Writing A Custom Constraint Solver for Blender with Cython

Maryanne Wachter

May 1, 2022

Agenda

💪 ***Motivation / Background***

Backend

- ❓ Why Cython?
- 🏗️ Toy problem
- 📦 Python package structure with Cython
- ⚙️ Compiling Cython packages

Agenda

Performance

- 🏃 Using Cython to release the GIL
- 📊 Profiling Cython

Frontend

- ❓ Why Blender?
- 👨‍💻 Blender Development
- 📓 Blender Add-ons

Motivation / Background

About Me

Maryanne Wachter

👷‍♀️ structural engineer

🐍 since 2015

📘 mclare.blog

About Me

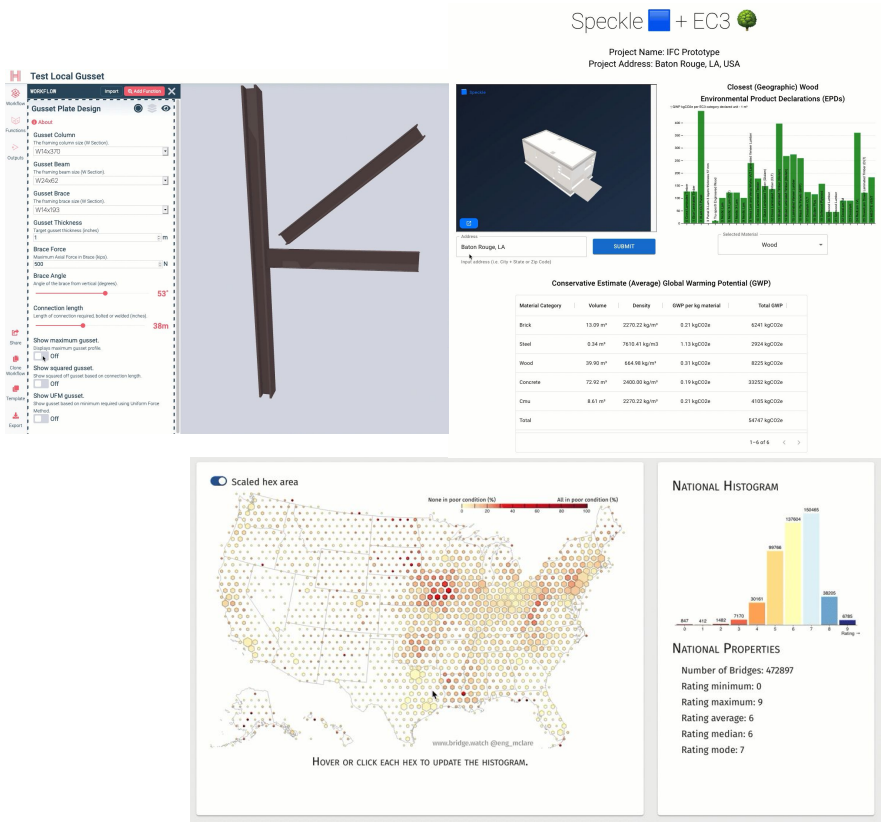
Maryanne Wachter

👷‍♀️ ~~structural~~ software engineer

🐍 since 2015

📘 mclare.blog

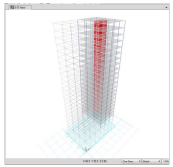

@eng_mclare



Software in the Built Environment

What I Used It For

- Building Analysis



- Building Design

- Building Data Exchange

What I Don't Like

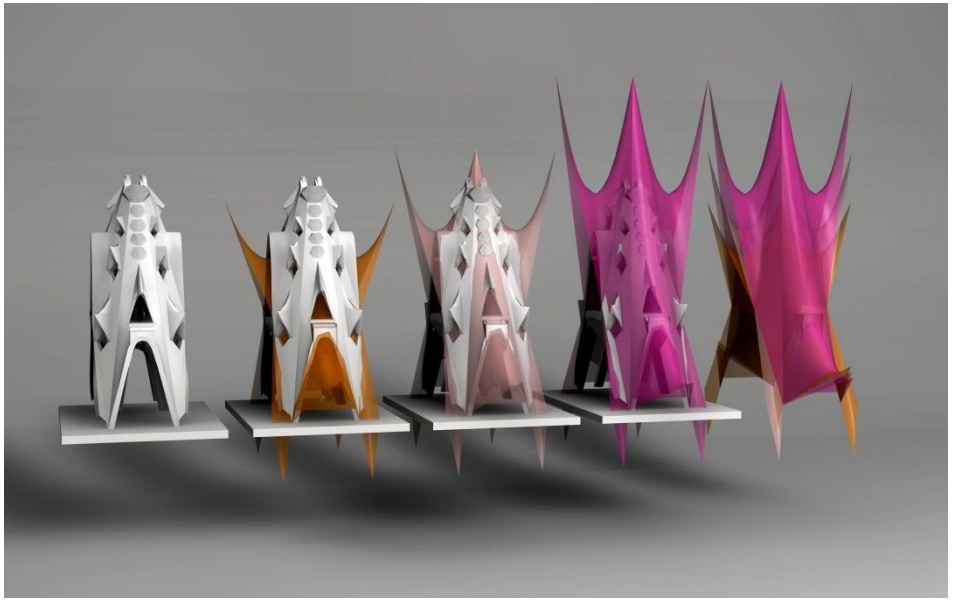
- Terrible UX/UI
- $$$
- Slow
- "Black box"

Physical and Digital Formfinding

Sagrada Familia - Antonio Gaudi
1882 - present

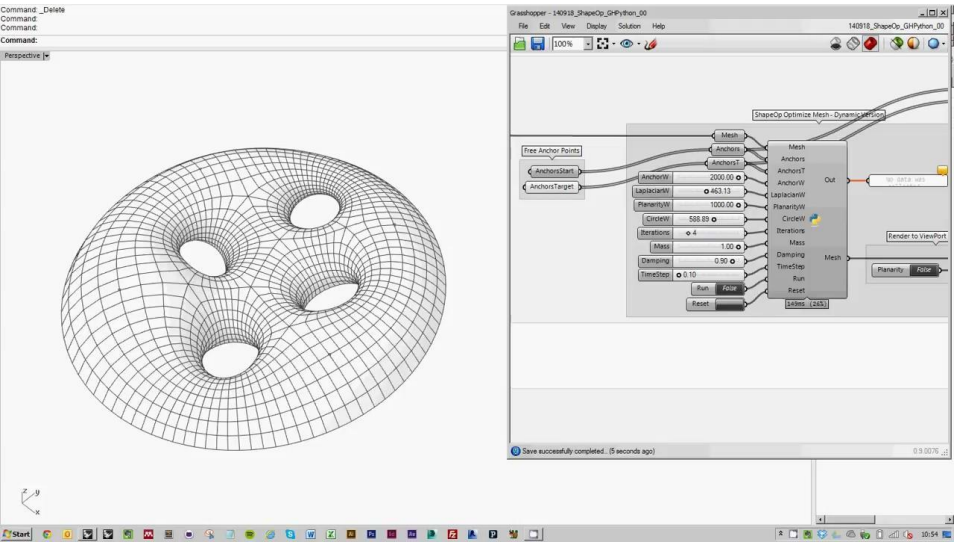
Physical and Digital Formfinding

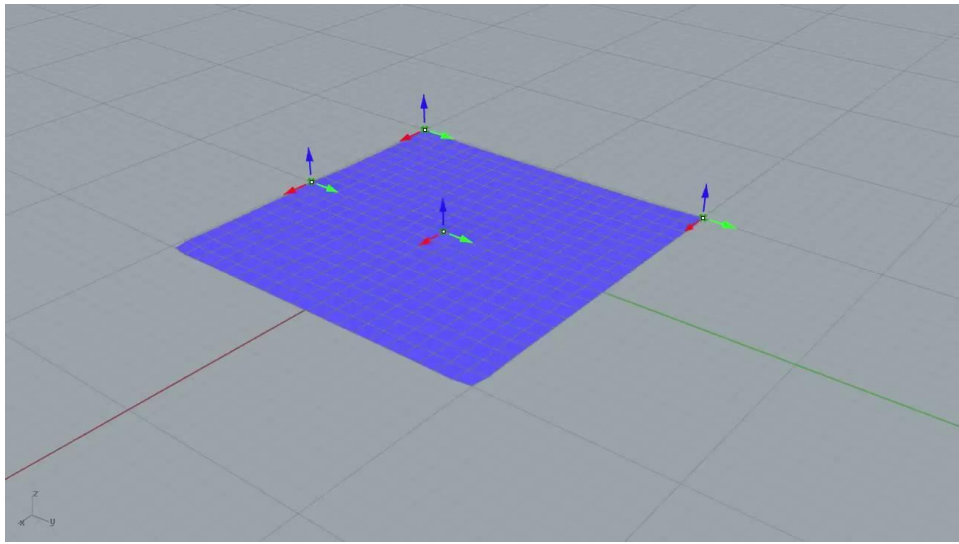
Sagrada Familia - Antonio Gaudi
1882 - present



Mark Burry - The Nave Roof
https://mcburry.net/the-west-transept/

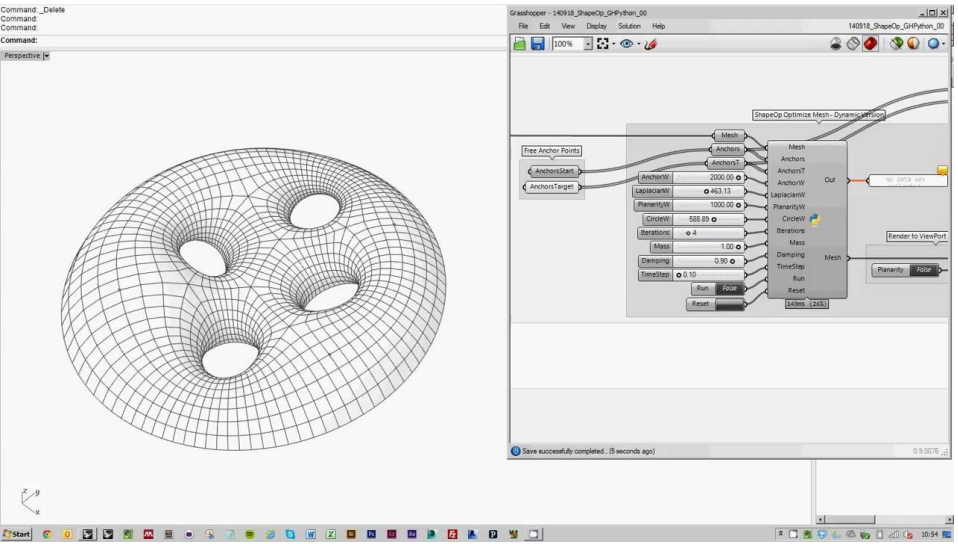
Constraint Solvers for 3d Geometry





LGG EPFL
https://www.shapeop.org

Constraint Solvers for 3d Geometry



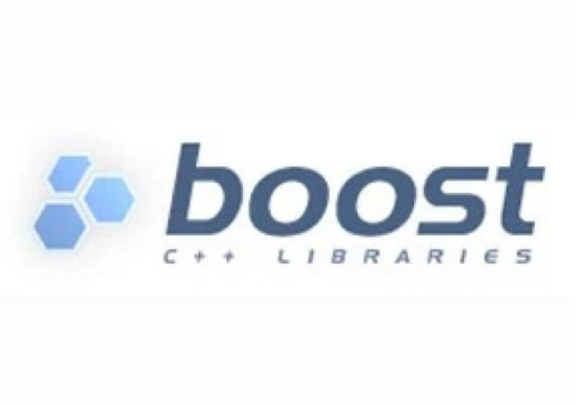
- Terrible UX/UI
- $$$
- Slow
- "Black box"

Backend

Building Pymaxion

https://xkcd.com/303/

Why Cython?



Why Cython?

- Significant speedups are possible (can release the GIL!)
- Can create general framework to build on
- Able to preserve a Python package structure
- Able to utilize existing C++ libraries
- Looks *mostly* like Python (with some additions, i.e. static typing)

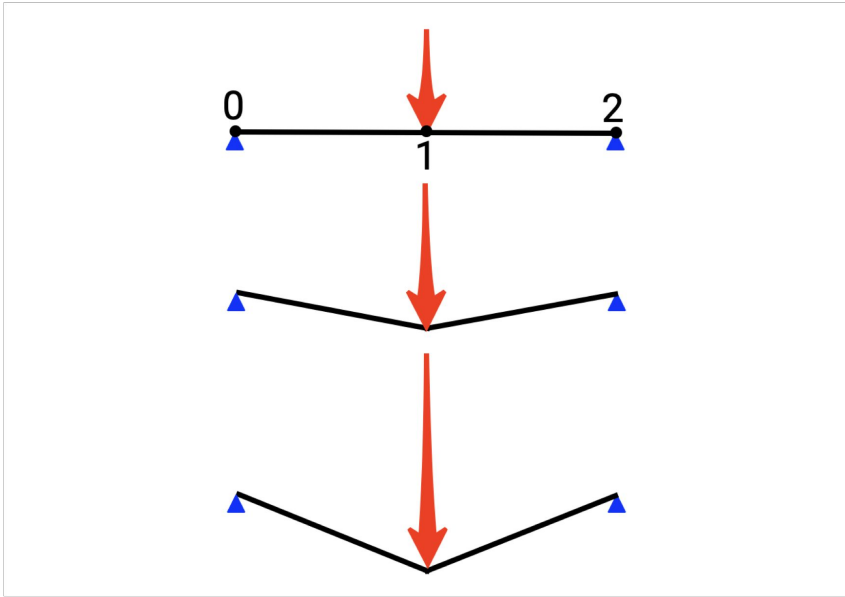
What is Cython?

- Optimizing Static Compiler
- Provides support for debugging in Python, Cython, and C/C++ code
- Superset of Python
- Enables easily writing C-extensions for Python

Toy Constraint Problem

Constraints

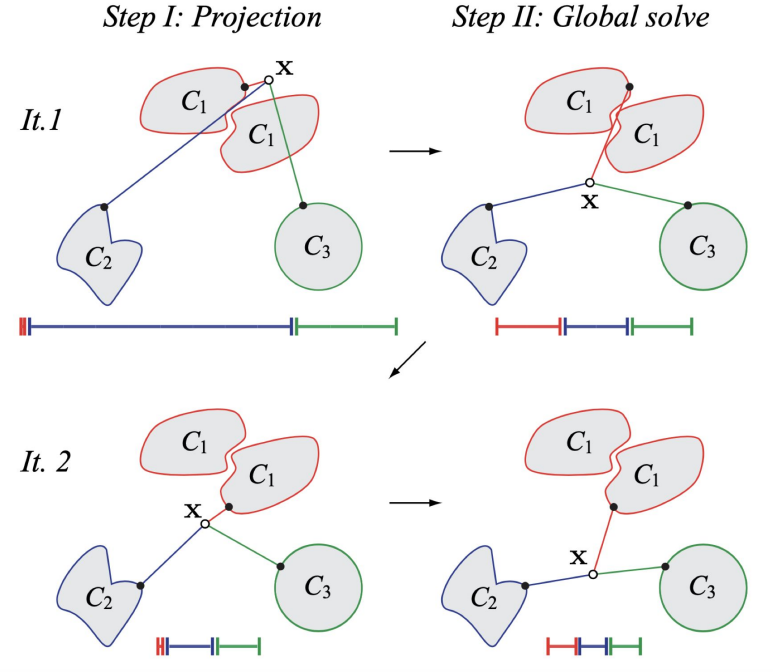
- Anchors (supports) - [0, 2]
- Forces (loads) - [1]
- Cables (lengthening) - [[0, 1], [1, 2]]



Toy Constraint Problem

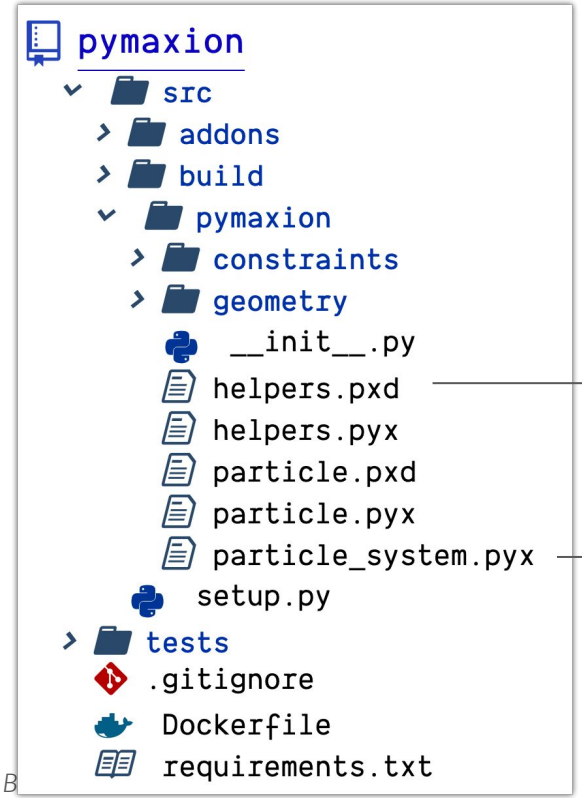
Constraints

- Anchors (supports) - [0, 2]
- Forces (loads) - [1]
- Cables (lengthening) - [[0, 1], [1, 2]]



Computational Methods for Fabrication-aware Modeling, Rationalization and Assembly of Architectural Structures
Deuss, 2015

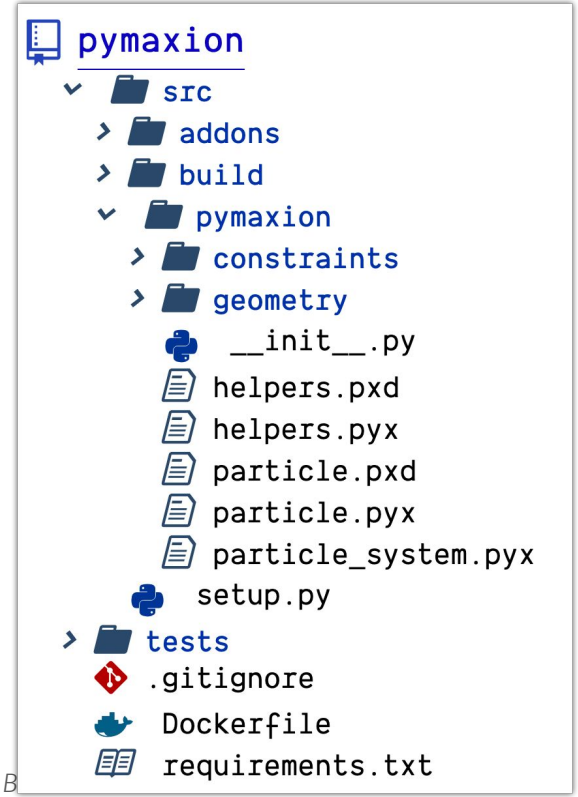
Package Structure and Modules with Cython

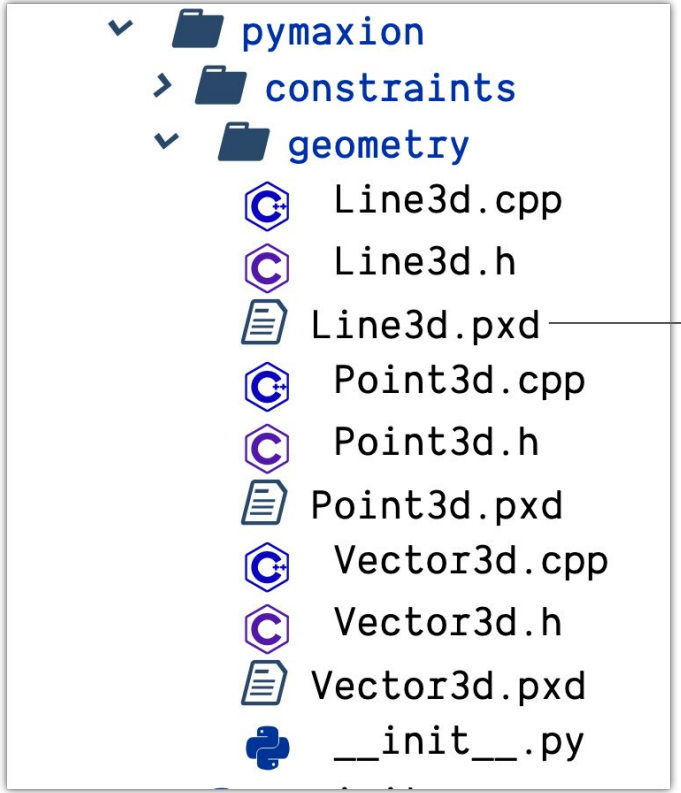


"Header" file (type declaration)

Class + methods

Package Structure and Modules with Cython

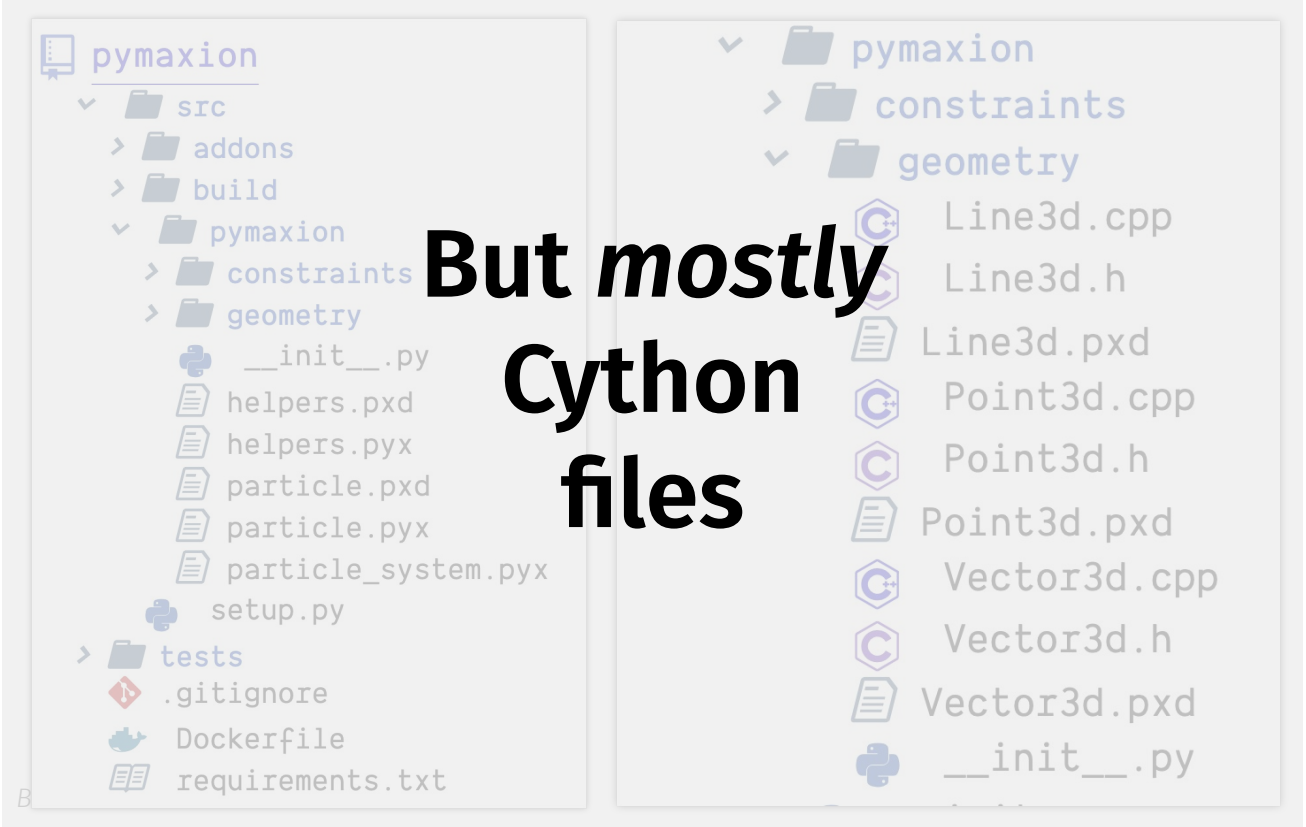




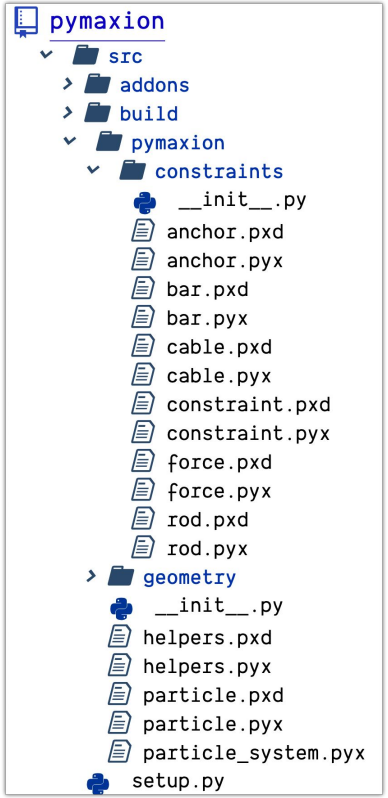
"Header" file

Ok a "little" C++ here...

Package Structure and Modules with Cython



But *mostly* Cython files



Package Structure and Modules with Cython

- Only one level of subclassing allowed!
- .pxd files are needed for importing into other modules at the same folder level (Cython "headers")

```pxd
from libcpp.vector cimport vector
from pymaxion.geometry.Vector3d cimport Vector3d
from pymaxion.geometry.Point3d cimport Point3d

cdef class Constraint:
    cdef public int constraint_n_particles
    cdef vector[int] *particle_index
    cdef vector[Vector3d] *move_vectors
    cdef vector[double] *weighting
    cdef vector[double] *strength
    cdef public list particles
    cdef void calculate(Constraint, double[:,:] arr) nogil
    cdef void sum_moves(Constraint, double[:,:] p_sum, double[:] w_sum) nogil
```

src/pymaxion/constraints/constraint.pxd

```pxd
from libcpp.vector cimport vector
from pymaxion.constraints.constraint cimport Constraint
from pymaxion.geometry.Point3d cimport Point3d

cdef class Anchor(Constraint):
    cdef vector[Point3d] *anchor_pt

    @staticmethod
    cdef Anchor from_Point3d(Point3d pt, double strength)
```

src/pymaxion/constraints/anchor.pxd

Package Structure and Modules with Cython

```
1  from libcpp.vector cimport vector
2  from pymaxion.geometry.Vector3d cimport Vector3d
3  from pymaxion.geometry.Point3d cimport Point3d
4
5  cdef class Constraint:
6      cdef public int constraint_n_particles
7      cdef vector[int] *particle_index
8      cdef vector[Vector3d] *move_vectors
9      cdef vector[double] *weighting
10     cdef vector[double] *strength
11     cdef public list particles
12     cdef void calculate(Constraint, double[:,:] arr) nogil
13     cdef void sum_moves(Constraint, double[:,:] p_sum, double[:] w_sum) nogil
```

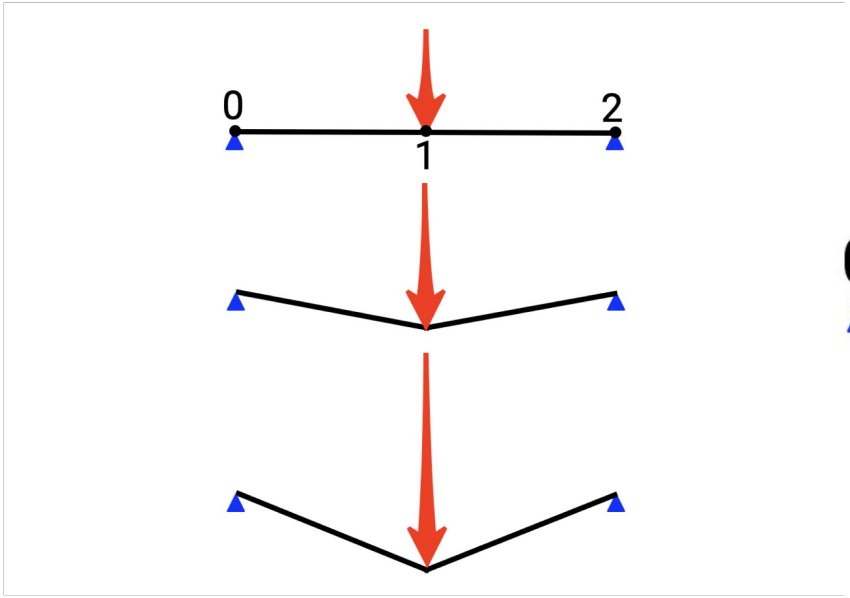
src/pymaxion/constraints/constraint.pxd

Package Structure and Modules with Cython

```
46      cdef void calculate(Constraint self, double[:,:] arr) nogil:
47          pass
48
49      cdef void sum_moves(Constraint self, double[:,:] p_sum, double[:] w_sum) nogil:
50          cdef int i
51          for i in range(self.constraint_n_particles):
52              p_index = self.particle_index[0].at(i)
53              curr_move = self.move_vectors[0].at(i)
54              curr_strength = self.strength[0].at(i)
55              w_sum[p_index] += curr_strength
56              p_sum[p_index, 0] += curr_move.x * curr_strength
57              p_sum[p_index, 1] += curr_move.y * curr_strength
58              p_sum[p_index, 2] += curr_move.z * curr_strength
```

src/pymaxion/constraints/constraint.pyx

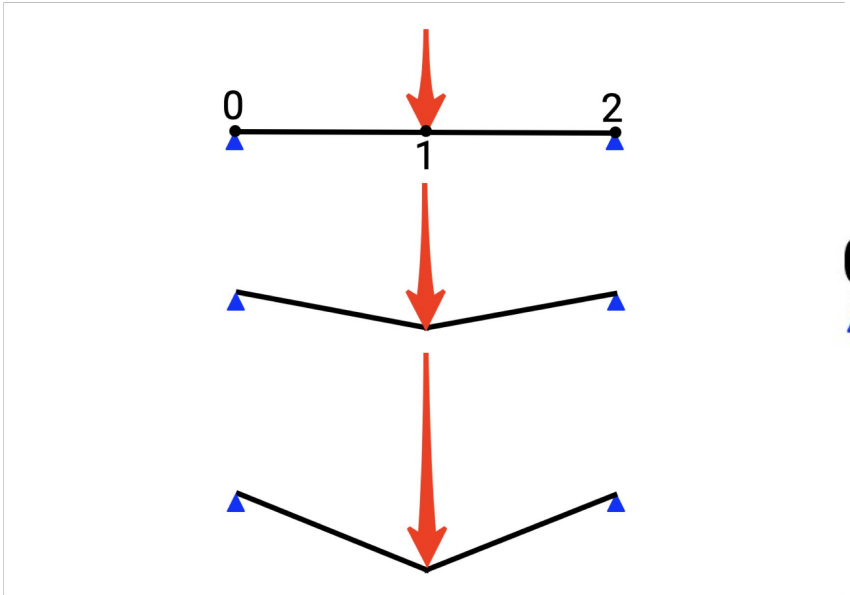
Package Structure and Modules with Cython

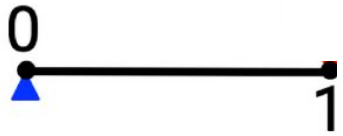



```
53      cdef void calculate(Anchor self, double[:,:] arr) nogil:
54          cdef Vector3d cur_pos
55          anchor_pt = self.anchor_pt[0].at(0)
56          p_index = self.particle_index[0].at(0)
57          cur_pos = Vector3d(anchor_pt.x, anchor_pt.y, anchor_pt.z)
58          new_pos = cur_pos.vector_subtract(Vector3d(arr[p_index, 0],
59                                                    arr[p_index, 1],
60                                                    arr[p_index, 2]))
61          self.move_vectors[0].at(0).set_value(new_pos.x,
62                                               new_pos.y,
63                                               new_pos.z)
```

src/pymaxion/constraints/anchor.pyx

Package Structure and Modules with Cython






```
48      cdef void calculate(Cable self, double[:, :] arr) nogil:
49          cdef Vector3d start_pt
50          cdef Vector3d end_pt
51          cdef double current_length
52          cdef double length_diff
53          cdef Vector3d force_start
54          cdef Vector3d force_end
55          cdef Vector3d force_dir
56
57          p1 = self.particle_index[0].at(0)
58          p2 = self.particle_index[0].at(1)
59          start_pt = Vector3d(arr[p1, 0], arr[p1, 1], arr[p1, 2])
60          end_pt = Vector3d(arr[p2, 0], arr[p2, 1], arr[p2, 2])
61
62          force_dir = end_pt.vector_subtract(start_pt)
63          current_length = force_dir.length()
64          force_dir.vector_unitize()
65          length_diff = current_length - self.rest_length
66          if length_diff > 0.0:
67              force_start = force_dir.vector_scale(0.5 * length_diff)
68              force_end = force_dir.vector_scale(-0.5 * length_diff)
69          else:
70              force_start = Vector3d(0, 0, 0)
71              force_end = Vector3d(0, 0, 0)
72
73          self.move_vectors[0].at(0).set_value(force_start)
74          self.move_vectors[0].at(1).set_value(force_end)
```

src/pymaxion/constraints/cable.pyx

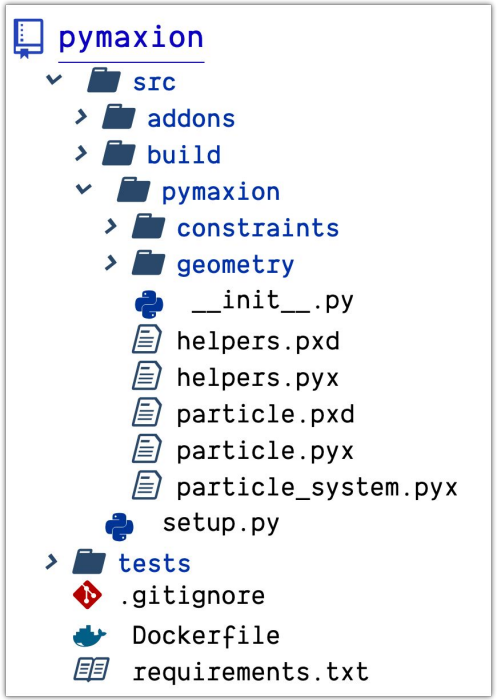
Compiling Cython

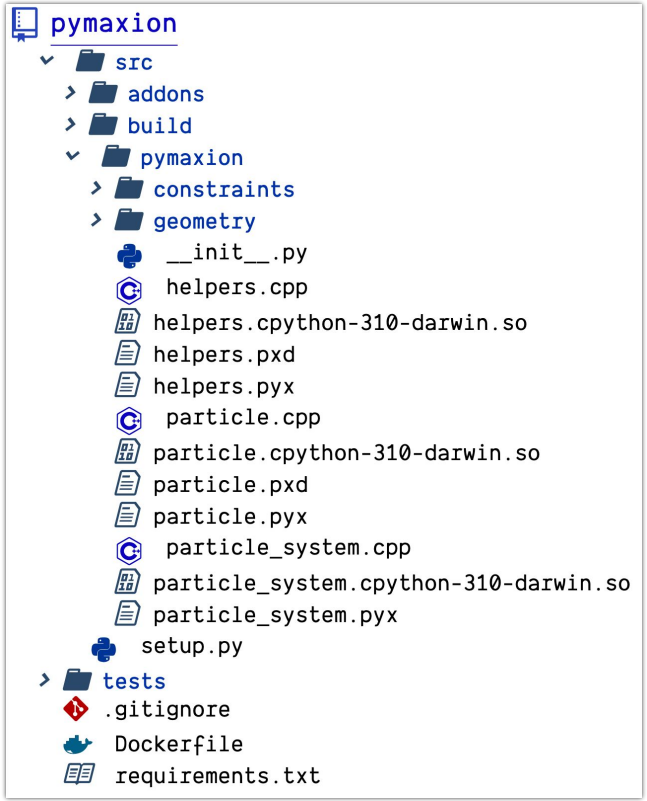
```
11 def scandir(dir, files=[]):
12     for file in os.listdir(dir):
13         path = os.path.join(dir, file)
14         if os.path.isfile(path) and path.endswith("pyx"):
15             files.append(path.replace(os.path.sep, ".")[:-4])
16         elif os.path.isdir(path):
17             scandir(path, files)
18     return files
19
20 def makeExtension(extName):
21     extPath = extName.replace(".", os.path.sep) + ".pyx"
22     print(extName)
23     return Extension(
24         name=extName,
25         sources=[extPath],
26         include_dirs=[numpy_include, ".", geo_include],
27         extra_compile_args=["-fopenmp"],
28         extra_link_args=["-fopenmp"],
29         language="c++",
30     )
31
32 # Find all includes
33 numpy_include = numpy.get_include()
34 # Add includes for C++ only libraries (no .pyx wrapped)
35 geo_include = os.path.abspath("./pymaxion/geometry")
36 extNames = scandir("pymaxion")
37
38 # build up set of Extension objects
39 extensions = [makeExtension(name) for name in extNames]
40
41 # Create a dictionary of arguments for setup
42 setup_args = {
43     "name": "pymaxion",
44     "packages": ["pymaxion", "pymaxion.goals", "pymaxion.geometry"],
45     "py_modules": [],
46     "ext_modules": cythonize(extensions,
47                              emit_linenums=True,
48                              compiler_directives={'language_level': 3}),
49     "requires": [],
50     "cmdclass": {"build_ext": build_ext},
51 }
52
53 setup(**setup_args)
```

```
20 def makeExtension(extName):
21     extPath = extName.replace(".", os.path.sep) + ".pyx"
22     print(extName)
23     return Extension(
24         name=extName,
25         sources=[extPath],
26         include_dirs=[numpy_include, ".", geo_include],
27         extra_compile_args=["-fopenmp"],
28         extra_link_args=["-fopenmp"],
29         language="c++",
30     )
```

```
$ python setup.py build_ext -inplace
```

Compiling Cython





Performance

Before you release the GIL...

- Use Cython for static typing!
- Use special comment blocks
- Write Python wrappers for Cython cdef functions and C attributes

Before you release the GIL...

- Use Cython for static typing!
- Use special comment blocks
- Write Python wrappers for Cython cdef functions and C attributes

```
1  from libcpp.vector cimport vector
2  from pymaxion.geometry.Vector3d cimport Vector3d
3  from pymaxion.geometry.Point3d cimport Point3d
4
5  cdef class Constraint:
6      cdef public int constraint_n_particles
7      cdef vector[int] *particle_index
8      cdef vector[Vector3d] *move_vectors
9      cdef vector[double] *weighting
10     cdef vector[double] *strength
11     cdef public list particles
12     cdef void calculate(Constraint, double[:,:] arr) nogil
13     cdef void sum_moves(Constraint, double[:,:] p_sum, double[:] w_sum) nogil
```

src/pymaxion/constraints/constraint.pxd

Before you release the GIL...

- Use Cython for static typing!
- Use special comment blocks
- Write Python wrappers for Cython cdef functions and C attributes

```
# distutils: language = c++
# cython: cdivision = True
# cython: boundscheck = False
# cython: wraparound = False
# cython: linetrace = True
# cython: language_level = 3


from libc.stdlib cimport free
from pymaxion.geometry.Vector3d cimport Vector3d
from pymaxion.particle cimport Particle
import numpy as np
```

src/pymaxion/particle_system.pyx

Before you release the GIL…

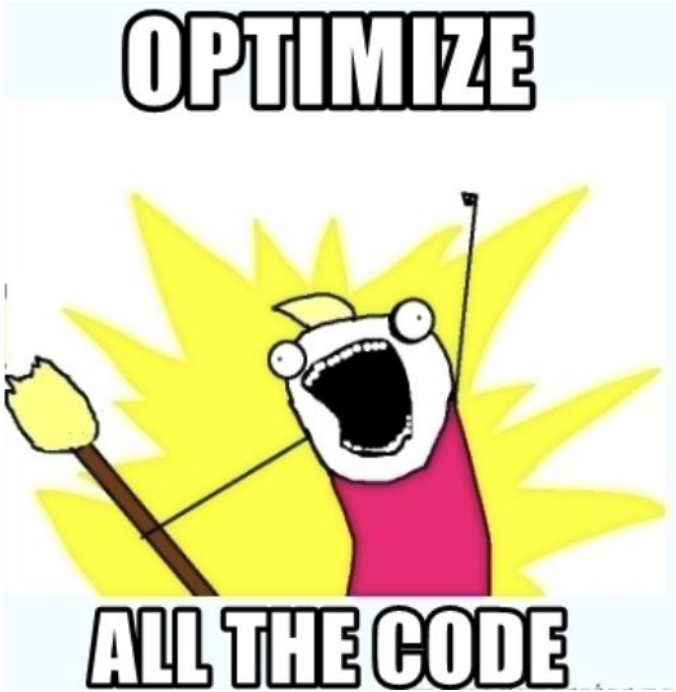
- Use Cython for static typing!
- Use special comment blocks
- Write Python wrappers for Cython cdef functions and C attributes

```
74      @property
75      def weighting(Constraint self):
76          return [weighting for weighting in self.weighting[:1]][0]
77
78      @property
79      def strength(Constraint self):
80          return [strength for strength in self.strength[:1]][0]
81
82      def py_calculate(Constraint self, double[:, :] arr):
83          return self.calculate(arr)
84
85      def py_sum_moves(Constraint self, double[:,:] p_sum, double[:] w_sum):
86          return self.sum_moves(p_sum, w_sum)
```

src/pymaxion/constraints/constraint.pyx

Releasing the GIL!

- Utilize cdef methods only (signature must match in .pxd and .pyx file)
- Use `nogil` in function signature (only indicates it can be used without GIL)



Releasing the GIL!

- Check python interference with `cython --cplus $file.pyx -a`

```
+009: import numpy as np
 010: from cpython cimport PyObject
 011: from cython.parallel import parallel, prange
 012: from libc.stdlib cimport malloc, free
 013: from numpy cimport ndarray
 014: from libc.stdio cimport printf
 015:
 016: from pymaxion.goals.goal cimport Goal
 017: from pymaxion.goals.anchor cimport Anchor
 018: from pymaxion.particle cimport Particle
 019: from pymaxion.geometry.Point3d  cimport Point3d
 020: from pymaxion.geometry.Vector3d cimport Vector3d
 021:
+022: cdef class ParticleSystem(object):
 023:     cdef int n_goals
 024:     cdef int n_particles
 025:     cdef PyObject **goals
 026:     cdef PyObject **particles
+027:     cdef public list ref_goals
+028:     cdef public list ref_particles
+029:     cdef public ndarray particle_positions
+030:     cdef public ndarray particle_sum_moves
+031:     cdef public ndarray particle_sum_weights
+032:     cdef public ndarray particle_velocities
 033:
+034:     def __cinit__(ParticleSystem self):
+035:         self.goals = NULL
+036:         self.particles = NULL
+037:         self.n_goals = 0
+038:         self.n_particles = 0
 039:
+040:     def __init__(ParticleSystem self):
+041:         self.ref_goals = []
+042:         self.ref_particles = []
```

Releasing the GIL!

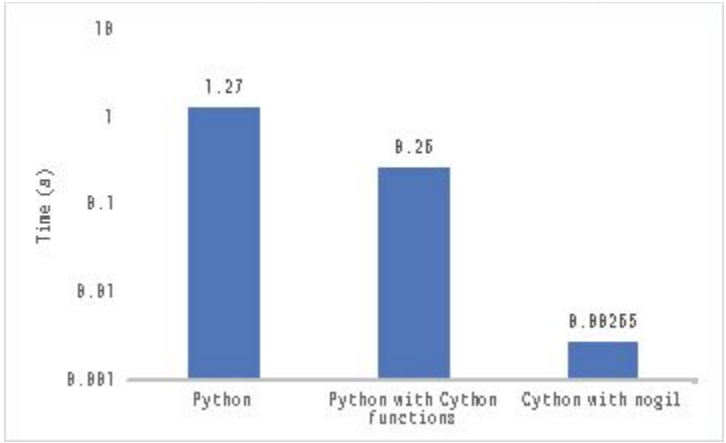
- Use memoryviews `double[:, :] arr` for fast Numpy array access

```
211         # memory view must be created before nogil
212         cdef double [:, :] p_pos = self.particle_positions
213         cdef double [:, :] p_vel = self.particle_velocities
214         cdef double [:, :] p_moves = self.particle_sum_moves
215         cdef double [:]  p_weights = self.particle_sum_weights
216         # set up C++ only objects for nogil
217         self.constraints = <PyObject **>malloc(self.n_constraints*cython.sizeof(
218                                                cython.pointer(PyObject)))
```

src/pymaxion/particle_system.pyx

Releasing the GIL!


```
211         # memory view must be created before nogil
212         cdef double [:, :] p_pos = self.particle_positions
213         cdef double [:, :] p_vel = self.particle_velocities
214         cdef double [:, :] p_moves = self.particle_sum_moves
215         cdef double [:]  p_weights = self.particle_sum_weights
216         # set up C++ only objects for nogil
217         self.constraints = <PyObject **>malloc(self.n_constraints*cython.sizeof(
218                                                cython.pointer(PyObject)))
219         for i in range(self.n_constraints):
220             self.constraints[i] = <PyObject*>self.ref_constraints[i]
221         with nogil:
222             while flag == False:
223                 for j in range(self.n_particles):
```



Evaluating Cython performance - py-spy

Prep work:

```
"ext_modules": cythonize(extensions, emit_linums=True)
```

pymaxion/src/setup.py

```
#line 212 "pymaxion/particle_system.pyx"
```

pymaxion/src/*.cpp (Cython generated C++ files)

```
# cython: linetrace = True
```

pymaxion/src/*.pyx

https://github.com/benfred/py-spy

Evaluating Cython performance - py-spy

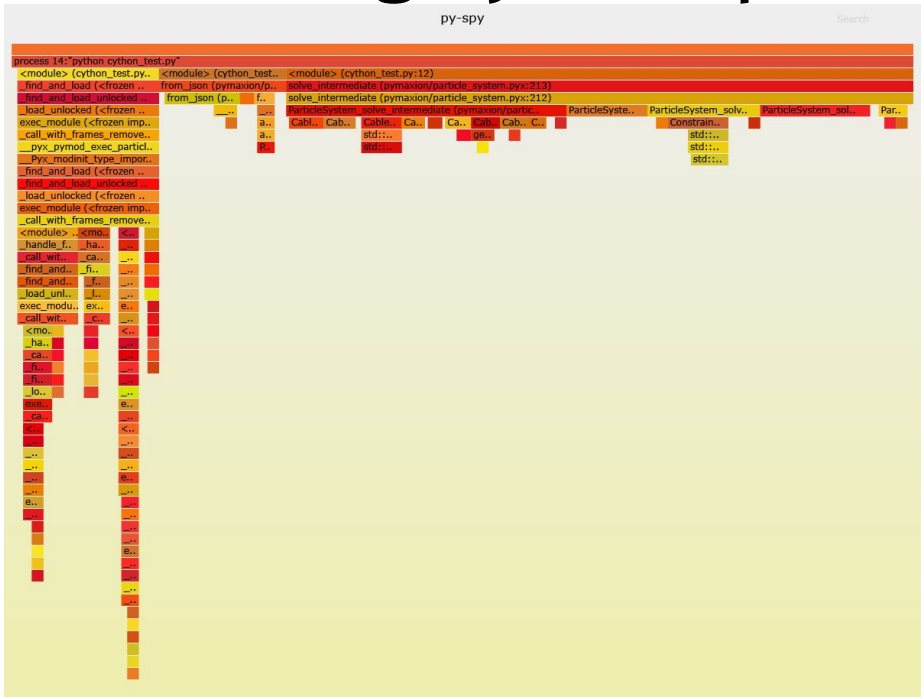

```
$ docker run -td -cap-add SYS_PTRACE $IMAGE_NAME
```

```
$ sudo docker exec -t $CONTAINER_NAME bash -c "PYTHONPATH=/code/src &&
py-spy record --subprocesses --native -o profile.svg -- python
cython_test.py
```

```
$ docker cp $CONTAINER_NAME:/code/tests/open_mp_tests/profile.svg
~/open_mp_tests/
```

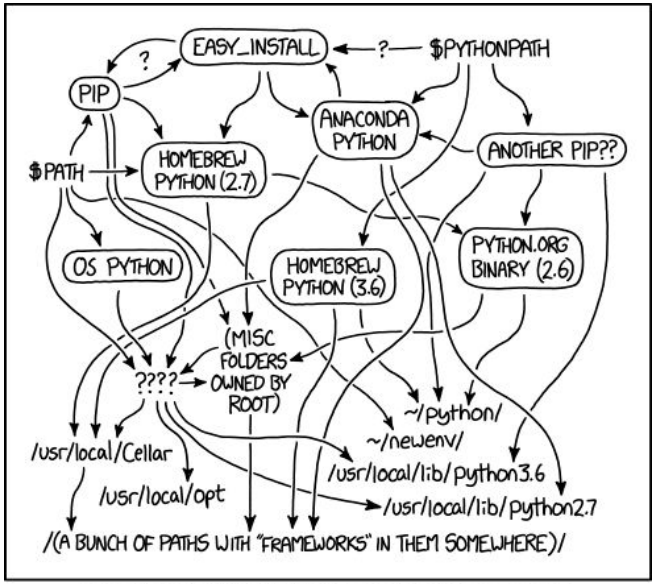
https://github.com/benfred/py-spy

Evaluating Cython performance - py-spy



Frontend

BlenderPymaxion

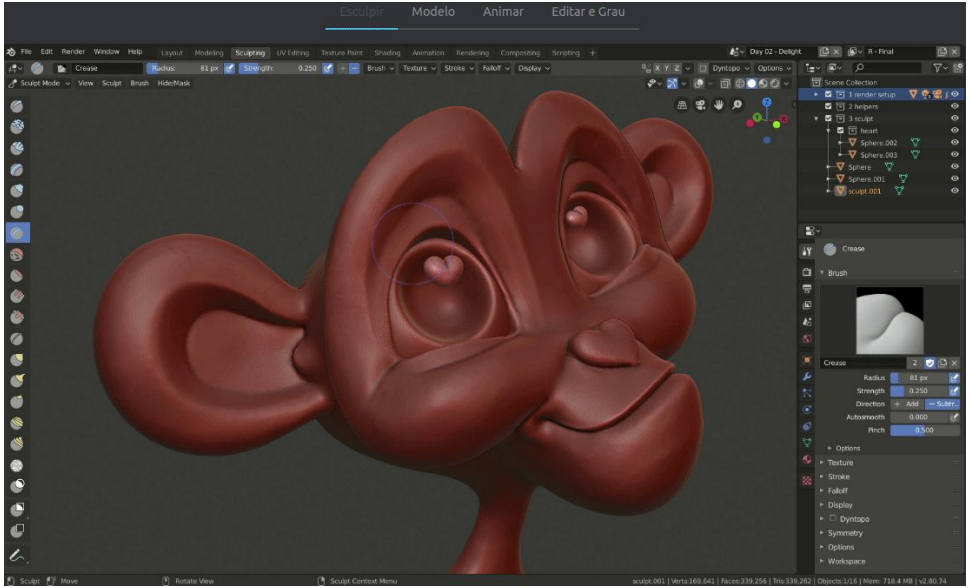


MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

https://xkcd.com/1987/

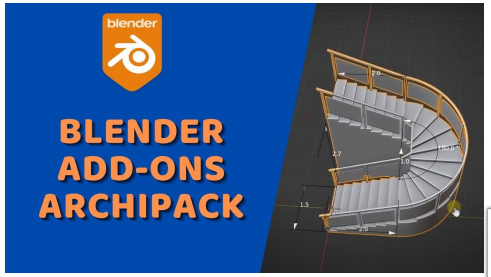
What is Blender?

- 3d Computer graphics software
- Free and Open Source (GNU GPL licensed)
- Possibilities include 3D modelling, fluid and smoke simulation, soft body simulation



Blender Add-ons

- Community made plugins
- Some free, some paid
- Extend capabilities of Blender



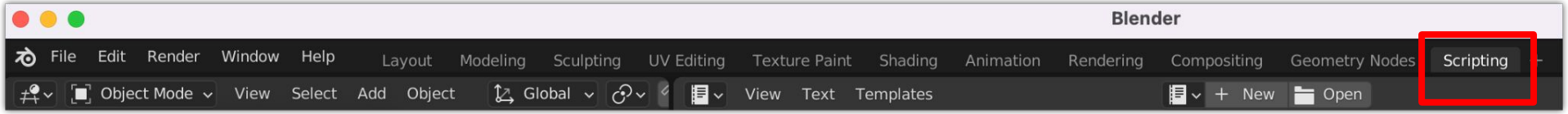
Sverchok

Sverchok CI passing | Build documentation pages passing
chat 66 online | stackoverflow invalid | Proposal issues 37 open
bug issues 36 open

BLENDERBIM ADD-ON

Adding your Cython package to Blender

- Build with Cython using the same Blender python version
  - (use whatever flavor virtualenv rather than the Blender python)
- Check your blender version from scripting tab interactive console `print(sys.version_info)`



Adding your Cython package to Blender

- Add Cython package to site-packages using symlinks

```
ln -s
/Users/mclare/workspaces/current/pymaxion/src/pymaxion
/Users/mclare/Blender.app/…/site-packages/pymaxion
```

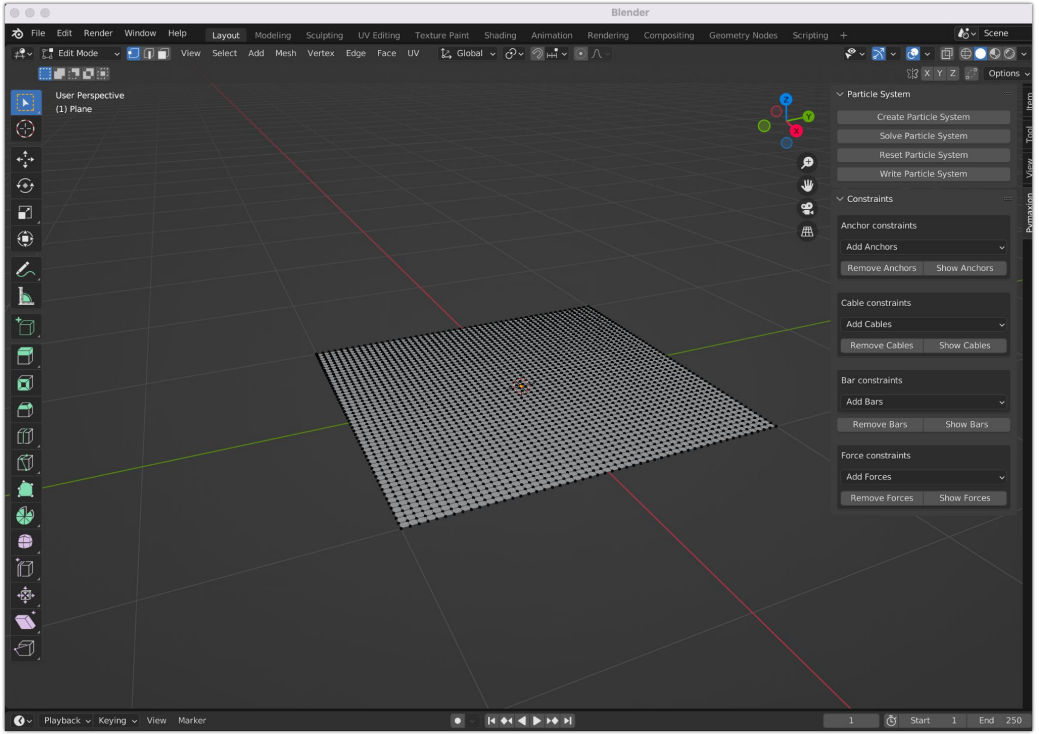
Adding your Add-on to Blender

- Lots of disagreement about best way to develop with Blender
  - script hot reloading
  - close and relaunch Blender
- *At minimum* add Blender at the command line (easier to see and debug)
- Add Blender UI package to add-ons directory using symlinks

```
ln -s /Users/mclare/workspaces/current/blenderPymaxion
/Users/mclare/Blender.app/Contents/Resources/3.2/scripts/addons
```

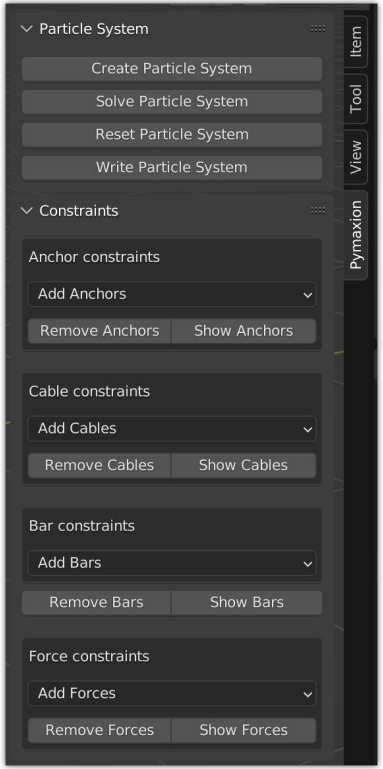
Blender Add-on Interface Creation

- Typical files:
  - __init__.py
  - operator.py
  - ui.py
  - properties.py



Blender Add-on Interface Creation

- Typical files:
  - __init__.py
  - operator.py
  - ui.py
  - properties.py



Blender Operators

- where you define all the actions taken in the UI
- where you import your external Cython package!
- specific Blender class structure in terms of attributes and methods

```
1  import bpy
2  from bpy.props import EnumProperty
3  from bpy.types import Operator
4  from profilehooks import profile
5  import json
6  from math import pi
7
8  # Pymaxion imports
9  from pymaxion.constraints.anchor import Anchor
10 from pymaxion.particle_system import ParticleSystem
11 from pymaxion.particle import Particle
12 from pymaxion.constraints.cable import Cable
13 from pymaxion.constraints.force import Force
14 from pymaxion.constraints.bar import Bar
```

blender_pymaxion/operator.py

Blender Operators

- where you define all the actions taken in the UI
- where you import your external Cython package!
- specific Blender class structure in terms of attributes and methods

```python
class solve_particle_system(Operator):
    bl_idname = "pymaxion_blender.solve_particle_system"
    bl_label = "Solve Particle System"

    def __init__(self):
        print("Start")

    def __del__(self):
        print("End")

    def execute(self, context):
        if bpy.data.objects["Pymaxion Particle System"]:
            obj = bpy.data.objects["Pymaxion Particle System"]
            bpy.context.view_layer.objects.active = obj
            new_obj = obj.copy()
            new_obj.data = obj.data.copy()
            obj.name = "Pymaxion Initial Particle System"
            new_obj.name = "Pymaxion Particle System"
            bpy.context.collection.objects.link(new_obj)
            obj.hide_set(True)
            print("Found particle system!")
            data = self.profile_run(new_obj)

        return {"FINISHED"}
```

blender_pymaxion/operator.py

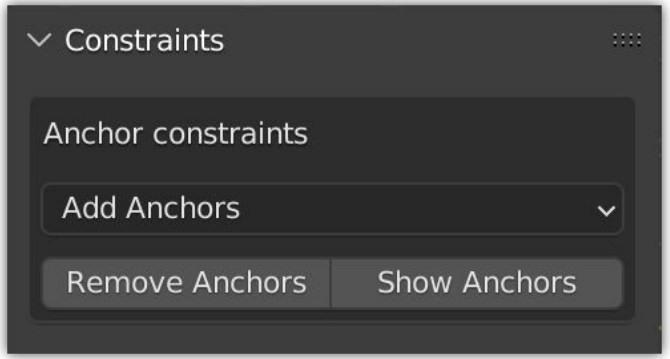
Blender UI

- Blender specific class prefix naming
- Blender specific class methods and attributes

```python
class PYMAXION_PT_constraints(Panel):
    bl_idname = "PYMAXION_PT_constraints"
    bl_label = "Constraints"
    bl_space_type = "VIEW_3D"
    bl_region_type = "UI"
    bl_category = "Pymaxion"

    def draw(self, context):
        layout = self.layout
        scene = context.scene

        box = layout.box()
        box.label(text="Anchor constraints")
        box.popover("PYMAXION_PT_Anchor")
        row = box.row(align=True)
        row.operator(
            "pymaxion_blender.anchor_constraint", text="Remove Anchors"
        ).action = "REMOVE"
        row.operator(
            "pymaxion_blender.anchor_constraint", text="Show Anchors"
        ).action = "SHOW"
```



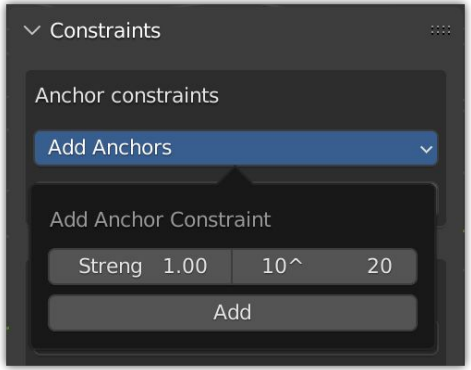
Blender UI

- Blender specific class prefix naming
- Blender specific class methods and attributes

```python
class PYMAXION_PT_Anchor(Panel):
    bl_label = "Add Anchors"
    bl_idname = "PYMAXION_PT_Anchor"
    bl_options = {"INSTANCED"}
    bl_space_type = "VIEW_3D"
    bl_region_type = "WINDOW"

    def draw(self, context):
        layout = self.layout
        scene = context.scene
        propertyGroup = getattr(scene, "Anchor")

        layout.label(text="Add Anchor Constraint")
        row = layout.row(align=True)
        row.prop(propertyGroup, "base", text="Strength Value")
        row.prop(propertyGroup, "power", text="10^")
        row = layout.row(align=True)
        row.operator("pymaxion_blender.anchor_constraint", text="Add").action = "ADD"
```



Blender Properties

- Typical files:
  - __init__.py
  - operator.py
  - ui.py
  - **properties.py**

```python
import bpy


class MyPropertyGroup(bpy.types.PropertyGroup):
    custom_1: bpy.props.FloatProperty(name="My Float")
    custom_2: bpy.props.IntProperty(name="My Int")


bpy.utils.register_class(MyPropertyGroup)

bpy.types.Object.my_prop_grp = bpy.props.PointerProperty(type=MyPropertyGroup)
```

https://docs.blender.org/api/current/bpy.types.PropertyGroup.html

Blender Properties

- Typical files:
  - __init__.py
  - operator.py
  - ui.py
  - ~~properties.py~~

```
"Anchor": [
  {
    "name": "base",
    "type": "float",
    "attr": "base",
    "min": 1,
    "max": 10,
    "default": 1,
    "precision": 2
  },
  {
    "name": "power",
    "type": "int",
    "attr": "power",
    "min": -30,
    "max": 30,
    "default": 20
  }
],
```

blender_pymaxion/assets/property_groups.json

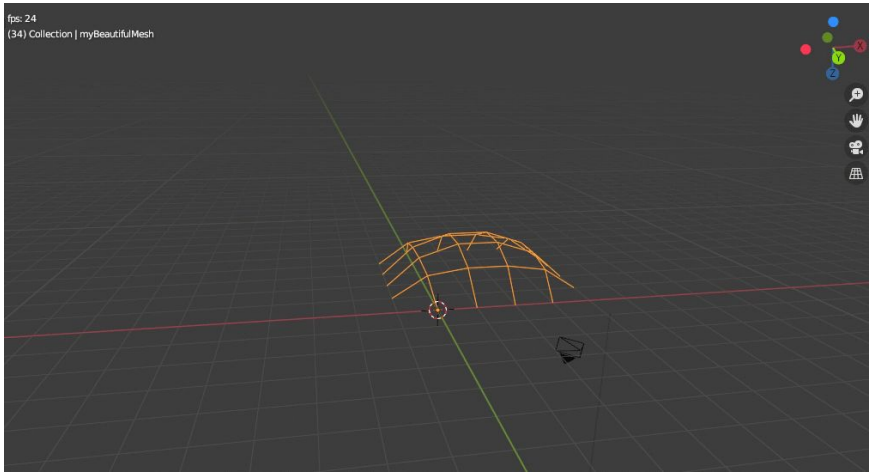
Blender Properties

- Typical files:
  - **__init__.py**
  - operator.py
  - ui.py
  - ~~properties.py~~

```
46      def register():
47          for cls in classes:
48              register_class(cls)
49
50          # Load custom property groups from json
51          dir_path = os.path.dirname(os.path.realpath(__file__))
52          with open(dir_path + '/assets/property_groups.json', 'r') as f:
53              properties = json.load(f)
54
55          for groupName, attributeDefinitions in properties.items():
56              attributes = {}
57              for attributeDefinition in attributeDefinitions:
58                  attType = attributeDefinition['type']
59                  attName = attributeDefinition['name']
60                  # Note: type of attribute must be removed for class
61                  if attType == 'float':
62                      attributeDefinition.pop('type')
63                      attributes[attName] = FloatProperty(**attributeDefinition)
64                  elif attType == 'int':
65                      attributeDefinition.pop('type')
66                      attributes[attName] = IntProperty(**attributeDefinition)
67                  else:
68                      raise TypeError('Unsupported type (%s) for %s on %s!' % (attType, attName, groupName)
    )
69              propertyGroupClass = type(groupName, (PropertyGroup,), {'__annotations__': attributes})
70              bpy.utils.register_class(propertyGroupClass)
71              setattr(bpy.types.Scene, groupName, PointerProperty(type=propertyGroupClass))
72              bpy.propertyGroups[groupName] = propertyGroupClass
```

blender_pymaxion/__init__.py

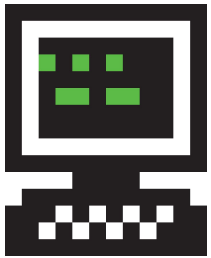
Pymaxion - Blender Demo!



https://github.com/m-clare/pymaxion
@eng_mclare

Thank you!

nyc.pyladies.com



recurse.com