Installation

Supported Platforms:

- Ubuntu (gcc 5+)
- Mac OS X (gcc 5+, clang 4.0+)
- Windows (Microsoft Visual Studio 2017)

Make sure you have python 3.5+.

1.1 Ubuntu, Arch Linux, and Mac OS X

```
wget https://raw.githubusercontent.com/yuanming-hu/taichi/master/install.py
python3 install.py
```

Note, if python complains that a package is missing, simply rerun install.py and the package should be loaded.

1.2 Windows

Download and execute this script with python3.

Additional environment variables: (assuming taichi is installed inDIR/taichi) Set TAICHI_REPO_DIR as DIR/taichi (e.g. E:/repos/taichi). Add %TAICHI_REPO_DIR%/python to PYTHONPATH, DIR/taichi/bin (e.g. E:/repos/taichi/bin) to PATH. Restart cmd or PowerShell, and you should be able to run command ti.
1.3 Build with Double Precision (64 bit) Float Point

```
export TC_USE_DOUBLE=1
ti build
```

1.4 Examples

Please see examples.

Run examples using `python3 [X.py]`. For example,

```
python3 projects/examples/simulation/3d/mqpcg_smoke_3d.py
python3 projects/examples/rendering/paper_cut.py
```

Please learn the python interface by examples for now. Detailed documentation coming soon.
2.1 Code Format

- We rely on clang-format-4.0 for code format.
- Make sure to format your code before you commit, since we have not set up a githook for it.

2.2 Scoping

2.3 Naming

- Variable names should consist of lowercase words connected by underscores, e.g. density_field.
- Class and struct names should consist of words with first letters capitalized, e.g. MultigridPreconditioner.
- Template classes should start with T, like TVector, TMatrix, TArray;
  - Reserve the name without T for specialized classes, e.g. using Vector=TVector<real, dim>.
- Macros should be capital start with TC, such as TC_INFO, TC_IMPLEMENTATION.
  - We do not encourage the use of macro, though there are cases where macros are inevitable.
- Filenames should consist of lowercase words connected by underscores, e.g. parallel_reduction.cpp.

2.4 File Organization

- Put in the projects folder
2.5 Object-Oriented Programming

2.6 Common Patterns

2.7 Casting

- We allow the use of old-style C casting e.g. `auto t = (int)x;`
  - Reason: `static_cast<type>(variable)` is too verbose.
- Think twice when you use `reinterpret_cast, const_cast`.
- Discussions on this in Google C++ Style Guide.

2.8 Do’s

- Be considerate to your users (this includes yourself in the near future).
- Use `auto` for local variables when appropriate.
- Mark `override` and `const` when necessary.

2.9 Dont’s

- C language legacies:
  - `printf` (use `fmtlib` instead).
  - `new & free`. Use smart pointers (`std::unique_ptr, std::shared_ptr` instead for ownership management).
  - Unnecessary dependencies.
- Prefix member functions with `m_` or `_`. Modern IDE can highlight members variables for you.
- Virtual function call in constructors/destructors.
- C++ exceptions
- `NULL`, use `nullptr` instead.
- `using namespace std;` in headers global scope.
- `typedef`. Use `using` instead.

2.10 Documentation

- To build the documentation: `ti doc cd docs && sphinx-build -b html . build`.
Acknowledgments

Taichi is based on many other open-source projects, which are shipped with taichi and users do not have to install manually: pybind11, Intel Embree, Intel TBB, fmt, Catch2, spdlog, stb_image, stb_image_write, stb_truetype, tiny-objloader, JIXIE::ImplicitQRSVD, dcraw, ffmpeg, miniz.

Current developers include:

- Yuanming Hu (project creator & main developer)
- Yu Fang (developer)