IMODAL: creating learnable user-defined deformation models

Leander Lacroix – leander.lacroix@pm.me
"Trajectoires développementales & psychiatrie" – U1299 – Inserm

Benjamin Charlier, Barbara Gris, Alain Trouvé

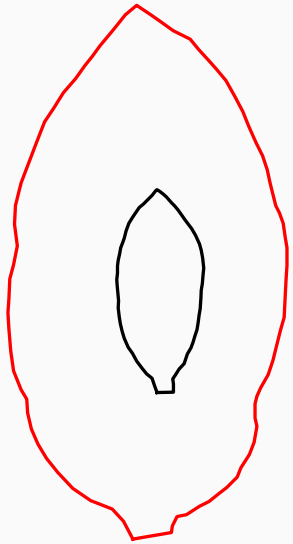
SMAI 2021

Leaf registration

$\rightarrow$ Goal: study of objects evolution by non-rigid registration

Our tools here:

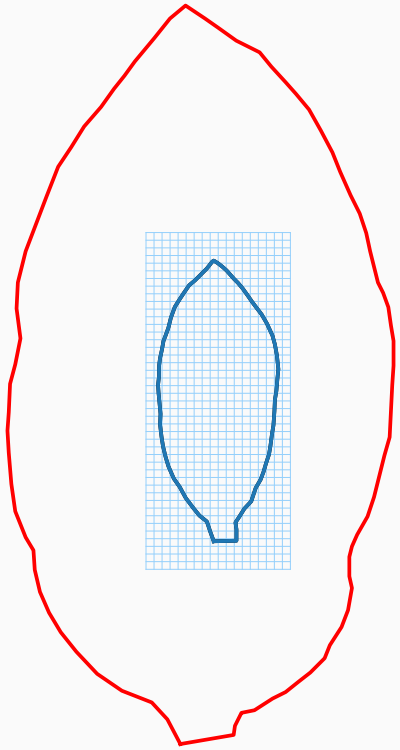
- LDDMM[1]
- Varifold attachment[2]



[1] Beg et al., "Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms".

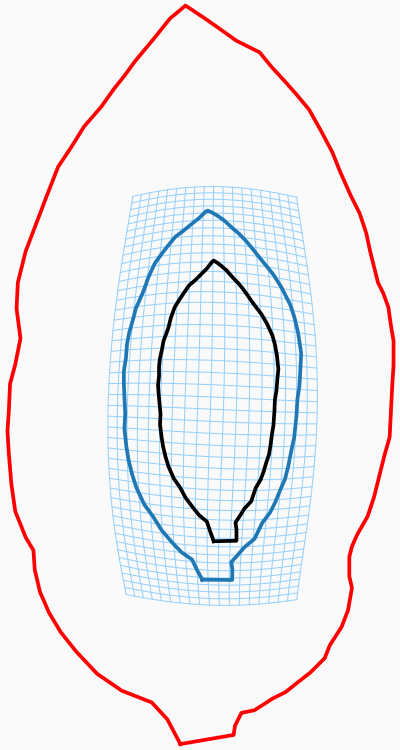
[2] Kaltenmark, Charlier, and Charon, "A general framework for curve and surface comparison and registration with oriented varifolds".

Leaf registration



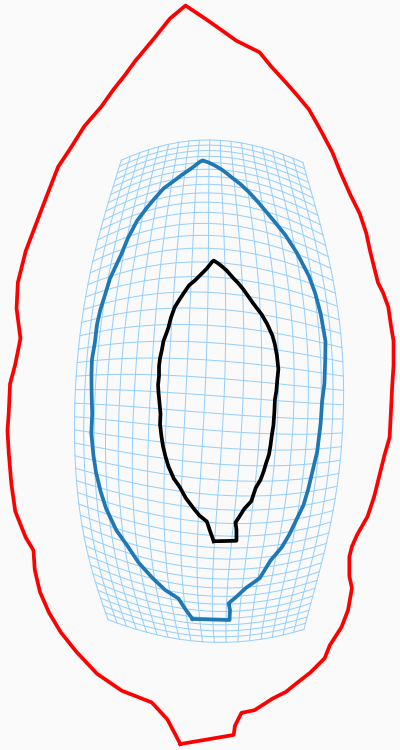
$t = 0$

Leaf registration



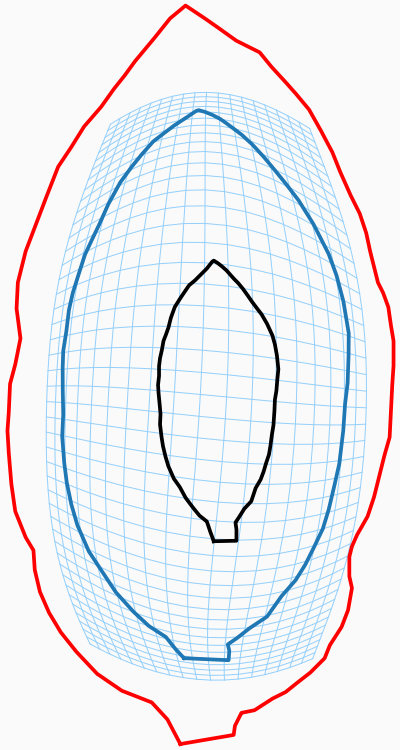
$t = 0.2$

Leaf registration



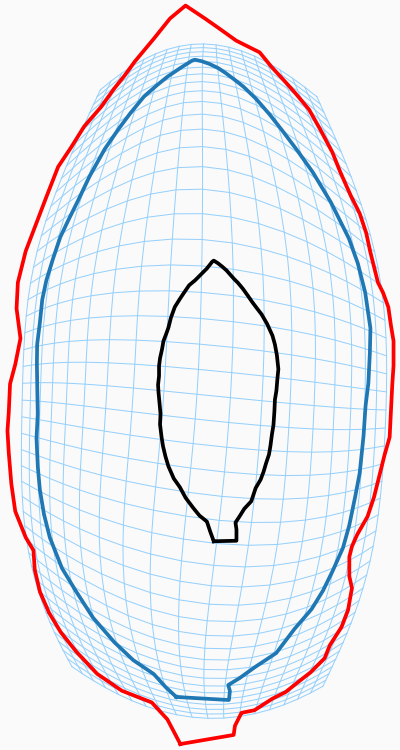
$t = 0.4$

Leaf registration



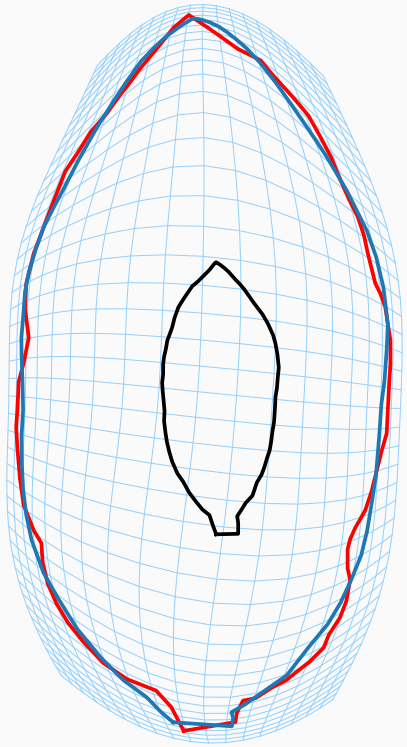
$t = 0.6$

Leaf registration



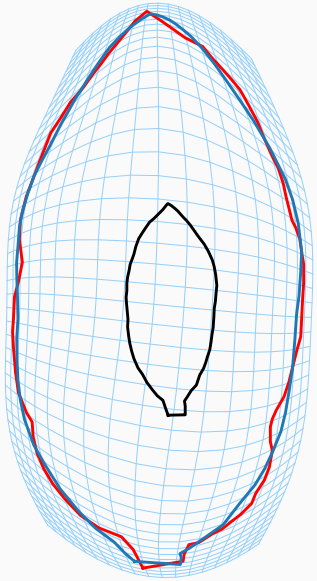
$t = 0.8$

Leaf registration

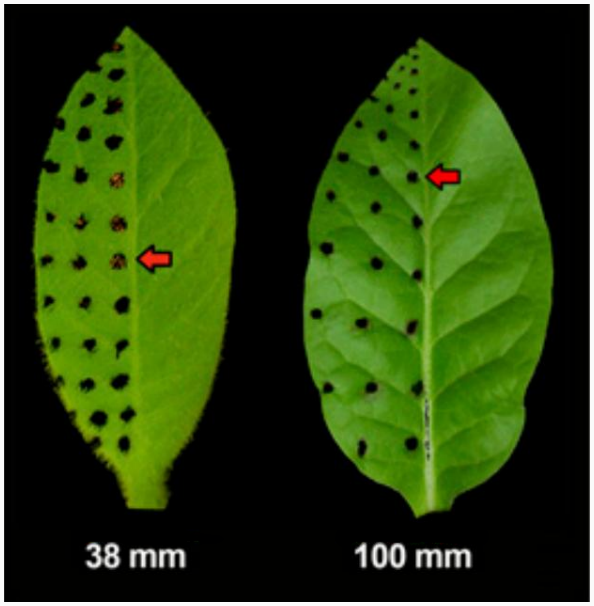


$t = 1$

Leaf registration: comparison with data



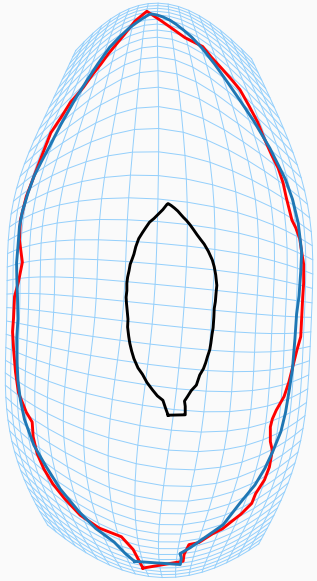
$t = 1$



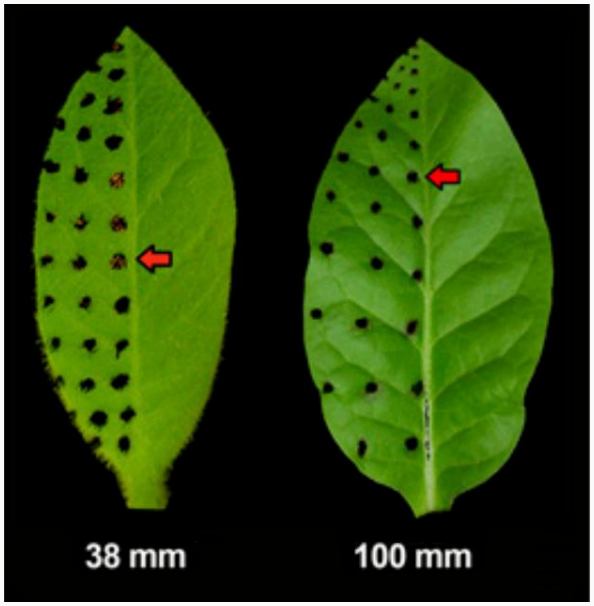
Basipetal growth illustrated by experiment[3]

[3] Gupta and Nath, "Divergence in patterns of leaf growth polarity is associated with the expression divergence of miR396".

Leaf registration: comparison with data



$t = 1$

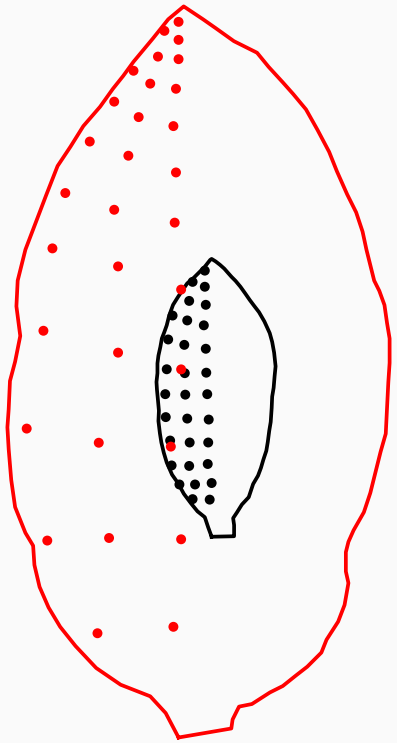


Basipetal growth illustrated by experiment[3]

$\rightarrow$ Let's add the dots in the data

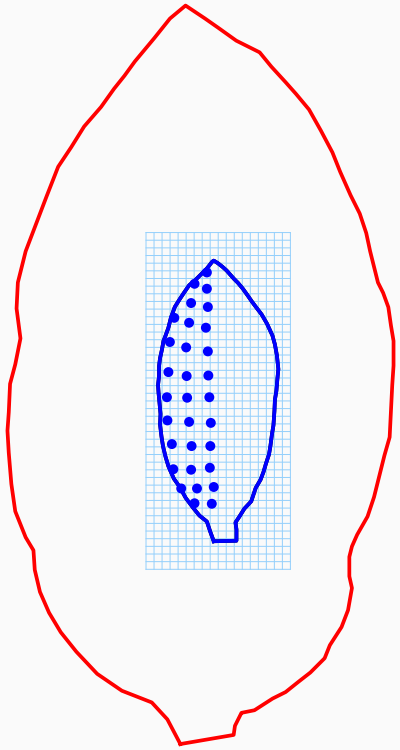
[3] Gupta and Nath, "Divergence in patterns of leaf growth polarity is associated with the expression divergence of miR396".

Leaf registration with dots



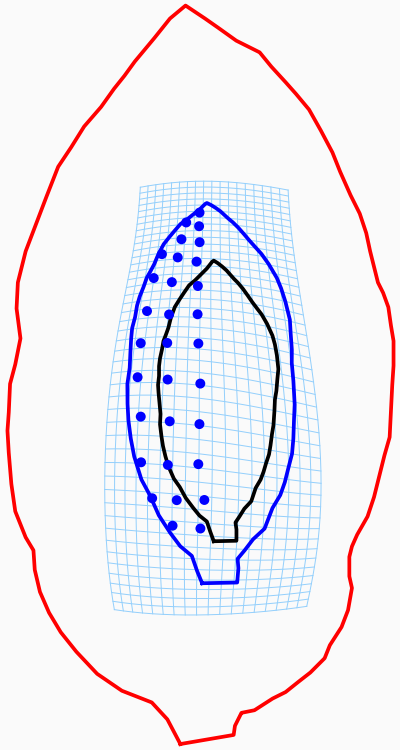
Adding dots

Leaf registration with dots



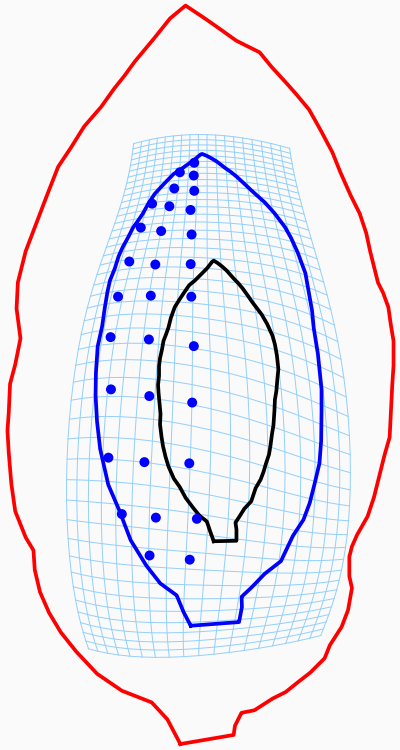
$t = 0$

Leaf registration with dots



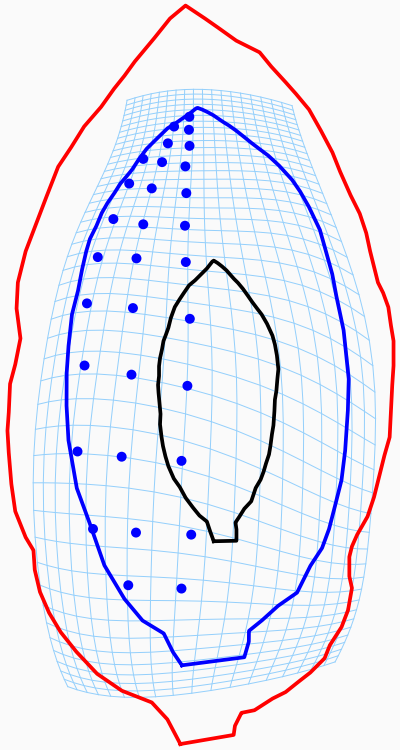
$t = 0.2$

Leaf registration with dots



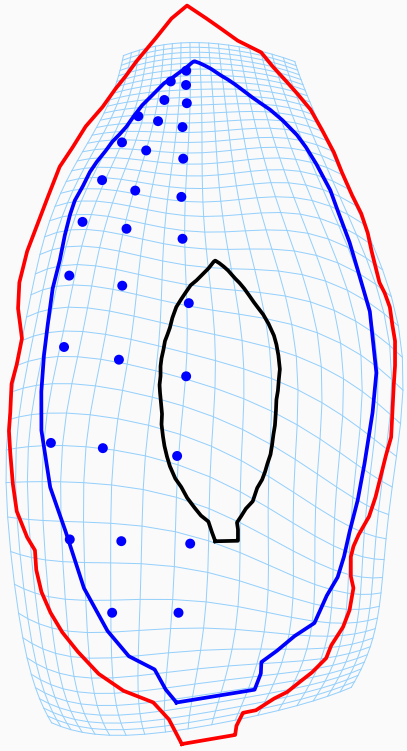
$t = 0.4$

Leaf registration with dots



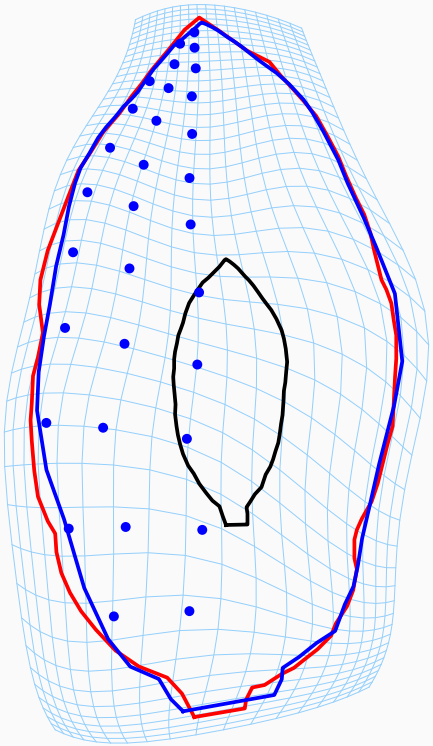
$t = 0.6$

Leaf registration with dots



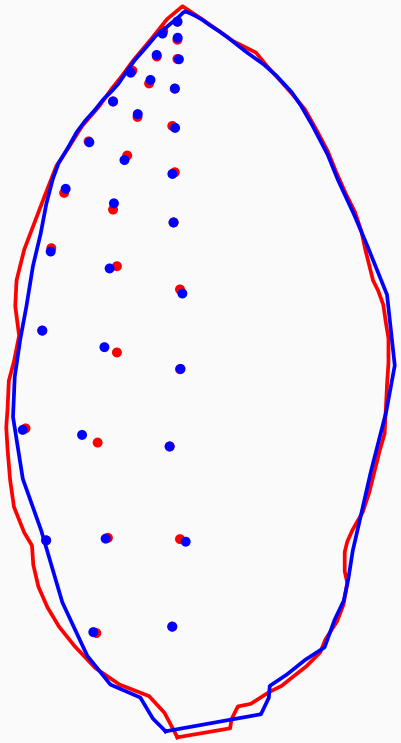
$t = 0.8$

Leaf registration with dots

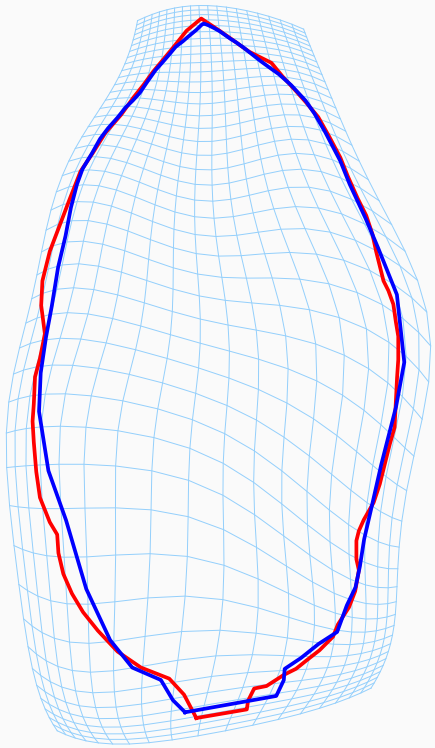


$t = 1$

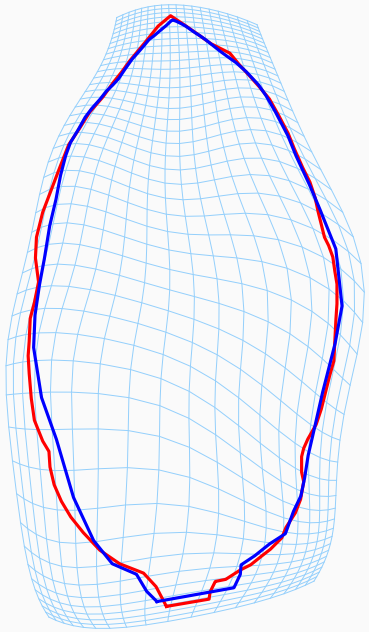
Leaf registration with dots

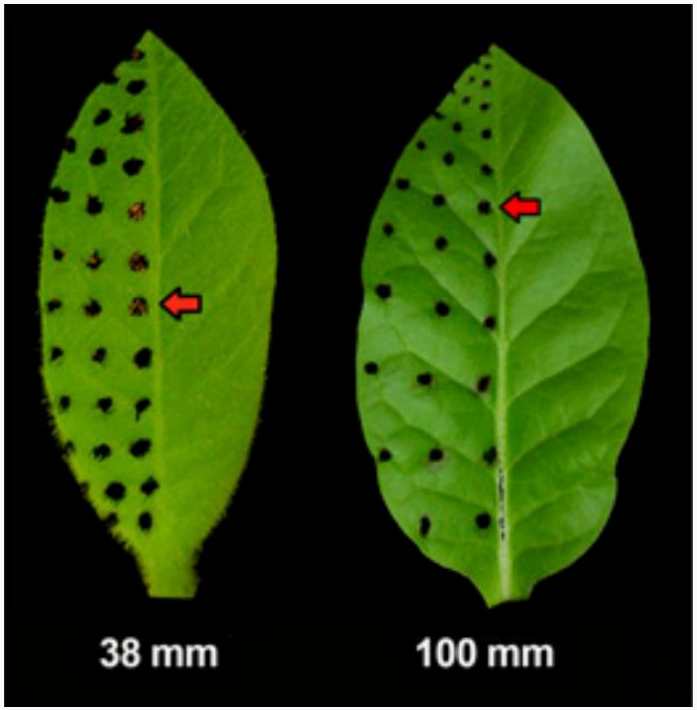


Leaf registration with dots



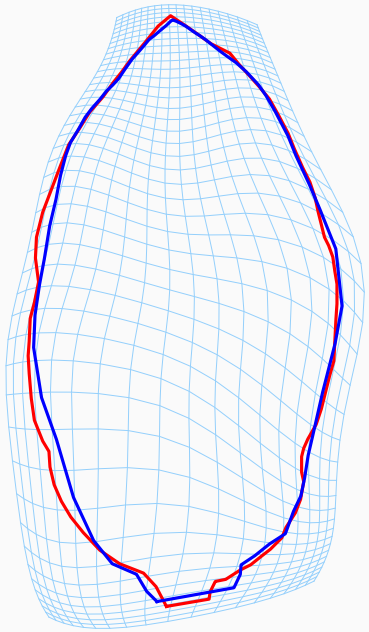
Leaf registration with dots: comparison with data

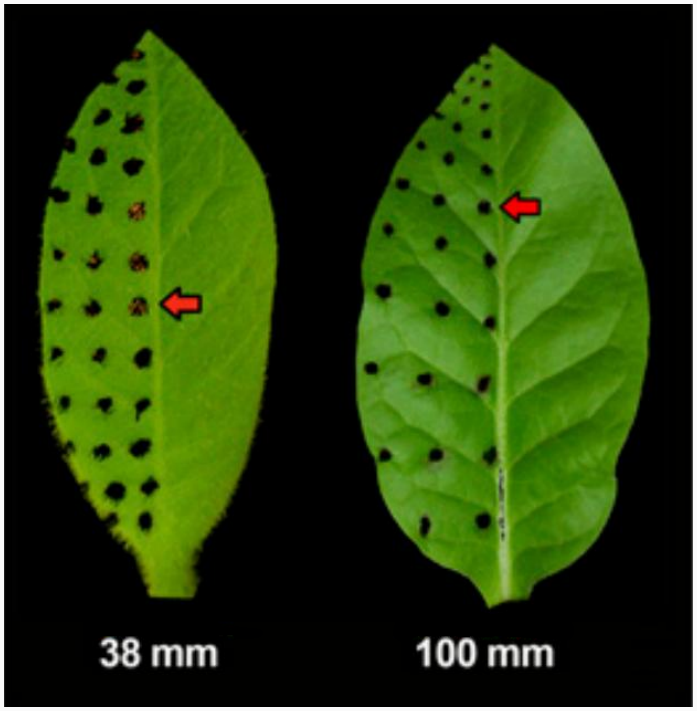




Basipetal growth illustrated by experiment

Leaf registration with dots: comparison with data





Basipetal growth illustrated by experiment

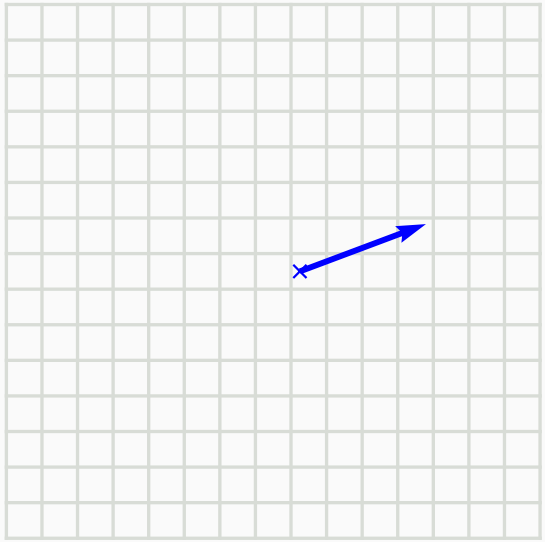
$\rightarrow$ Incorporate structure into the deformation model

IMODAL – A python library for modular deformations

Deformation modules: a way to incorporate structure into deformations

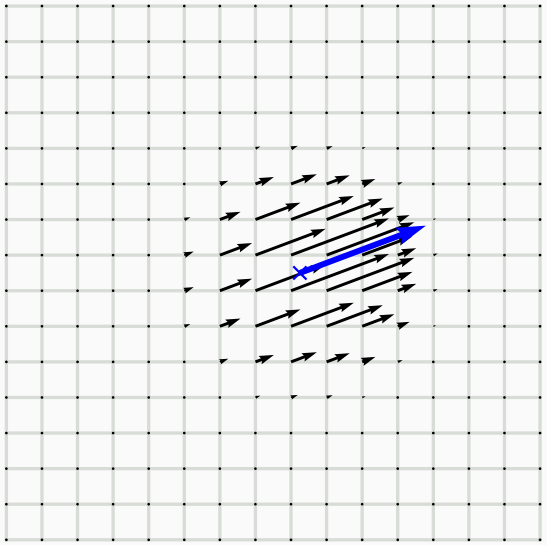
IMODAL: an implementation of the mathematical framework of deformation modules

Local translation deformation module



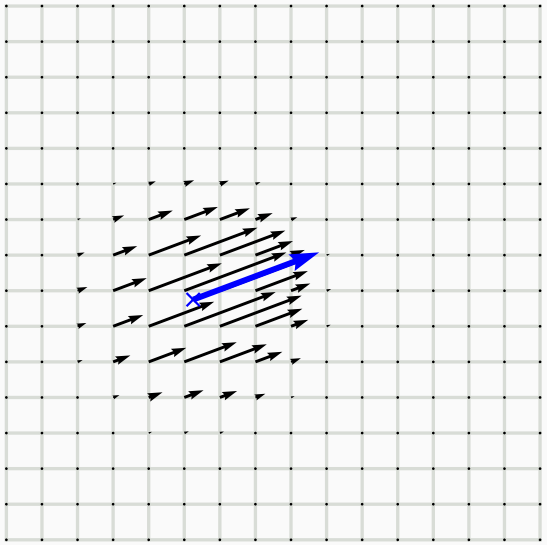
```
import imodal

translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[0.1, 0.]]))
translation.fill_controls(torch.tensor([[0.8, 0.3]]))
```

Local translation deformation module



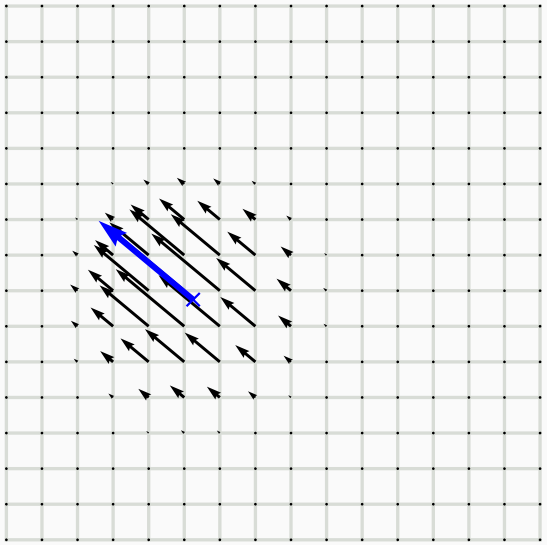
```python
import imodal

translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[0.1, 0.]]))
translation.fill_controls(torch.tensor([[0.8, 0.3]]))

vec = translation(grid_points)
```

Local translation deformation module



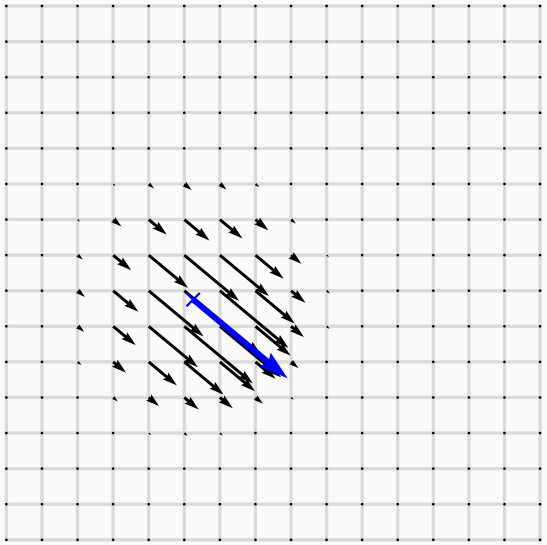
```python
import imodal

translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(torch.tensor([[0.8, 0.3]]))

vec = translation(grid_points)
```

Local translation deformation module



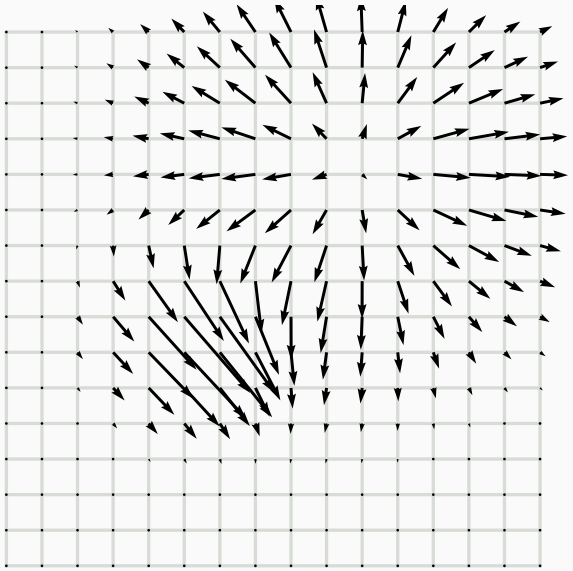
```
import imodal

translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(torch.tensor([[-0.6, 0.5]]))

vec = translation(grid_points)
```

Local translation deformation module



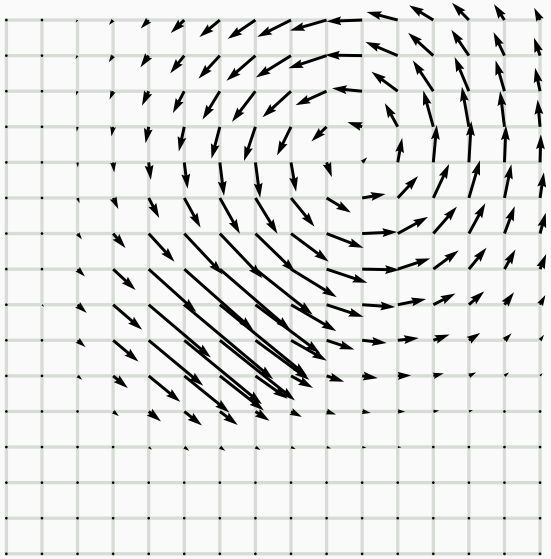
```
import imodal

translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(-torch.tensor([[-0.6, 0.5]]))

vec = translation(grid_points)
```

Local translation and scaling deformation modules



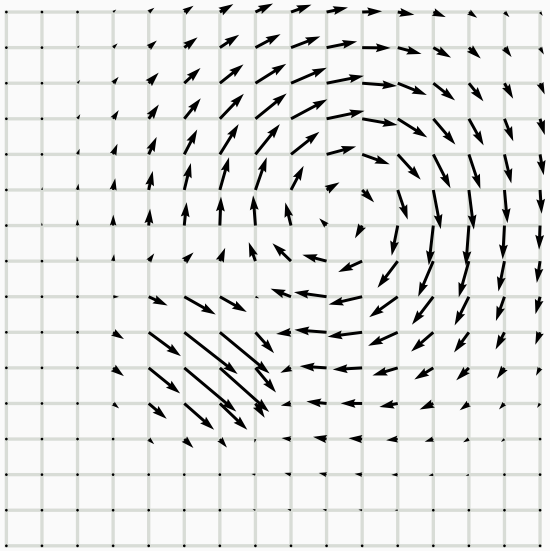
```
translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(-torch.tensor([[-0.6, 0.5]]))

scaling = imodal.DeformationModules.LocalScaling(2, sigma=0.3)
scaling.manifold.fill_gd(torch.tensor([[0.3, 0.5]]))
scaling.fill_controls(1.)

compound = imodal.DeformationModules.CompoundModule([translation, scaling])

vec = compound(grid_points)
```

Local translation and rotation deformation modules



```
translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(-torch.tensor([[-0.6, 0.5]]))

rotation = imodal.DeformationModules.LocalRotation(2, sigma=0.3)
rotation.manifold.fill_gd(torch.tensor([[0.3, 0.5]]))
rotation.fill_controls(torch.tensor([1.]))

compound = imodal.DeformationModules.CompoundModule([translation, rotation])

vec = compound(grid_points)
```

Local translation and rotation deformation modules



```
translation = imodal.DeformationModules.Translations(2, 1, sigma=0.2)
translation.manifold.fill_gd(torch.tensor([[-0.3, -0.1]]))
translation.fill_controls(-torch.tensor([[-0.6, 0.5]]))

rotation = imodal.DeformationModules.LocalRotation(2, sigma=0.3)
rotation.manifold.fill_gd(torch.tensor([[0.3, 0.5]]))
rotation.fill_controls(torch.tensor([-1.]))

compound = imodal.DeformationModules.CompoundModule([translation, rotation])

vec = compound(grid_points)
```

Flow integration into structured deformations

A structured diffeomorphism ϕ can be built by integrating the flow equation:

$$\begin{cases} \phi_0 = \mathrm{Id} \\ \dot{\phi}_t = v_t \circ \phi_t, & t \in [0,1] \end{cases}$$

With v_t the sum of all structured vector fields.

[4] Chen et al., "Neural Ordinary Differential Equations".

Flow integration into structured deformations

A structured diffeomorphism ϕ can be built by integrating the flow equation:

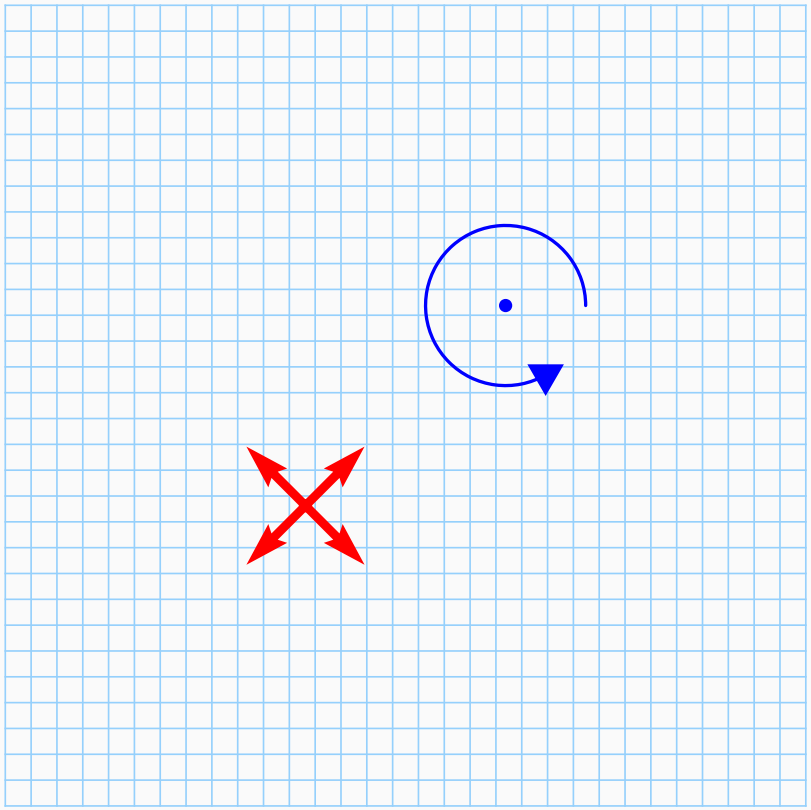
$$\begin{cases} \phi_0 = \mathrm{Id} \\ \dot{\phi}_t = v_t \circ \phi_t, & t \in [0, 1] \end{cases}$$

With v_t the sum of all structured vector fields.

Numerical integration using `torchdiffeq`[4]

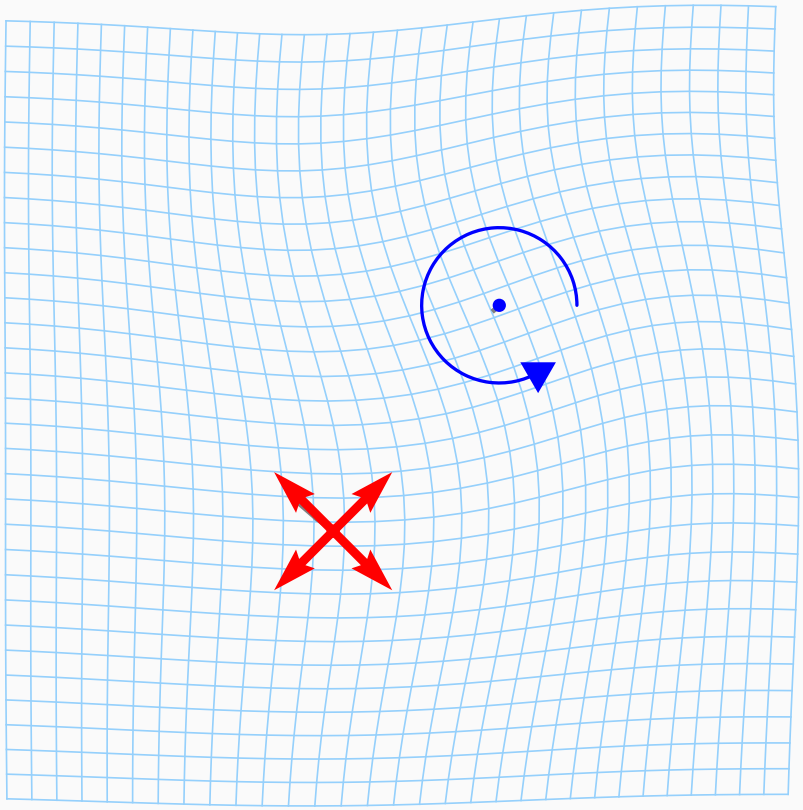
[4] Chen et al., "Neural Ordinary Differential Equations".

Integrating structured vector fields



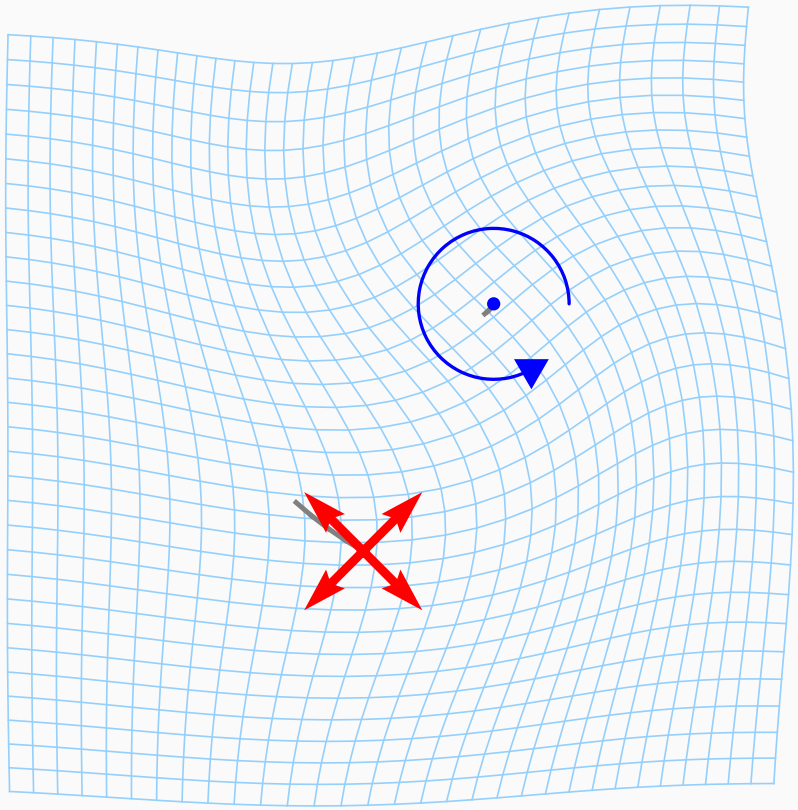
```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Integrating structured vector fields



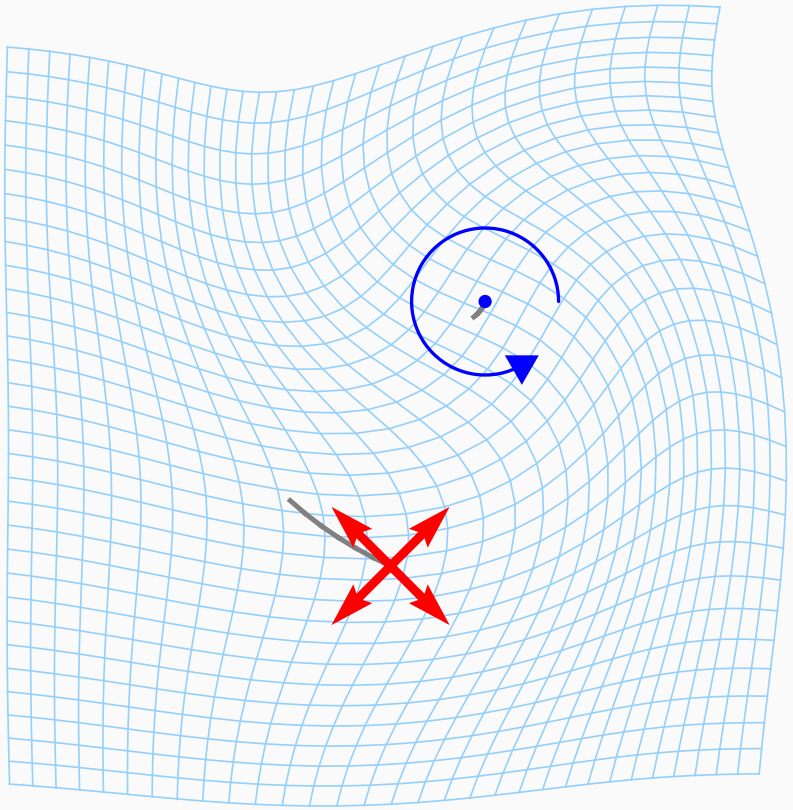
```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Integrating structured vector fields



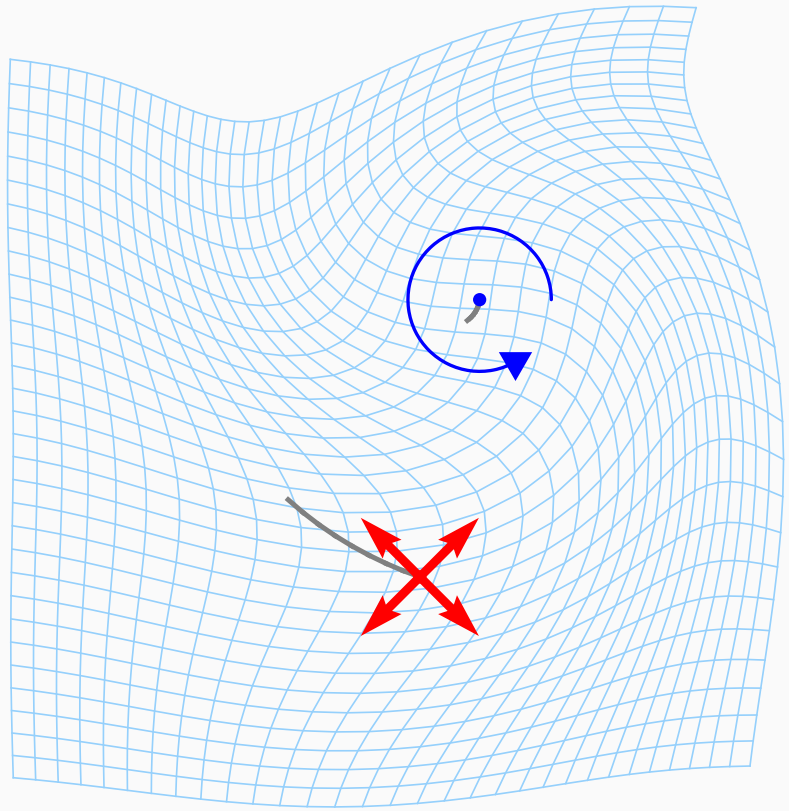
```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Integrating structured vector fields



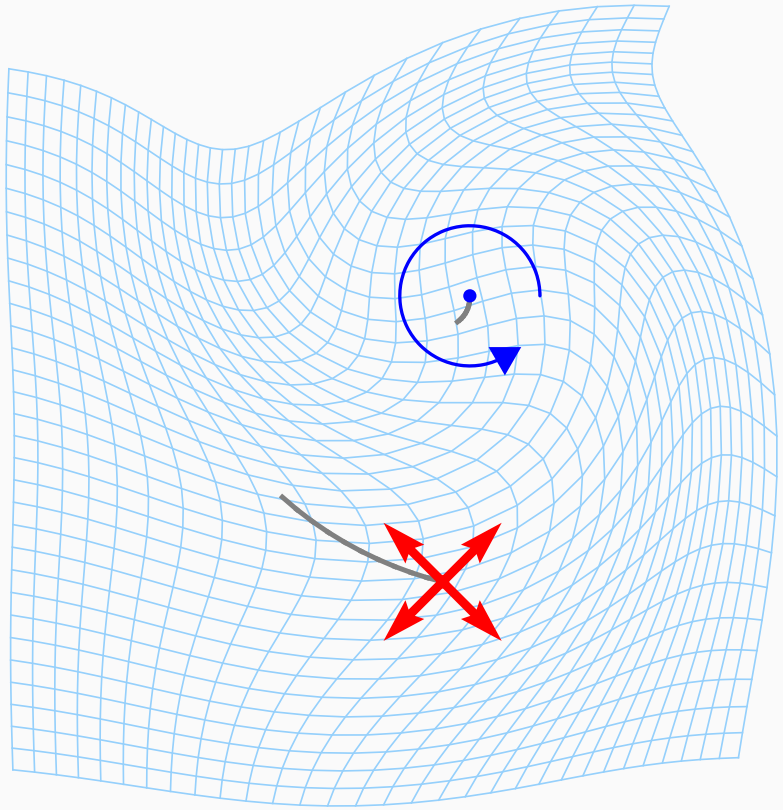
```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Integrating structured vector fields



```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Integrating structured vector fields



```
controls = [(torch.tensor(0.4), torch.tensor(2.), torch.tensor([]))]*it

H = imodal.HamiltonianDynamic.Hamiltonian(compound_module)
imodal.HamiltonianDynamic.shoot(H, solver='euler', it=10, controls=controls)
```

Shape registration

Goal: match source shape q_S towards target shape q_T

Shape registration

Goal: match source shape q_S towards target shape q_T

Strategy:

- Define a plausible deformation model by combining modules
- Numerical minimization of an energy function $\mathcal{J}$

$$\mathcal{J}(\phi; q_S, q_T) = \mathcal{U}(\phi; q_S, q_T) + \mathcal{R}(\phi)$$

With,

$\mathcal{U}$: similarity between the deformed source and the target

$\mathcal{R}$: regularity of the deformation ϕ, i.e. its cost

Shape registration

Define registration model:

```
model = imodal.Models.RegistrationModel(
    [deformable_shape_source, deformable_dots_source],
    [lddmm],
    [imodal.Attachment.VarifoldAttachment(2, [20., 120.], backend='torch'),
     imodal.Attachment.EuclideanPointwiseDistanceAttachment(10.)],
    lam=10.)
```

Shape registration

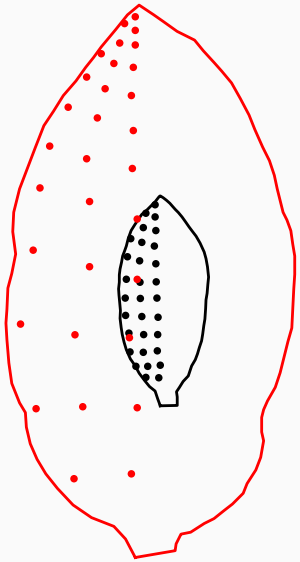
Define registration model:

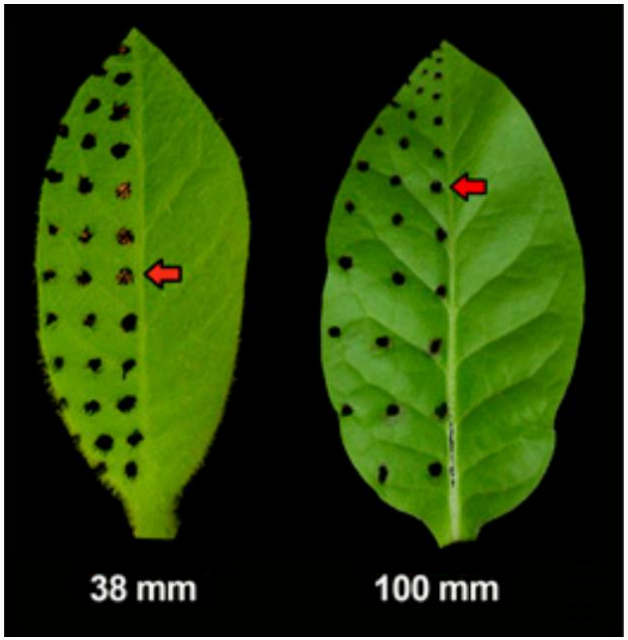
```
model = imodal.Models.RegistrationModel(
    [deformable_shape_source, deformable_dots_source],
    [lddmm],
    [imodal.Attachment.VarifoldAttachment(2, [20., 120.], backend='torch'),
     imodal.Attachment.EuclideanPointwiseDistanceAttachment(10.)],
    lam=10.)
```

Fit the model:

```
shoot_solver = 'euler'
shoot_it = 10

fitter = imodal.Models.Fitter(model, optimizer='torch_lbfgs')
fitter.fit([deformable_shape_target, deformable_dots_target], 100,
    options={'shoot_solver': shoot_solver, 'shoot_it': shoot_it,
    'line_search_fn': 'strong_wolfe'})
```

Back to basipetal leaf growth

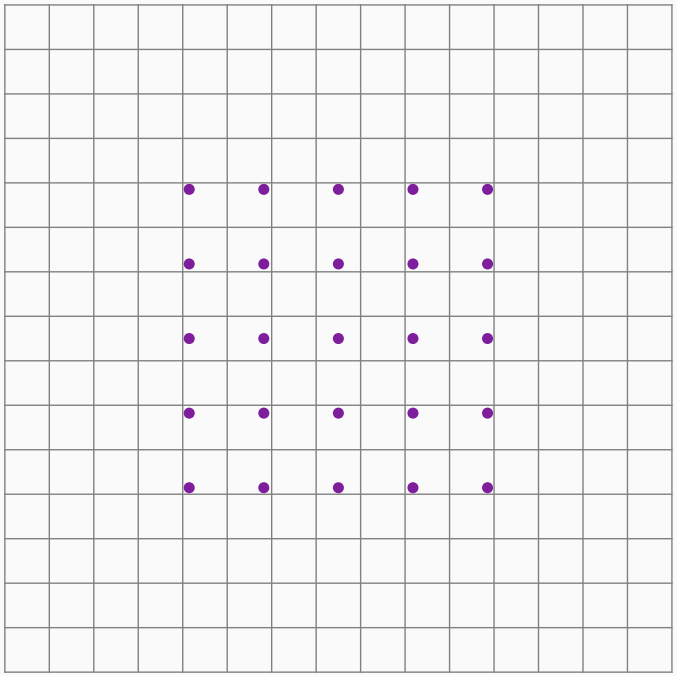




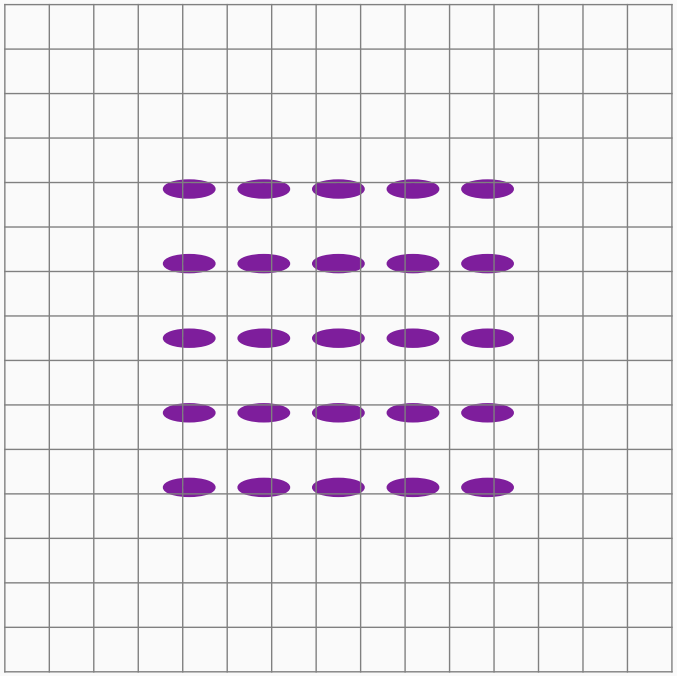
Basipetal growth illustrated by experiment

$\rightarrow$ How do we use modules to solve this problem ?

An answer: implicit module

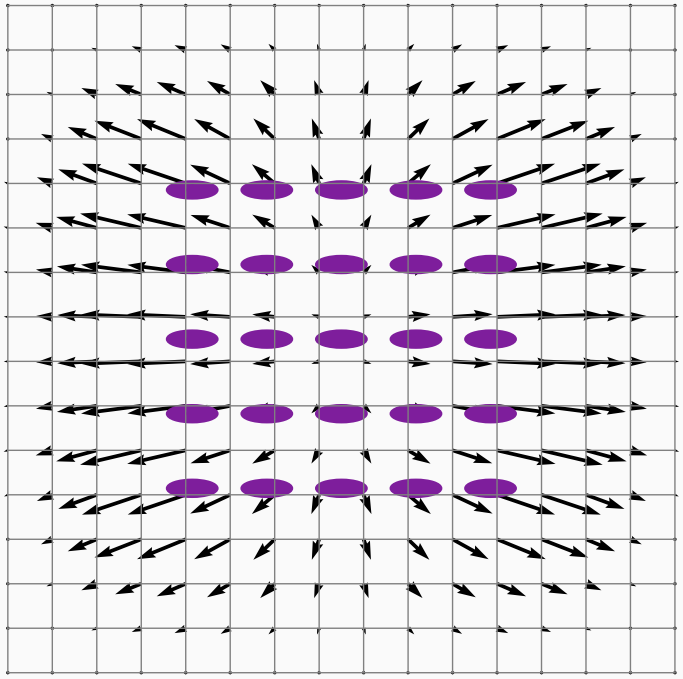


Implicit modules



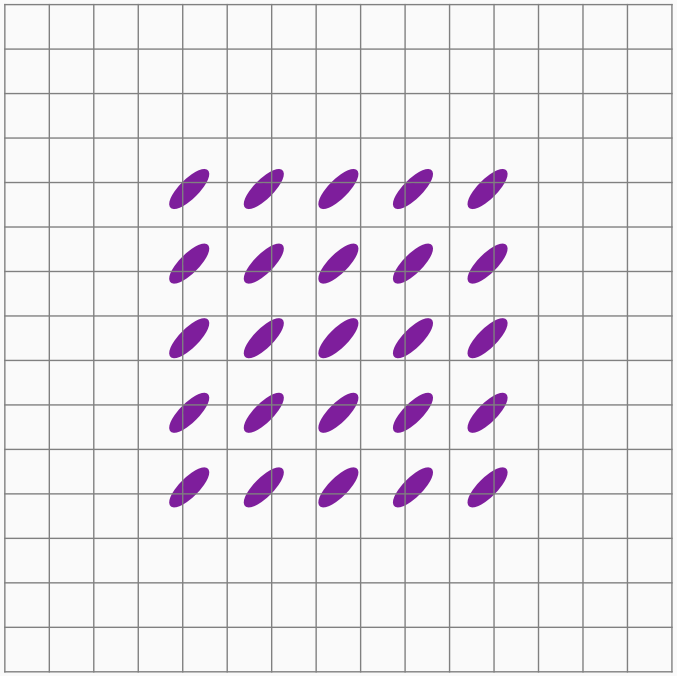
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(0.).repeat(len(positions), 1, 1)

growth_model = torch.zeros(positions.shape[0], 2, 1)
growth_model[:, 0, 0] = 3.
growth_model[:, 1, 0] = 1.
```

Implicit modules



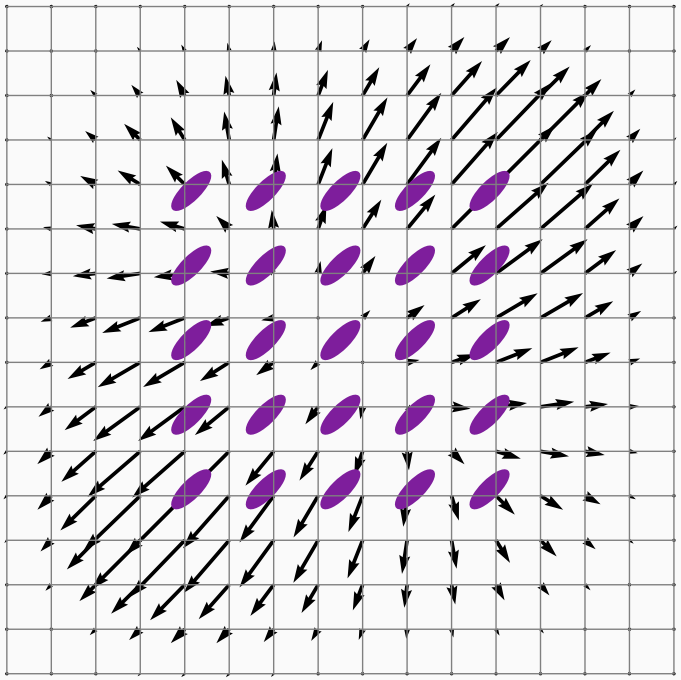
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(0.).repeat(len(positions), 1, 1)

growth_model = torch.tensor([[3.], [1.]]).repeat(len(positions), 1, )

implicit1 = imodal.DeformationModules.ImplicitModule1(2, len(implicit1_pts),
     sigma=0.2, C=growth_model, gd=(positions, local_frames))
implicit1.fill_controls(torch.tensor([1.]))

vec = implicit1(grid_points)
```

Implicit modules



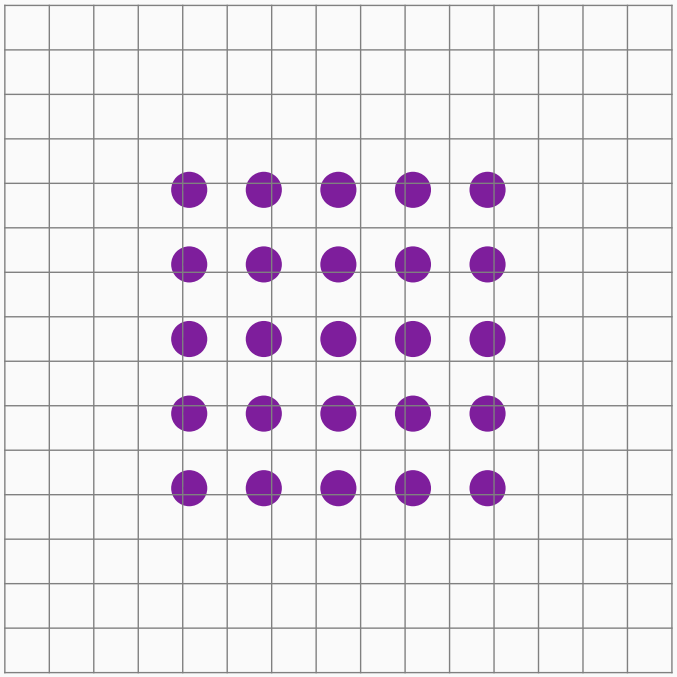
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(1./4.*math.pi).repeat(len(positions), 1,
    1)

growth_model = torch.tensor([[3.], [1.]]).repeat(len(positions), 1, )
```

Implicit modules



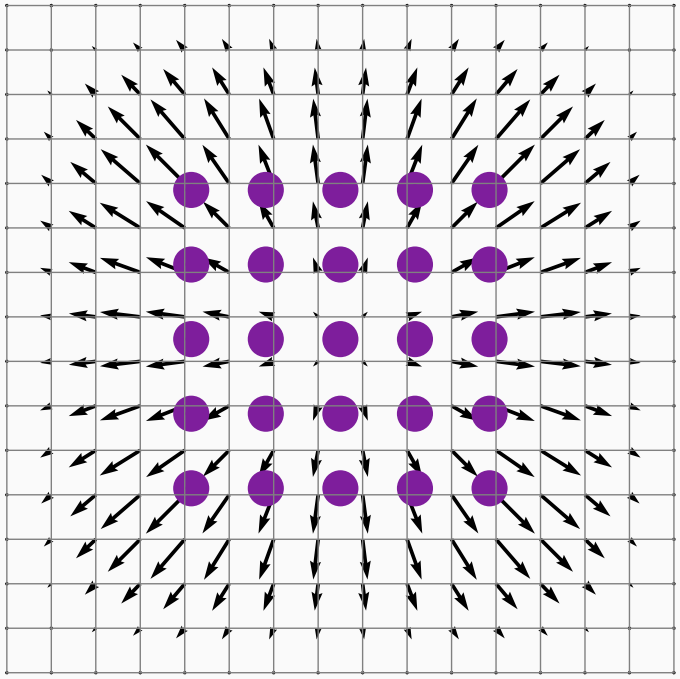
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(1./4.*math.pi).repeat(len(positions), 1,
    1)

growth_model = torch.tensor([[3.], [1.]]).repeat(len(positions), 1, )

implicit1 = imodal.DeformationModules.ImplicitModule1(2, len(mplicit1_pts),
    sigma=0.2, C=growth_model, gd=(positions, local_frames))
implicit1.fill_controls(torch.tensor([1.]))

vec = implicit1(grid_points)
```

Implicit modules



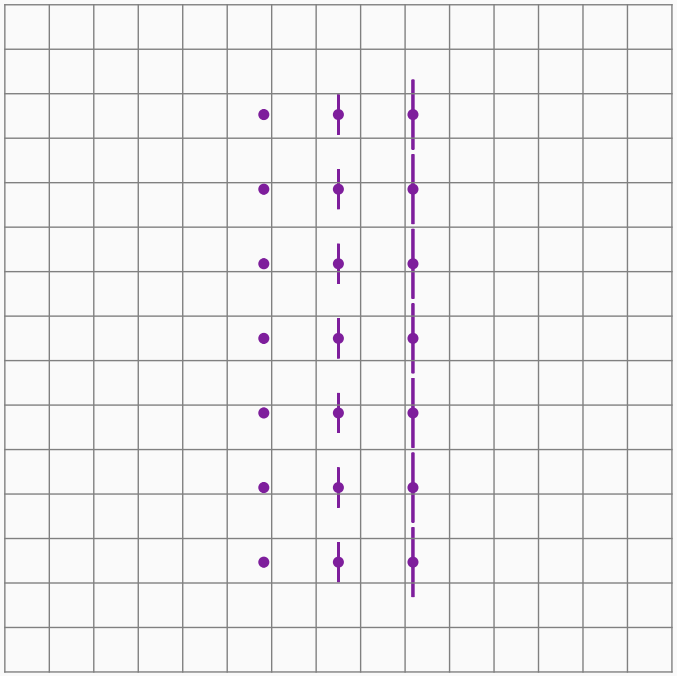
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(0.).repeat(len(positions), 1, 1)

growth_model = torch.ones(len(positions), 2, 1)
```

Implicit modules



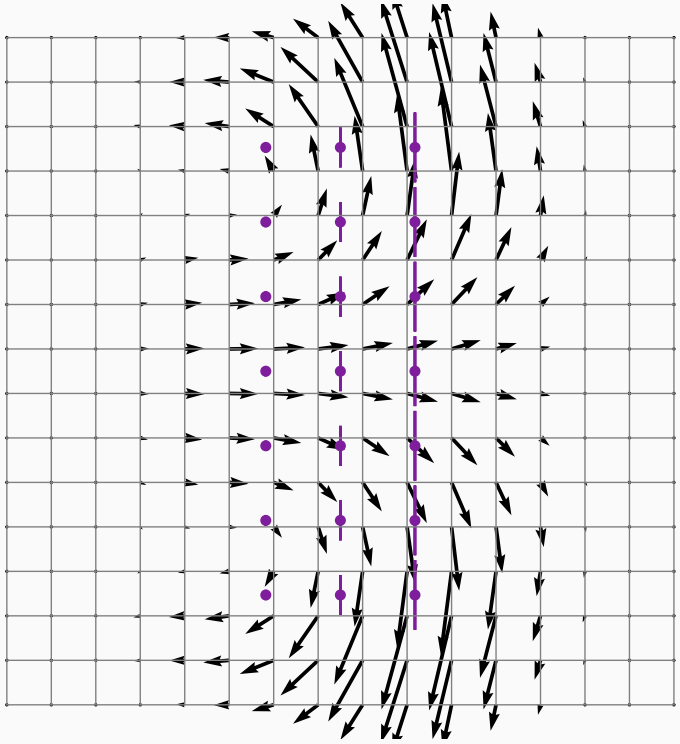
```
positions = aabb.fill_uniform_density(20.)
local_frames = imodal.Utilities.rot2d(0.).repeat(len(positions), 1, 1)

growth_model = torch.ones(len(positions), 2, 1)

implicit1 = imodal.DeformationModules.ImplicitModule1(2, len(mplicit1_pts),
     sigma=0.2, C=growth_model, gd=(positions, local_frames))
implicit1.fill_controls(torch.tensor([1.]))

vec = implicit1(grid_points)
```

Implicit modules



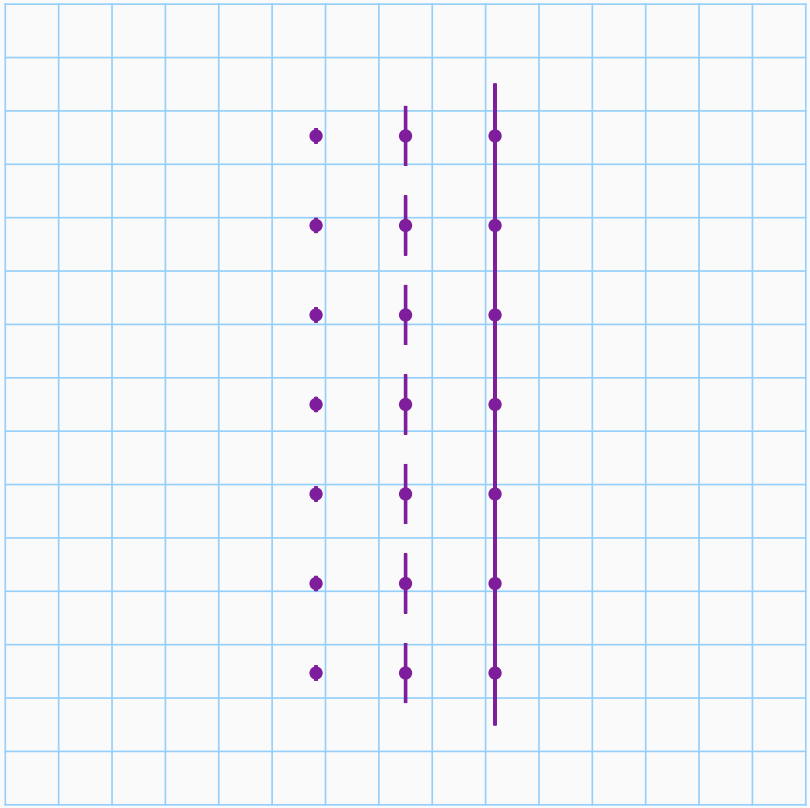
```
C = f(implicit1_pts)
```

Implicit modules

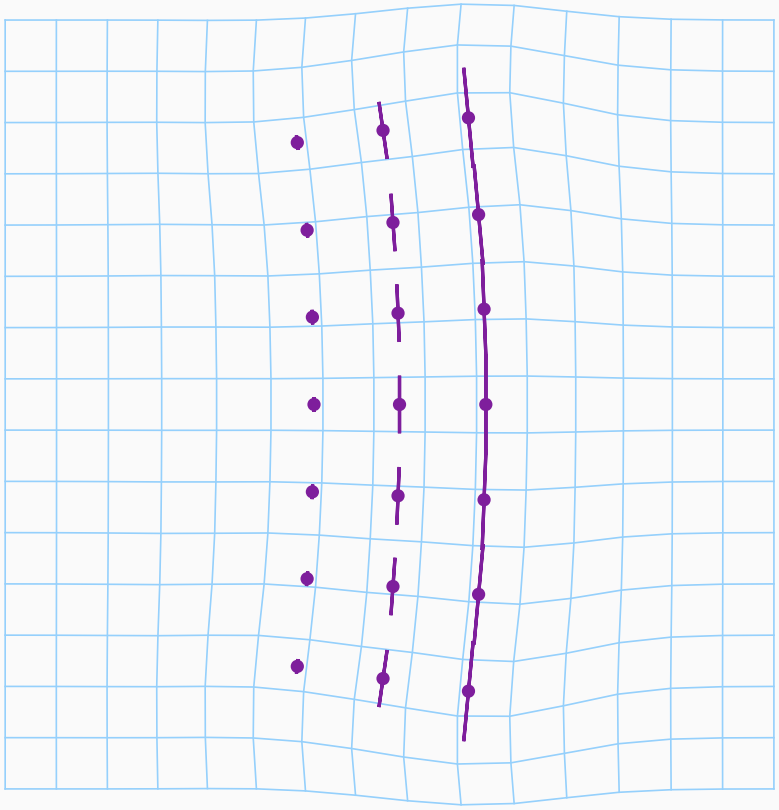


```
C = f(implicit1_pts)

implicit1 = imodal.DeformationModules.ImplicitModule1(2, implicit1_pts.shape[0],
      0.2, C, gd=(implicit1_pts, implicit1_r))
implicit1.fill_controls(torch.tensor([1.]))

vec = implicit1(grid_points)
```

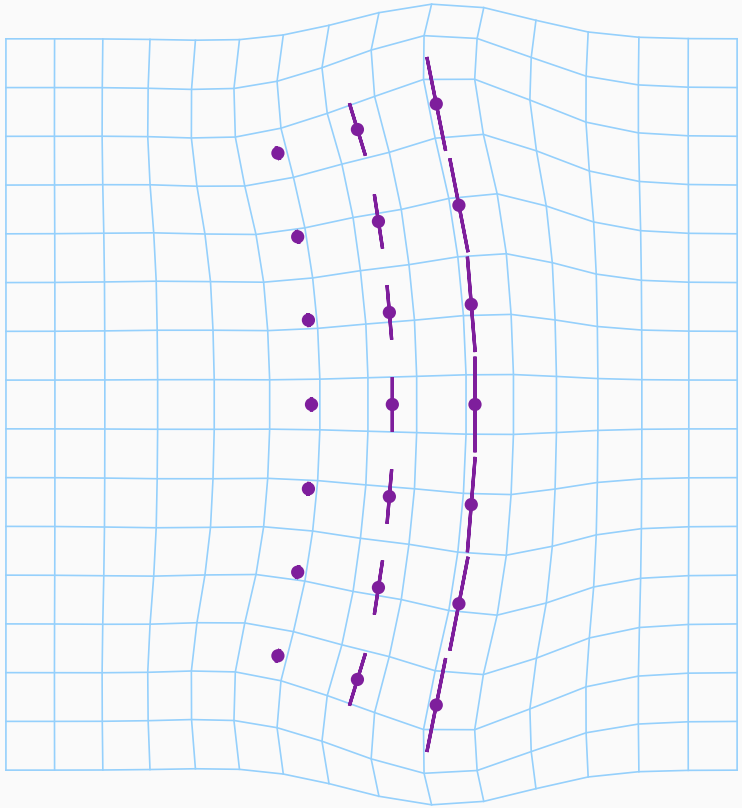
Bending using implicit modules



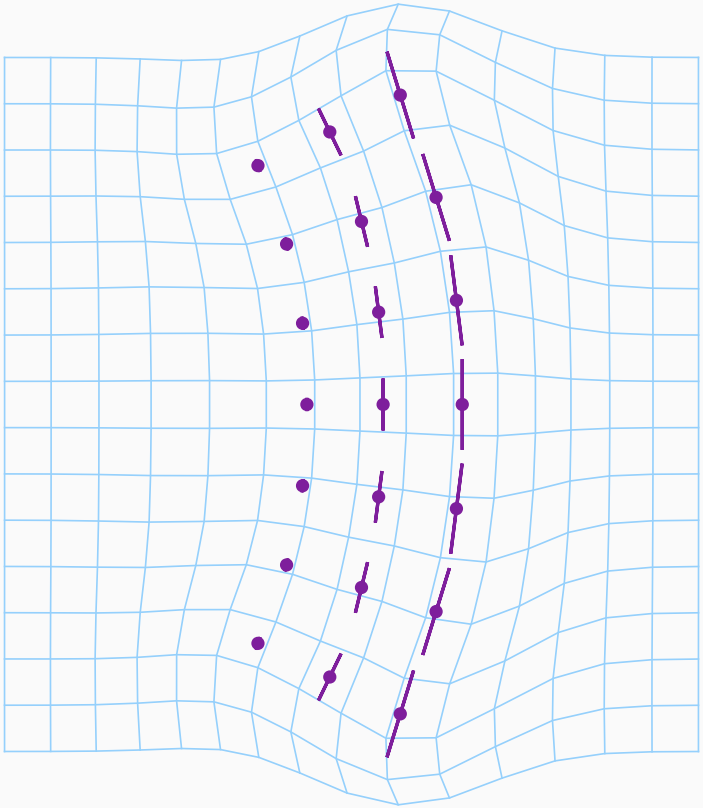
Bending using implicit modules



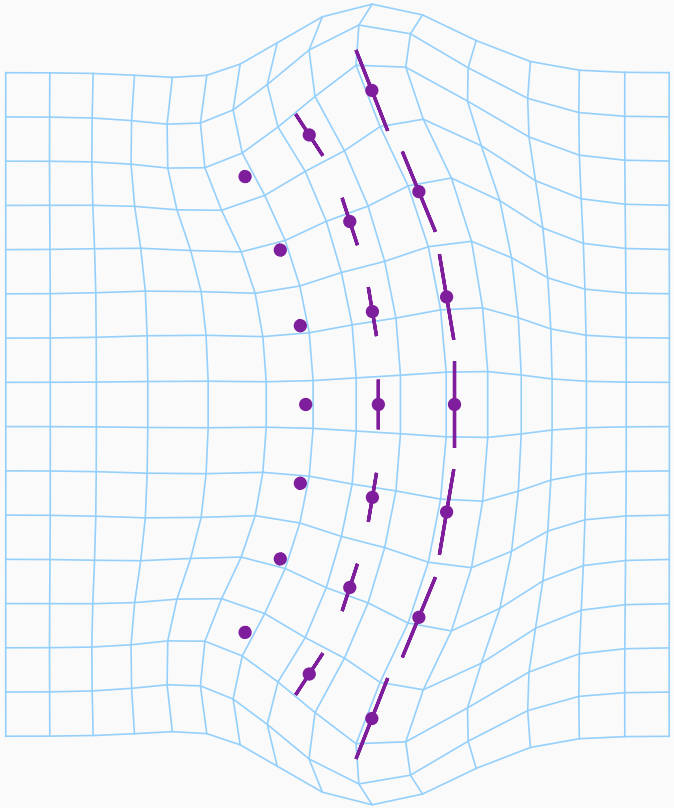
Bending using implicit modules



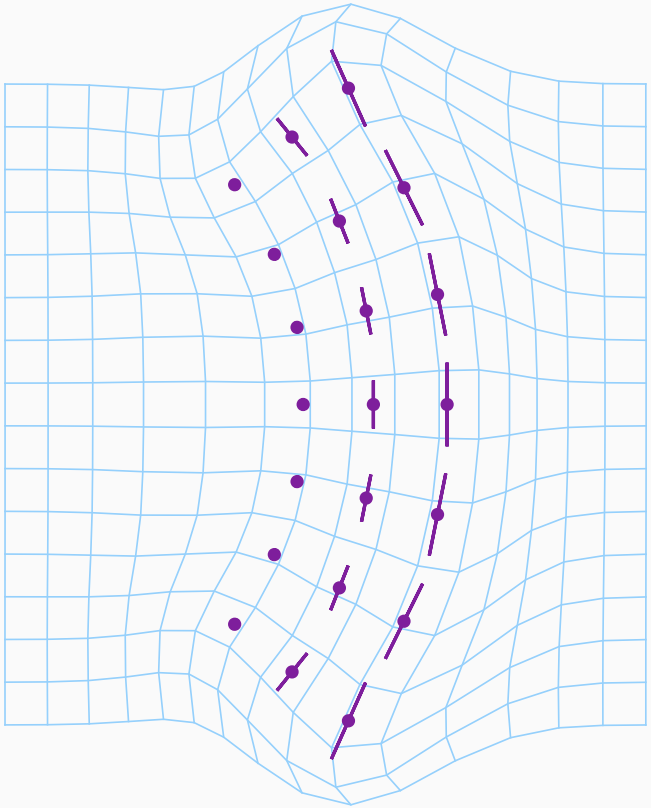
Bending using implicit modules



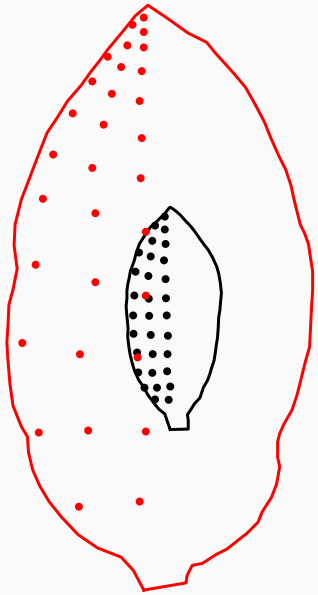
Bending using implicit modules

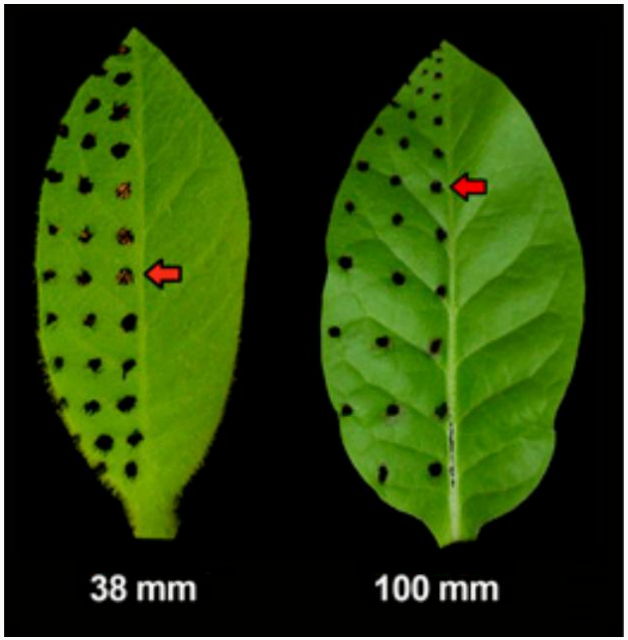


Bending using implicit modules



Basipetal growth model using implicit modules

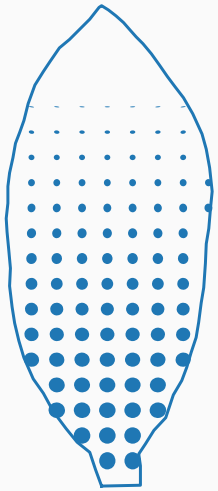




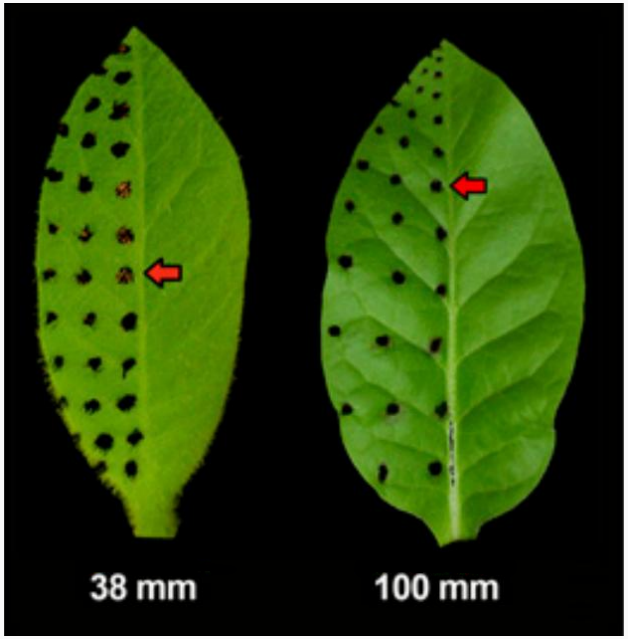
Basipetal growth illustrated by experiment

How do we use modules to solve this problem ?

Basipetal growth model using implicit modules



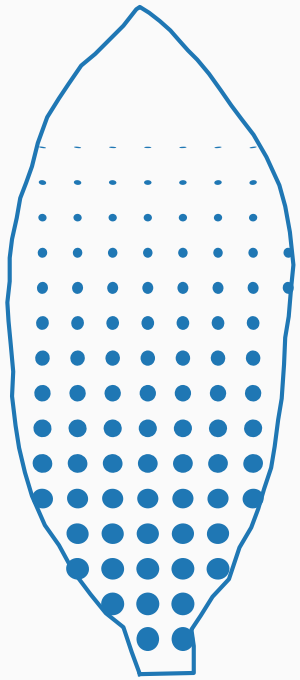
Growth parameters



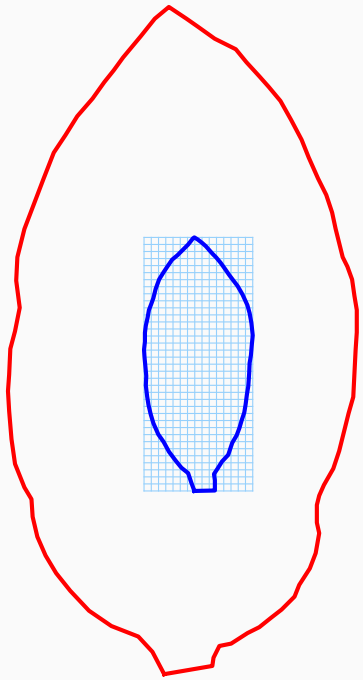
Basipetal growth illustrated by experiment

$\rightarrow$ Using an implicit module !

Basipetal growth using implicit modules

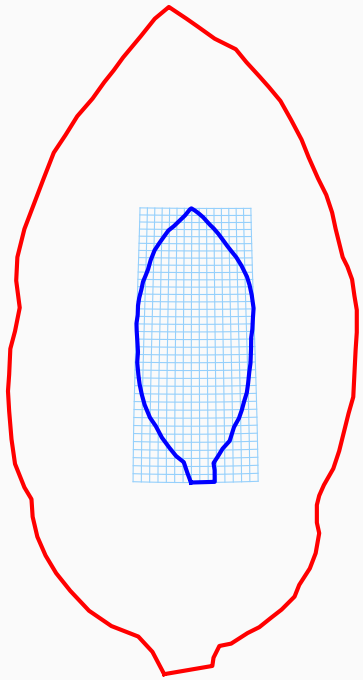


Basipetal growth using implicit modules



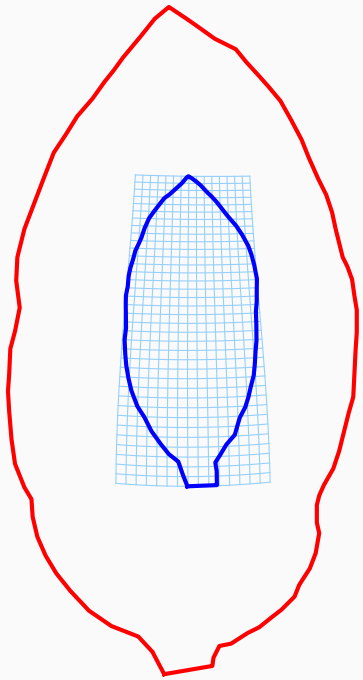
$t = 0$

Basipetal growth using implicit modules



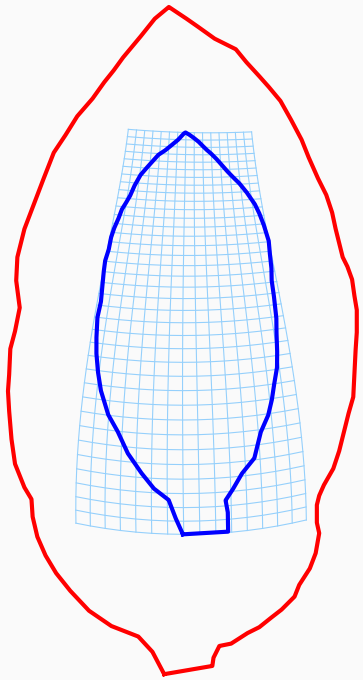
$t = 0.2$

Basipetal growth using implicit modules



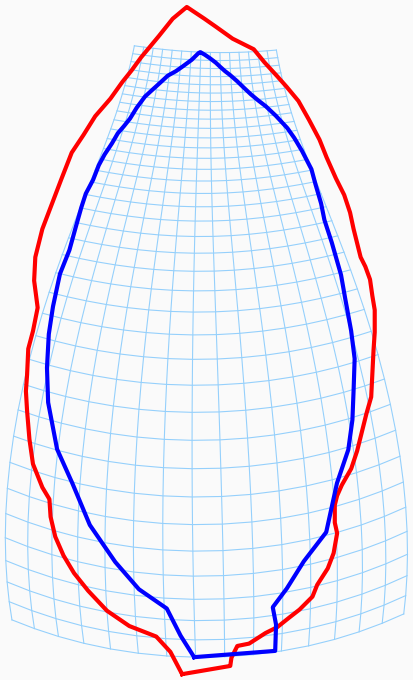
$t = 0.4$

Basipetal growth using implicit modules



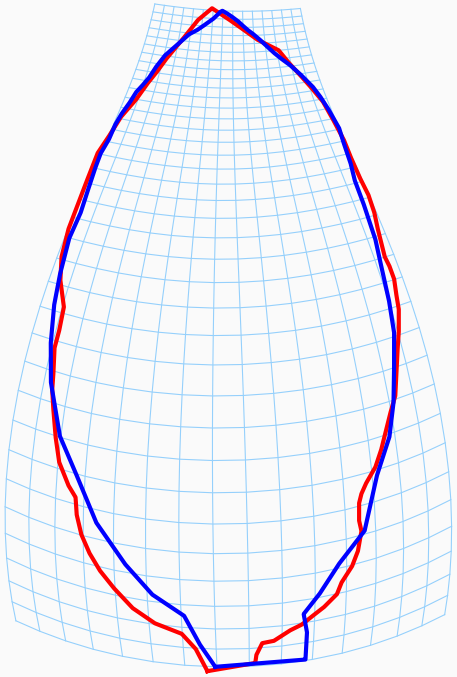
$t = 0.6$

Basipetal growth using implicit modules



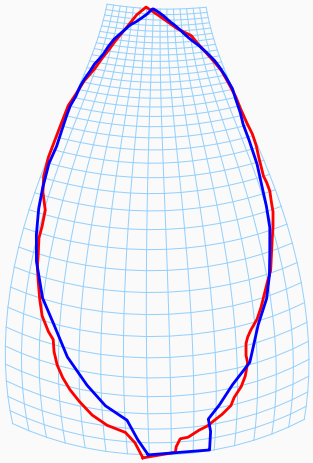
$t = 0.8$

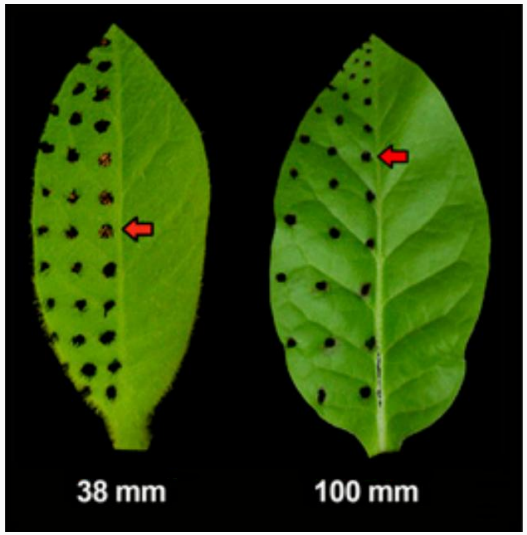
Basipetal growth using implicit modules



$t = 1$

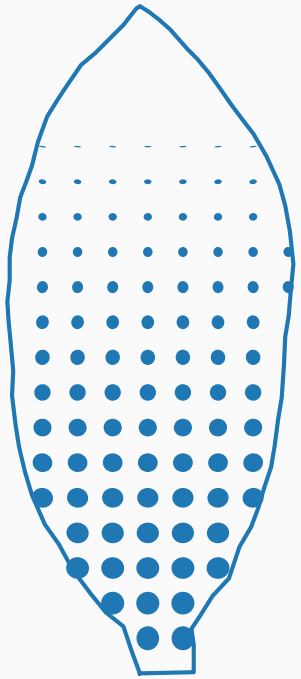
Basipetal growth model using implicit modules





Basipetal growth illustrated by experiment

Basipetal growth using implicit modules



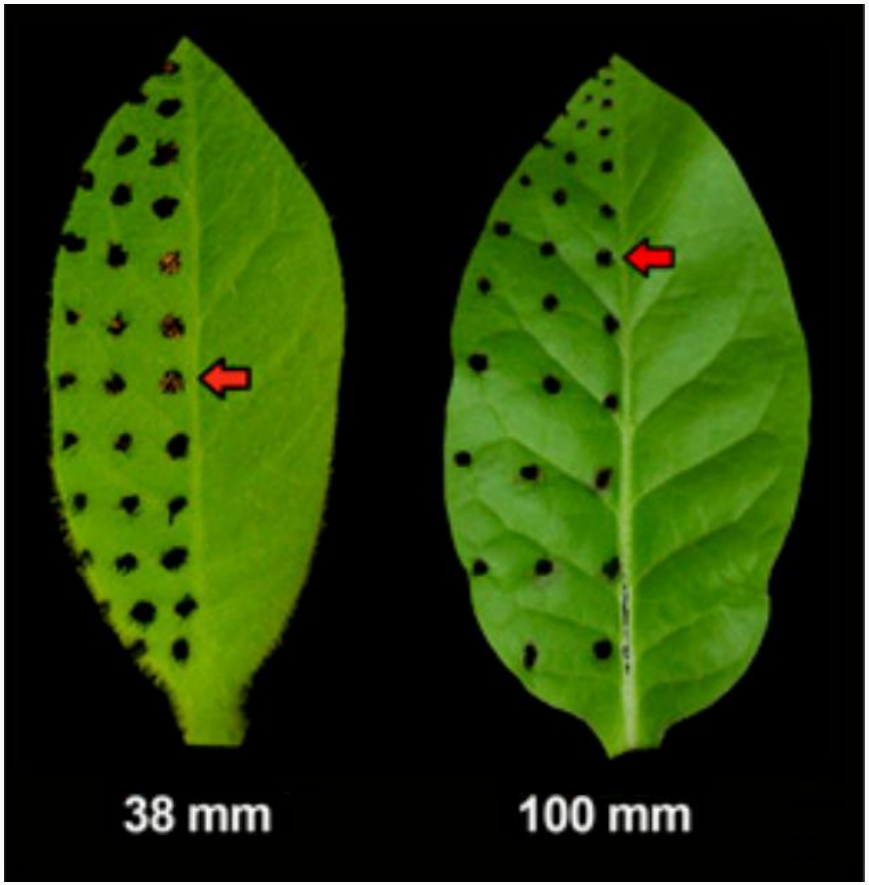
How did we get those growth parameters?

Model parameter fitting

IMODAL implements a rich interface for shape registration

- User callback before model evaluation
- Fit of geometrical descriptors (e.g. point positions)
- Fit of any external parameters

Basipetal growth pattern fitting



Prior for the growth parameter:

$$C(x, y)_i = a_i + b_i y + c_i y^2 + d_i y^3$$

Basipetal growth pattern fitting

Define appropriate callback function:

```
def callback_compute_c(init_manifold, modules, parameters, deformables):
    abcd = parameters['abcd']['params'][0]
    a = abcd[0].unsqueeze(1)
    b = abcd[1].unsqueeze(1)
    c = abcd[2].unsqueeze(1)
    d = abcd[3].unsqueeze(1)
    modules[3].C = a + b*pos[:, 1] + c*pos[:, 1]**2 + d*pos[:, 1]**3
```

Define model with callback function and polynome coefficients:

```
model = imodal.Models.RegistrationModel(
    [deformable_shape_source, deformable_dots_source],
    [global_translation, growth, small_scale_translations],
    [imodal.Attachment.VarifoldAttachment(2, [20., 120.], backend='torch'),
     imodal.Attachment.EuclideanPointwiseDistanceAttachment(10.)], lam=100.,
    other_parameters={'abcd': {'params': [abcd]}},
    precompute_callback=callback_compute_c)
```

IMODAL library

Object oriented design

Built on top of Pytorch

- GPU support
- Automatic differentiation

[5] Charlier et al., "Kernel Operations on the GPU, with Autodiff, without Memory Overflows".

IMODAL library

Object oriented design

Built on top of Pytorch

- GPU support
- Automatic differentiation

Supports:

- 2D, 3D point clouds, meshes
- 2D, 3D images

[5] Charlier et al., "Kernel Operations on the GPU, with Autodiff, without Memory Overflows".

IMODAL library

Object oriented design

Built on top of Pytorch

- GPU support
- Automatic differentiation

Supports:

- 2D, 3D point clouds, meshes
- 2D, 3D images

KeOps[5] for large kernel matrix reductions

[5] Charlier et al., "Kernel Operations on the GPU, with Autodiff, without Memory Overflows".

IMODAL library

Object oriented design

Built on top of Pytorch

- GPU support
- Automatic differentiation

Supports:

- 2D, 3D point clouds, meshes
- 2D, 3D images

KeOps[5] for large kernel matrix reductions

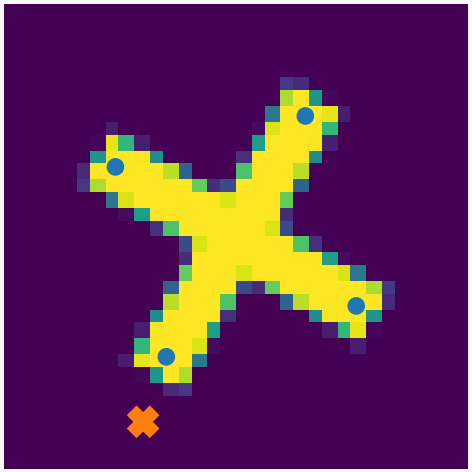
Large library of utility functions

- To help with point placements
- For plotting

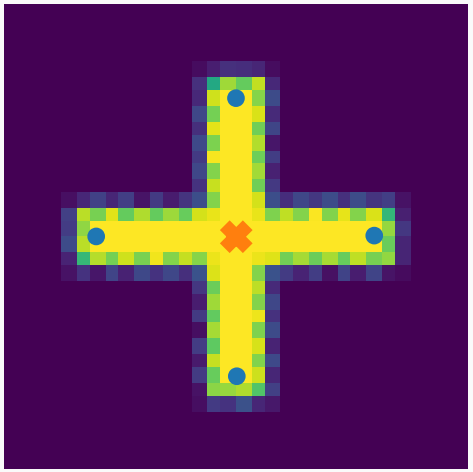
[5] Charlier et al., "Kernel Operations on the GPU, with Autodiff, without Memory Overflows".

Example – Fitting geometrical descriptors

Source image



Fitted image



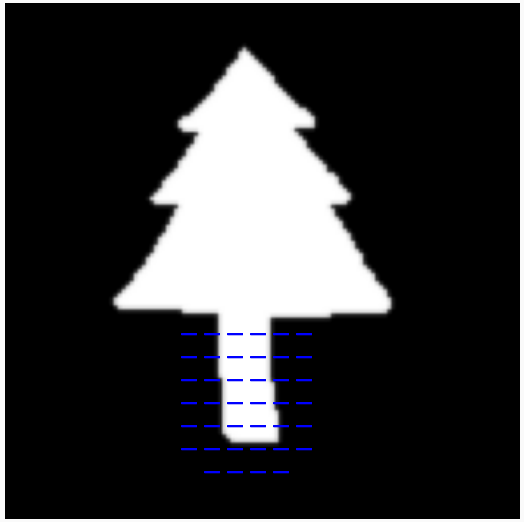
Target image

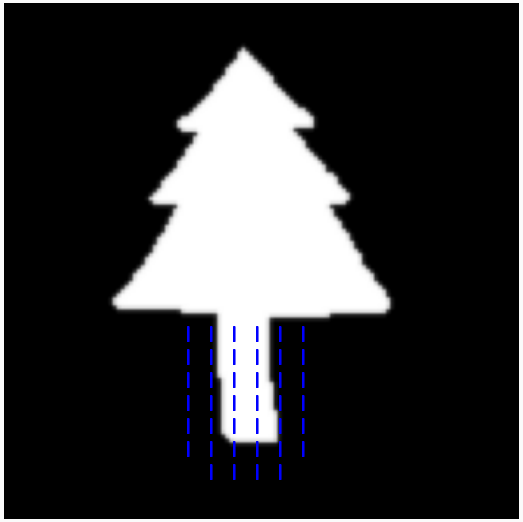
Example – Interpretable controls: fitting trees

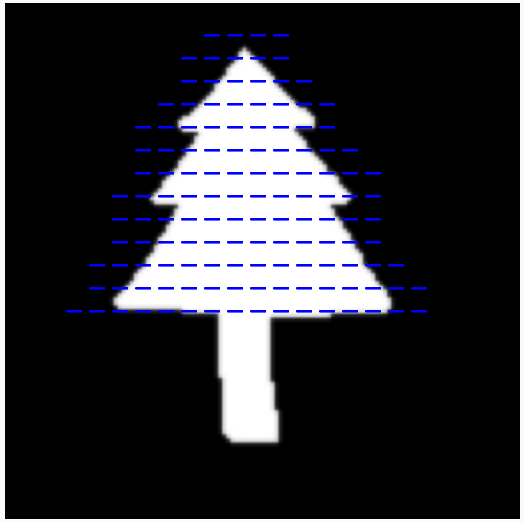
Source

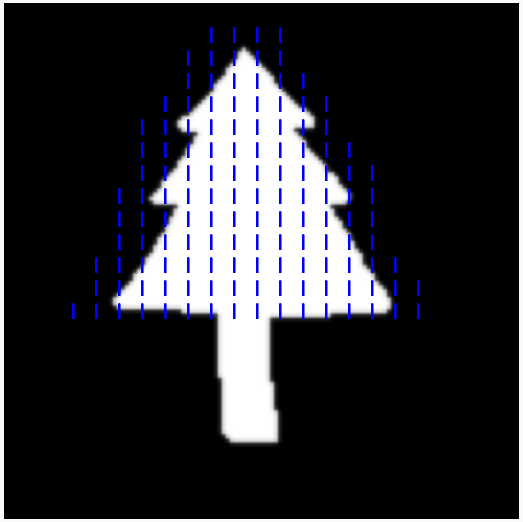
Target

Example –Interpretable controls: growth parameters

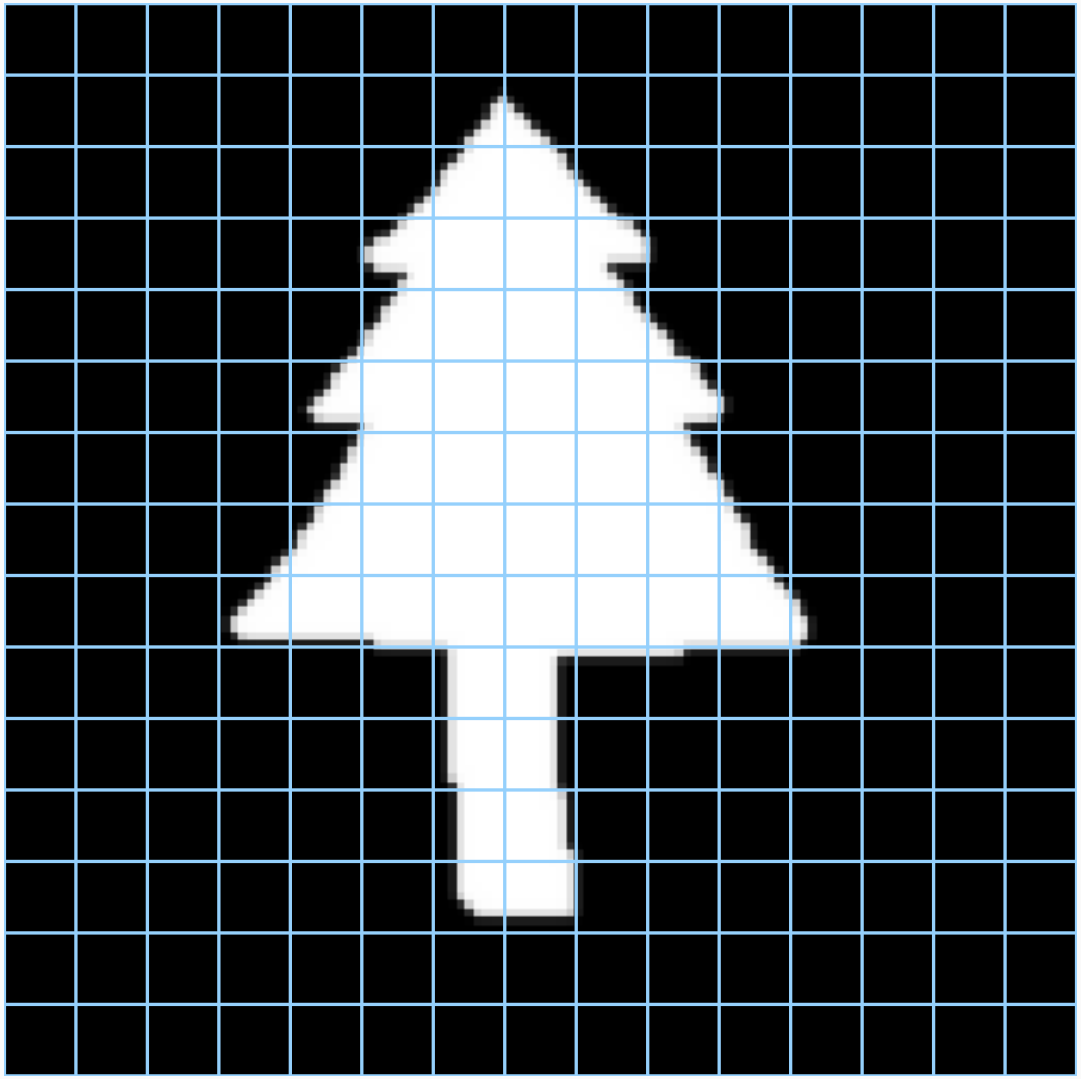




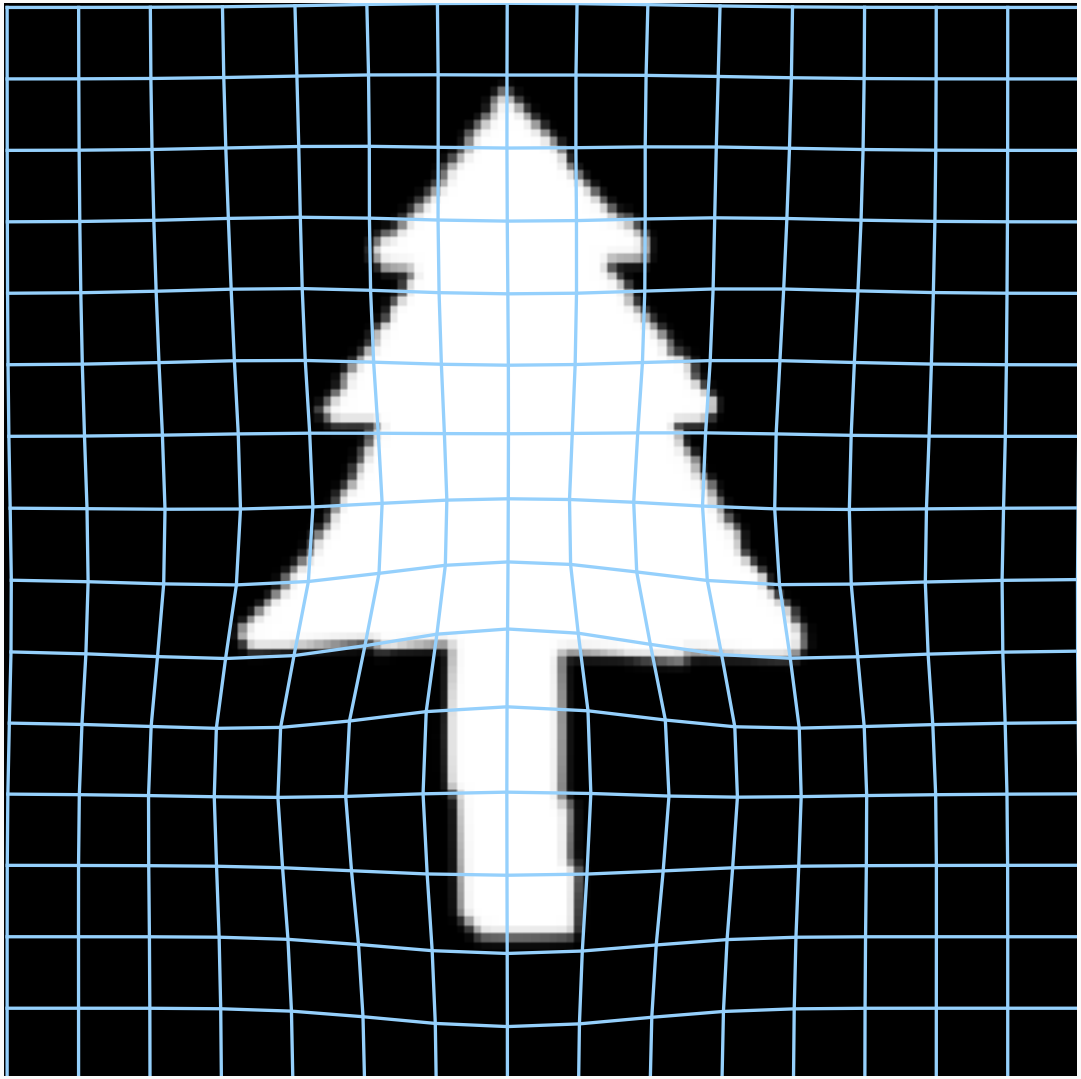




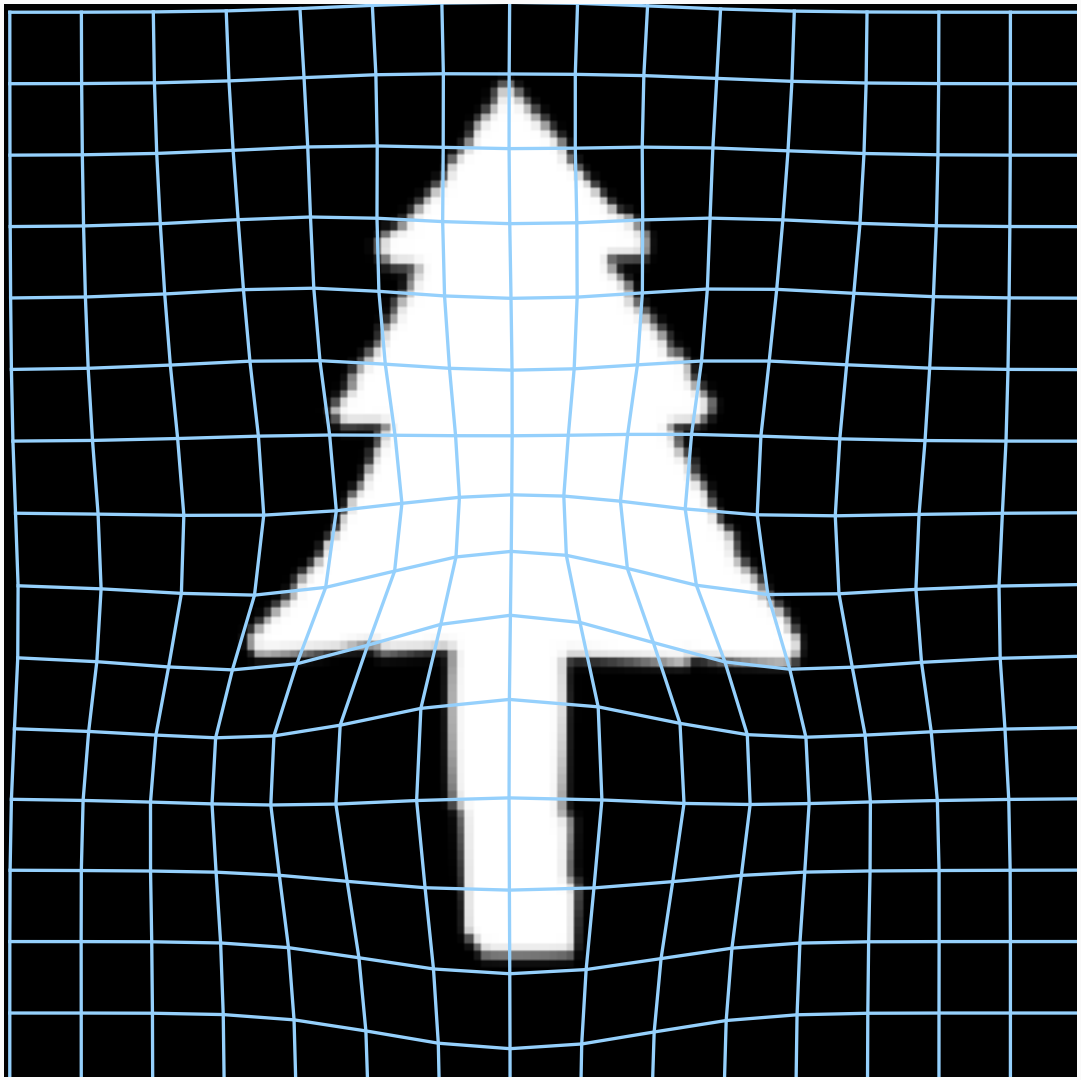
Example – Interpretable controls: fit



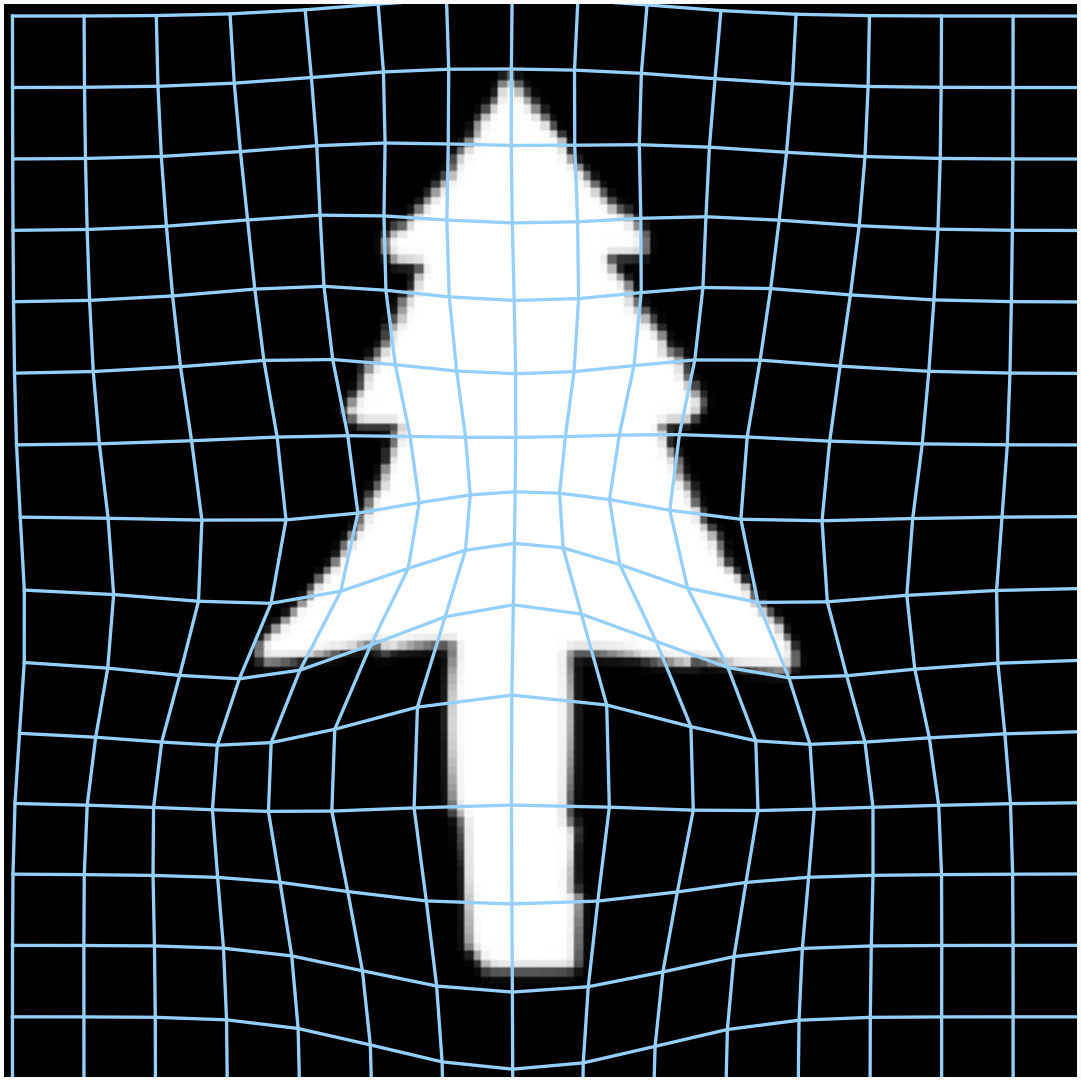
Example – Interpretable controls: fit



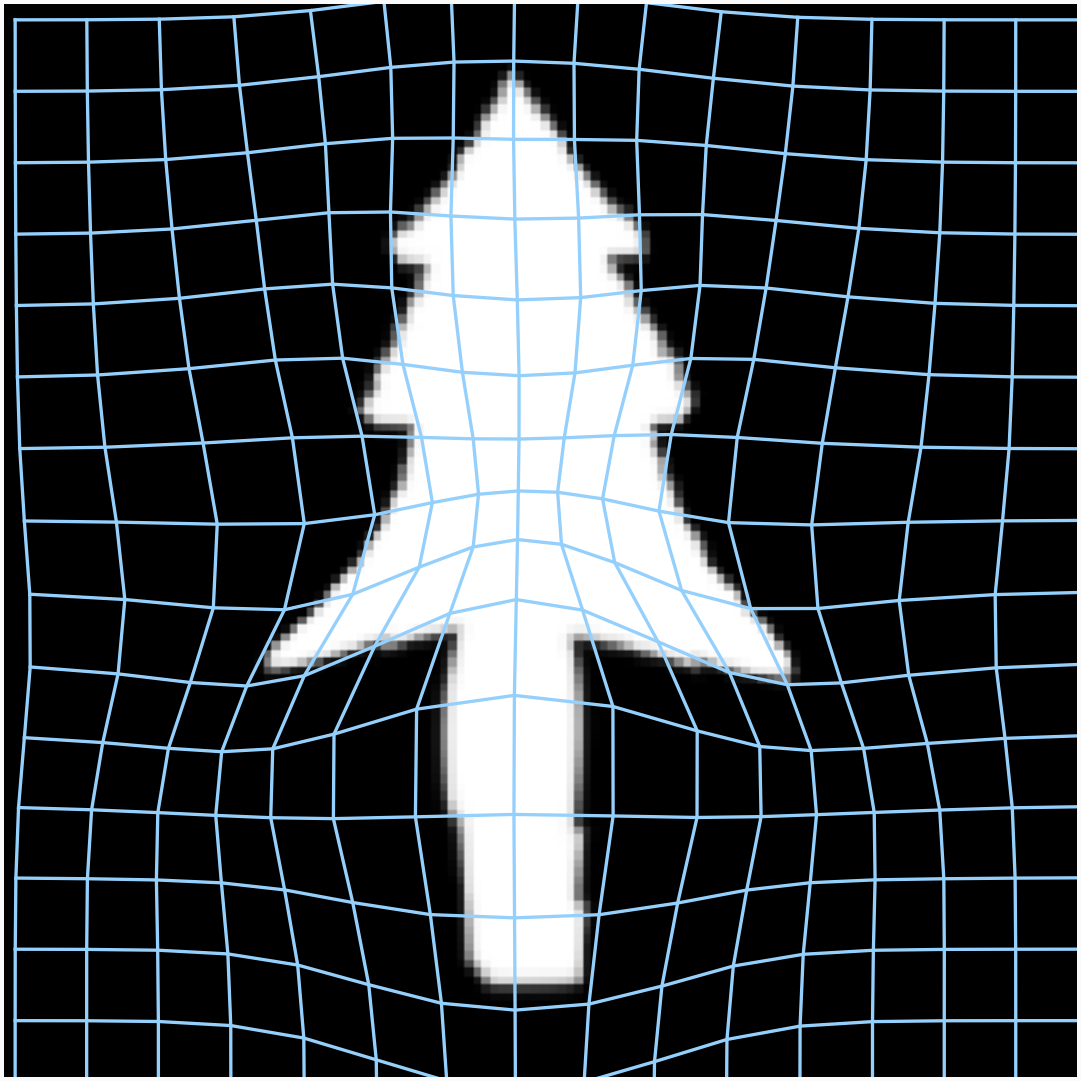
Example – Interpretable controls: fit



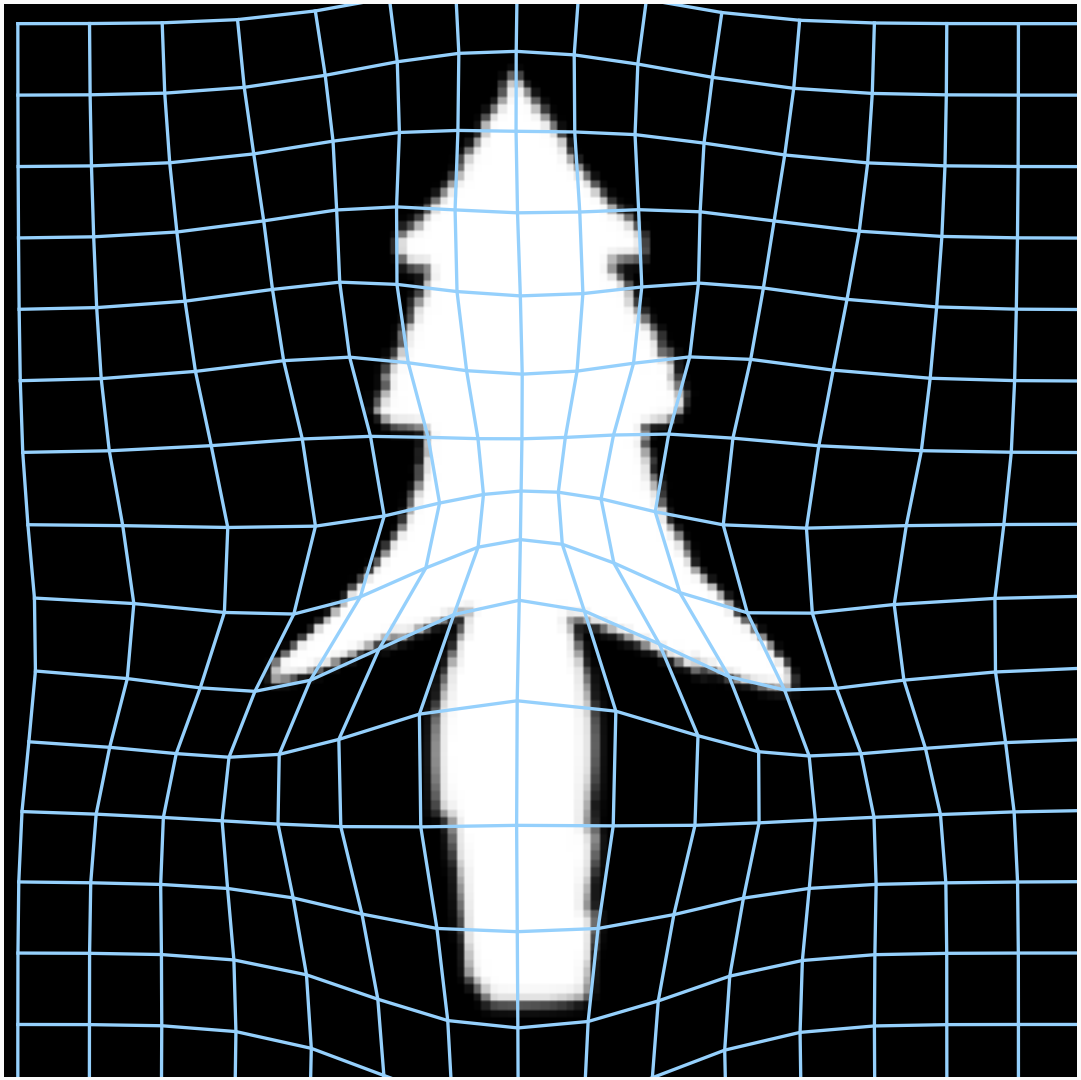
Example – Interpretable controls: fit



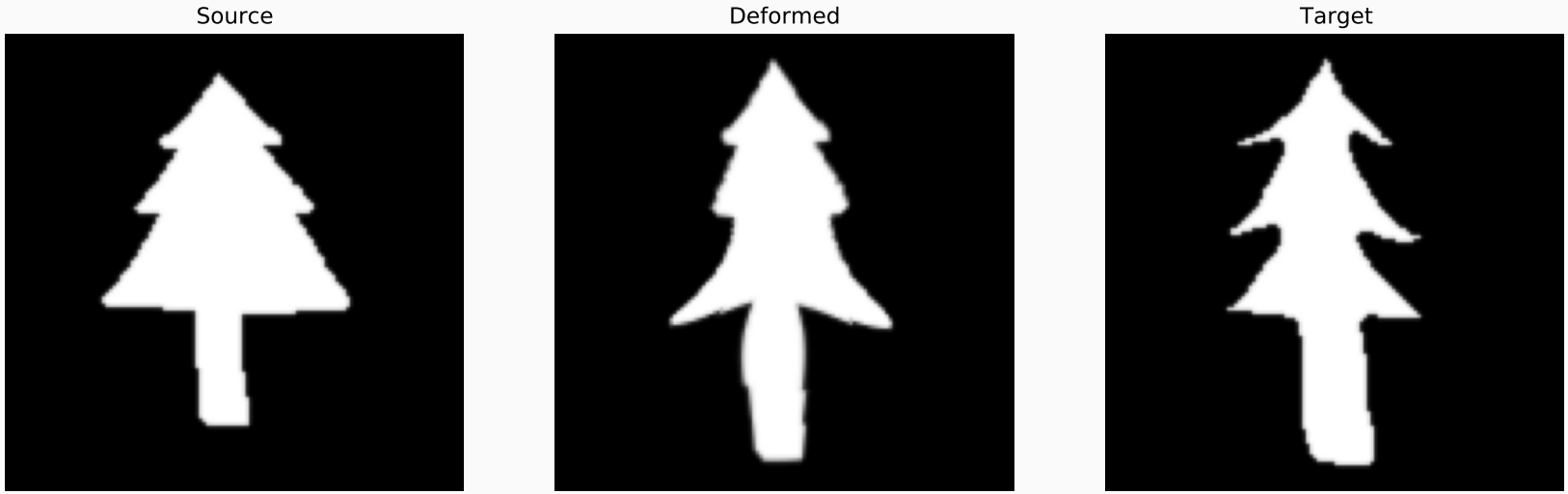
Example – Interpretable controls: fit



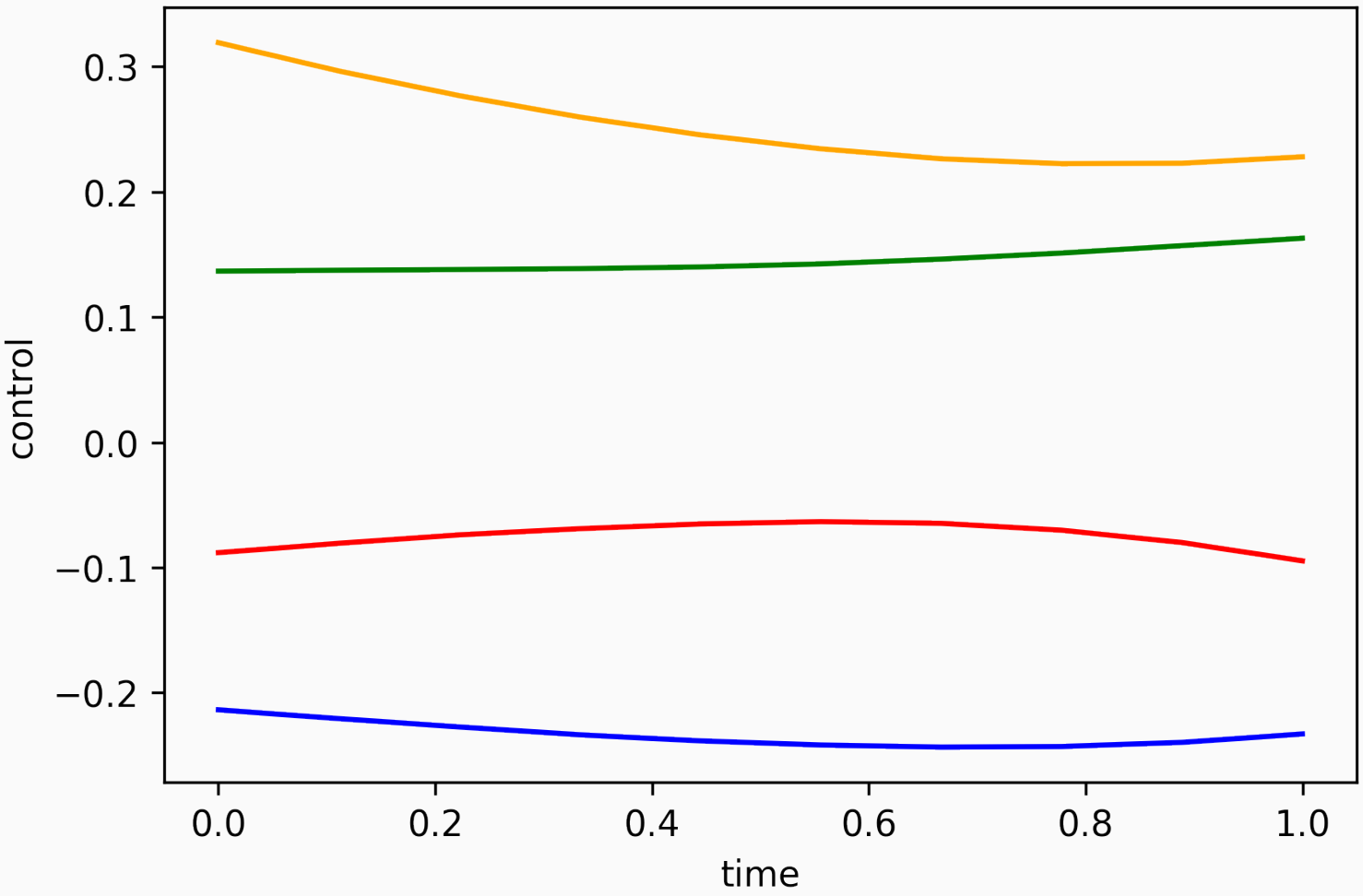
Example – Interpretable controls: fit



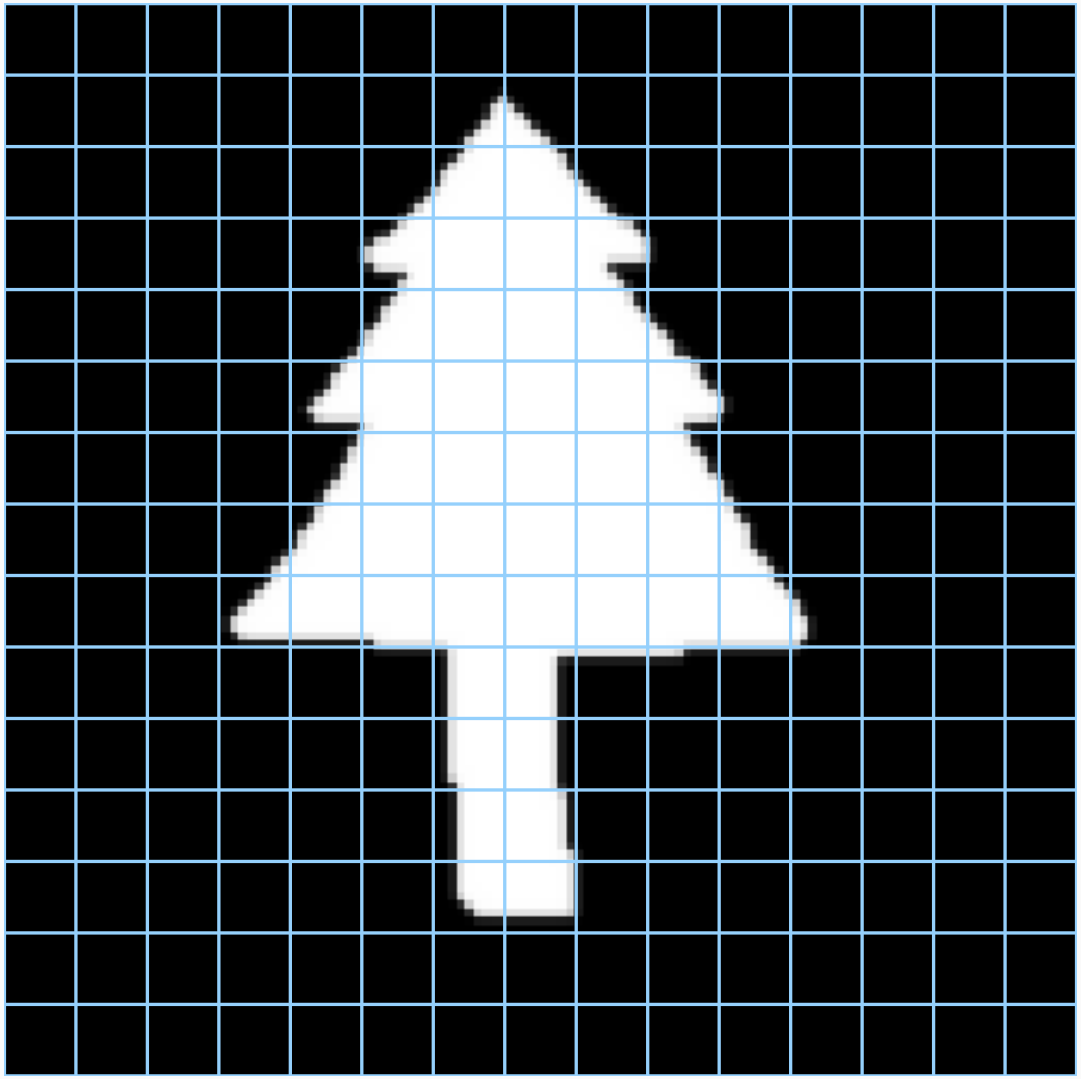
Example – Interpretable controls: fit



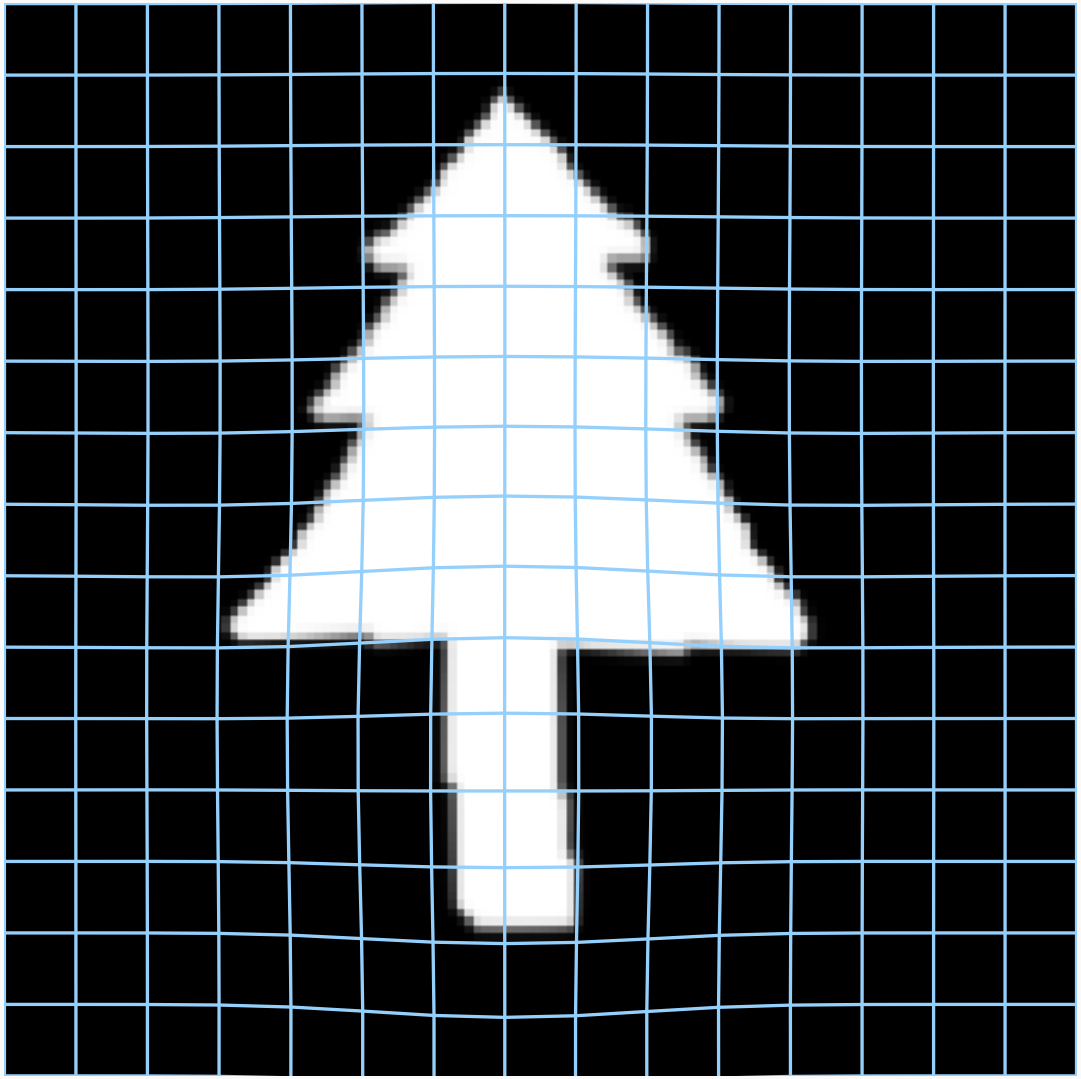
Example – Interpretable controls: control trajectory



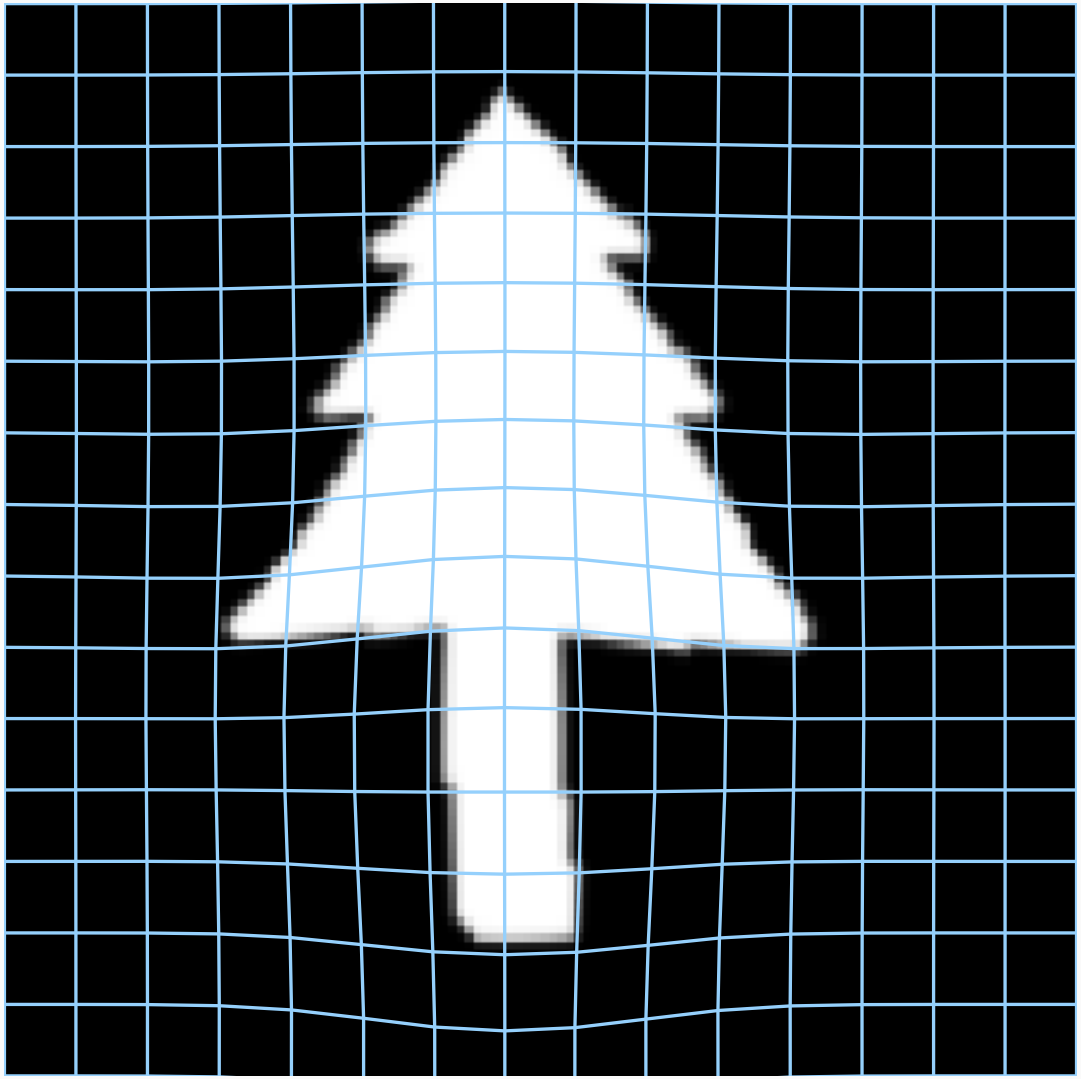
Example – Interpretable controls: trunk deformation



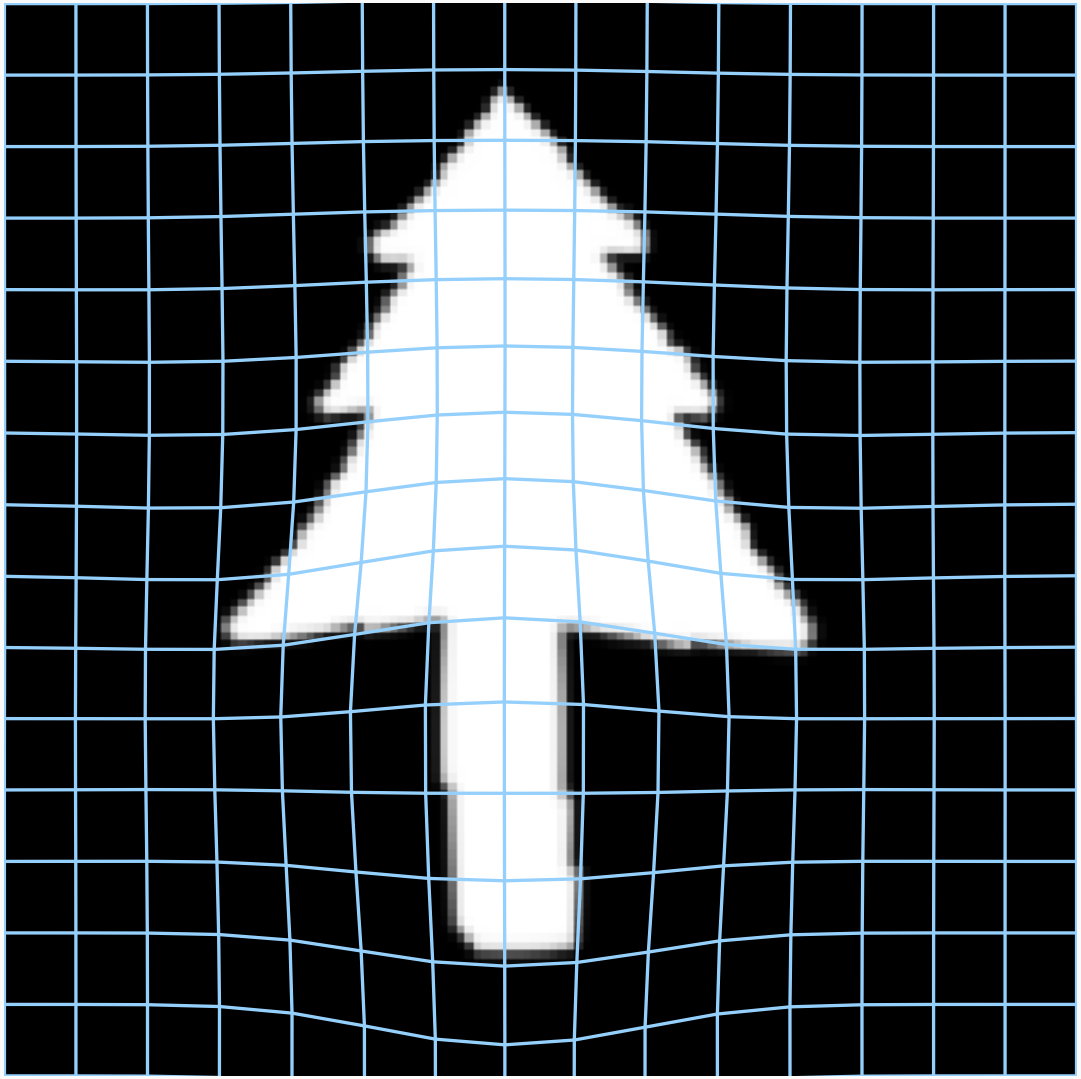
Example – Interpretable controls: trunk deformation



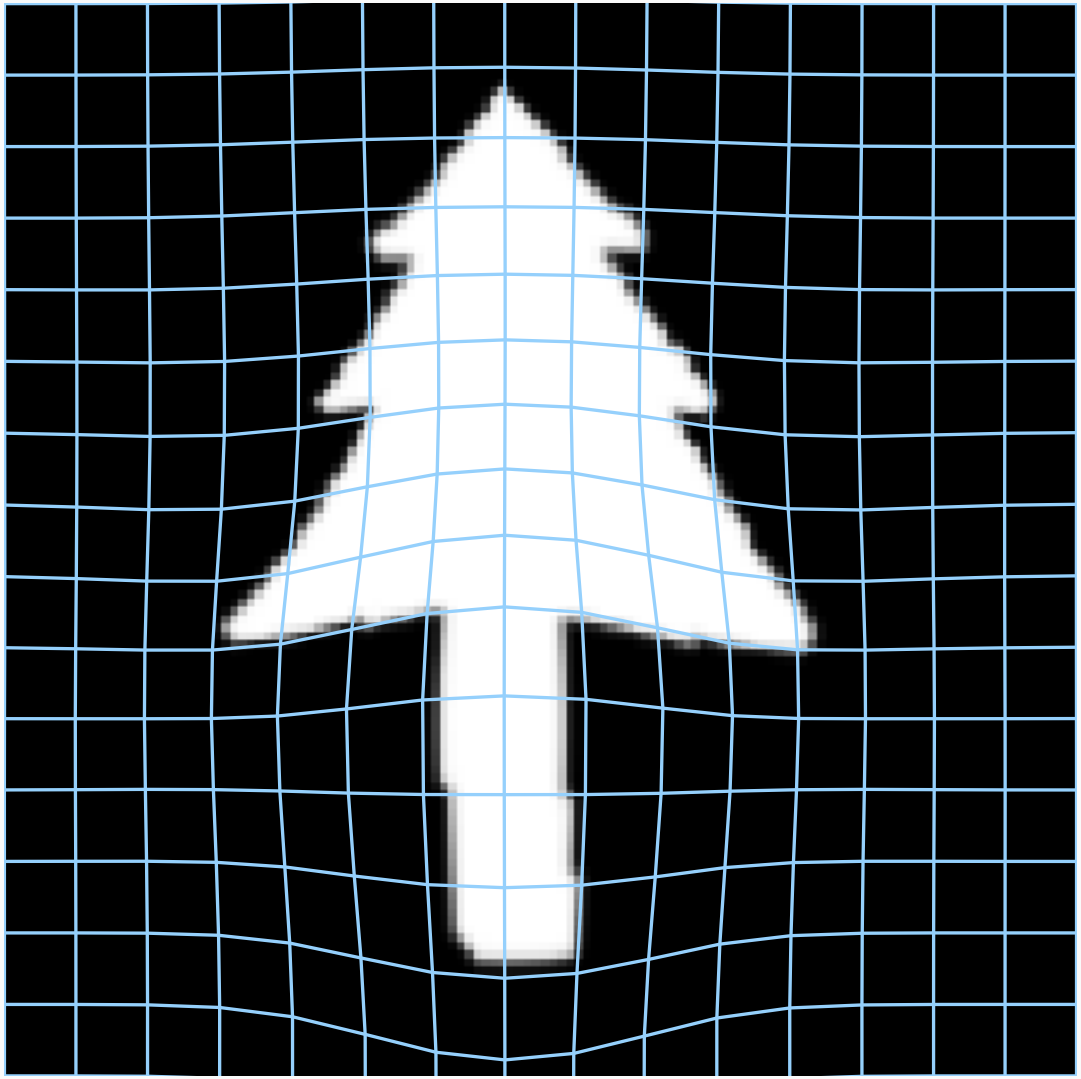
Example – Interpretable controls: trunk deformation



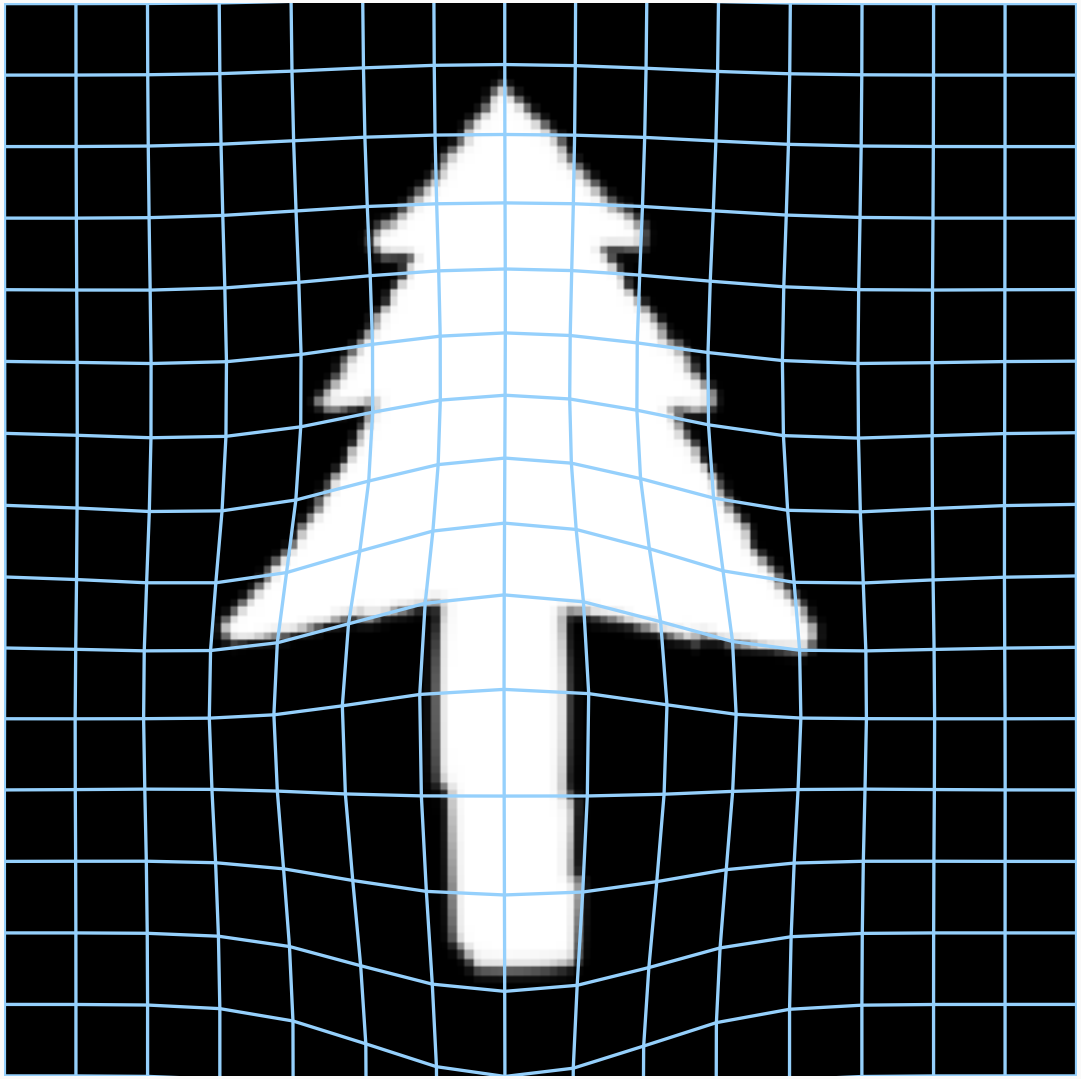
Example – Interpretable controls: trunk deformation



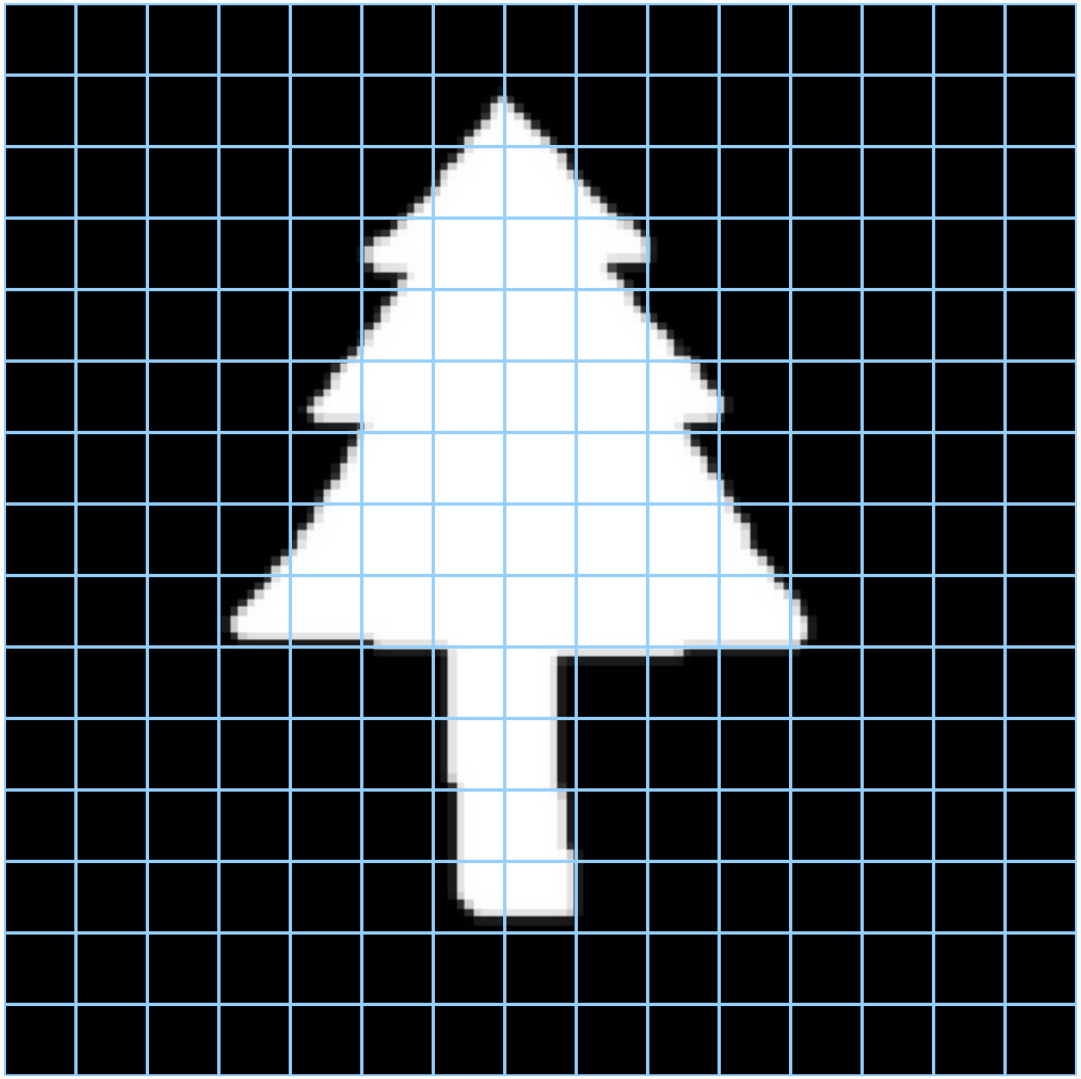
Example – Interpretable controls: trunk deformation



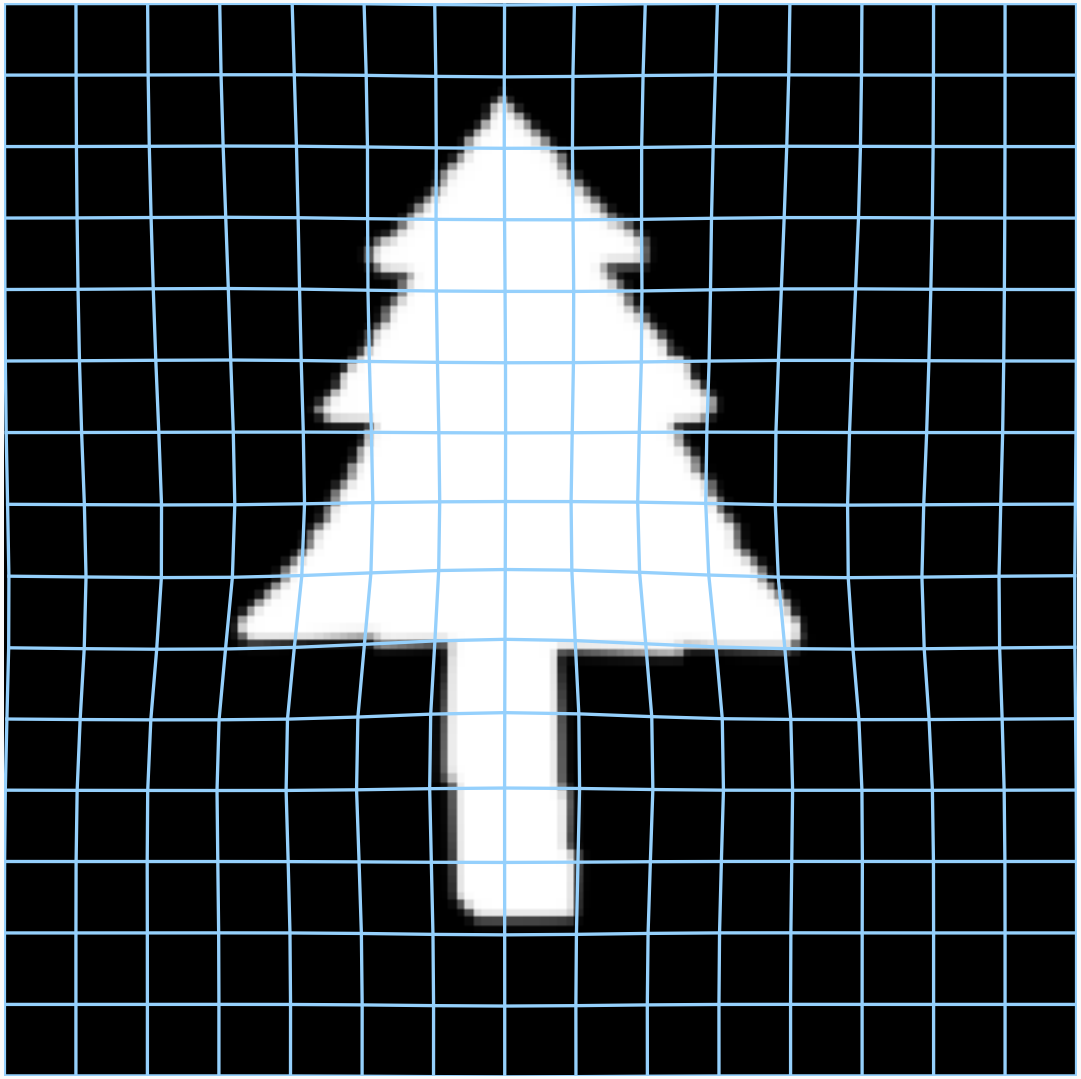
Example – Interpretable controls: trunk deformation



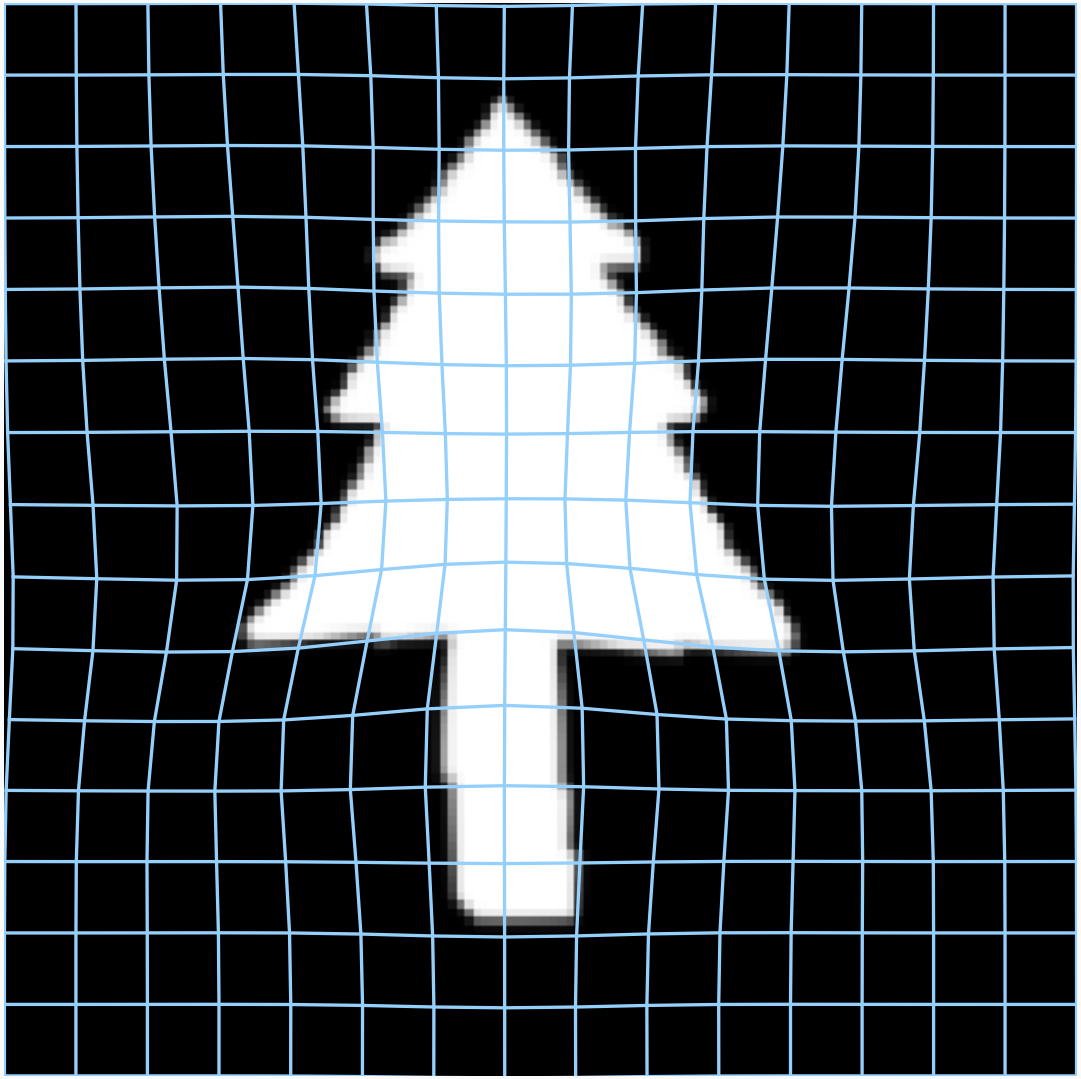
Example – Interpretable controls: crown deformation



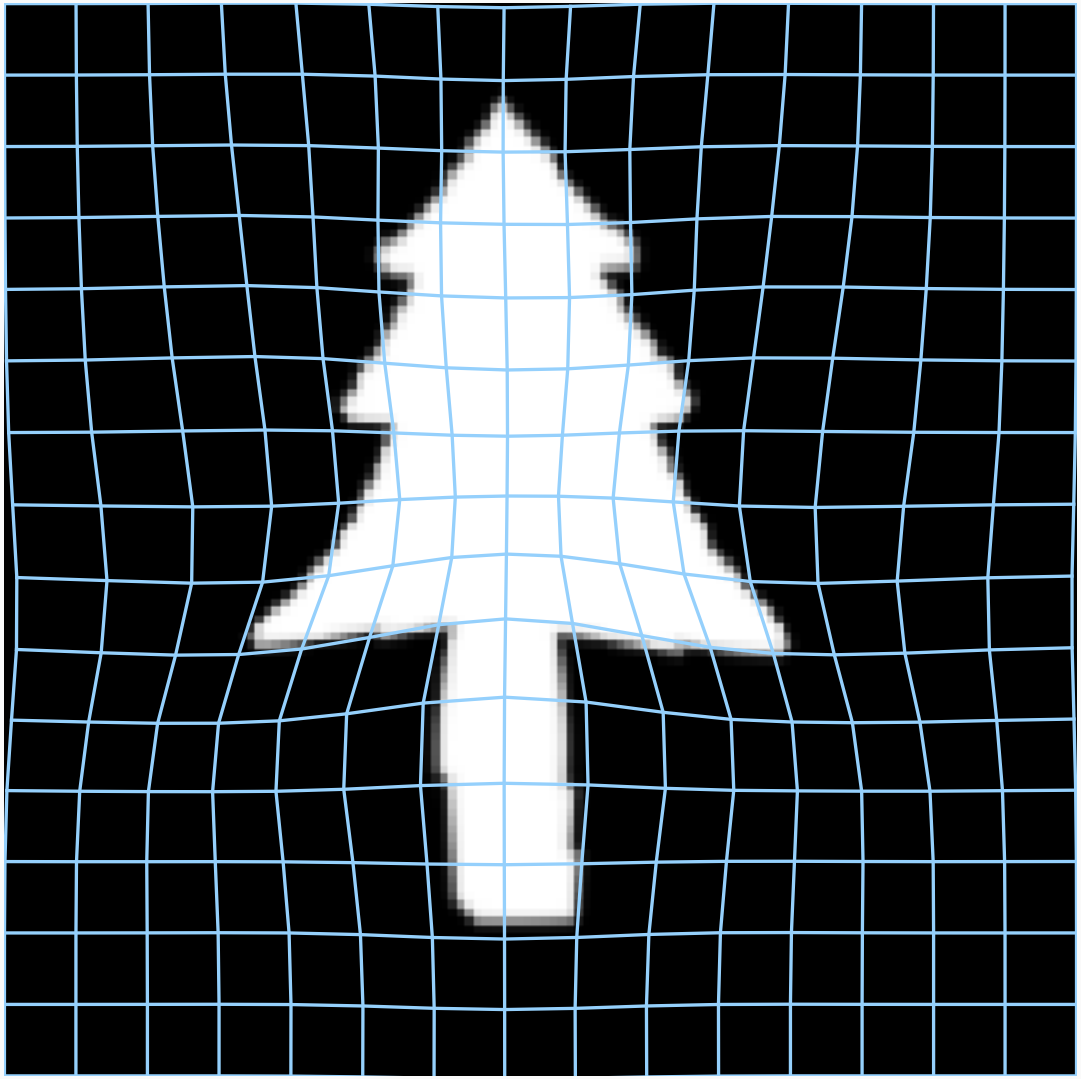
Example – Interpretable controls: crown deformation



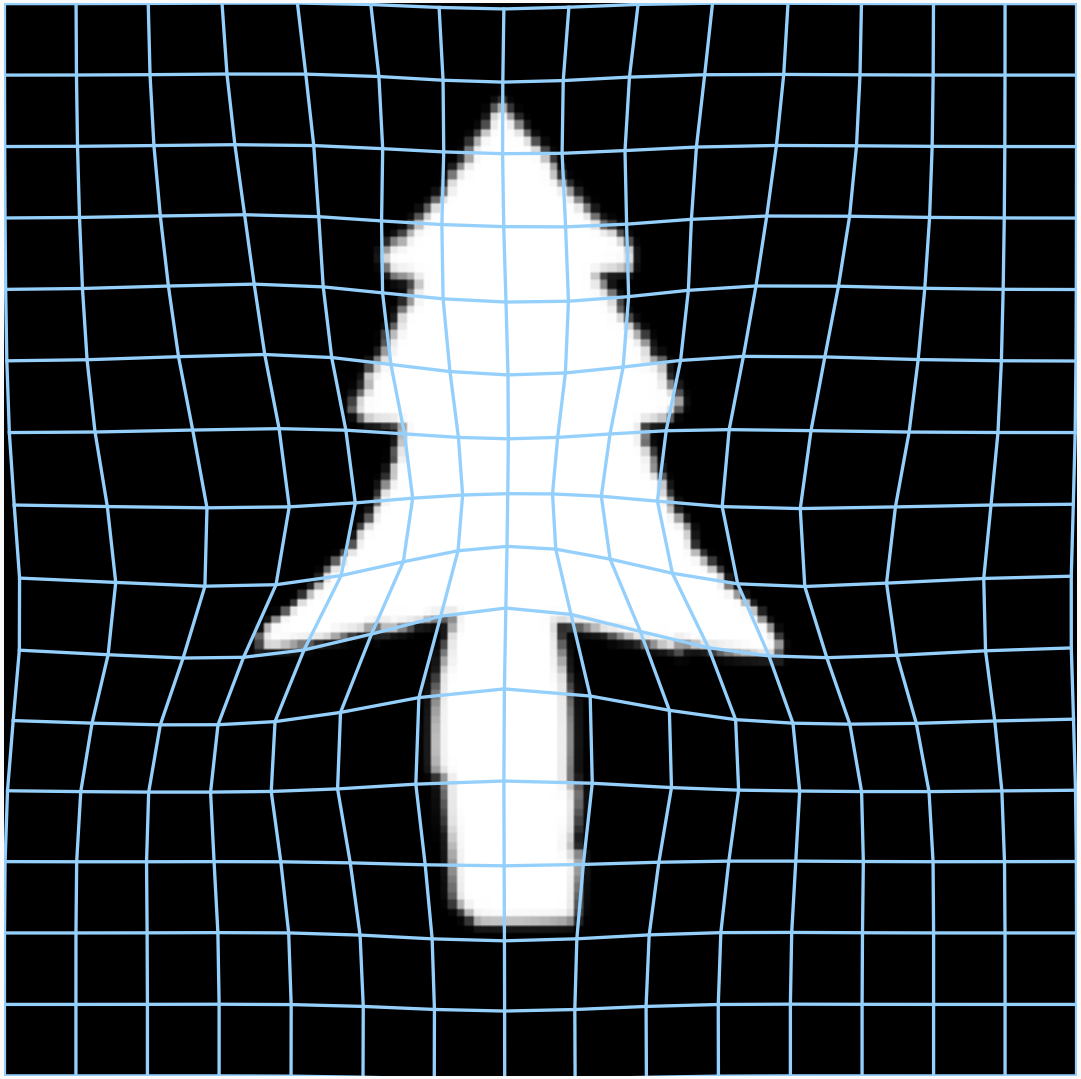
Example – Interpretable controls: crown deformation



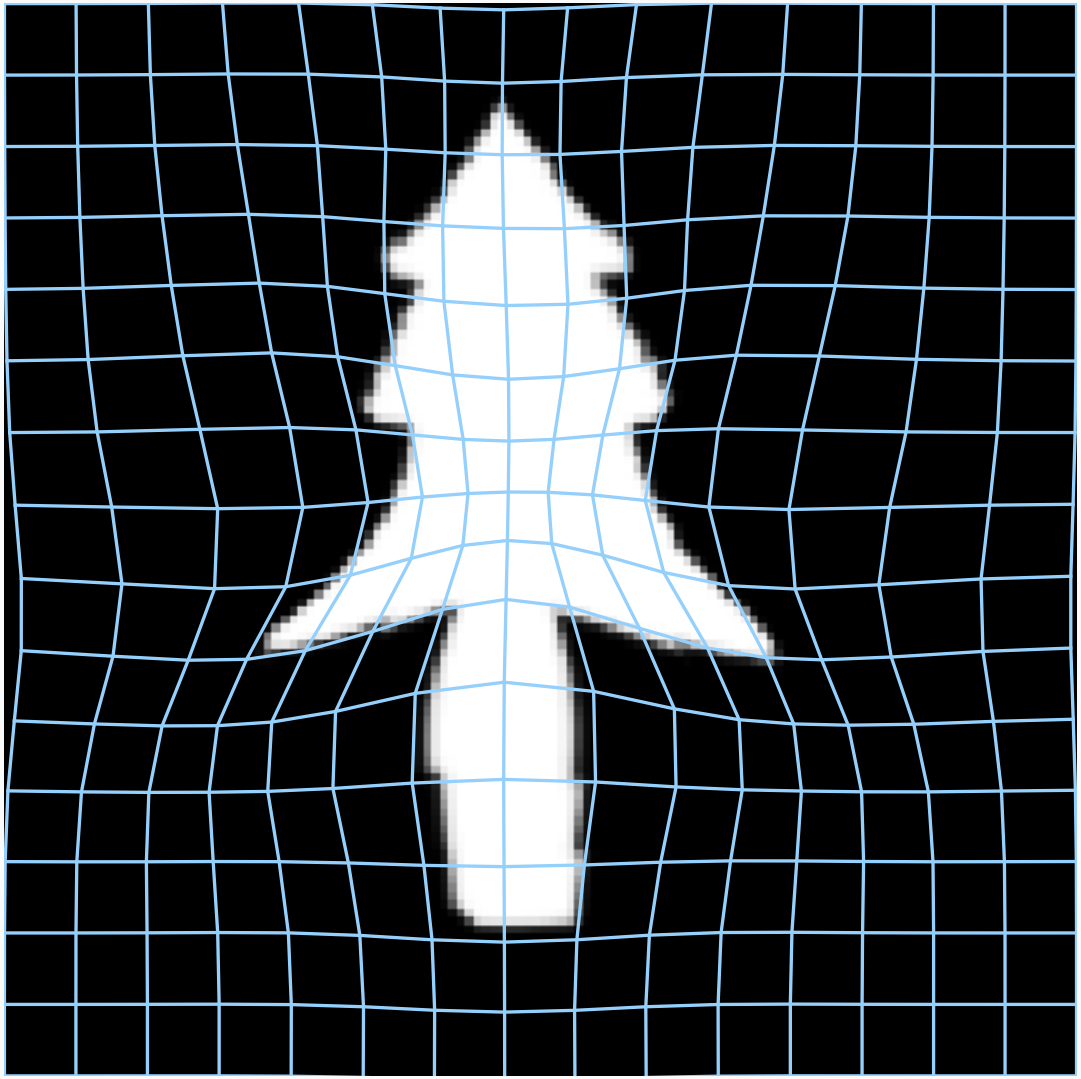
Example – Interpretable controls: crown deformation



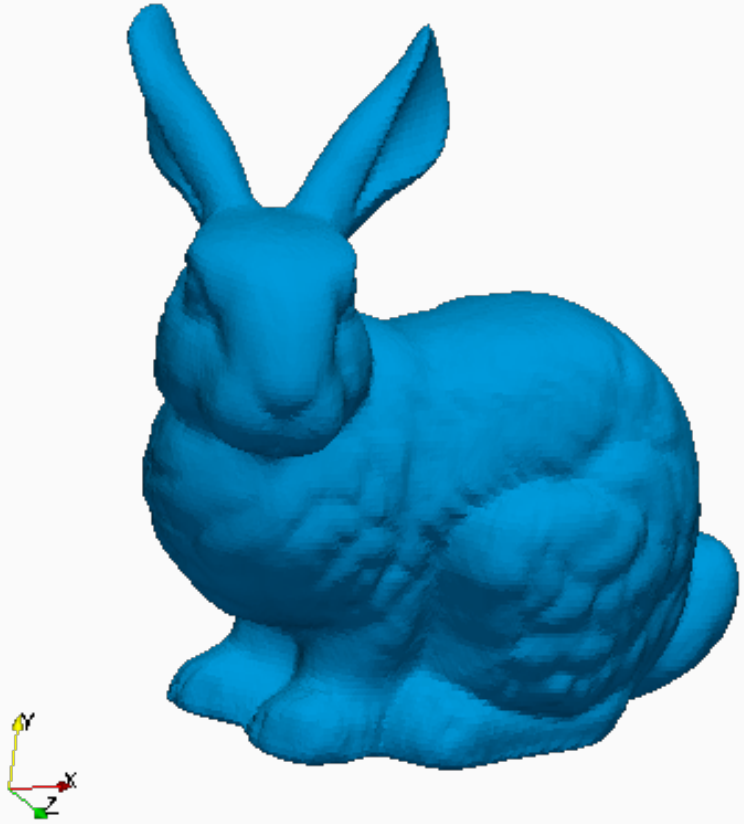
Example – Interpretable controls: crown deformation



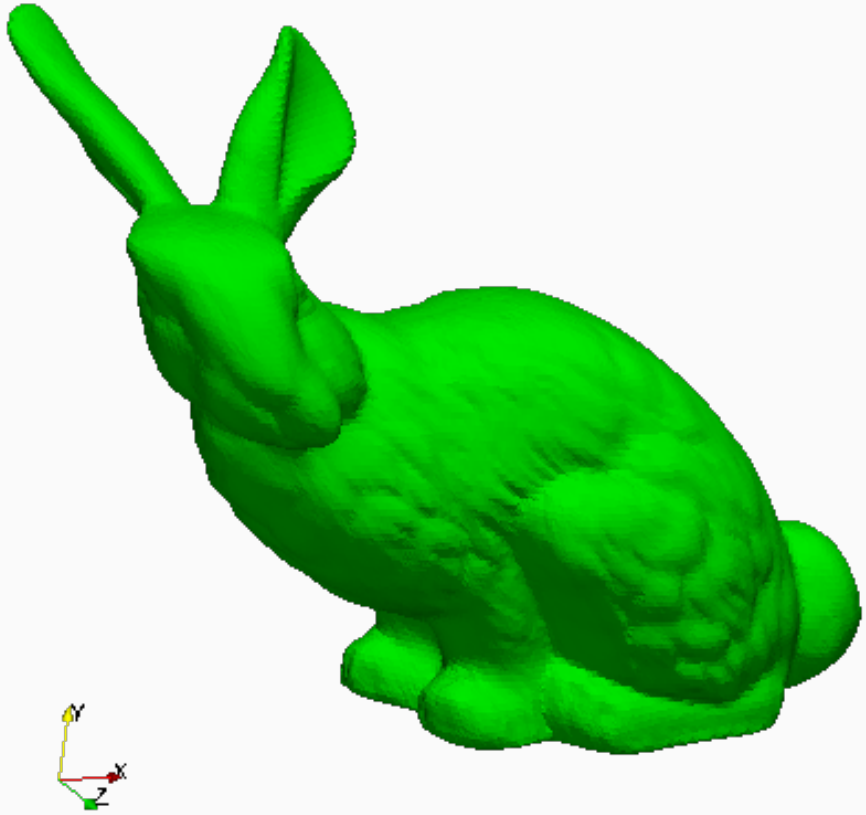
Example – Interpretable controls: crown deformation



Example – Fitting local frames in 3D: data

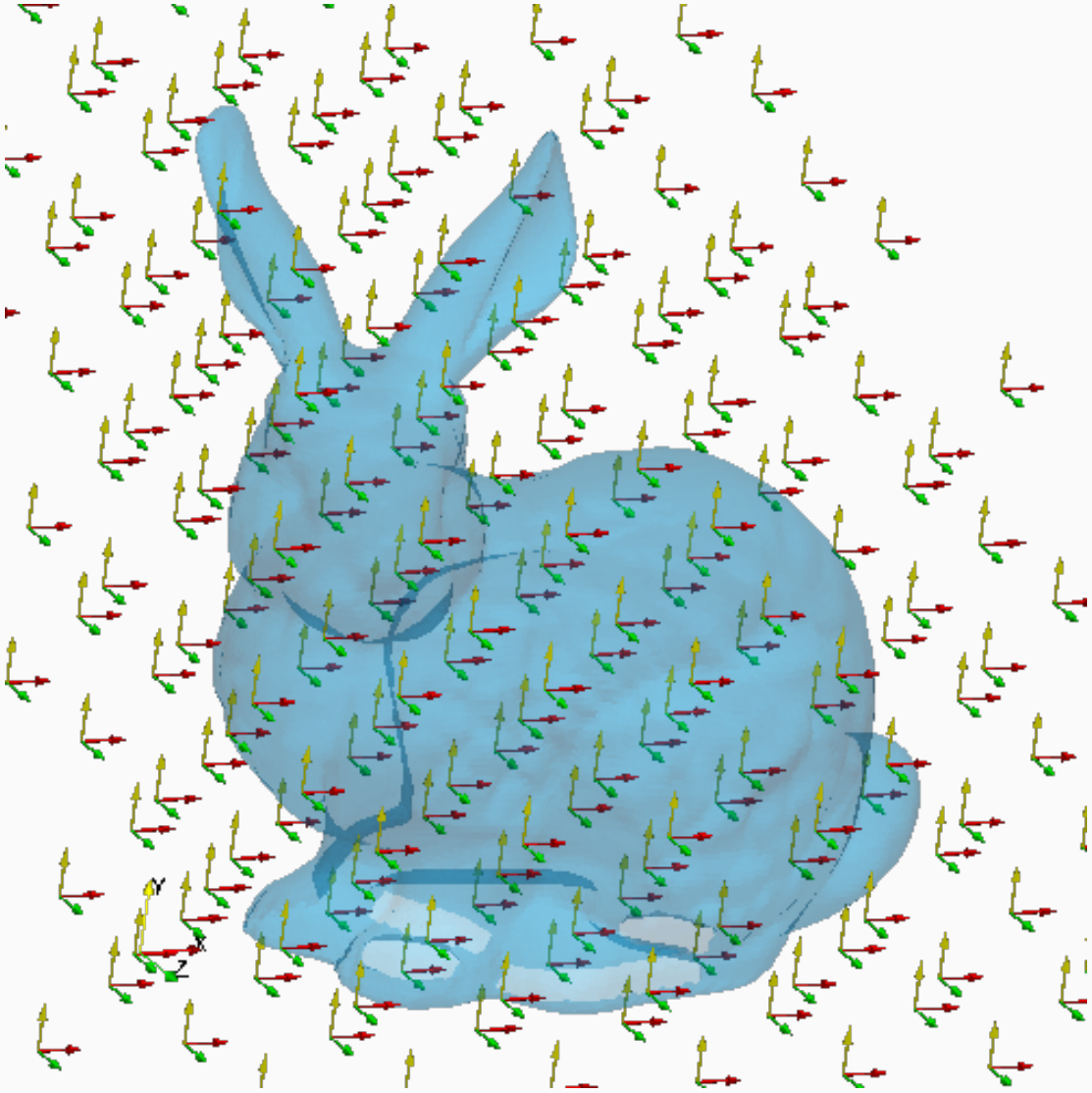


Source



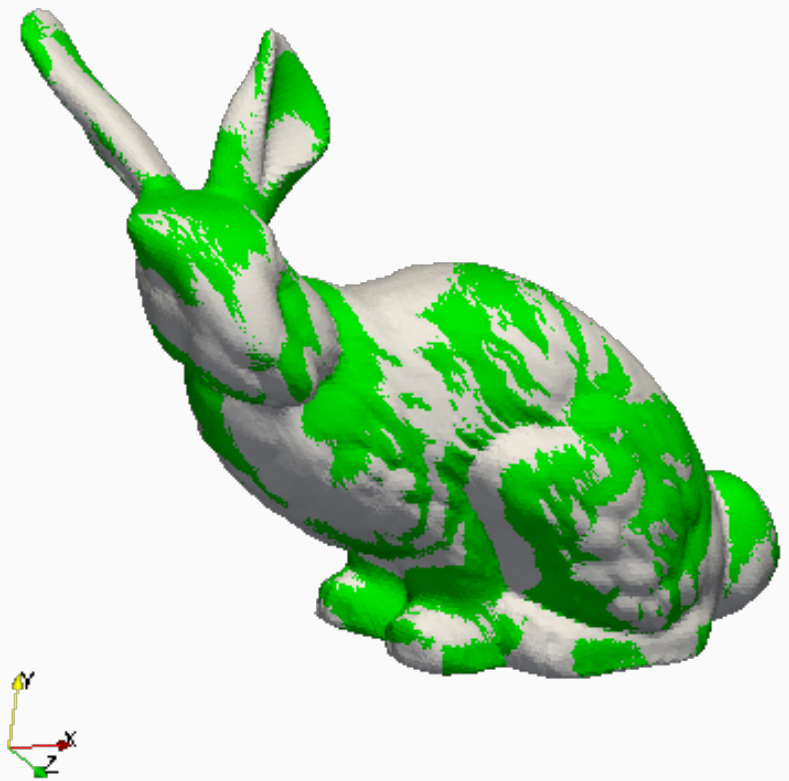
Target

Example – Fitting local frames in 3D: initial local frames



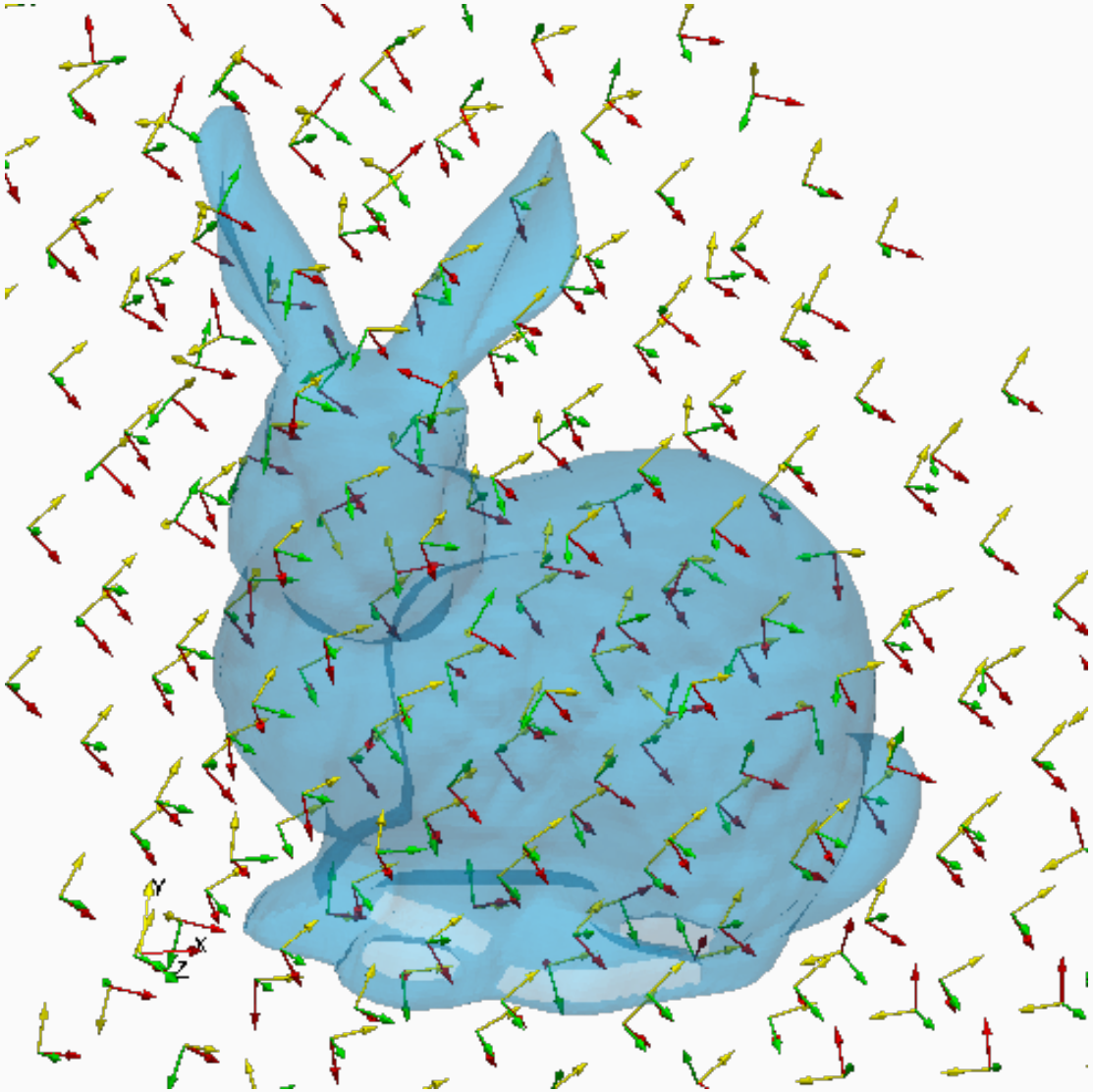
Initial local frames

Example – Fitting local frames in 3D: fit



Deformed source fitting the target

Example – Fitting local frames in 3D: optimized local frames



Fitted local frames

Future goals

Experimental features

- Atlas
- Template fitting

Future goals

Experimental features

- Atlas
- Template fitting

Technical difficulties

- Stable ODE solver (symplectic solver)
- Adapted optimisation schemes

Future goals

Experimental features

- Atlas
- Template fitting

Technical difficulties

- Stable ODE solver (symplectic solver)
- Adapted optimisation schemes

More real world examples

- Registration with more than 2 time points
- Collaborations with potential users

Future goals

Experimental features

- Atlas
- Template fitting

Technical difficulties

- Stable ODE solver (symplectic solver)
- Adapted optimisation schemes

More real world examples

- Registration with more than 2 time points
- Collaborations with potential users

CLI and GUI

Thank you

Paper, documentation, code available on the github page:
`https://github.com/imodal`

Backup: Implemented deformation modules

- Local translation
- Local constrained translation
    - Local scaling
    - Local rotation
- Local oriented translation
- Global translation
- Silent
- Silent grid
- Linear (unstable)
- Implicit modules
    - Order 0 (local translation with regularization)
    - Order 1 (elastic constraints)