ReIncarnate Artifact, ICFP 2018

Goals of the artifact

In our paper, we made the following contributions (`Section 1`, last paragraph):

1. A purely functional programming language model for 3D CAD along with denotational semantics for both CAD and triangular mesh.

2. A meaning preserving compilation algorithm from 3D CAD to mesh along with a proof sketch for compiler correctness.

3. A synthesis algorithm that can reverse engineer 3D CAD programs from meshes.

In support of these contributions, this artifact will demonstrate:

- An early prototype of the compiler (`Section 4`) from the core 3D CAD language (`Figure 8` in the paper) to mesh. The goal is to show that our compiler is capable of generating valid triangular meshes as we described in the paper. To that end we provide 5 CAD programs which cover all the core CAD language features we described in `Figure 8` of the paper: 3D primitives, affine transformations, binary operations, and their combinations.

- An early prototype of the synthesis tool or reverse compiler (`Section 5`) from 3D mesh to CAD. The goal is to show that the synthesis tool is capable of synthesizing CAD programs from meshes for the case studies we described in `Section 6` of the paper. Later in this document, we provide instructions on how to view the 3D renderings of the CAD programs our tool synthesizes. You can compare them with the figures we show in the paper, namely in `Figures 3, 20, 21` and verify that the renderings look the same.

This document contains the following parts:

- System requirements

- Getting started.

- How to run the compiler and synthesis tool.

- Some notes and remarks

- How to set up ReIncarnate on a different machine (this is also how we set up the VM).

System requirements

- We provide the artifact as a virtual machine image. To open it you need virtual box version `5.2.12`, which can be downloaded here.

- In the machine where we tested the VM, we have 16 GB RAM and 500 GB hard disk.

Getting started

- Please download the `.ova` file from the link and open it with Virtual Box by going to `File -> import appliance` and giving the path to the `.ova` file and clicking on `continue`. In the next window that pops up, click on `Import`. It should take a few minutes to import.

- Next, please open the virtual machine image in virtual box by clicking on the green `Start` button.

- Login is automatic, but in case needed, the password is: `icfp2018`.

- The terminal should be open at startup. The project repository is already cloned. Navigate to the `reincarnate` directory. All the required packages are already installed.

- Type `cd src` from the `reincarnate` directory and then type `make`. This will build all the tools.

Running the tools

Compiler: cad3 -> mesh3

We provide 5 CAD programs and a script that compiles them using our CAD compiler to generate 3D meshes. These CAD programs are in the directory: `aec/cads-to-compile/cad3`. We compile the CAD programs to our mesh format (.mesh3 files) and also the industry standard format, STL (.stl files). The `.mesh3` files will be saved in the `aec/compiled-meshes/mesh3` directory. The `.stl` files will be saved in the `aec/compiled-meshes/stl` directory.

Note: Before running any scripts, feel free to check that the directories `aec/compiled-meshes/mesh3` and `aec/compiled-meshes/stl` are empty since that is where the meshes will be saved.

- To run the compiler, run the following from the `src` directory: `./scripts/compile.sh`. This should take 2-3 minutes to finish.

- In order for you to verify that the mesh our compiler generated corresponds to the same CAD program it started with, we recommend viewing the renderings before and after compilation.

    - To facilitate viewing the rendering before compilation, we provide the directory `aec/cads-to-compile/scad` that contains the CAD programs from our CAD language (`.cad3` files) pretty printed to OpenSCAD's language (`.scad` files). OpenSCAD is another programmatic CAD language which has a 3D renderer. We have already installed OpenSCAD in the VM. Simply click on the `scad` files in the directory `aec/cads-to-compile/scad` and click the `Render` button above the Editor (the icon looks like a small cube with an hourglass at the bottom corner).
    - To view the rendering after compilation, we again recommend using OpenSCAD: click on the OpenSCAD icon on the vertical panel on the left and then click on `New`. To render an `stl` file, type: `import("/home/reincarnate/reincarnate/src/aec/compiled-meshes/stl/example-name.stl");`

Then click on the `Render` button on top and compare with the rendering of the corresponding `.scad` file. They should look the same.

Synthesis: mesh3 -> cad3

In order to show the working synthesis tool, we provide the case studies we showed in the paper (`Section 6`).

Note: Before running any scripts, feel free to check that the directories `aec/synthed-cads/cad3` and `aec/synthed-cads/scad` are both empty.

- We recommend first running the script `./scripts/basic-synth.sh` to run the synthesis tool on the 5 meshes our compiler generated. This is just a sanity check. It should finish in ~ 7 minutes and a successful run indicates that it is possible to write CAD programs in our CAD language, compile them to mesh using our compiler, and then synthesize CAD programs back from the meshes.

- To run the case studies in the paper (`Section 6`), run `./scripts/paper-synth.sh`. Please let this script run for two hours. *Note:* for the hexholder, we run a smaller version of it from this script which has fewer holes than the one in `Figure 21` of the paper. This is because the one in the paper is very big and it takes about 9 hours to complete. We provide the mesh for the very big one in the directory: `aec/paper-synth/bighexholder.mesh3`. In case you are interested to let it run for 9 hours, the command you need to run is:

  ```
  ./Main.native --src aec/paper-synth/bighexholder.mesh3 --tgt aec/synthed-cads/cad3/bighexholder.cad3 --glue os-mesh --no-invariants --fuel x
  ```

  `fuel` is a parameter used by the synthesis algorithm shown in `Figure 18` in `Section 5.1` of the paper. It is used to ensure termination of the algorithm. You can set it to `--fuel 1000`. We explain the flags `--glue os-mesh` and `--no-invariants` in the next section.

  You can further experiment to generate the corresponding `scad` file. The command for that is:

  ```
  ./Main.native --src aec/synthed-cads/cad3/bighexholder.cad3 --tgt aec/synthed-cads/scad/bighexholder.scad
  ```

If you just want to look at the code our tool synthesized for this big example, we have included them in the directories: `aec/pre-run-big-hexholder/cad3/bighexholder.cad3` and `aec/pre-run-big-hexholder/scad/bighexholder.scad` (to open in OpenSCAD anc view).

- All the synthesized CAD programs will be in the directory `aec/synthed-cads`. Our script will generate both `.cad3` files and `.scad` files in dedicated sub directories within `aec/synthed-cads`. The `.cad3` files correspond to the CAD programs synthesized in our CAD language. The `.scad` files correspond to equivalent CAD programs in the OpenSCAD language. We do this so that you can use the OpenSCAD GUI to view the rendered CAD programs. Clicking on the files will open them in OpenSCAD from where you can click the `Render` button to view the rendering.

- There are some other small examples we provide for synthesis that you can try to run yourself if you are interested. The meshes for these are in the `aec/extra-synth` directory. One is a cylinder primitive, another is a cube with one hole, and the third one is a another small hexholder with 3 holes.

Notes and remarks

As we have explained in `Section 4.2.2` of the paper, the design of our tool is fully functorial. This has been extremely helpful in managing complexity as the codebase has grown beyond 20000 lines. Some advantages of this design decision is being able to parametrize our compiler and synthesis tool over different number systems, and swapping our compiler with other CAD compilers for synthesis (see `Section 4.2.2`). There are several implementations of number systems (see `NumSys.ml`, `MPFRNumSys.ml`, `ExactArith.ml`) which are instantiated in `Main.ml`. `Glue.ml` contains several configurations (called glues) for using our compiler or an external compiler (e.g. OpenSCAD) for synthesis. It is possible to choose the configuration from the command line. Type `./Main.native -h` to see all the options.

For the compiler experiments, we of course use the compiler that we have built. We also check all the invariants (see `Section 3.2.1` of the paper) to ensure that the meshes our compiler produces are valid.

Currently for the synthesis experiments, we use the OpenSCAD compiler (indicated by the flag `--glue os-mesh`), and also disable our invariant (indicated by the flag `--no-invariants`) checks in order to avoid rounding errors. As we explained in Section `8.1` of the paper, rounding errors creep in very frequently in CAD compilation and as part of our future work, we have already started to work on ways to fix it (e.g. exact arithmetic). Since these numerical issues are still work in progress and not a contribution of this paper, for the purpose of demonstrating our synthesis tool, we leverage the fully functorial design of our tools and plug in the OpenSCAD compiler.

Setup instructions (for setting up ReIncarnate in a different machine)

1. Install system dependencies. On macOS with [Homebrew]:

```
$ brew install coreutils autoconf gnu-time gawk parallel git git-lfs graphviz gnuplot
$ brew cask install openscad
$ brew install ocaml opam
```

The version of Ocaml we use is `4.05`.

On Linux (Ubuntu 18.04) without root, you need to install [OpenSCAD] and build gnuplot from source. Then using [Linuxbrew]:

```
$ brew install autoconf parallel git git-lfs graphviz
$ brew install ocaml opam
```

On Linux (Ubuntu 18.04) with root and `apt`:

```
$ apt-get install autoconf parallel git git-lfs graphviz gnuplot openscad
$ apt-get install ocaml opam
```

2. Install [opam] packages:

```
$ opam init
$ eval `opam config env`
$ opam install mlgmpidl zarith hashcons menhir js_of_ocaml
```

3. Run GNU parallel once interactively and acknowledge that you will cite the authors:

```
$ parallel --citation

```

4. Build the compiler and synthesis tool:

```
$ cd reincarnate/src
$ make
```