A LOCOMOTION TASKS

A.1 MOVEMENT SPEED

In this task, the goal of the robot is to move as fast as possible along a pre-defined direction. We compute the average velocity of all existing (non-zero-mass) particles on robot body and project it to the target direction to obtain a scalar estimate.

A.2 TURNING

In this task, the robot is encouraged to turn as fast as possible counter-clockwise about the upward direction of robot's canonical pose. We first compute the relative position of all particles on robot body with respect to its center of mass. We then compute the cross product of the upward direction followed by unit-vector normalization to obtain tangential directions for all particle while turning. Finally, we compute velocity projection of every particle along the tangent and return the average as the measure of turning performance.

A.3 VELOCITY TRACKING

In this task, the robot is required to track a series of timestamped velocities. We use quintic polynomials formulated as a function of time for path parameterization since it generates smooth trajectories with easy access to different orders of derivative. A path can be fully specified with target position, velocity, and acceleration. After setting a target state, we query the first-order derivative of the polynomial at each environment time step as the target velocity. Furthermore, due to the deforming nature of soft robot, we need to estimate the heading of the robot in order compare it with the target velocity. We manually label particles corresponding to a head and a tail of the robot *a priori*. During robot motion, we extract rotation from deformation gradient of these particles and inversely transform them back to material space to compute heading direction. Velocities of all particles on robot body are then projected to the heading direction and averaged in order to compare to the target velocity. We separate the measurement of magnitude and direction as this allows different weighting of the two terms. We put more emphasis on the alignment of direction since it better indicates maneuverability.

A.4 WAYPOINT FOLLOWING

In this task, the robot needs to follow a sequence of waypoints. The waypoints are generated by the above-mentioned quintic polynomial method. We compute root mean squared error between robot center of mass and the target waypoint for each time step. Remark that well-performing velocity tracking can induce large waypoint following error but trace out similar-shaped yet different-scaled trajectories. Both serve a role as distinct aspects of evaluating path following.

B PLATFORM COMPARISON

A comprehensive comparison to existing soft robot platform is shown in Table 5.

Table 5: Comparison to existing soft robot platforms.

Platform	Simulation Method	Tasks	Design	Control	Differentiability	Multiphysical Materials
SoMoGym (Graule et al., 2022)	Rigid-link System	Mostly Manipulation		✓		
DiffAqua (Ma et al., 2021)	FEM	Swimmer	✓	✓	✓	
EvoGym (Bhatia et al., 2021)	2D Mass-spring System	Locomotion Manipulation	✓	✓		
SoftZoo (Ours)	MPM	Locomotion	✓	✓	✓	✓

Table 6: The physical phenomenons that each environment covered in our multiphysical simulation.

Environment	Elasticity	Plasticity	Fluid	Friction
Ground				High
Desert	✓	✓		
Wetland	✓	✓	Mixed	
Clay	✓	✓		
Ice				Low
Snow	✓	✓		
Shallow Water			Shallow	
Ocean			Deep	

C Continuum Mechanics Simulation

We formulate the continuum mechanics simulation in the framework of the moving least squares material point method (MLP-MPM) (Hu et al., 2018), whose governing equations are characterized by:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{f}_{\text{ext}} \tag{1}$$

$$\rho \frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}, \tag{2}$$

where ρ is the density of the material, $\mathbf{v}$ is the velocity, $\boldsymbol{\sigma}$ is the Cauchy stress of the energy, and $\mathbf{f}_{\text{ext}}$ is the external forces applied, which is the gravity in our cases. We solve these equations for an equilibrium between different materials coupled in our environments. We will not dive further into continuum mechanics and point the interested reader to Gonzalez & Stuart (2008) for more details. In terms of the implementation of the differentiable physics-based simulation, we massively use DiffTaichi (Hu et al., 2019a) as the backbone.

D Multiphysical Materials

For result validation and visual entertainment, we present a diverse set of environments spanned by different material setups and tasks. Here we illustrate the materials that each environment covered in Table 6. Note that even though *Desert*, *Clay*, and *Snow* share the same composition of material types, we distinguish their elastoplasticity by imposing different parameters and models (*e.g.*, friction cone). We will release the code-level implementation of all materials for reproducibility.

E Gradient Checkpointing

Simulating environments like ocean, desert, etc requires a significant amount of particles. This poses a challenge in differentiation as the computation graph needs to be cached for backward pass, leading to considerably high memory usage. Accordingly, we implement gradient checkpointing that allows very large-scale simulation with gradient computation. Instead of caching simulation state at every single step, we only store data every N steps. When doing backward pass at $i \times N$ steps, we perform recomputation of forward pass from $(i-1) \times N$ to $i \times N$ to reconstruct the computation graph in-between for reverse-mode automatic differentiation.

F Optimization

In this section, we describe the implementation details of each optimization method.

F.1 DIFFERENTIABLE PHYSICS

Model-based gradient provides much accurate searching direction and thus considerably more efficient optimization. However, gradient information is susceptible to local optimum and often leads to bad convergence without proper initialization. Hence, we adopt a simple yet effective approach that samples 8 random seeds, performs optimization with differentiable physics for all runs, and picks the best result. For the large-scale benchmark with biologically-inspired design (Table 1), we use learning rate 0.1 and training iterations 30. For all control-only and design-only optimization, we use learning rate 0.01 and training iterations 100. For co-design, we use learning rate 0.01 for both control and design with training iterations 250. We use Adam as the optimizer.

F.2 REINFORCEMENT LEARNING

RL is only used in control optimization. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) with the following hyperparameters: number of timesteps 10^5, buffer size 2048, batch size 32, GAE coefficient 0.95, discounting factor 0.98, number of epochs 20, entropy coefficient 0.001, learning rate 0.0001, clip range 0.2. We use the same controller parameterization as all other experiments throughout the paper.

F.3 EVOLUTION STRATEGY

We implement a fully-ES-based method as a co-design baseline. The genome fitness function is set as the episode reward of the environment. We pose a constraint on connected component of robot body. For CPPN, we use a set of activation functions including *sigmoid*, *tanh*, *sin*, *gaussian*, *selu*, *abs*, *log*, *exp*. The inputs of CPPN include x, y, z coordinates along with distance along xy, xz, yz planes and radius from the body center. We use HyperNEAT (Stanley et al., 2009) for design optimization and CMA-ES (Hansen et al., 2003) for control optimization with initial standard deviation as 0.1. We don't use an inner-outer-loop scheme for co-design. Instead, HyperNEAT and CMA-ES share the same set of population with population size as 10. We run ES for 100 generations for the co-design baseline.

G CONTROLLER PARAMETERIZATION

Locomotion often exhibits cyclic motion and thus control optimization can significantly benefit from considering periodic functions in controller parameterization. Specifically, we use a set of sine functions with different frequency and phase (offset) as bases. The controller is hence parameterized with a set of weights on these bases along with bias terms. We use 4 different phases with frequency 20 and 80 rad/s throughout the paper. While all experiments presented are not confined to using this sinewave basis controller, we empirically found it extremely efficient to generate reasonable results.

H DESIGN SPACE REPRESENTATION

In this section, we provide more implementation details of design space representations.

H.1 PARTICLE-BASED REPRESENTATION.

Given a base particle set, we instantiate two trainable scalars followed by sigmoid for geometry and stiffness, and a K-dimensional vector followed by softmax for muscle placement with a fixed muscle direction along the canonical heading direction.

H.2 VOXEL-BASED REPRESENTATION.

We voxelize the given base particle set to obtain a voxel grid and follows similar modelling technique to particle-based representation in voxel level.

H.3 IMPLICIT FUNCTION

We extend the idea of OccupancyNet (Mescheder et al., 2019) to predicting robot geometry, stiffness, and muscle placement. It is modeled by a multi-layer perceptron (MLP) with 2 layers and 32 dimensions for each. We use *tanh* activation. The MLP takes in x, y, z coordinates, distance along xy, xz, yz planes and radius from the body center. The network outputs occupancy as geometry using sigmoid, stiffness multiplier using sigmoid, and a K-dimensional vector using softmax for muscle placement.

H.4 DIFF-CPPN

Diff-CPPN is a differentiable version of Compositional Pattern Producing Networks (CPPN) (Stanley, 2007), following similar concept in (Fernando et al., 2016). CPPN is a graphical model composed of a set of activation functions with interesting geometric properties (e.g., sine, tanh) that takes in particle or voxel coordinates and output occupancy or other properties. It is originally designed to be optimized with varying graph topologies. We use a meta graph to allow gradient flow and mimic the augmenting topolgies process in NEAT yet in a differentiable manner. The model takes in x, y, z coordinates, distance along xy, xz, yz planes and radius from the body center, and outputs occupancy as geometry using sigmoid, stiffness multiplier using sigmoid, and a K-dimensional vector using softmax for muscle placement. We use *sin* and *sigmoid* activation functions with 3 hidden layers and 20 graph nodes in each layer.

H.5 SDF-LERP

Given a base particle set (i.e., a point cloud representation that span the robot design workspace), we compute SDF of every particle for the shape of all design primitives. The shape of the robot design is then determined by a set of coefficients weighting the SDFs from design primitives. In other words, the trainable parameters for robot geometry only construct a N-dimensional vector, where N is number of design primitives. We then compute weighted sum of the SDF bases and extract robot body with the final SDF smaller or equal to zero. We use a low-temperature sigmoid in implementation to keep gradient flow. For stiffness, we can directly perform linear interpolation. For muscle group membership, we use linear interpolation upon the one-hot vectors from design primitives and effectively realize a soft muscle group assignment. For muscle direction, we adopt interpolation designed for rotation matrices (Brégier, 2021).

H.6 WASSERSTEIN BARYCENTER

This method also uses design primitives. First, it adopts the same approach for stiffness and muscle placement as *SDF-Lerp*. We use a fixed muscle direction along the canonical heading direction. The major difference is the way to represent robot geometry. Following (Ma et al., 2021), we define a probability simplex (i.e., a set of coefficients with length as the number of design primitives) that serves as a weighting in the sense of Wasserstein distance among different shapes. It better preserves the volume from the shape of design primitives. We refer the reader to the original paper for more details.

I VISUALIZATION OF BIOLOGICALLY-INSPIRED DESIGN

In this section, we demonstrate the biologically-inspired designs used in this paper. In Figure 7, we show the four animals used in the main paper, including *Baby Seal*, *Caterpillar*, *Fish*, and *Panda*. In Figure 8, we show the set of design primitives used in *SDF-Lerp* and *Wasserstein Barycenter*.

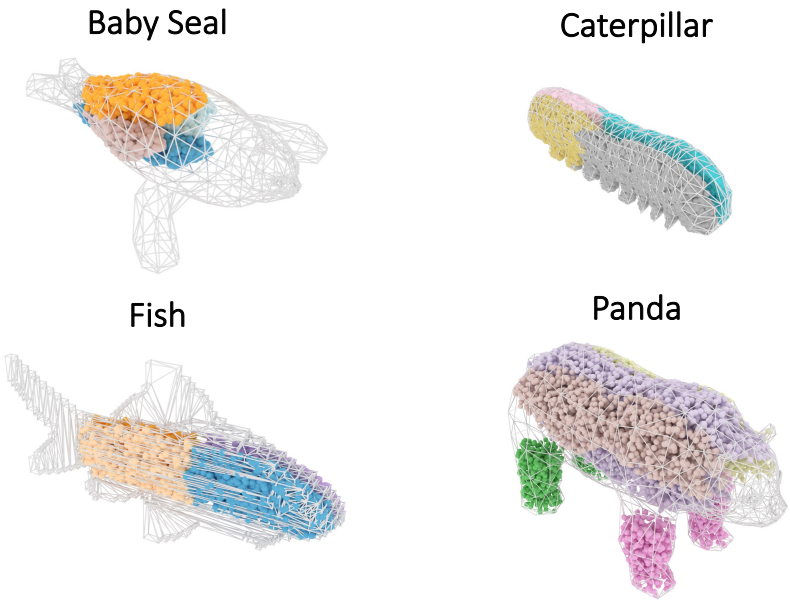

Figure 7: Visualization of animals for biologically-inspired design.

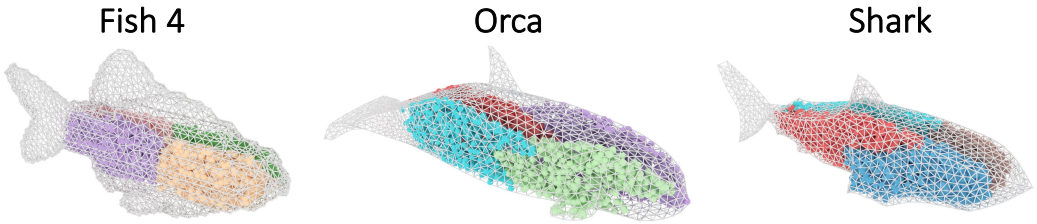

Figure 8: Visualization of fish-like design primitives.