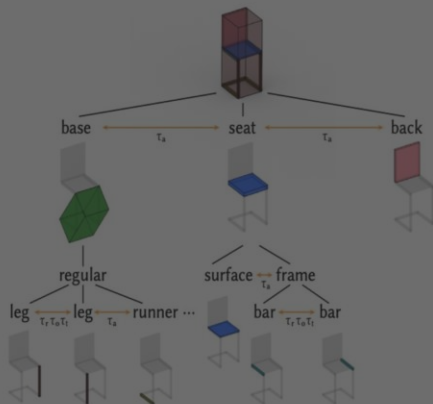
## Shapes to Training Programs (Section 5)



## Learning to Generate Programs (Section 6)

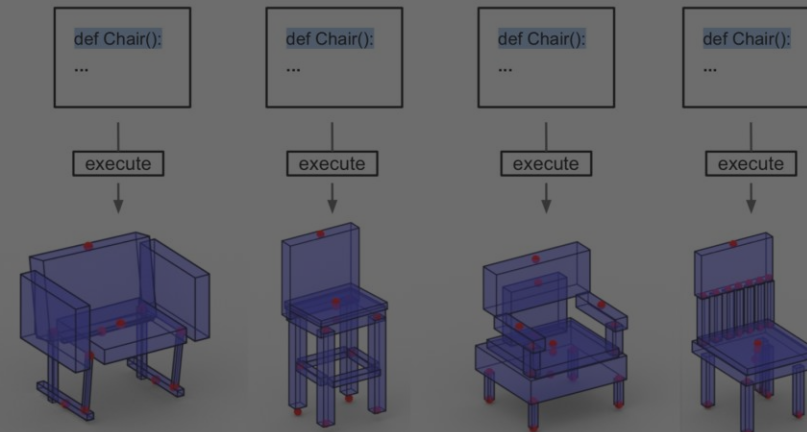


## Input Hierarchical Part Graphs



```
def Chair():  
  bbox = Cuboid(.7, 1.7, .5, True)  
  prog1 = Program1(.7, .6, .5, True)  
  prog2 = Program2(.7, .9, .05, True)  
  cube2 = Cuboid(.7, .15, .5, True)  
  attach(prog1, bbox, .5, 0, .5, .5, 0, .5)  
  attach(cube2, prog1, .5, 0, .5, .5, 1, .5)  
  squeeze(prog2, bbox, cube2, top, .5, .1)
```

```
def Program1(l, w, h, aligned):  
  bbox = Cuboid(.7, .6, .5, True)  
  prog3 = Program3(.05, .6, .5, True)  
  squeeze(prog3, bbox, bbox, top, 0, .5)  
  reflect(prog3, X)  
  ...
```



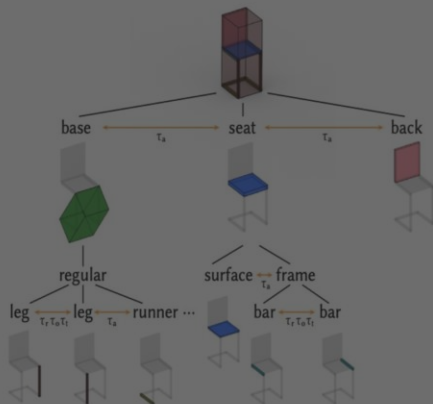
# A Neurosymbolic 3D Modeling Pipeline

## ShapeAssembly DSL (Section 4)

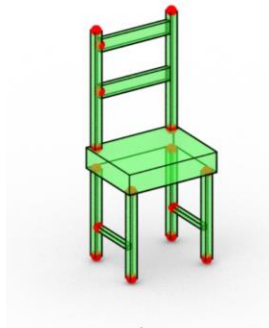
```

Start → BBlock; CBlock; ABlock; SBlock;
BBlock → bbox = Cuboid( $l, h, w$ , True)
CBlock →  $c_n$  = Cuboid( $l, w, h, a$ ) ; CBlock | None
ABlock → Attach ; ABlock | Squeeze ; ABlock | None
SBlock → Reflect ; SBlock | Translate ; SBlock | None
Attach → attach( $c_{n_1}, c_{n_2}, x_1, y_1, z_1, x_2, y_2, z_2$ )
Squeeze → squeeze( $c_{n_1}, c_{n_2}, c_{n_3}, f, u, v$ )
Reflect → reflect( $c_{n_1}, \text{axis}$ )
Translate → translate( $c_{n_1}, \text{axis}, m, d$ )
 $f \rightarrow \text{right} \mid \text{left} \mid \text{top} \mid \text{bot} \mid \text{front} \mid \text{back}$ 
 $\text{axis} \rightarrow X \mid Y \mid Z$ 
 $l, h, w \in \mathbb{R}^3$ 
 $x, y, z, u, v, d \in [0, 1]^2$ 
 $a \in [\text{True}, \text{False}]$ 
 $n, m \in \mathbb{Z}^+$ 
    
```

## Input Hierarchical Part Graphs



## Shapes to Training Programs (Section 5)

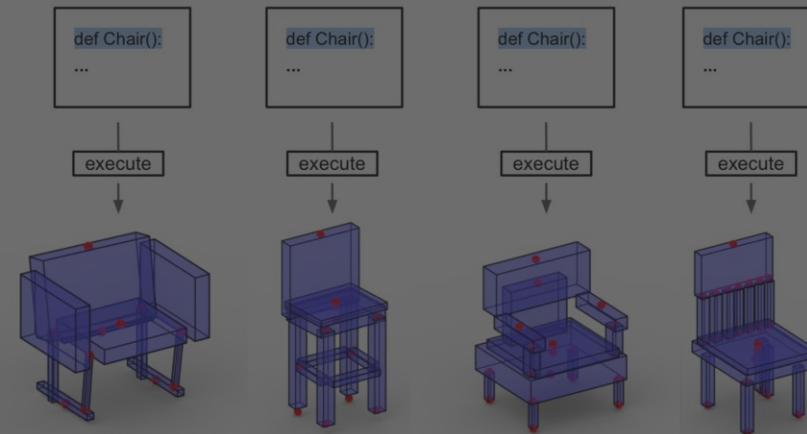
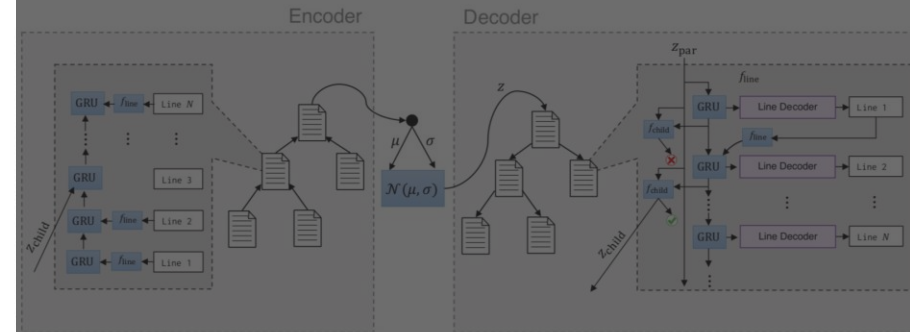


```

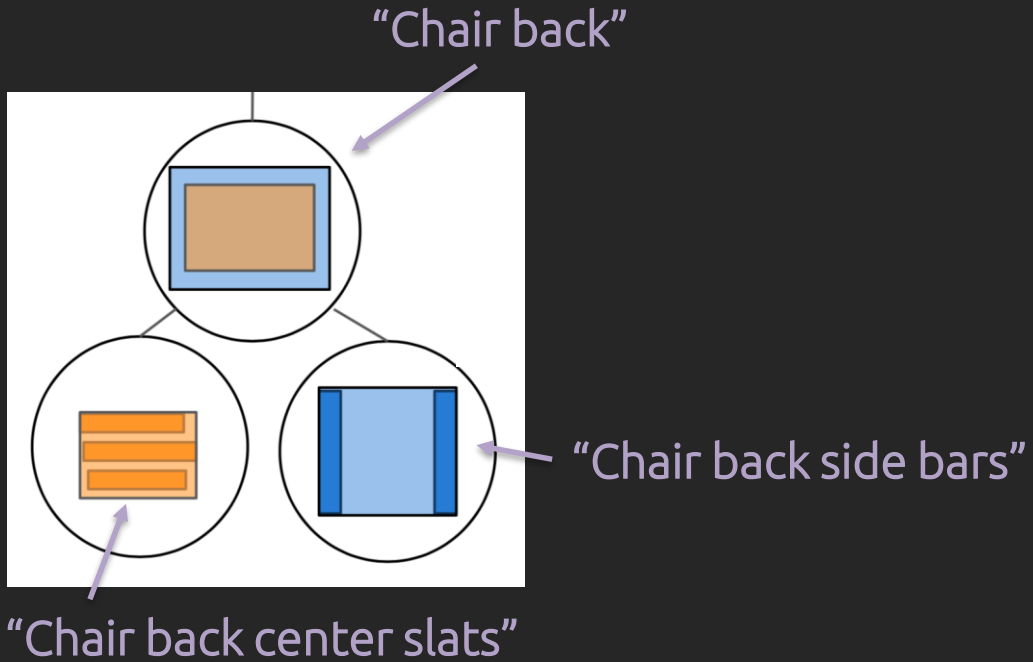
def Chair():
    bbox = Cuboid(.7, 1.7, .5, True)
    prog1 = Program1(.7, .6, .5, True)
    prog2 = Program2(.7, .9, .05, True)
    cube2 = Cuboid(.7, .15, .5, True)
    attach(prog1, bbox, .5, 0, .5, .5, 0, .5)
    attach(cube2, prog1, .5, 0, .5, .5, 1, .5)
    squeeze(prog2, bbox, cube2, top, .5, .1)

def Program1( $l, w, h, \text{aligned}$ ):
    bbox = Cuboid(.7, .6, .5, True)
    prog3 = Program3(.05, .6, .5, True)
    squeeze(prog3, bbox, bbox, top, 0, .5)
    reflect(prog3, X)
    ...
    
```

## Learning to Generate Programs (Section 6)

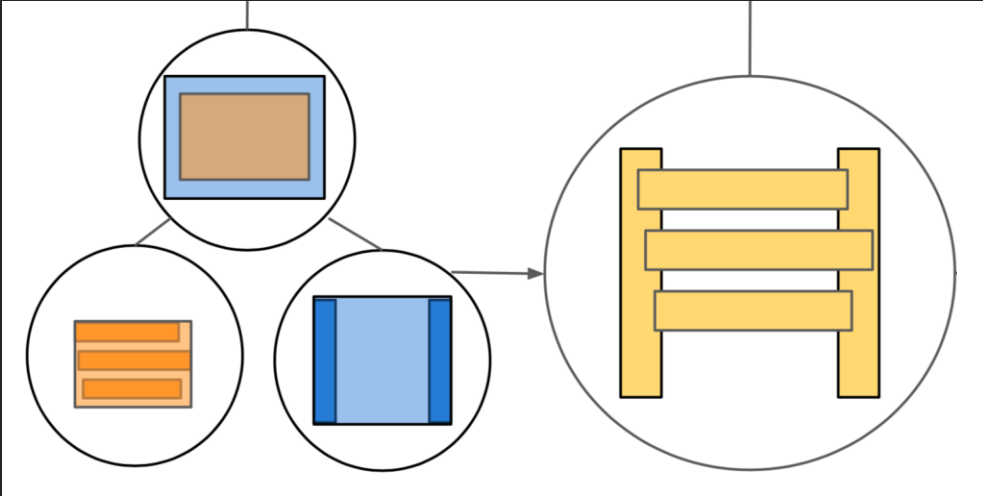


# Extracting Programs from Shapes



Local region of an  
input hierarchical  
part graph

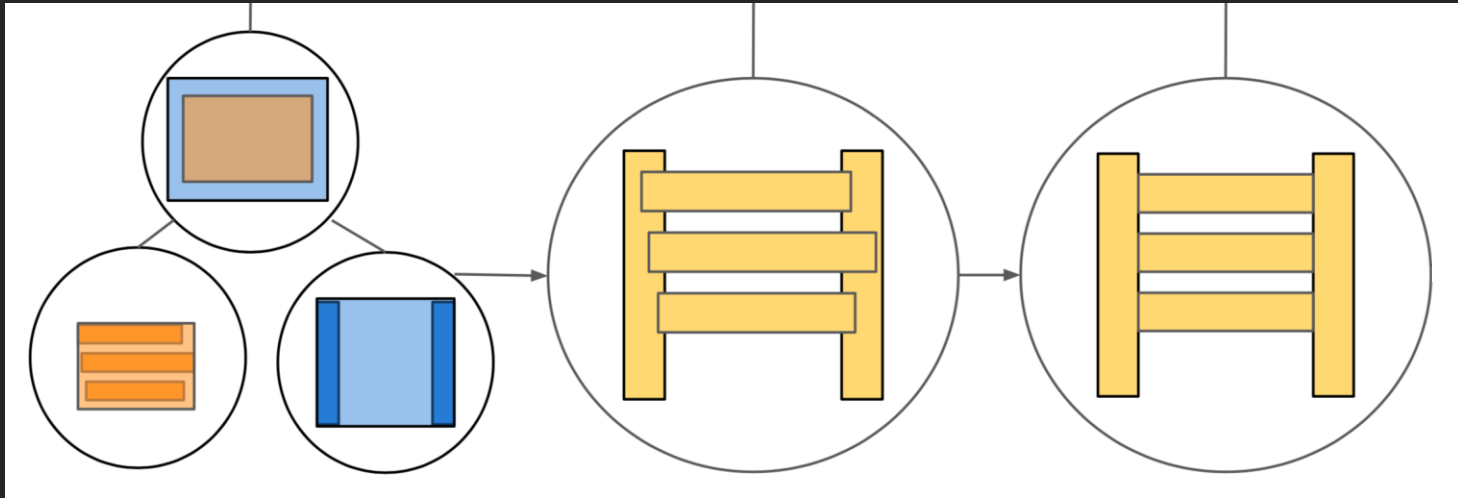
# Extracting Programs from Shapes



Locally flattening  
the hierarchy to  
make interacting  
leaf parts siblings

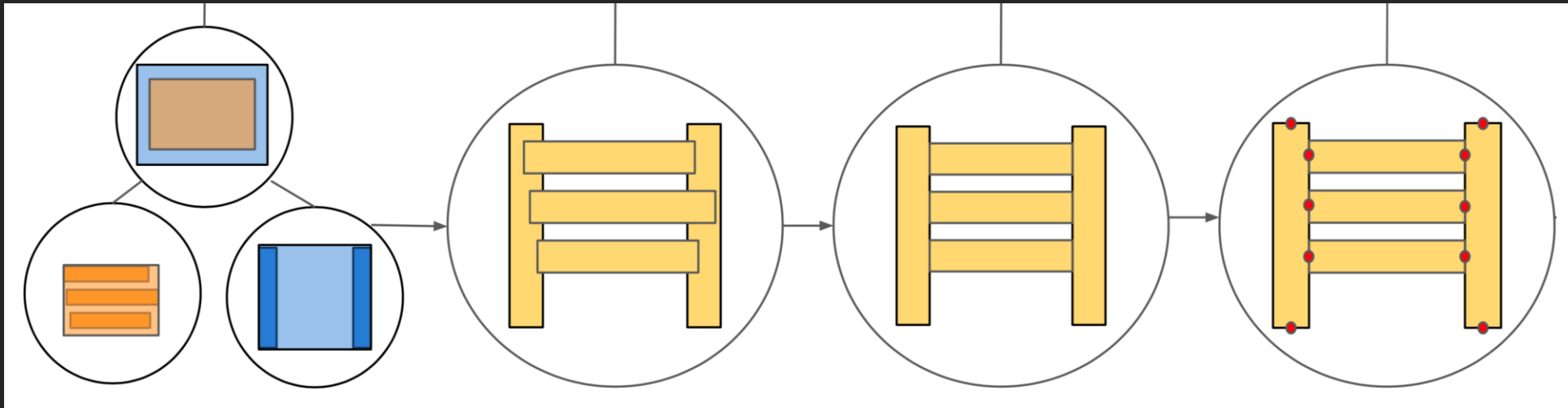


# Extracting Programs from Shapes



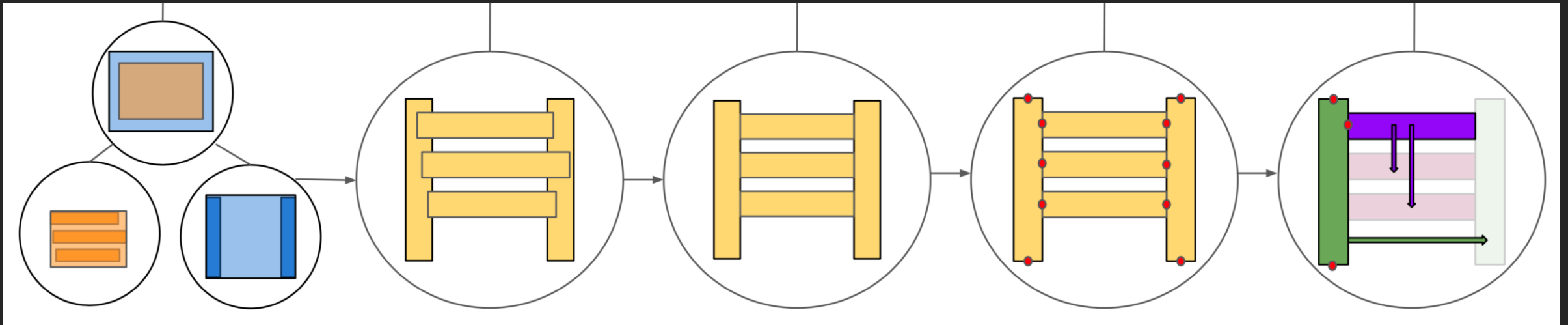
Shortening leaf  
parts that intersect  
other leaf parts

# Extracting Programs from Shapes



Locating  
attachment points  
between parts

# Extracting Programs from Shapes

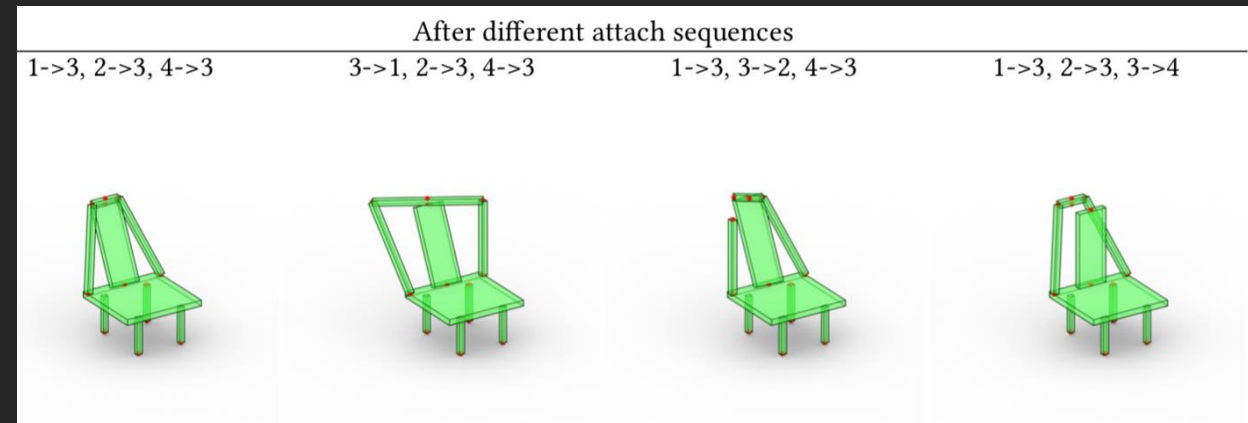
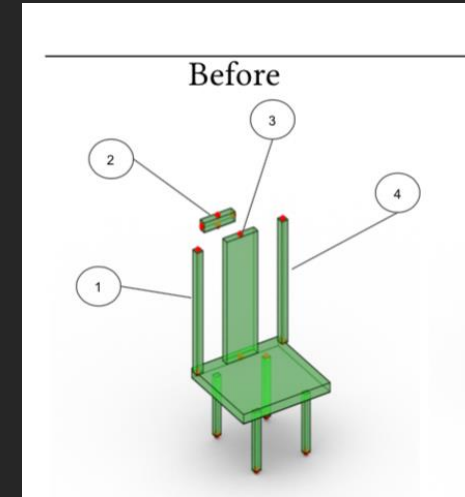


Forming leaf parts  
into symmetry  
groups

# Extracting Programs from Shapes

## Ordering Attachments

- Due to imperative semantics, attach order matters
- Heuristics to prune possible orders, then check which one produces output that best fits the shape



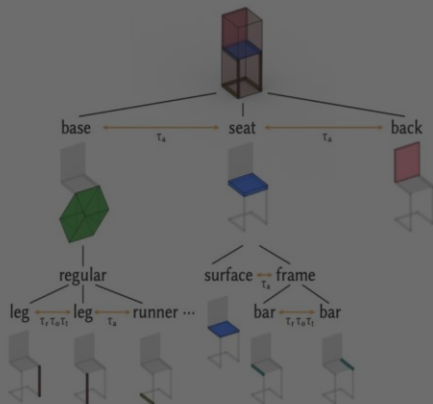
# A Neurosymbolic 3D Modeling Pipeline

## ShapeAssembly DSL (Section 4)

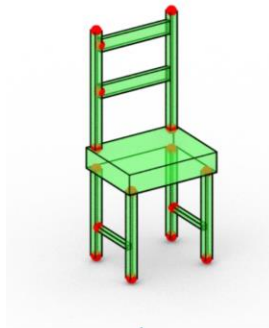
```

Start → BBlock; CBlock; ABlock; SBlock;
BBlock → bbox = Cuboid( $l, h, w$ , True)
CBlock →  $c_n$  = Cuboid( $l, w, h, a$ ) ; CBlock | None
ABlock → Attach ; ABlock | Squeeze ; ABlock | None
SBlock → Reflect ; SBlock | Translate ; SBlock | None
Attach → attach( $c_{n_1}, c_{n_2}, x_1, y_1, z_1, x_2, y_2, z_2$ )
Squeeze → squeeze( $c_{n_1}, c_{n_2}, c_{n_3}, f, u, v$ )
Reflect → reflect( $c_{n_1}, \text{axis}$ )
Translate → translate( $c_{n_1}, \text{axis}, m, d$ )
 $f \rightarrow \text{right} \mid \text{left} \mid \text{top} \mid \text{bot} \mid \text{front} \mid \text{back}$ 
 $\text{axis} \rightarrow X \mid Y \mid Z$ 
 $l, h, w \in \mathbb{R}^3$ 
 $x, y, z, u, v, d \in [0, 1]^2$ 
 $a \in [\text{True}, \text{False}]$ 
 $n, m \in \mathbb{Z}^+$ 
    
```

## Input Hierarchical Part Graphs



## Shapes to Training Programs (Section 5)

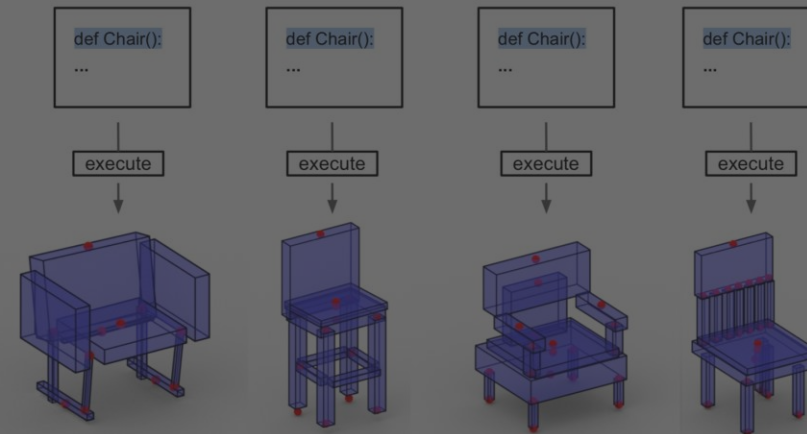
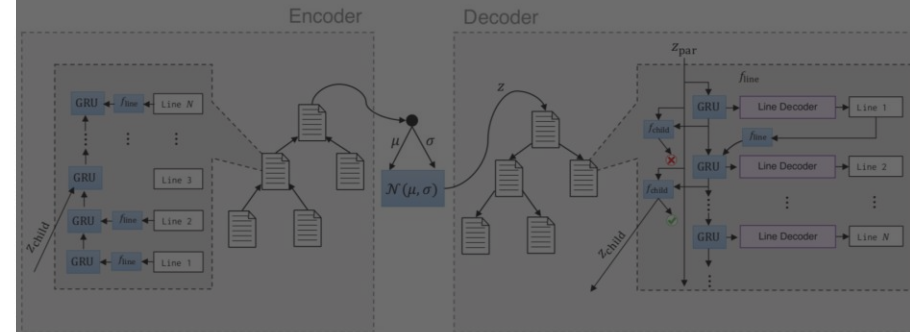


```

def Chair():
    bbox = Cuboid(.7, 1.7, .5, True)
    prog1 = Program1(.7, .6, .5, True)
    prog2 = Program2(.7, .9, .05, True)
    cube2 = Cuboid(.7, .15, .5, True)
    attach(prog1, bbox, .5, 0, .5, .5, 0, .5)
    attach(cube2, prog1, .5, 0, .5, .5, 1, .5)
    squeeze(prog2, bbox, cube2, top, .5, .1)

def Program1( $l, w, h, \text{aligned}$ ):
    bbox = Cuboid(.7, .6, .5, True)
    prog3 = Program3(.05, .6, .5, True)
    squeeze(prog3, bbox, bbox, top, 0, .5)
    reflect(prog3, X)
    ...
    
```

## Learning to Generate Programs (Section 6)



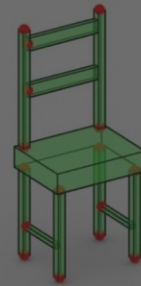


# A Neurosymbolic 3D Modeling Pipeline

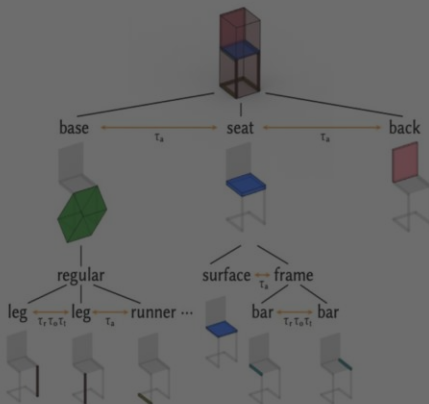
## ShapeAssembly DSL (Section 4)

Start  $\rightarrow$  BBlock; CBlock; ABlock; SBlock;  
 BBlock  $\rightarrow$  bbox = Cuboid( $l, h, w$ , True)  
 CBlock  $\rightarrow$   $c_n$  = Cuboid( $l, w, h, a$ ) ; CBlock | None  
 ABlock  $\rightarrow$  Attach ; ABlock | Squeeze ; ABlock | None  
 SBlock  $\rightarrow$  Reflect ; SBlock | Translate ; SBlock | None  
 Attach  $\rightarrow$  attach( $c_{n_1}, c_{n_2}, x_1, y_1, z_1, x_2, y_2, z_2$ )  
 Squeeze  $\rightarrow$  squeeze( $c_{n_1}, c_{n_2}, c_{n_3}, f, u, v$ )  
 Reflect  $\rightarrow$  reflect( $c_{n_1}, \text{axis}$ )  
 Translate  $\rightarrow$  translate( $c_{n_1}, \text{axis}, m, d$ )  
 $f \rightarrow$  right | left | top | bot | front | back  
 axis  $\rightarrow$  X | Y | Z  
 $l, h, w \in \mathbb{R}^3$   
 $x, y, z, u, v, d \in [0, 1]^2$   
 $a \in [\text{True}, \text{False}]$   
 $n, m \in \mathbb{Z}^+$

## Shapes to Training Programs (Section 5)



## Input Hierarchical Part Graphs

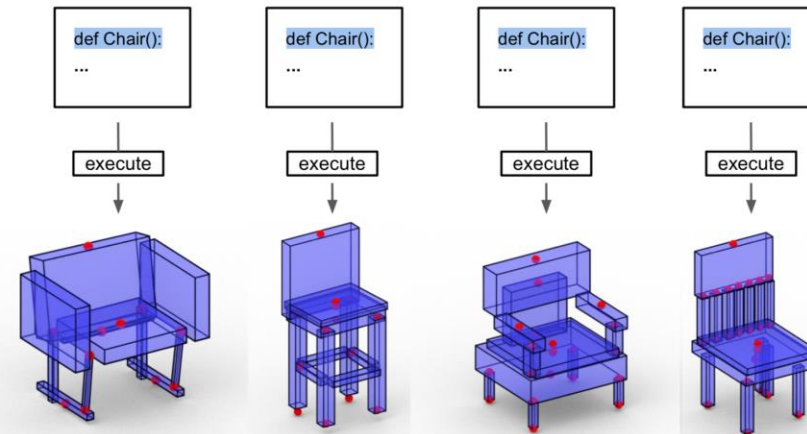
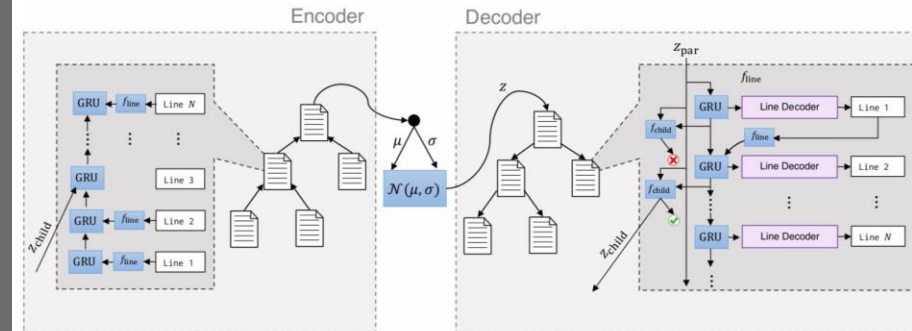


```

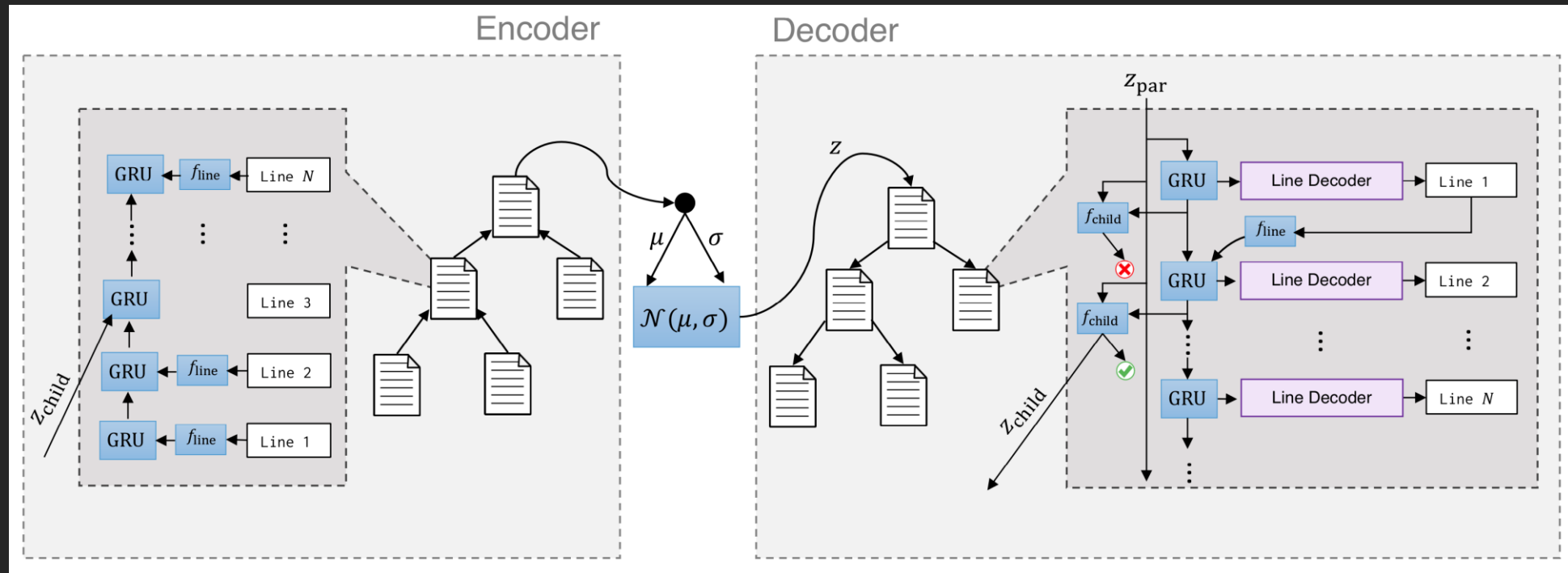
def Chair():
    bbox = Cuboid(.7, 1.7, .5, True)
    prog1 = Program1(.7, .6, .5, True)
    prog2 = Program2(.7, .9, .05, True)
    cube2 = Cuboid(.7, .15, .5, True)
    attach(prog1, bbox, .5, 0, .5, .5, 0, .5)
    attach(cube2, prog1, .5, 0, .5, .5, 1, .5)
    squeeze(prog2, bbox, cube2, top, .5, .1)

def Program1(l, w, h, aligned):
    bbox = Cuboid(.7, .6, .5, True)
    prog3 = Program3(.05, .6, .5, True)
    squeeze(prog3, bbox, bbox, top, 0, .5)
    reflect(prog3, X)
    ...
    
```

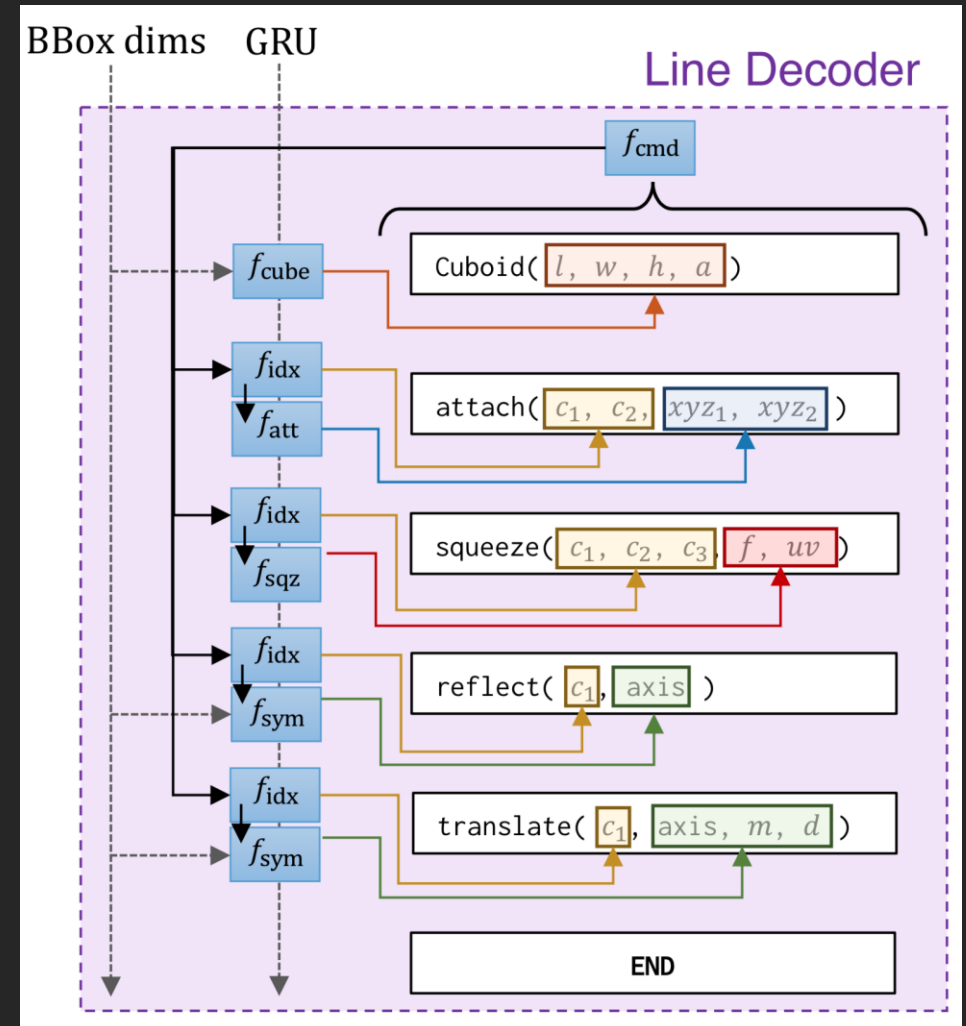
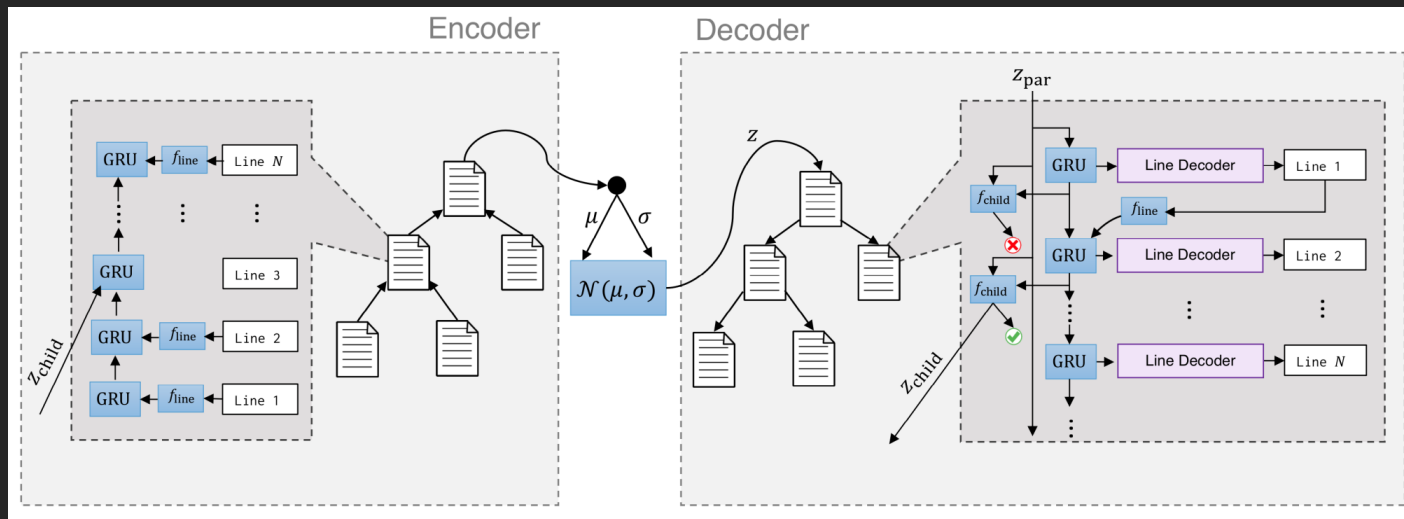
## Learning to Generate Programs (Section 6)



# Learning to Write ShapeAssembly Programs



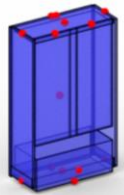
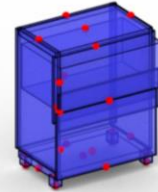
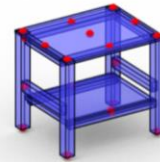
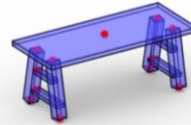
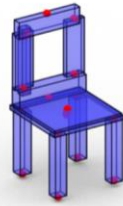
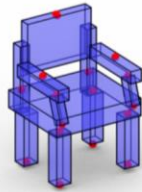
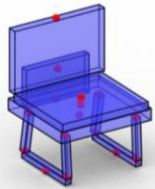
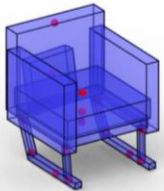
# Learning to Write ShapeAssembly Programs



**WHAT CAN YOU DO WITH IT?**

# Novel Shape Generation

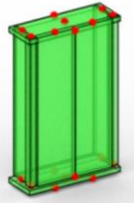
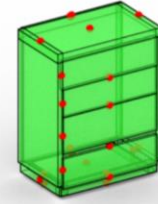
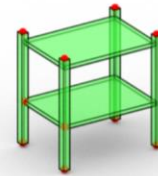
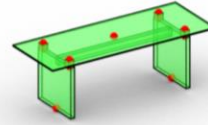
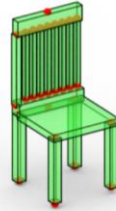
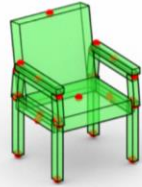
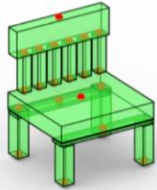
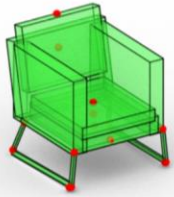
Sample



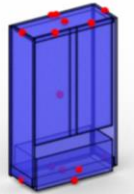
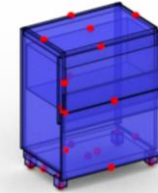
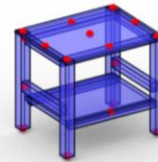
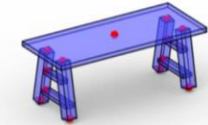
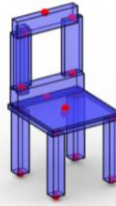
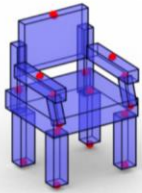
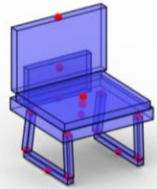
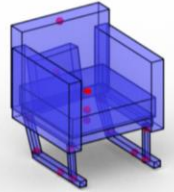


# Novel Shape Generation

Geom NN

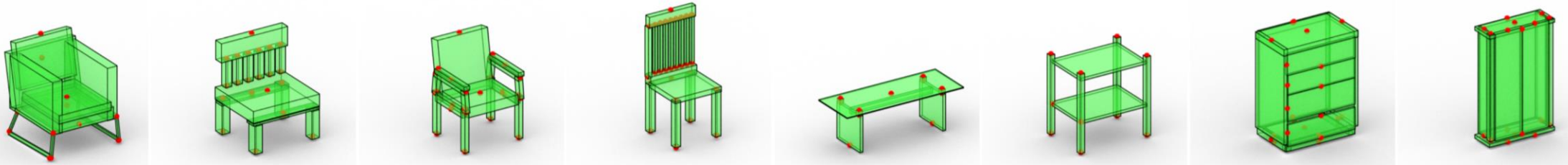


Sample

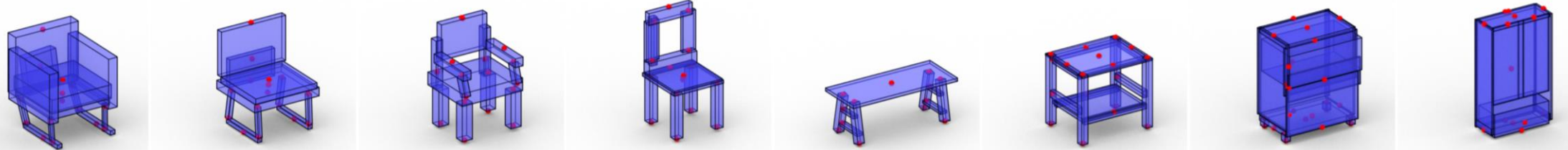


# Novel Shape Generation

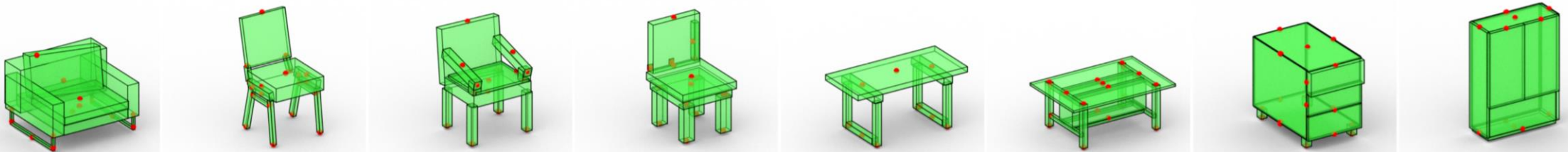
Geom NN



Sample

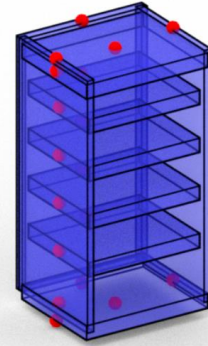
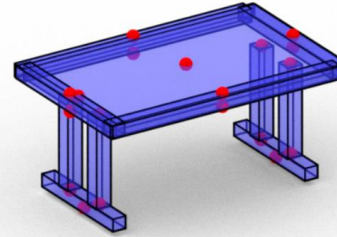
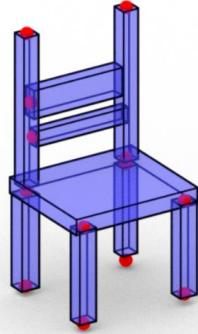
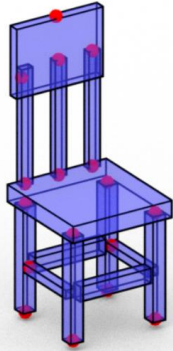


Prog NN



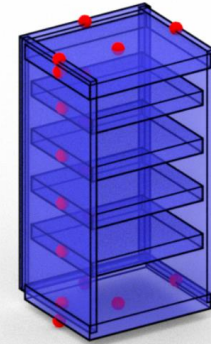
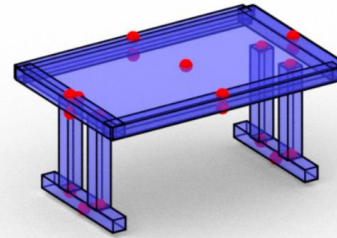
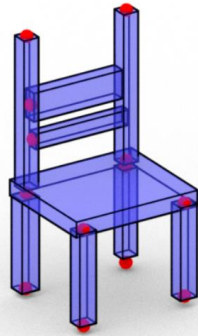
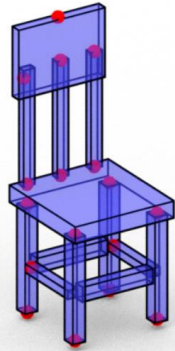
# Editing Generated Programs

Sample

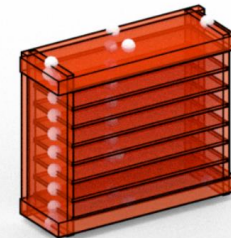
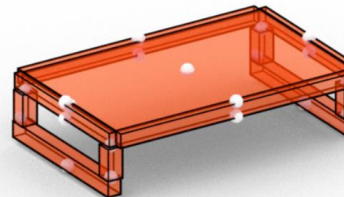
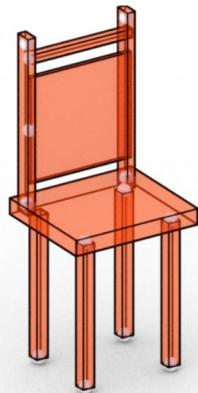
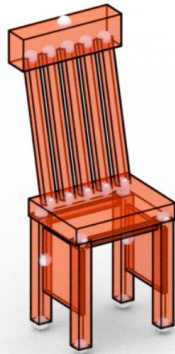


# Editing Generated Programs

Sample



Variant



# Comparison Conditions

- **3D PRNN:**  
Sequence of boxes, but no hierarchy or relations
- **StructureNet:**  
Hierarchy of boxes w/ symmetry relations, but no explicit parametric attachments

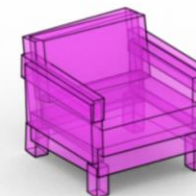
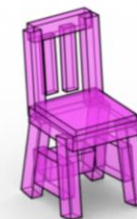
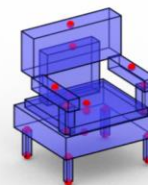
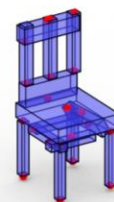
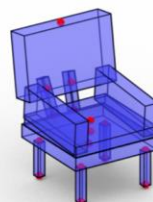
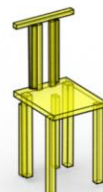
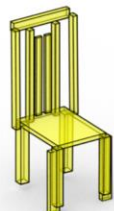
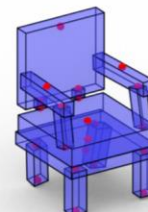
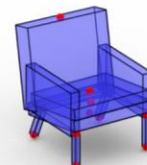
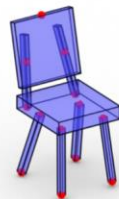
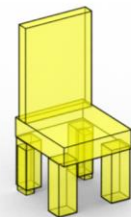
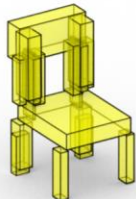
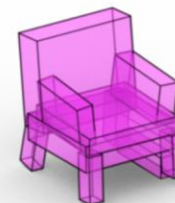
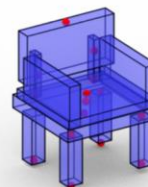
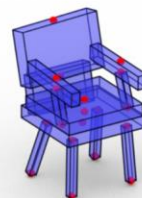
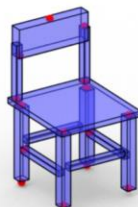
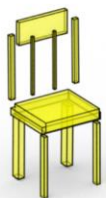


# Ours Generates Better Novel Shapes

3D-PRNN

Ours

## StructureNet



## Chair



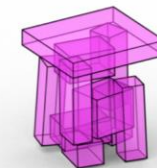
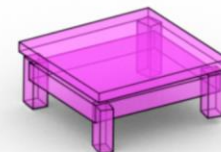
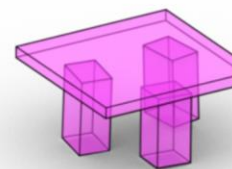
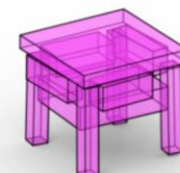
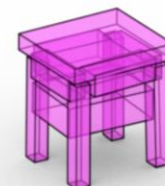
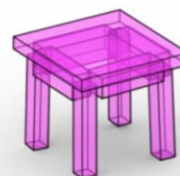
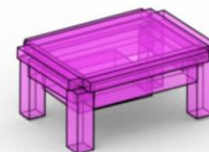
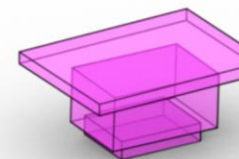
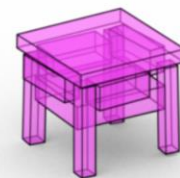
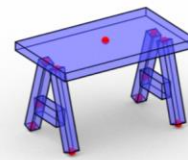
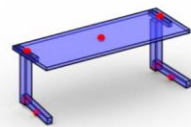
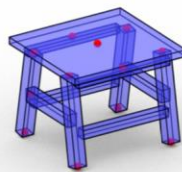
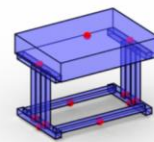
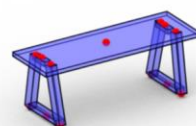
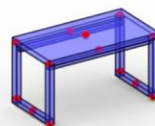
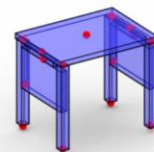
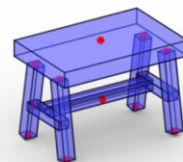
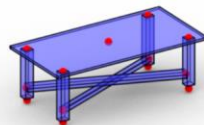
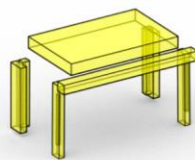
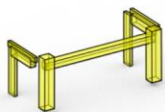
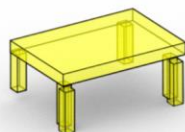
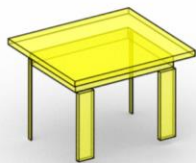
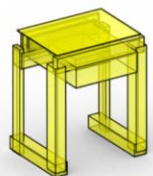
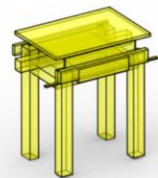
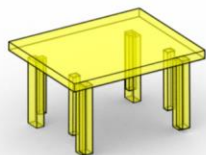
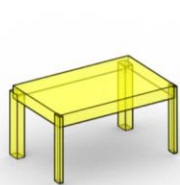
# Ours Generates Better Novel Shapes

3D-PRNN

Ours

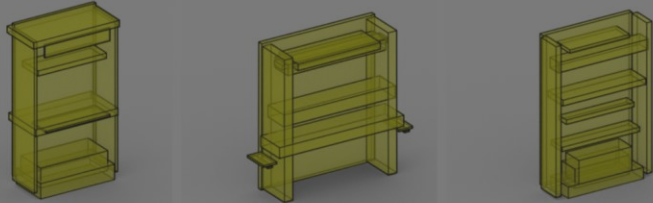
StructureNet

Table

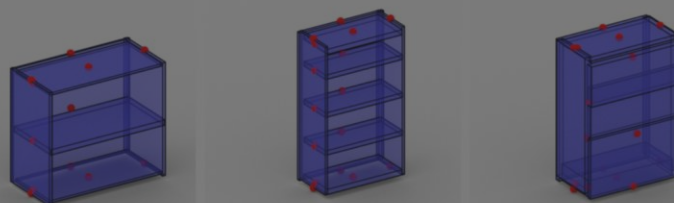


# Ours Generates Better Novel Shapes

3D-PRNN



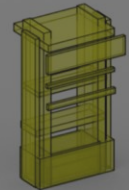
Ours



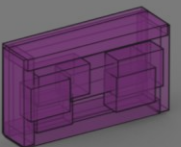
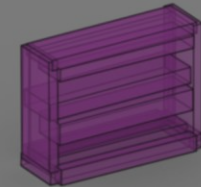
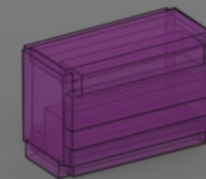
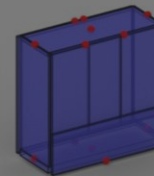
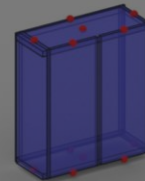
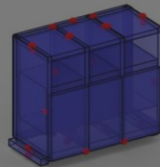
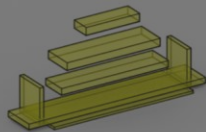
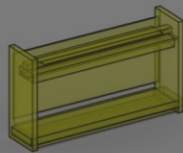
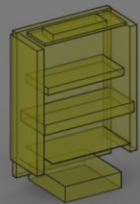
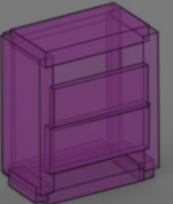
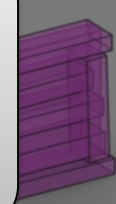
StructureNet



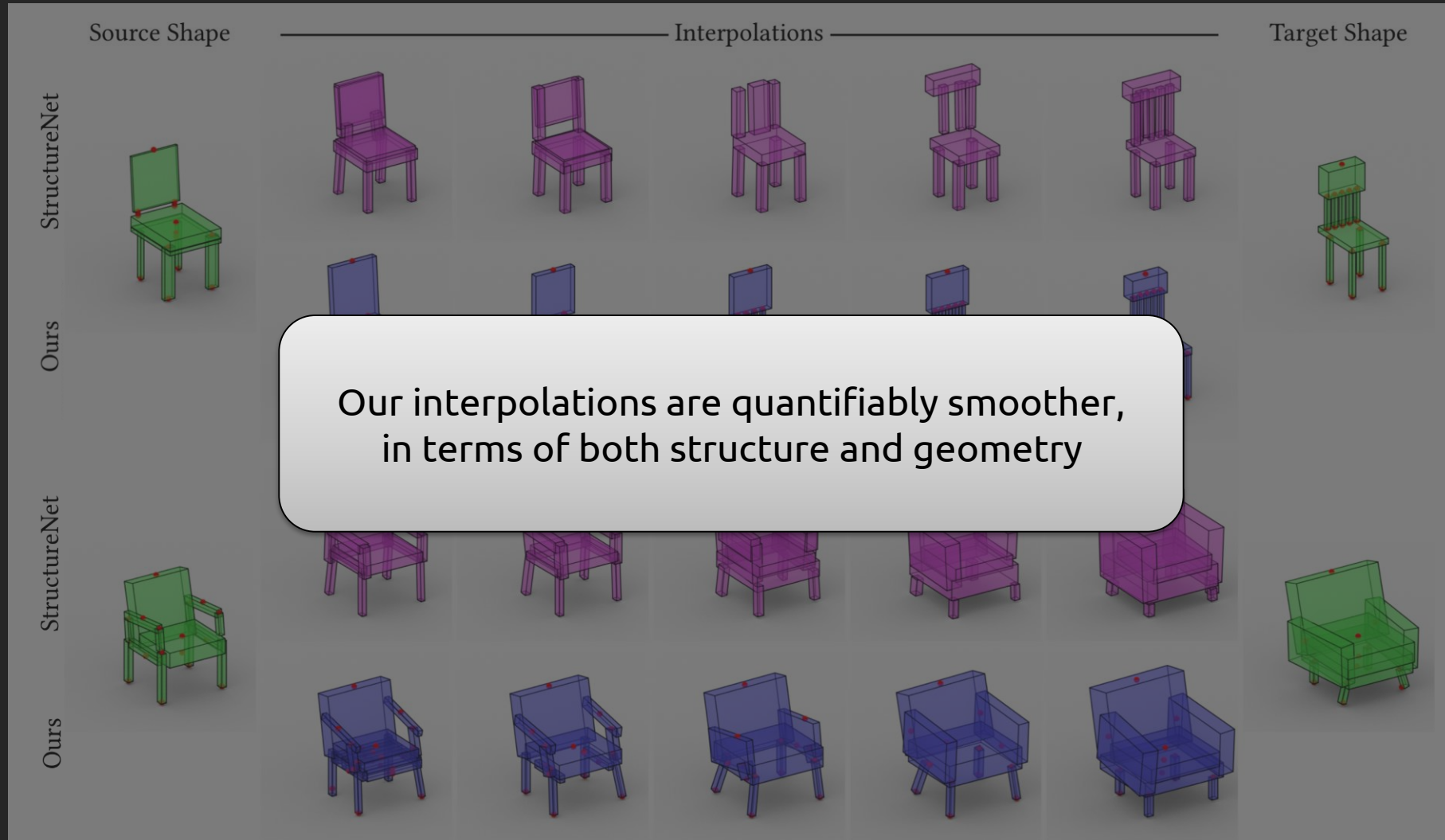
Storage



Ours are also quantifiably more compact, physically stable, and distributionally similar to a held-out validation set



# Ours Produces Better Interpolation

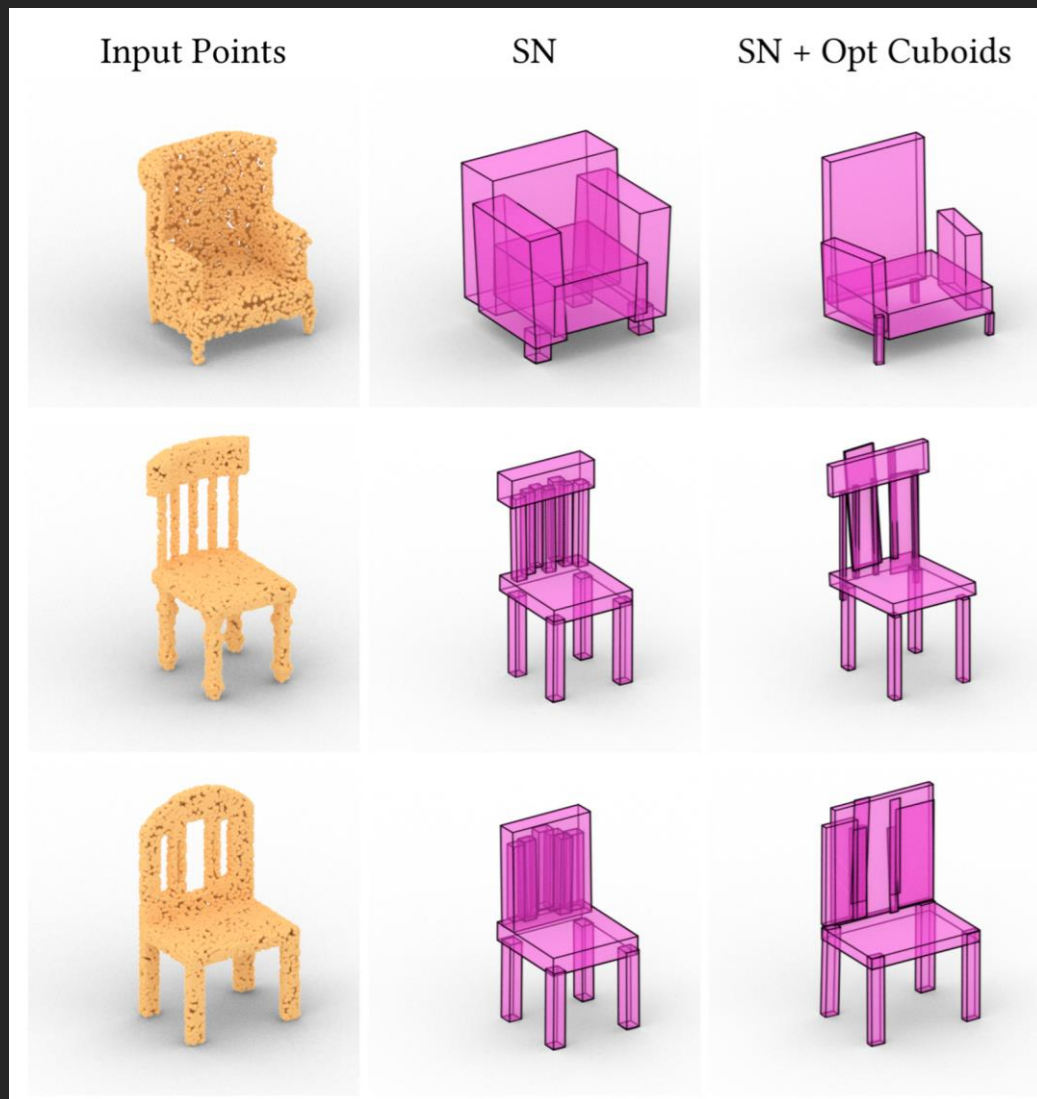


# Point Cloud “Parsing”

Input Points

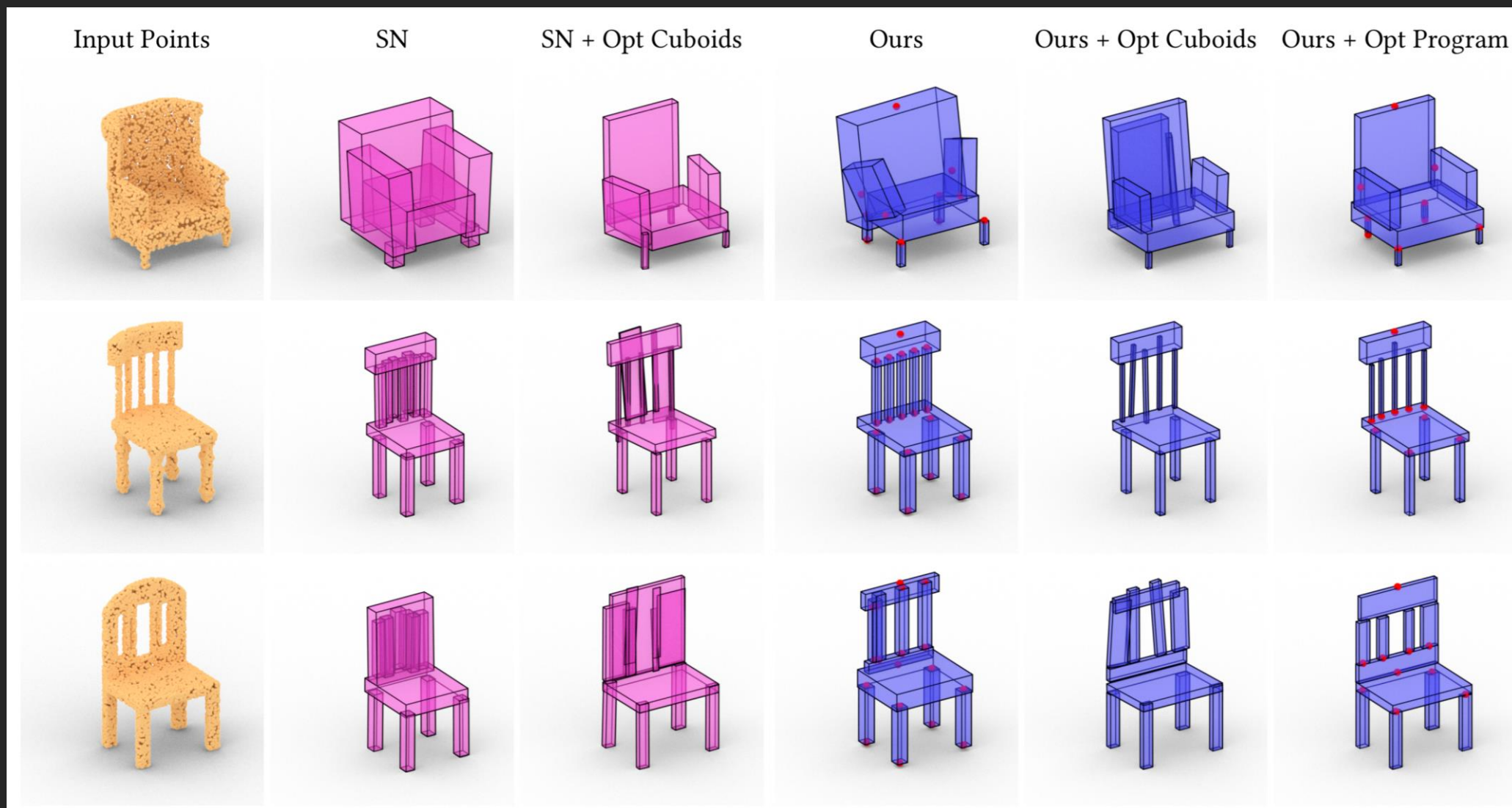


# Point Cloud “Parsing”





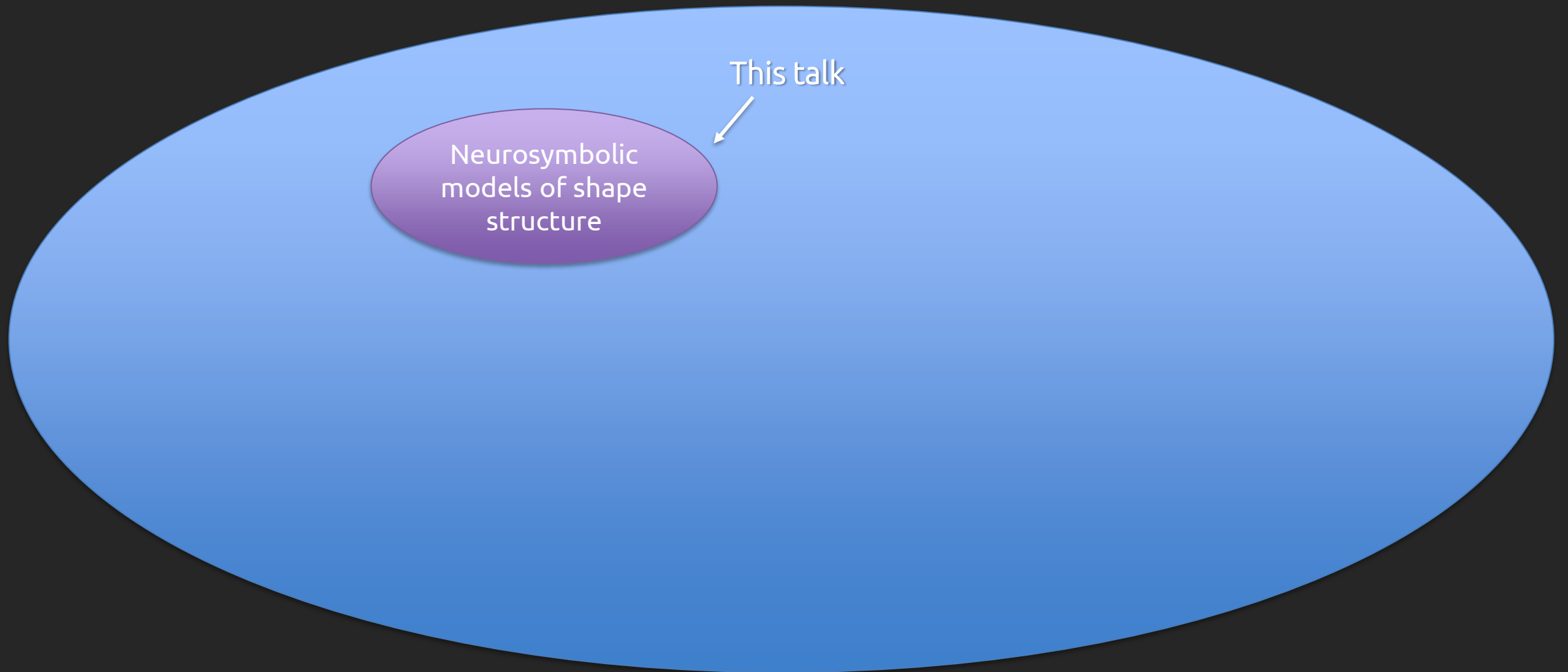
# Point Cloud “Parsing”



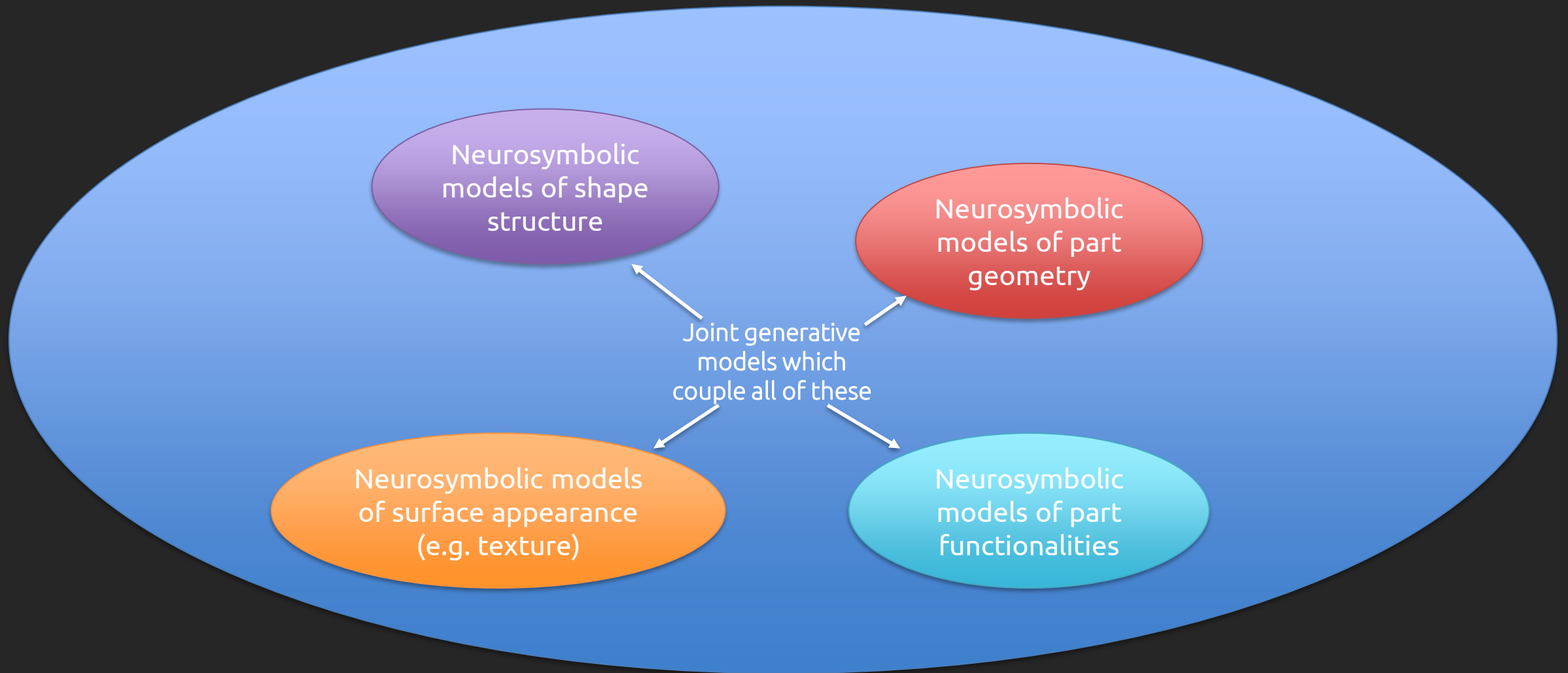


**WHAT'S NEXT?**

# Neurosymbolic 3D Model Design Space



# Neurosymbolic 3D Model Design Space



**THANKS!**