Generalized Matryoshka
Computational Design of Nesting Objects

Alec Jacobson
University of Toronto
Previous work enables computational design of reconfigurables [Garg et al. 2016]
Previous work enables computational design of reconfigurables

[Garg et al. 2016]
Previous work enables computational design of reconfigurables

[Garg et al. 2016]
[Zvyozdochkin & Malyutin 1890]
We present a method to generalize Matryoshka to arbitrary shapes
We present a method to generalize Matryoshka to arbitrary shapes
Nesting requires strict enclosure…

loose enclosure
Nesting requires strict enclosure…
Nesting also requires removal

loose enclosure

enclosed, but not removable
Nesting also requires *removal*

- loose enclosure
- enclosed, but not removable
Nesting also requires *removal*

loose enclosure  
enclosed, but not removable
Nesting also requires *removal*

- loose enclosure
- enclosed, but not removable
- enclosed and removable
We present highly parallelizable methods to...

- determine feasibility of nesting,
We present highly parallelizable methods to...

• determine feasibility of nesting,
• find maximum scale,
We present highly parallelizable methods to...

- determine feasibility of nesting,
- find maximum scale, and
- optimize nesting scale over some or all parameters
Our optimization utilizes rigid motion for tighter nesting

fixed position+rotation

39%
Our optimization utilizes rigid motion for tighter nesting

fixed position+rotation

fixed rotation

39%

53%
Our optimization utilizes rigid motion for tighter nesting

fixed position+rotation  fixed rotation  free

39%  53%  63%
We define valid self-nesting

Given:
1. shape \( A \),
We define valid self-nesting

Given:
1. shape $\mathcal{A}$,
2. similarity transform $T$, 

We define valid self-nesting

Given:
1. shape $\mathcal{A}$,
2. similarity transform $T$,
3. cut plane $\mathcal{P}$, and
We define *valid self-nesting*

Given:
1. shape $\mathcal{A}$,
2. similarity transform $T$,
3. cut plane $\mathbf{p}$, and
4. removal *trajectories*
We define valid self-nesting

Given:
1. shape $A$, 
2. similarity transform $T$, 
3. cut plane $P$, and 
4. removal trajectories directions
We define valid self-nesting

Must have:

1. $T(A) \subset A$, and
2. no collisions along either direction after cutting $A$ by $P$
We define valid self-nesting

Must have:

1. $T(A) \subset A$, and
2. no collisions along either direction after cutting $A$ by $P$

Definition depends on choice of cut plane and removal directions.
Some configurations admit *perfect self-nesting* convex shapes?
Some configurations admit *perfect self-nesting*

convex shapes?

enclosure is easy ....
Some configurations admit *perfect self-nesting*

- convex shapes?
- enclosure is easy ....
- but removal depends on cut plane!
[Zvyozdochkin & Malyutin 1890]
valid
Perfect self-nesting requires **visibility** of cut plane at all points along removal directions.
Our tool explores nesting of *arbitrary* solid 3D shapes
Our tool explores nesting of *arbitrary* solid 3D shapes
Our tool explores nesting of *arbitrary* solid 3D shapes
We cast this as a computational design problem.

Manual design with traditional tools would be tortuous.
We cast this as a *computational design* problem

Manual design with traditional tools would be tortuous
We cast this as a *computational design* problem

Manual design with traditional tools would be tortuous
We cast this as a \textit{computational design} problem

Manual design with traditional tools would be tortuous
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Take a clue from order-independent transparency by “depth peeling”
Take a clue from order-independent transparency by “depth peeling”
Take a clue from order-independent transparency by “depth peeling”
Take a clue from order-independent transparency by “depth peeling”

a.k.a. K-Buffer, Layered Depth Images

transparency
shape diameter
image-based rendering
CNC milling
intersection volume
swept volumes
collision detection
CSG operations

[Everitt 2001, Bavoil et al. 2007]
[Baldacci et al. 2016]
[Shade et al. 1998]
[Inui & Ohta 2007]
[Faure et al. 2008]
[Kim et al. 2002]
Take a clue from order-independent transparency by “depth peeling”

a.k.a. K-Buffer, Layered Depth Images

transparency
shape diameter
image-based rendering
CNC milling
intersection volume
swept volumes
collision detection
CSG operations

[Everitt 2001, Bavoil et al. 2007]
[Baldacci et al. 2016]
[Shade et al. 1998]
[Inui & Ohta 2007]
[Faure et al. 2008]
[Kim et al. 2002]


Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering

Bad “codes”:
• blue before orange
• orange before green
• orange before front-facing blue
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering
Step 1: we determine feasibility in real-time by exploiting orthographic rendering.
Step 1: we determine feasibility in real-time by exploiting orthographic rendering

Feasible!
• all green
Step 1: we determine feasibility in real-time by exploiting orthographic rendering

“ping-pong” with 2 buffers

GL_SAMPLES_PASSED

Feasible!
• all green
Step 2: binary search to maximize scale

Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation
Step 2: binary search to maximize scale

Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation
Step 2: binary search to maximize scale

Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation
Step 2: binary search to maximize scale

Assume momentarily that shape is convex

Fix cut plane, center of mass, rotation
Step 2: binary search to maximize scale

Assume *momentarily* that shape is convex.

Fix cut plane, center of mass, rotation.
Step 2: binary search to maximize scale

Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation
Step 2: binary search to maximize scale

For non-convex shapes, binary search is conservative, but in practice optimal.
Step 2: binary search to maximize scale

For non-convex shapes, binary search is conservative, but in practice optimal.
Step 2: binary search to maximize scale

For non-convex shapes, binary search is conservative, but in practice optimal.
Step 2: binary search to maximize scale

For non-convex shapes, binary search is conservative, but in practice optimal.
Step 2: binary search to maximize scale

For non-convex shapes, binary search is conservative, but in practice optimal.
Step 3: optimize over all parameters

maximize scale subject to nesting constraint

non-convex energy landscape
Step 3: optimize over all parameters via particle swarm optimization

\[ k \text{ parameter vector as point in } nD \quad x_i \in \mathbb{R}^n \]
Step 3: optimize over all parameters via particle swarm optimization

$k$ parameter vector as point in $n$D \( \mathbf{x}_i \in \mathbb{R}^n \)

update each iteration according to “velocity”

\[
\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,
\]
Step 3: optimize over all parameters via particle swarm optimization

$k$ parameter vector as point in $n$D \( \mathbf{x}_i \in \mathbb{R}^n \)

pull velocity toward personal best and global best of swarm

\[
\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + \phi_p r_p (\mathbf{x}^p_i - \mathbf{x}_i) + \phi_g r_g (\mathbf{x}^g - \mathbf{x}_i),
\]

\[
\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,
\]
Step 3: optimize over all parameters via *particle swarm optimization*

A parameter vector as point in $n$D: $\mathbf{x}_i \in \mathbb{R}^n$

\[
\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + \phi_p r_p(\mathbf{x}_i^p - \mathbf{x}_i) + \phi_g r_g(\mathbf{x}_i^g - \mathbf{x}_i),
\]

\[
\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,
\]

*random perturbations*
Naive P-Swarm would treat scale as just another parameter (coordinate)...
... instead optimize over all others,

$$\max_{R, c, P, a^+, a^-} f(R, c, P, a^+, a^-)$$

all other parameters
... instead optimize over all others,

\[
\max_{R,c,P,a^+,a^-} f(R, c, P, a^+, a^-)
\]

where

\[
f(R, c, P, a^+, a^-) = \max_s s
\]

such that \( T(\mathcal{B}) \) nests in \( \mathcal{A} \) w.r.t. \( P, a^+, a^- \)
... instead optimize over all others, and \textit{search} for max scale

$$\text{maximize } f(R, c, P, a^+, a^-)$$

where

$$f \approx \text{search}(R, c, P, a^+, a^-)$$
... instead optimize over all others, and search for max scale

$$\max_{R, c, P, a^+, a^-} f(R, c, P, a^+, a^-)$$

where

$$f \approx \underset{s}{\text{search}}(R, c, P, a^+, a^-)$$

abort search early if upper bound < best
Our optimization enables fully automatic Matryoshka generation…

63% fully optimized
... or partially constrained interactive design

63%  
60%  
fixed upright orientation
… or partially constrained interactive design
Tool performs fast enough for interaction
Tool performs fast enough for interaction

Fixed Rotation
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We validate our results via 3D printing
We accommodate printer tolerances by nesting *within* an offset surface.
Our tools trivially generalize to nesting disparate shapes
Limitations & Future Work

- no global optimum guarantee
Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don’t rigidly nest well
Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don’t *rigidly* nest well
- deformable nesting?
Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don’t *rigidly* nest well
- deformable nesting?
  1. deform during design

Prevost et al.
Limitations & Future Work

• no global optimum guarantee
• search assumption too conservative
• thin shapes don’t *rigidly* nest well
• deformable nesting?
  1. deform during design
  2. nest soft physical objects

Bickel et al.
Acknowledgements…

David Levin
NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938)
Connaught Fund (NR-2016-17)
Adobe Systems Inc.
Kevin Gibson, Masha Shugrina, Michael Tao, and Alex Tessier
Generalized Matryoshka
Computational Design of Nesting Objects

Alec Jacobson
jacobson@cs.toronto.edu