# Using machine learning to augment artistic creativity

There has been a long running debate about whether machines can be truly creative.  While this debate probably doesn’t have an end, there is no doubt that the artistic processes can make use of technology, whether through better brushes for painting, or 3D animation software. Recently there has been growing interest in how machine learning in general, and neural networks in particular could be used to generate art.

In this article I will be exploring the general question of how to use a neural network to augment artistic expression and how that is different from previous approaches. This will involve a discussion of how this changes the role of the artist. More specifically I will be presenting a method to develop a generative model that tries to capture a set of relational constrains from some artistic samples and then uses then to generate a space of possibilities for artistic expression. This can be though of as a kind of gamification of artistic expression through the combination of machine learning and artistic sampling and exploration.

A simple example that I will use for illustration, is a 3D model of an opening flower petal for different morphological shapes. The model uses an encode/decode pattern common in many generative architectures, but will introduce some additional ideas draw from information theory and the genomic/phenomic separation of biological evolution. While the aspirations of this article are somewhat pretentious in proportion to the simple example used to illustrate its principles, hopefully it will provide a starting point for more elaborate examples.

## How machine learning has been used to generate art

Initial attempts at generating art with machine learning have taken several different forms.  One obvious approach is to automate the generation of art by training a network using existing works of art as input. Then the network can be used to generate outputs different that the training data.  This can be done for literary texts, musical sounds, visual representations of images etc. Inputs for culinary recipes, perfumes or dance can also be imagined.  The immediate challenges then become, how to represent the inputs as numbers, how to structure the algorithm and how to define an error signal and loss function the steers the training towards what would be recognized legitimate works of art.  As you can imagine, there have been many awkward as well as “interesting” initial attempts.

Some of the most publicized attempts have included:

* generating text from existing works of prose
* generating images of faces for people that do not exist
* generating music that mimics a composer’s style
* taking an existing painting or photograph and transforming it into a different visual style.

## The current role of the artist in machine learning approaches

Instead of focusing on the technical issues of these approaches, I would first like to address the role of the artist and where creativity is recognized and assigned.

With an artist and his paint brushes, this is obvious, but with many artists increasingly using a team of assistants or fabricators it is less clear.  Usually the directing artist is given the creative credit even if he was only tangentially involved in the production of the work. With “found object” art the creativity is simply in the recognition of an object as art, or the choosing of what is art without any actual production of the work.

When looking at machine learning through the lens of automating the role of the art assistant or fabricator, creativity would seem to stay with the artist.  But the allure of creating a machine that has true creativity is the holy grail of AI art. In a similar way to what Jaron Lanier described for machine learning approaches to language translation, machine learning usually uses big data to process thousands, millions or billions of input samples into an algorithm that uses some emergent process of interpolation and extrapolation to generate a result. This creates a kind of Star Trek Borg Cube that assimilates vast quantities of material and erases their source identity, transferring the assignment of creativity to an emergent black box. This can be seen as promoting a mechanical assistant until the “student becomes the master” and the artist is deprecated and replaced with an algorithm.

## Trying to support the true source of creativity

The problem with this vision is that it still depends on the real artists for its input, even if their identity and importance is obscured. The second problem is that there is no one to drive the process of creating value, which is always holistic and subjective in nature. “The distinction between first-order expression and derivative expression is lost on true believers in the hive. First-order expression is when someone presents a whole, a work that integrates its own worldview and aesthetic. It is something genuinely new in the world. Second-order expression is made of fragmentary reactions to first-order expression.” (Jaron Lanier) [<https://medium.com/@ramurrio/the-future-of-content-eecc45d46c2a>

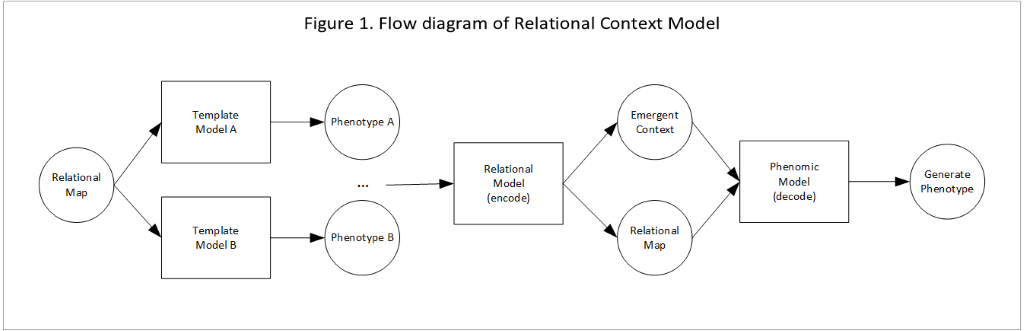
There is also another technical problem with these kinds of algorithms usually identified as the “The curse of dimensionality”.  Organic forms are made up of a large number of parameters or coordinates that all depend on each other in a complex web of many to many relationships. Changing one point requires the adjustment of a large part of the rest of the system in order to maintain its functional coherence. As a flower petal opens and expands, every point on its surface must track with every other point in its shape, to maintain a holistic pattern. In 3D phenomenal space, as the complexity of an object’s shape increases (requires more coordinates), the complexity of possible 3D coordinate combinations increases exponentially, and the percentage of those coordinate combinations that represent coherent shapes diminishes exponentially. Even more importantly, trying to extrapolate or interpolate between these values will become exponentially useless.

“In high dimensions, almost all of the data is in the outer shell. This means, among other things, that “neighborhoods” typically stretch to the outer edges of multiple dimensions, that machine learning models almost always need to extrapolate, and that few other data points will be “similar” to any given data point.” ([Aaron Lipeles](https://towardsdatascience.com/@aaron.lipeles?source=post_page-----f07c66128fe1----------------------)) <https://towardsdatascience.com/the-curse-of-dimensionality-f07c66128fe1>

The mathematical details of this are outside the scope of this article, but the ability of machine learning to scale to increasing number of features is often overlooked by starting with problems where an unlimited amount of data can be simulated, for example in games, or by using relatively simple examples, with ambiguous “artistic” results. Overcoming these problems requires reinstating an actual artist as the coherent source of input, even if this involves starting with derivative or “found” samples and then have the algorithm develop a space of new creative possibilities that is dense enough to be navigated by an artist’s subjective value, without getting lost in the vast empty space of the “curse of dimensionality”.

## Machine learning as artistic augmentation

To address these problems a Relational Context Model shown in Figure 1. will be presented. **While most current generative approaches use neural networks as a black box to distill vast quantities of sample data without the guidance of an artist, this new approach will instead try and more directly support or augment the creative expression of an active artist.** There will be three steps to this process. First human/manually generated content is captured in neural networks that will be call template models. Then two or more template models are used to train a generative model. This generative model has two parts, a “Relational Context Model” and a “Phenomenal Model”. Once the generative model has been trained, it can be used to generate new objects or phenomenal forms.



## Artist generated templates

The starting point for this example is for an artist to create a set of objects that capture the changing morphology of a shape over time in a template, analogous to the still frames of a movie. For this example, I am using 3D coordinates for a computer graphics imagery (CGI) mesh.  These artist created samples, can then be used to generate a continuous template model (neural network), capturing the morphological changes over time of an object. A different template model is created for each object type, using a common relational map as input. Figure 2 shows the 3D rendering of several morphological points from the relational map of one template using a single time index. To illustrate the process, we will use CGI mesh shapes similar to the opening of a flower petal. Each petal represents a point in time and consist of a 7 by 9 mesh of 3D (x,y,z) coordinates with a separate mesh for 7 points of time, progressing from a closed bloom to a fully open shape.

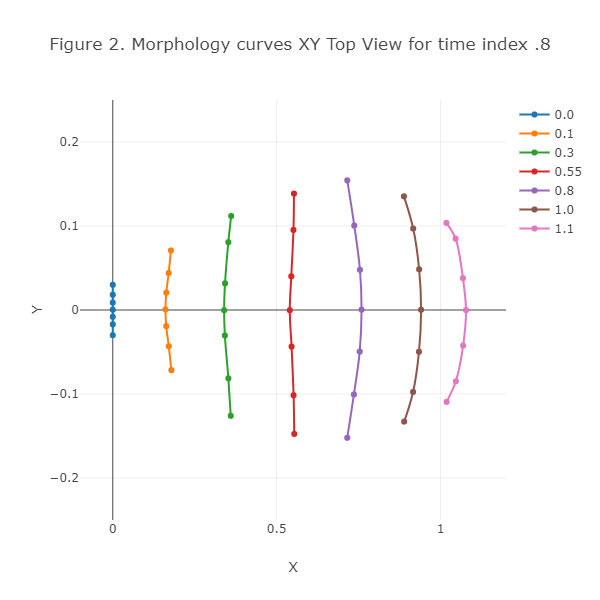
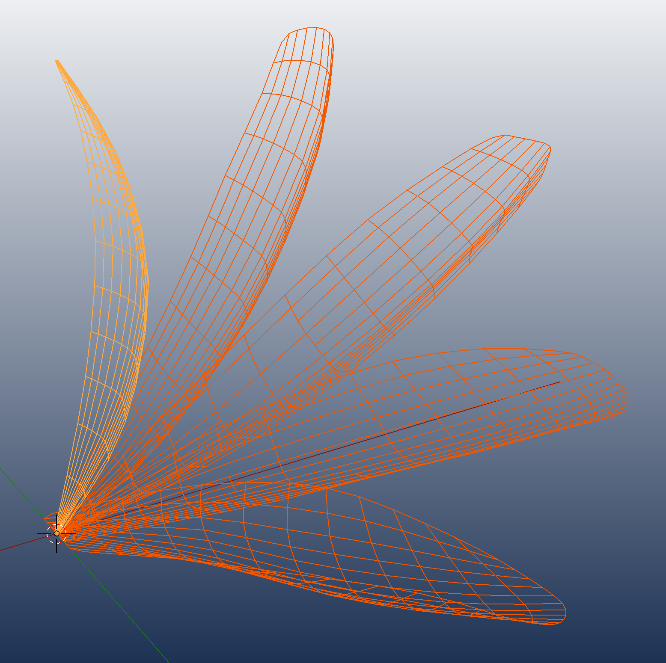
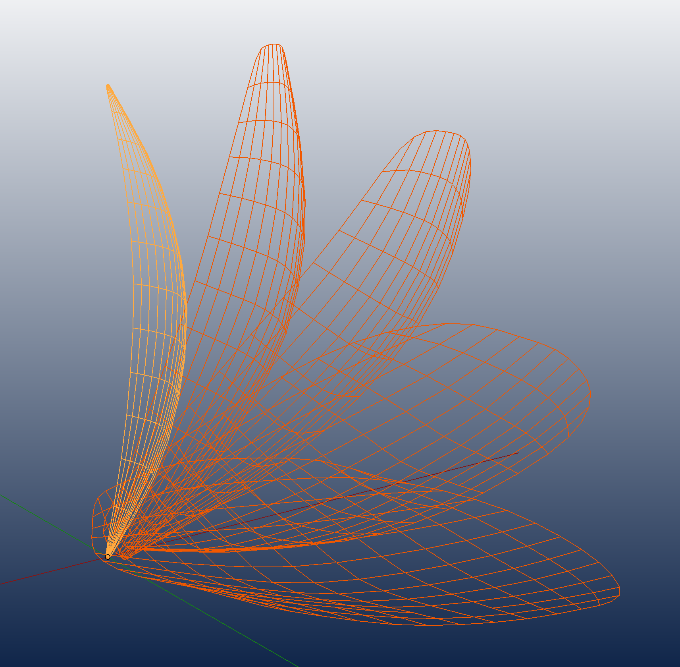


Figure 3. Shows 3D mesh for two template shapes with subsurf smoothing.

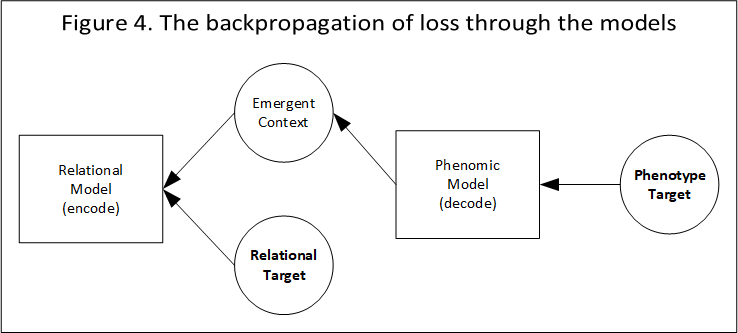


In our simple example the inputs to the template model are a 2D relational map consisting of two index values. One is an index of the time step and the other is the index of the morphology step. The output of a template models are the mesh coordinates for the object at a specific morphological step and point in time. A morphological step references the 3D coordinates of a “row” of points in a grid of points organized in a row and column like structure (Figure 2).  This provides a mapping between the temporal/relational structures of an object that is independent of it shape, to the actual coordinates of the objects in physical space. This separation between the relational constraints that give organic form its holistic character and the rendering those relationships into a specific form, will be critical to the generative process. Once these template models have been trained, they can be used to generate an unlimited amount of continuous training data for our generative model.

## Creating an inverse relational map

The second step of the process, trains the Relational Model using training data from multiple template models, creating an inverse transform of the generated template model data into a single relational model. This Relational (Rel) model takes phenomenal space shape coordinates and outputs the same relational space coordinates uses as input to the template models. **This allows us to take a shape in 3D space and translate it into locations on a “relational map” in time/morphology space.** These relational map coordinates are then transformed back into 3D phenomenal space coordinates by a Phenomic (Ph) model. The problem is that all the templates use the same relational map, making its inverse into a single model ambiguous. This problem is usually solved by “tagging” or one-hot-coding the data with an index to disambiguate the data. Otherwise the generative Ph model will average together the different output values for the same relational index.

However instead of explicitly setting a target template index, the model will let a context value emerge from the training process. This allows one or more emergent “context dimensions” to be output from the Rel model to provide values to the Ph model necessary to preserve the Ph template information and maintain its reversibility. During training the backpropagation of error from the Ph output targets will move back through the Ph model and then back through the weights in the Rel model, as shown in Figure 4.



This will allow the Relational model to differentiate the context values necessary to make the relational to phenomenal transformation reversible back to the original input shape**.  In the example a single “relational contextual” value will emerge from the training process for each morphological step, allowing the Phenomic model to differentiate what 3D shape to generate.**

If templates share the same 3D shape, for time and morphological indexes, then there is nothing to distinguish, and the context values will be the same, producing the same shape, from the same relational constraints.  As the 3D shapes diverge, the templates “trace” in context space will also diverge along different manifolds. Context points between the manifolds will interpolate between the different template shapes for a given relational set of constraints. If you project from the context average outward beyond the template context trace boundaries, you would expect to see that emergence of new shapes, diverging from the relational constraints of the templates. **If these emergent shapes are deemed to have artistic value, they could be used to identify new template models in an evolutionary bootstrap ecology of increased artistic value, diversity and complexity**.

To see how the context values change across the relational map, we can input the reference Ph values used to train the templates into the Relational Model and observe the resulting context outputs.

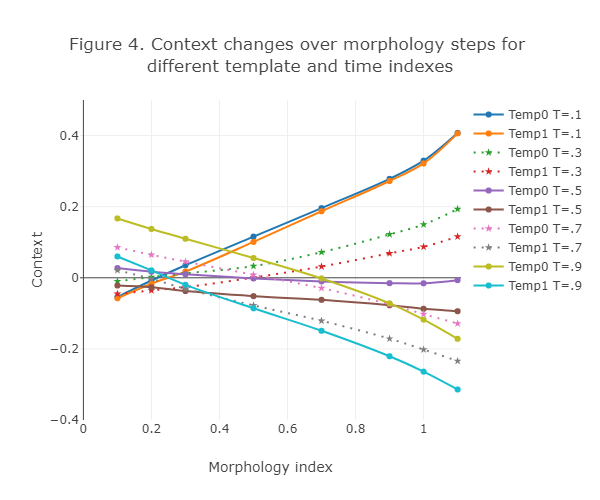


Figure 4. shows how the context values change across morphological steps for lines that represent different time indexes for the two template models. At time step .1 both templates model a closed petal with the same mesh. Since the mesh curves are the same, the context values are the same. As the petal expands over time the meshes diverge, and the context values separate.

## Using context values to explore the generation of new shapes

This provide a model for shape generation that can be used to interpolate/extrapolate between the two templates. **However, in order to have any hope of finding artistically valuable new shapes, you need to project phenomenal values to a much smaller and compact “latent” space of coherent configurations.** There are two levels to this problem. First you need to be able to interpolate between coherent shapes, and second you want those shapes to be of as high artistic value as possible. This crosses over into a discussion of reinforcement learning and genetic algorithms, that will have to be deferred to another article, but it emphasizes the important of mapping to a compact space that respects the implicit relational constraints of the forms, and is able to identify context values that support high value exploration across that space. Instead of exploring in the space of phenomenal shapes, which quickly becomes computationally intractable, the relational morphological/time map that was created with the templates, provides a completely compact set of indices. Every point in the space represents a valid morphological stage at some point in time. The context values that emerges during the learning process can now be used as a kind of “game controller” like parameter to interpolate and extrapolate from training points in context space and projected out into 3D object space, in search of new shapes of artistic value.

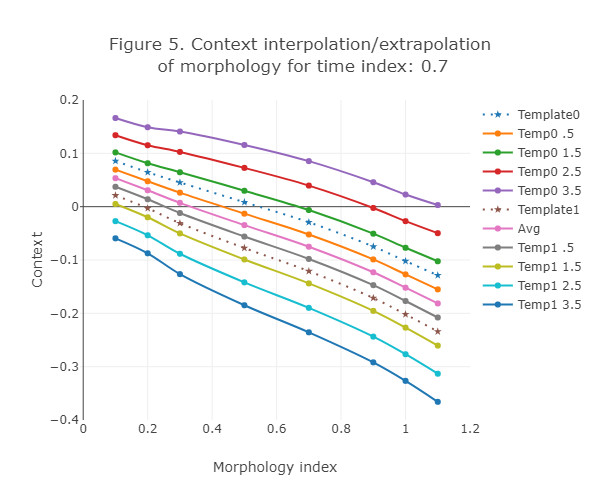


Figure 5. shows the context curves for the reference template shapes at time index 0.7 as dotted lines. Context values between the two reference curves (dotted lines) will interpolate between those two reference shapes when projected into a 3D space.

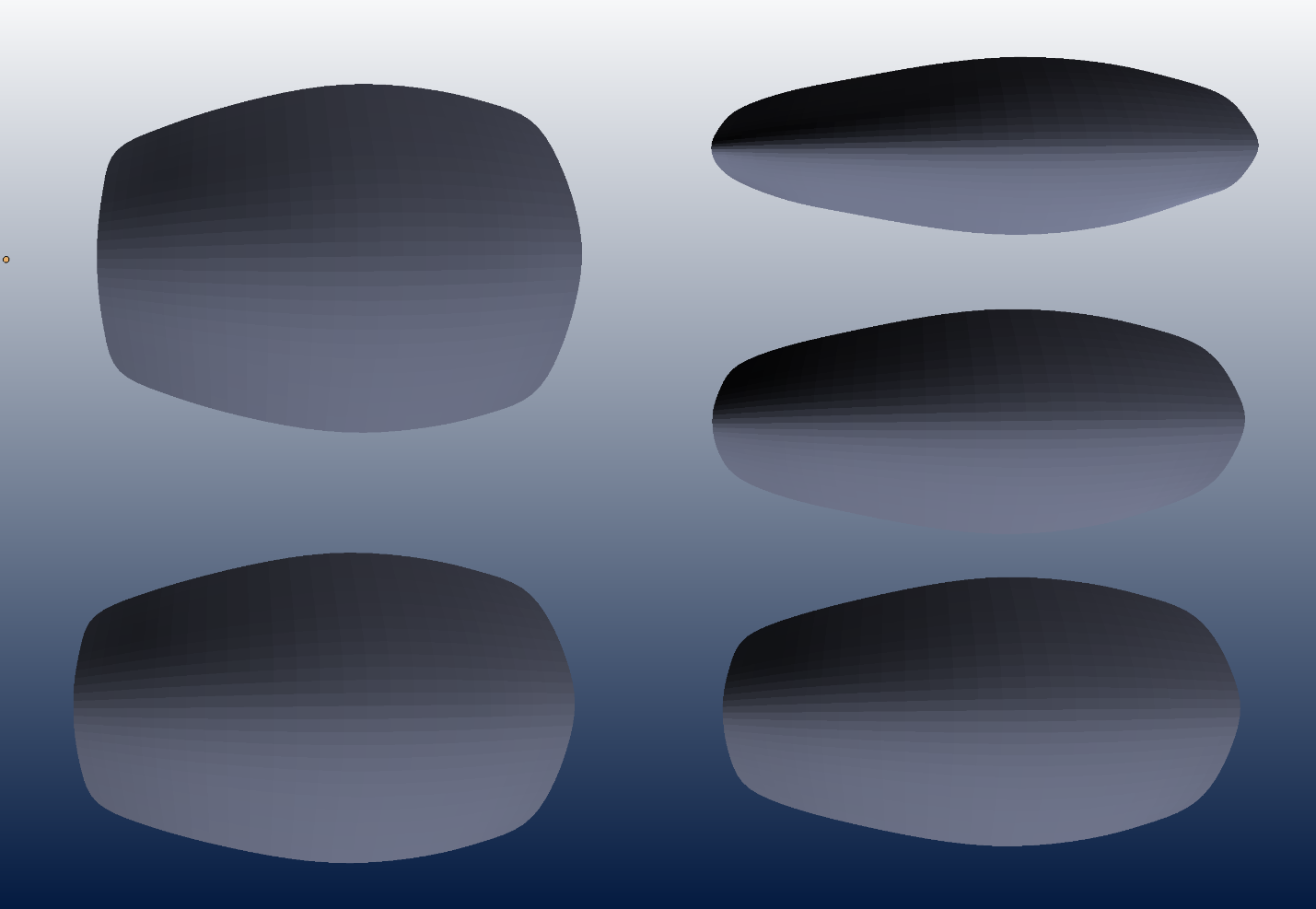


Figure 6. Top view 3D renderings of Phenomic model interpolating between reference context values

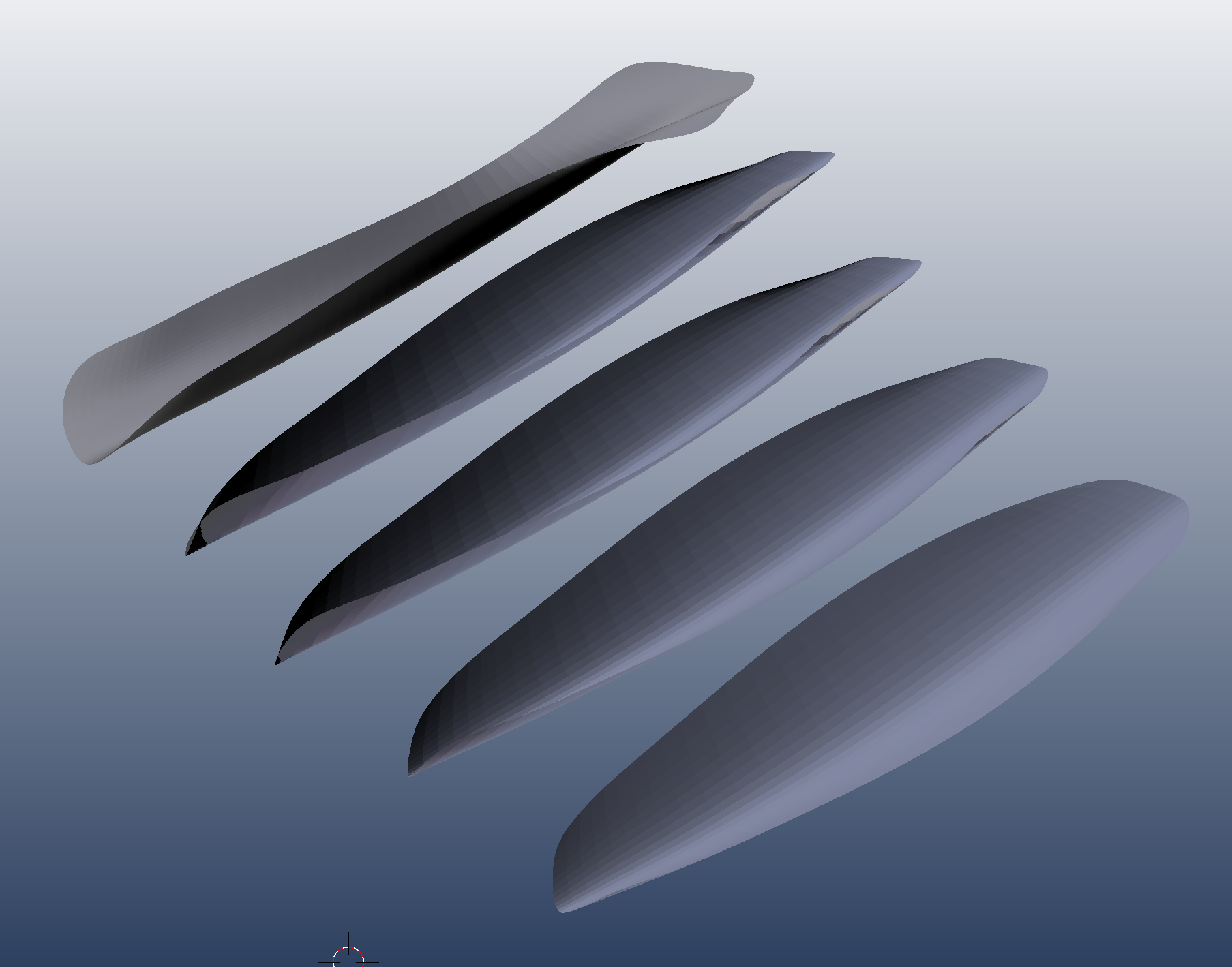


Figure 7. 3D renderings of Phenomic model outputs using an exploratory context path

The context lines outside the reference curves extrapolate beyond the constraints in the relational map. This in turn allows the Ph model to produce shapes beyond what it was trained to produce, while enforcing the relational constraints, to a diminishing degree. Figure 7 show one path of context values along the normal vector outside the Template model 1 context values.

## Example code

In conclusion I would like to highlights some aspects of the code used in this article. While the code presents a fairly trivial example of generating a “low poly“ mesh to support a 3D model of an expanding flower petal. Hopefully it will allow for the clear presentation of the issues required to scale up to more complex models.  While a complete review of the code is outside the scope of this article, I want to highlight some more unique features of the model. The full code will be available on a GitHub repository to support a future article.

First, I want to show the network model that was used. One notable feature is the use of the ELU activation function that seems to work better for small problems like this, where large scale GRU performance isn’t an issue. For this problem the CPU actually ran faster than using the GPU.

## A simple PyTorch Model structure:

class Net(nn.Module):

def \_\_init\_\_(self, layerSizes, name):

super(Net, self).\_\_init\_\_()

self.shape = layerSizes

self.name = name

self.seq = nn.Sequential(

nn.Linear(layerSizes[0],layerSizes[1]),

nn.ELU(),

nn.Linear(layerSizes[1],layerSizes[2]),

nn.ELU(),

nn.Linear(layerSizes[2],layerSizes[3]),

nn.ELU()

)

def forward(self, x):

return self.seq(x)

## Model initialization and parameters:

#initialize network

manualSeed = 1

np.random.seed(manualSeed)

# random.seed(manualSeed)

torch.manual\_seed(manualSeed)

path=CURRENT\_MODEL\_SAVE\_DIR+'templateModel'

tmModel0 = Net([2,10,15,21],'First template model')

tmModel1 = Net([2,10,15,21],'Second template model')

tmModel0.load\_state\_dict(torch.load(path+'0.pth'))

tmModel1.load\_state\_dict(torch.load(path+'1.pth'))

tmModel0.eval()

tmModel1.eval()

#tmModel0.shape, tmModel1.shape

relModel = Net([21,15,10,3],'Relational Model with 1 Context output')

phModel = Net([3,10,15,21],'Phenomic Model with 1 Context input')

relModel.train()

phModel.train()

# device = torch.device("cuda:0")

device = torch.device("cpu")

if device:

relModel.to(device)

phModel.to(device)

tmModel0.to(device)

tmModel1.to(device)

print("Transfer to use ", device)

#relModel.shape, phModel.shape

parms={ 'epochs':100000,'lrR':.0001,'lrP':.0002,'numSamples':1,'rFilter':torch.Tensor([1,1,.05]),

'tModels':[tmModel0,tmModel1],'device':device}

## Training loop

For the training loop there is a lot more going on. The function gy.createPhRcTargetsFromModelList creates relational Map and 3D target coordinate values that are sampled from randomly weighted values of the two template models. A more notable issue that was not discussed in the article, is how gradient information is passed back through the Relational and Phenomic models. This requires that you use “retain\_graph=True” when moving back through the Phenomic model. You also want the Phenomic model gradient to influence the context value without disrupting the relational map of the Relational model targets. Trying to do this in a high-level framework like PyTorch turned out to be challenging. After many hours trying to develop a custom autograd function, I hit on a much simpler solution. I simply mask the values passed into the loss criterion of the Relational model. A small .05 amount of the gradient was allowed to pass through the mask in order to help scale the context value towards its default value of zero.

def phRcPhTrain(relModel,phModel, parms):

epochs = parms['epochs']

lrR = parms['lrR']

lrP = parms['lrP']

tModels = parms['tModels']

device = parms['device']

numSamples = parms['numSamples']

rFilter = parms['rFilter']

numContextDims=1 #extra emergent dimensions

minLossP = 1000

maxLossP = -10

minLossR = 10

maxLossR = 0

criterion = torch.nn.MSELoss(reduction='sum')

relOptimizer = torch.optim.Adam(relModel.parameters(), lr=lrR)

phOptimizer = torch.optim.Adam(phModel.parameters(), lr=lrP)

for epoch in range(epochs):

#Each epoch get a new training set based on random input to the template models

relTarget, phTarget, \_ = gy.createPhRcTargetsFromModelList(tModels, numSamples, numContextDims)

phOptimizer.zero\_grad()

relOptimizer.zero\_grad()

relOutput = relModel(phTarget)

phOutput = phModel(relOutput)

lossP = criterion(phOutput, phTarget)

lossR = criterion(relOutput\*rFilter, relTarget\*rFilter)

(lossP).backward(retain\_graph=True)

(lossR).backward()

phOptimizer.step()

relOptimizer.step()

minLossP = min(lossP.data.item(), minLossP)

maxLossP = max(lossP.data.item(), maxLossP)

minLossR = min(lossR.data.item(), minLossR)

maxLossR = max(lossR.data.item(), maxLossR)

if (epoch) % (epochs/10) == 0:

print("Epoch {0:,.0f} - lossR: {1:,.5f} - lossP: {2:,.5f}".format(

epoch, lossR/numSamples, lossP/numSamples))

print("Epoch {0:,.0f} - lossR: {1:,.5f}:{2:,.5f} - lossP: {3:,.5f}:{4:,.5f}".format(epoch+1,

minLossR/numSamples, maxLossR/numSamples, minLossP/numSamples, maxLossP/numSamples))

return