Functional Programming for Compiling and Decompiling Computer-Aided Design

Chandrakana Nandi, James R. Wilcox, Pavel Panchekha, Taylor Blau, Dan Grossman, Zachary Tatlock
ICFP 2018
The 3D Printing Revolution

No Donor Required: 5 Body Parts You Can Make With 3-D Printers

ENABLING THE FUTURE

3D-printed prosthetic limbs: the next revolution in medicine
Democratized Fabrication

PLSE printed!

Hardware cost has gone down!
Democratized Fabrication

Design tools challenges

- steep learning curve
- lack of specifications
- expensive
Democratized Fabrication

Design tools challenges
- steep learning curve
- lack of specifications
- expensive

Online repositories
- Thingiverse
- YM
- GRABCAD
A mesh for a hex holder

Triangular mesh
A mesh for a hex holder

Bent wrench not parallel to the rest :(  
Need to rotate the fifth hole
A mesh for a hex holder

Simple mesh editing broke model

No abstraction

Move around the vertices manually
A mesh for a hex holder

CAD (Computer Aided Design) editing is easier
Higher level of abstraction
Easier to visualize outcome
A mesh for a hex holder

Higher level of abstraction

Easier to visualize outcome

Automatically infer CAD from Mesh!

: \textit{mesh} \rightarrow \textit{cad}
Mesh

1600 LOC
difference (
  scale (97.0, 25.0, 5.0) cube
  trans (49.0, 13.0, 2.5) (
    scale (7.0, 6.06, 5.0) (polyhedron 6)
  )
... )
Rotating by 35 degrees solved the problem!
Rotating by 35 degrees solved the problem!
Automatically infer CAD from Mesh!

: \textit{mesh} \rightarrow \textit{cad}
How?

Key insight: view the computational fabrication pipeline as a compiler
How?

Key insight: view the computational fabrication pipeline as a compiler

PL foundations applied to computational fabrication to provide *clarity* and *usefulness*

**Clarity**

- Denotational semantics, inductive definitions
- Proof of correctness of a compiler from CAD to mesh

**Usefulness**

- Program synthesis to reverse engineer CAD from mesh
Talk Outline

• 3D Printing Workflow

• *Clarity* achieved by applying FP to fabrication

• *Usefulness*: the *first* decompiler from mesh to CAD
Talk Outline

• 3D Printing Workflow

• **Clarity** achieved by applying FP to fabrication

• **Usefulness**: the first decompiler from mesh to CAD
3D printing workflow
1. Design

CAD
difference ( 
scale (97, 25, 5) ( 
cube 
) 
trans (10, 13, 0) ( 
scale (5, 5, 5) ( 
polyhedron 6 
) 
... 
))
1. Design

Idea

2. Compile

CAD
difference (scale (97, 25, 5) (cube)
  trans (10, 13, 0) (scale (5, 5, 5) (polyhedron 6)
  ...
))

Mesh
1. Design

Idea

2. Compile

CAD
difference (scale (97, 25, 5) (cube
  trans (10, 13, 0) (scale (5, 5, 5) (polyhedron 6
  ...
))

3. Slice

Mesh

G-code

G1 X79.629 Y66.912 E0.0020 F1200
G1 X81.530 Y65.814 E0.0875
G1 X81.581 Y65.800 E0.0896
G1 X118.419 Y65.800 E1.5231
G1 X118.469 Y65.814 E1.5251
G1 X120.371 Y66.912 E1.6106
G1 X120.409 Y66.949 E1.6126
G1 X138.827 Y98.051 E3.0461
G1 X138.841 Y98.902 E3.0482
G1 X138.841 Y101.098 E3.1336
G1 X138.827 Y101.149 E3.1357
G1 X120.409 Y133.051 E4.5692
G1 X120.371 Y133.088 E4.5712
G1 X118.469 Y134.186 E4.6567
G1 X118.419 Y134.200 E4.6588
G1 X81.581 Y134.200 E6.0923
G1 X81.530 Y134.186 E6.0943
G1 X79.629 Y133.088 E6.1798
1. Design
2. Compile
3. Slice
4. Print
1. Design

Idea

2. Compile

CAD

difference (
  scale (97, 25, 5) (cube
  trans (10, 13, 0) (scale (5, 5, 5) (polyhedron 6
  ...))
)

3. Slice

Mesh

4. Print

Part

G-code

5. OK?
1. Design

G-code

difference ( scale (97, 25, 5) ( cube 
  trans (10, 13, 0) ( scale (5, 5, 5) ( polyhedron 6 
  ... 
)) )

2. Compile

3. Slice

 iterate

4. Print

Part

Mesh

5. OK?
1. Design

2. Compile

3. Slice

4. Print

5. OK?

CAD

difference (scale (97, 25, 5) (cube
trans (10, 13, 0) (scale (5, 5, 5) (polyhedron 6))...))

Mesh

Part

Mesh

G-code

iterate

SAILFISH

Marlin
Talk Outline

- 3D Printing Workflow

- Clarity achieved by applying FP to fabrication

- Usefulness: the first decompiler from mesh to CAD
Talk Outline

- 3D Printing Workflow

- *Clarity* achieved by applying FP to fabrication
  - Denotational semantics for CAD and mesh
  - Inductive compiler definition
  - Proof of correctness for compiler

- *Usefulness*: the *first* decompiler from mesh to CAD
CAD

3D primitive representing a cube of unit length
CAD

cube

3D primitive representing a cube of unit length

polyhedron 6

3D primitive representing a hexagonal prism
scale (5, 5, 5) polyhedron 6
difference (  
  scale (97, 25, 5) (  
    cube  
  )  
  trans (10, 13, 0) (  
    scale (5, 5, 5) (  
      polyhedron 6  
    )  
  )  
)
\[ c ::= \begin{align*} & \text{Empty} \\ & \text{Cube} \\ & \text{Polyhedron } \mathbb{N} \\ & \text{Affine } \mathbb{R}^{3 \times 3} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \\ & \text{Binop } op \begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} \end{align*} \]

\[ op ::= \begin{align*} & \text{Union} \\ & \text{Difference} \\ & \text{Intersection} \end{align*} \]
**CAD**: Denotational semantics

\[
[c]_{cad} : \{ \text{all points inside } c \}
\]

\[
[Empty]_{cad} = \{ \}
\]

\[
[Cube]_{cad} = \{ (0, 1)^3 \}
\]

\[
[\text{Affine } p q c]_{cad} = \{ pv + q | v \in [c]_{cad} \}
\]

\[
[\text{Binop Union } c_1 c_2]_{cad} = [c_1]_{cad} \cup [c_2]_{cad}
\]

Regular opens
Mesh

List of faces
A face is a triangular plane

Convenient for geometric operations
CAD tool agnostic
Standard format for 3D models
Widely shared in online repositories
Mesh: Denotational semantics

$[m]_{mesh} : \{\text{all points inside } m\}$

Key insight: view the computational fabrication pipeline as a compiler

CAD is based on solid geometry, mesh is based on surface geometry

We need tools that relate these two different representations

$pt$ is inside $m$ iff a ray starting at $pt$ crosses an odd number of faces of $m$
Talk Outline

• 3D Printing Workflow

• *Clarity* achieved by applying FP to fabrication
  
  • Denotational semantics for CAD and mesh
  
  • Inductive compiler definition
  
  • Proof of correctness for compiler

• *Usefulness*: the *first* decompiler from mesh to CAD
Talk Outline

• 3D Printing Workflow

• *Clarity* achieved by applying FP to fabrication
  - Denotational semantics for CAD and mesh 
  - Inductive compiler definition
  - Proof of correctness for compiler

• *Usefulness*: the *first* decompiler from mesh to CAD
Compiling CAD to mesh

\[ \text{compile} : c \rightarrow m \]

\[ \text{compile cube} = m_{\text{cube}} \]

\[ \text{compile } (\text{Affine } p \ q \ c) = \text{map}_{\text{vertex}}(\lambda v \cdot p v + q) \ (\text{compile } c) \]

\[ \text{compile } (\text{Binop } \cup \ c_1 \ c_2) = \text{meshBinop}(\cup) \ (\text{compile}(c_1), \text{compile}(c_2)) \]

Regular opens at mesh level too: check normals!
Compiler on an example

union
cube
trans (2, 0, 0) cube
Compiler on an example

union

cube

trans (2, 0, 0) cube

→

$C$

$\begin{bmatrix}
\text{union} \\
[]
\text{trans} (2, 0, 0) \text{ cube}
\end{bmatrix}$

$\begin{bmatrix}
\text{cube}
\end{bmatrix}$
Compiler on an example

Evaluation context
Compiler on an example

union
cube
trans (2, 0, 0) cube

Evaluation context
Compiler on an example

union 

cube

trans (2, 0, 0) cube

\[\text{union} \]
\[\text{trans (2, 0, 0) cube} \]

\[\text{union} \]
\[\text{trans (2, 0, 0) cube} \]

\[\text{Mesh [m}_c\text{ube}] \]

\[\text{union} \]
\[\text{Mesh [m}_c\text{ube}] \]

\[\text{trans (2, 0, 0) cube} \]
Compiler on an example

Evaluation context
Compiler on an example

union

cube

trans (2, 0, 0) cube

union

cube

trans (2, 0, 0) cube

Mesh [m_cube]
Compiler on an example

union
cube
trans (2, 0, 0) cube

Evaluation context

union
Mesh [m_{cube}]
Mesh [m_{trans (2, 0, 0) cube}]

union
Mesh [m_{cube}]
union
Mesh [m_{cube}]
union
Mesh [m_{cube}]

union
Mesh [m_{trans (2, 0, 0) cube}]
union
Mesh [m_{trans (2, 0, 0) cube}]
union
trans (2, 0, 0) cube
Compiler on an example

TRICKY when faces overlap!

Need to split meshes w.r.t one another to resolve overlaps

Need to determine which faces from $\text{compile}(c_1)$ and $\text{compile}(c_2)$ to keep in the final mesh
Talk Outline

- 3D Printing Workflow
- *Clarity* achieved by applying FP to fabrication
  - Denotational semantics for CAD and mesh
  - Inductive compiler definition
  - Proof of correctness for compiler
- *Usefulness*: the *first* decompiler from mesh to CAD
Talk Outline

- 3D Printing Workflow
- *Clarity* achieved by applying FP to fabrication
  - Denotational semantics for CAD and mesh
  - Inductive compiler definition
- Proof of correctness for compiler
- *Usefulness*: the *first* decompiler from mesh to CAD
Compiler Proof

Thm : \( \forall e, [\text{compile } e]_{\text{mesh}} = [e]_{\text{cad}} \)

Proof: By induction on e

- Case Primitives: …
- Case Affine Transformations: …
- Case Union:
Compiler Proof

Thm: $\forall e, \left[\text{compile } e\right]_{mesh} = \left[e\right]_{cad}$

Proof: By induction on $e$

- Case Primitives: …
- Case Affine Transformations: …
- Case Union:

Union lemma

$\left[\text{meshBinop (Union) } (m_1, m_2)\right]_{mesh} = \left[m_1\right]_{mesh} \cup \left[m_2\right]_{mesh}$
Compiler Proof

Thm : \( \forall e, \left[ \text{compile } e \right]_{mesh} = \left[ e \right]_{cad} \)

Union lemma

\[
\left[ meshBinop (Union) (m_1, m_2) \right]_{mesh} = \left[ m_1 \right]_{mesh} \cup \left[ m_2 \right]_{mesh}
\]

let \( m_3 = mBop (Union) (m_1, m_2) \)

\[
pt \in \left[ m_3 \right]_{mesh} \iff pt \in \left[ m_1 \right]_{mesh} \cup \left[ m_2 \right]_{mesh}
\]

Use Ray casting
Case: overlapping meshes
Case: overlapping meshes
Case: overlapping meshes

Sufficient to show that $h$ crosses an odd number of faces of $m_3$ iff it crosses an odd number of edges of $m_1$ or $m_2$
Case: overlapping meshes

Divide $h$ into 4 sub-regions

Since meshes are split, sufficient to show that $h_i$ is inside $m_3$ iff it is inside $m_1$ or $m_2$

See paper for more details
Talk Outline

- 3D Printing Workflow
- *Clarity* achieved by applying FP to fabrication
  - Denotational semantics for CAD and mesh ✔
  - Inductive compiler definition ✔
- Proof of correctness for compiler ✔
- *Usefulness*: the *first* decompiler from mesh to CAD
Talk Outline

• 3D Printing Workflow

• *Clarity* achieved by applying FP to fabrication
  • Denotational semantics for CAD and mesh
  • Inductive compiler definition
  • Proof of correctness for compiler

• *Usefulness*: the *first* decompiler from mesh to CAD
Synthesis: flip the arrows!

union
cube
trans (2, 0, 0) cube

union
cube
trans (2, 0, 0) cube

Mesh [...]
Synthesis: flip the arrows!

union
cube
trans (2, 0, 0) cube

union
[.] trans (2, 0, 0) cube
cube

union
[.] trans (2, 0, 0) cube

union
Mesh [m_{cube}]

union
Mesh [m_{trans (2, 0, 0) cube}]

Mesh […]
Synthesis: flip the arrows!

- Computational fabrication pipeline is a compiler.
- We have small step operational semantics.
- Get synthesis by “just flipping the arrows”!
Synthesis: flip the arrows!

- Loss of information
- Geometric challenges
- Non-deterministic
Synthesis: flip the arrows!

- Loss of information
- Geometric challenges
- Non-deterministic

Evaluation context
Synthesis: flip the arrows!

- Loss of information
- Geometric challenges
- Non-deterministic

Evaluation context

Geometric oracles
Synthesis: flip the arrows!

Evaluation context

- Loss of information
- Geometric challenges
- Non-deterministic
Eval context is geometric context!

synthesize
Eval context is geometric context!

synthesize

synthesize
Eval context is geometric context!

In the context of union, the top can be replaced by a sphere, even though it does not match a sphere primitive.
Synthesis: flip the arrows!

- Loss of information
- Geometric challenges
- Non-deterministic

Geometric oracles
Oracles: Primitive Detection

\[ p \in \Omega_{\text{prim}}(m) \]

\[ \text{Mesh } m \rightarrow_{\Omega} p \]
Oracles : Primitive Detection

Rotation:
- Find an object coordinate system
- Align it with the world’s coordinates

Scale:
- Scale the object down to unit dimensions

Translate:
- Move the object back to origin

Apply affine transformations to primitive in reverse order

cube
  |> rotateX (30)
  |> rotateY (45)
  |> rotateZ (60)
  |> scale (2, 3, 4)
  |> translate (1, 2, 3)
Oracles: Subtractive

\[ (m_1, m_2) \in \Omega_{sub}(m) \]

Mesh \( m \rightarrow_{\Omega} \text{Binop Diff (Mesh } m_1) \text{ (Mesh } m_2) \)
Oracles: Subtractive

Many possible bounding primitives
Oracles: Subtractive

\( \Omega_{sub} \)

\( m \) → Bounding primitive → Remaining mesh

\( \rho_{best} = \arg \min_{\rho} \text{volume of difference}(\rho, m) \)
Oracles: Additive

\[ (m_1, m_2) \in \Omega_{\text{add}}(m) \]

Mesh \( m \rightarrow_{\Omega} \text{Binop Union} \ (\text{Mesh} \ m_1) \ (\text{Mesh} \ m_2) \]
Infinite ways to split a mesh!
Oracles: Additive

Infinite ways to split a mesh!

Disjoint splits
Convex splits
Infinite ways to split a mesh!

Disjoint splits

Convex splits

Oracles: Additive

$\Omega_{add}$

Disjoint split
Oracles: Additive

Infinite ways to split a mesh!

Disjoint splits

Convex splits

Disjoint split

Convex split: Split mesh along plane where convexity changes
Synthesis: flip the arrows!

- Loss of information
- Geometric challenges
- Non-deterministic
Synthesis: try all three steps

Mesh [...]
Synthesis: try all three steps
Synthesis: try all three steps

Mesh [...] \[ \Omega_{add} \] + \[ \Omega_{sub} \] —
Synthesis: try all three steps

Mesh [...] → \( \Omega_{\text{add}} \) → \( \Omega_{\text{sub}} \) → No match
Synthesis: try all three steps

\( \Omega_{\text{prim}} = \text{cube} \)

\( \Omega_{\text{add}} \)

\( \Omega_{\text{sub}} \)

\( \Omega_{\text{prim}} = \text{trans (1, 0, 0) cube} \)

Mesh [...]
Synthesis: try all three steps

No match

$\Omega_{prim} = scale \ (3, \ 1, \ 1) \ cube$  
$\Omega_{prim} = trans \ (1, \ 0, \ 0) \ cube$  

union  
cube  
trans \ (2, \ 0, \ 0) \ cube

difference  
scale \ (3, \ 1, \ 1) \ cube  
trans \ (1, \ 0, \ 0) \ cube
Synthesis: try all three steps

\[ \Omega_{prim} = \text{cube} \]

\[ \Omega_{prim} = \text{trans (1, 0, 0) cube} \]

Mesh [...]  

\[ \Omega_{add} \]

\[ \Omega_{sub} \]

\[ \Omega_{prim} \]

No match

\[ \Omega_{prim} = \text{scale (3, 1, 1) cube} \]

\[ \Omega_{prim} = \text{trans (1, 0, 0) cube} \]

Pick best based on ranking function
Implementations in OCaml

**25,000 LOC:** Supports 1D, 2D, 3D CAD & Mesh

- Floating points, MPFR, Exact Arithmetic
- Points, lines, planes, intersections, area
- Affine transf, binary ops, hull, mesh split
- CAD primitives, compiler implementation
- Geom oracles, search algorithm, ranking

Number Systems
Geometry
Meshes
CADs
Synthesis

https://github.com/uwplse/reincarnate-aec
Implementations in OCaml

- Floating points, MPFR, Exact Arithmetic
- Points, lines, planes, intersections, area
- Affine transform, binary ops, hull, mesh split
- CAD primitives, compiler implementation
- Geom oracles, search algorithm, ranking

25,000 LOC: Supports 1D, 2D, 3D CAD & Mesh

Number Systems
Geometry
Meshes
CADs
Synthesis

Pavel Panchekha
NPFL, Thursday

https://github.com/uwplse/reincarnate-aec
ICFP (our design)

Hexagonal Candle Holder
Ultimate 22 Hex-Wrench Holder
40mm Cube Test Object
25mm Calibration with Empty Top
Measuring Cylinder
Basic Box with Lid
Modular Memory Holder (USB)
Circle Cell Block Generator
Jewelry Box with Inlay
SD Card Rack
Gordian knot 3D Puzzle
Conclusions

Functional PL for fabrication (3D printing)

**Clarity:** semantics and compiler correctness

**Usefulness:** the first decompiler from mesh to CAD
Functional PL for fabrication (3D printing)

**Clarity:** semantics and compiler correctness

**Usefulness:** the *first* decompiler from mesh to CAD

Check out our web IDE! — Adam Anderson

http://reincarnate.uwplse.org/

https://github.com/uwplse/reincarnate-aec