Mario Plays on a Manifold: Generating Functional Content in Latent Space through Differential Geometry

Miguel González-Duque[1], Rasmus Berg Palm, Søren Hauberg[2], Sebastian Risi[1,3]
[1]*Creative AI Lab, IT University of Copenhagen*
[2]*Cognitive Systems, Technical University of Denmark*
[3]*modl.ai Denmark*
migd@itu.dk, rasmusbergpalm@gmail.com, sohau@dtu.dk, sebr@itu.dk

***Abstract*— Deep generative models can automatically create content of diverse types. However, there are no guarantees that such content will satisfy the criteria necessary to present it to end-users and be functional, e.g. the generated levels could be unsolvable or incoherent. In this paper we study this problem from a geometric perspective, and provide a method for reliable interpolation and random walks in the latent spaces of Categorical VAEs based on Riemannian geometry. We test our method with "Super Mario Bros" and "The Legend of Zelda" levels, and against simpler baselines inspired by current practice. Results show that the geometry we propose is better able to interpolate and sample, reliably staying closer to parts of the latent space that decode to playable content.**

***Index Terms*—Variational Autoencoders, Differential Geometry, Uncertainty Quantification, Deep Generative Models**

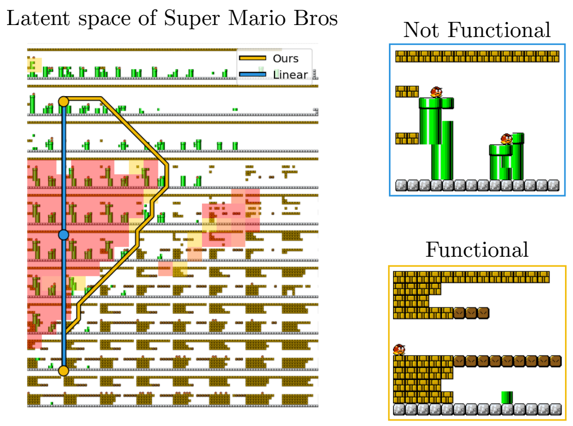


Fig. 1: Functional content in latent space. When using deep generative models, there are no guarantees that functional content will be produced. This figure shows the result of decoding a grid of levels in the latent space of a Variational Autoencoder, highlighting a functional level (i.e. one solvable by an AI agent) and a non-functional one. Superimposed to the levels is a finer grid showing which regions of the latent space correspond to functional (transparent) and non-functional (red) content. We propose a method for performing interpolations and random walks that stay within functional content. This figure shows a comparison between an interpolation using the proposed approach, and a linear interpolation.

I. Introduction

Deep latent variable models, such as Variational Autoencoders (VAEs) or Generative Adversarial Networks (GANs), learn a low-dimensional reprensentation of a given dataset. Such methods have found great application in the game AI community, mainly to provide a continuous space in which to search for relevant content using evolutionary techniques or random sampling [1]–[3], or as a tool for game blending [4]–[7]. Even though these methods excel at replicating a given distribution and generating novel samples, they sometimes fail to generate **functional** content [8]. For example, when using them to create levels for tile-based games, there are no guarantees for the generated content to be solvable by players [2], restricting the possibility to serve content directly from latent space. Fig. 1 shows an example of this problem for the latent space of a VAE trained on Super Mario Bros (SMB) levels: only some of the regions of the latent space correspond to functional levels (defined in this case as levels that are solvable for a human player, or an artificial agent), thus making it difficult to reliably sample or interpolate functional content. Fig. 1 shows, for example, a linear interpolation that crosses non-functional regions of latent space.

We propose a heuristic for safe interpolation and random walks based on differential geometry. Interpolations and random walks allow us to gradually modify one type of content to another, making them great tools for designers interested in creating functional content [9]. In this novel perspective, we can construct a discrete graph of the points in latent space that correspond to playable content, and perform interpolations and random walks therein. Fig. 1 shows an example of our interpolation approach. These techniques are inspired by advancements in the uncertainty quantification community, where methods like these are applied to stay close to where training points are [10], [11]. Our insight is that these same tools could be applied for staying close to playable content instead of the support of the data.

High-level description of our method. As discuss above, we provide a method for interpolating and performing random walks based on considering a discrete graph of only the

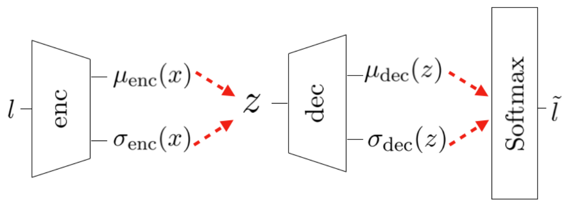


Fig. 2: Description of the VAE. In our experiments, we use a one-layer hierarchical VAE. This means that we first decode to a Normal distribution, whose samples are then passed through a Softmax activation function to generate levels. Red dashed arrows indicate sampling from a normal distribution.

playable content in latent space. To construct it, we first train a Variational Autoencoder with one hierarchical layer on tile-based game content (e.g. SMB levels), and we regularize its decoder to be uncertain in non-playable regions. This regularization process, developed for differential geometry, assigns high volume to non-functional parts of latent space, thus forcing shortest paths and random walks to avoid them. As inputs, our method requires a Variational Autoencoder (with one hierarchical layer and low-dimensional latent space) trained on tile-based content, and a way to test for the functionality of decoded content.

We test our method against simpler baselines (two types of random walks based on taking Gaussian steps and on following the mass of playable levels, and linear interpolation), and we find that we are able to present functional content more reliably than them. These methods are tested on two domains: grid-based representations of levels from Super Mario Bros and The Legend of Zelda, using different definitions of functionality (e.g. a solvable level, or a solvable level involving jumps).

II. Background

From a bird eye's view, our method can be described by the following steps: (i) train a Variational Autoencoder (VAE) on tile-based data (using a low-dimensional latent space), (ii) explore its latent space for functional content using grid searches, and (iii) use this knowledge to build a discrete graph that connects only the functional content. To construct this graph, we leverage differential geometry: we modify the decoder such that non-playable areas have high volume. This section describes all the theory and technical tools involved.

A. Variational Autoencoders on tile-based data

Variational Autoencoders (VAEs) [12], [13] are a type of latent variable deep generative model (i.e. a method for approximating the density $p(x)$ of a given dataset, assuming that the generative process involves a low dimensional representation given by latent codes z). Opposite to GANs [14], VAEs model this density explicitly using a variational training process. This means that the conditional distribution $p(x|z)$ (which models the probability density of a certain datapoint x given a certain training code z) and the posterior distribution of the training codes $p(z|x)$ (which models the likelihood of the code of a certain datapoint) are approximated using parametrized

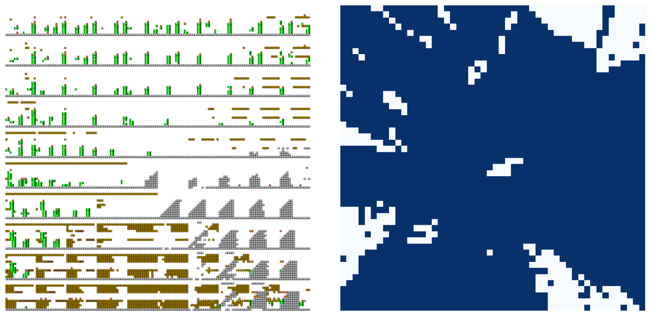

(a) (b)

Fig. 3: How playability is distributed in the latent space of SMB. After training the VAE, we decode in levels in a 10×10 grid of equally spaced points in the $[-5, 5] \times [-5, 5]$ square (Fig. 3a). Even in two dimensions, our model is able to segment functionally different levels in distant parts of the latent space. These levels seem to vary smoothly with respect to playability: Fig. 3b shows the measured playability after decoding a 50×50 grid of levels in latent space. Blue squares correspond to playable levels, and white to non-playable.

families of distributions. A common choice is to approximate the posterior density of the training codes using a Gaussian distribution $q_\phi(z|x)$, parametrized using neural networks. Since we will focus on tile-based content, the conditional $p_\theta(x|z)$ will be modeled using a Categorical distribution parametrized by a neural network.

In practical terms, this translates to having an autoencoder with *probabilistic* encoder and decoder, both learning the parameters of the distributions $q_\phi(z|x)$ and $p_\theta(x|z)$ respectively.

For our geometric tools to work, we need a deep generative model that allows for manipulating the uncertainty of our decoded outcomes. Thus, we consider a one-layer hierarchical VAE which first decodes to a Normal distribuition, which is then sampled to create logits for the final Categorical distribution $p(x|z)$. This hierarchical layer computes the mean $\mu_{\text{dec}}(z)$ and standard deviations $\sigma_{\text{dec}}(z)$ of the logits of the Categorical distribution. In other words, the probabilities of this distribution are giving by

$$\text{decode}(z) = \text{Softmax}(\mu_{\text{dec}}(z) + \varepsilon \odot \sigma_{\text{dec}}(z)^2), \qquad (1)$$

where $\epsilon \sim N(0, I_D)$ and D is the dimension of the data. With this formulation, we can manipulate the uncertainty of our Categorical by having $\sigma_{\text{dec}}(z) \to \infty$. A diagram that shows how this network is structured can be found in Fig. 2.

Summarizing, our model is as follows

$$q(z \mid x) = \mathcal{N}\left(z \mid \mu_{\text{enc}}(x), \sigma_{\text{enc}}(x)^2\right), \qquad (2)$$
$$p(x \mid z) = \text{Cat}(x \mid \text{decode}(z)), \qquad (3)$$

where μ_{enc}, μ_{dec}, σ_{enc} and σ_{dec} are usually parametrized using neural networks whose weights are optimized by maximizing the Evidence Lower Bound (ELBO) [12].

B. An intuitive introduction to differential geometry

Our method relies on the theory of smooth and Riemannian manifolds (known as differential geometry). We will explain it by giving an intuition for what Riemannian manifolds are, what a metric is, and how a smooth map can allow us to "pull back" a metric from one Riemannian manifold to another. This explanation focuses on intuition at the expense of technical precision. For a thorough and theoretical explanation we recommend [15], [16].

Intuitively, a Riemannian manifold is a smooth surface that carries a **metric**: a tool for defining distances accounting for how curved the surface might be. For example, distances along a sphere are different from distances on Euclidean space. In loose terms, a metric is equivalent to a dot product, allowing us to define norms (e.g. $\|x\| = \sqrt{x \cdot x}$ in $\mathbb{R}^n$), lengths and angles between tangent vectors to the surface. One key subtlety is that this metric is defined locally around a given point, instead of globally as in $\mathbb{R}^n$.

Consider our decoder dec between the latent space $\mathcal{Z}$ and the Euclidean data space $\mathbb{R}^D$. This decoder induces a metric on $\mathcal{Z}$ called the **pullback**, which is given by $M(z) = J_{\mathrm{dec}}(z)^\top J_{\mathrm{dec}}(z)$ where J_{dec} is the Jacobian of the decoder. To understand this definition in context, we can compute a curve's length in $\mathcal{Z}$ by using lengths in $\mathbb{R}^D$: let $c : [0,1] \to \mathcal{Z}$ be a curve, its length is then given by

$$\begin{aligned}\mathrm{Length}[c] &= \int_0^1 \left\| \frac{\mathrm{d}}{\mathrm{d}t} d(c(t)) \right\| \mathrm{d}t \\ &= \int_0^1 \sqrt{(J_d(c(t))c'(t)) \cdot (J_d(c(t))c'(t))} \mathrm{d}t \\ &= \int_0^1 \sqrt{c'(t)^\top M(c(t)) c'(t)} \mathrm{d}t\end{aligned}$$

where we used the chain rule and the definition of the usual norm in Euclidean space. In other words, the pullback $M(z)$ is mediating local lengths in $\mathcal{Z}$. The quantity $\det(M(z))$ also plays a special role, since it measures the **volume** of a region locally around z [16]. Our method modifies the decoder in such a way that $M(z)$ has high volume close to non-playable parts of latent space.

III. Defining a geometry in latent space

A. Playability-induced geometry in latent space: Motivation

After training a VAE on SMB levels, we wondered how playability was distributed in the latent space. For this, we used Robin Baumgarten's super-human A* agent [17] to simulate a 50×50 grid in $[-5, 5]^2$ in latent space. This agent returns telemetrics, including whether the level was playable and how many jumps the agent performed. Simulations are supposed to be deterministic, but we found in practice that they were not and thus we performed 5 simulations per level.[1]

Fig. 3b shows the VAE's latent space illuminated by the mean playability of each level. Notice that unplayable regions are localized in different parts of the latent space, and can be thought of as "obstacles" when doing interpolations or random searches. This empirical observation is supported by looking at different training runs. A similar phenomena can be observed in the latent space of Zelda: if we define functionalty as decoding coherent and connected levels, we see structure in the latent space (see Figs 4a and 4d).

Our goal, then, is to use this playability in latent space (or an approximation thereof) to define a meaningful geometry in latent space that allows us to do informed interpolations and diffusions (i.e. random walks).

This section describes how to use the playability information from a coarse grid in latent space to define algorithms for interpolation and random walks in latent space. On a high-level, this geometry is a discrete approximation of the one induced by $J_{\mathrm{dec}}(z)^\top J_{\mathrm{dec}}(z)$, where J_{dec} is the Jacobian of a "calibrated" version of the decoder. The quantity $J_{\mathrm{dec}}(z)^\top J_{\mathrm{dec}}(z)$ has a clear geometric meaning: it defines a metric on the latent space $\mathcal{Z}$ by "pulling back" the Euclidean metric on the data space $\mathbb{R}^D$ (see Sec. II-B). We will guide the reader using a VAE trained on The Legend of Zelda levels as an example (see Fig. 4).

B. Calibrating the decoder

In Sec. II-B we saw that we can use the Jacobian of the decoder dec to define a geometry in the latent space. Let's develop more intuition about what the Jacobian is, and how we can exploit its definition to make traversing through non-playable levels "more expensive" for our distances. The Jacobian plays the role of a first order derivative for functions of more than one variable, and it can be approximated by finite differences just like the usual derivative:

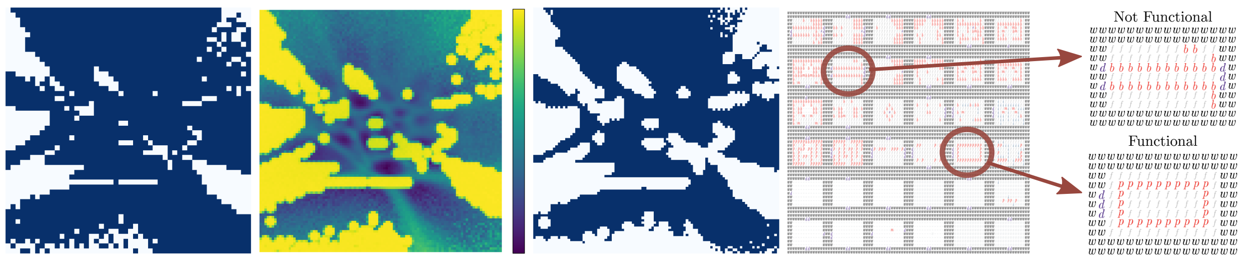
$$J_{\mathrm{dec}}(z) \approx \left[\frac{\mathrm{dec}(z + dz_i) - \mathrm{dec}(z)}{\|dz_i\|} \right]_{i=1}^{\dim(\mathcal{Z})} \quad (4)$$

where dz is a small vector on the ith direction and $\|dz_i\|$ is its norm. To increase the volume of the metric (and thus have higher local distances in latent space), we need a decoder in which small changes with respect to the latent code will result in large changes after decoding. Thus, we calibrate our decoder to output what it learned during training when close to playable levels (i.e. $\mathrm{Softmax}(\mu_{\mathrm{dec}}(z) + \epsilon\sigma_{\mathrm{dec}}(z))$, and to extrapolate to noise when close to non-playable levels (replacing $\sigma_{\mathrm{dec}}(z)$ for 10^5). In other words, whenever we walk close to non-playable levels, we pass through levels of very different forms making the numerator of Eq. (4) explode. This method was first introduced for VAEs in [10] and was applied to Categorical distributions in [18].

Numerically speaking, the way in which we transition from decoding the levels we learned to decoding highly noisy levels is by computing the distance to the closest non-playable level, and transforming that distance into a number $\alpha(z)$ between 0 and 1, where $\alpha(z) = 0$ means that the latent code is close to non-playable regions and $\alpha(z) = 1$ means the opposite. That number is then used as a semaphore in a sum

$$\mathrm{dec}(z) = \alpha(z)\,\mathrm{dec}_{\mu,\sigma}(z) + (1 - \alpha(z))\,\mathrm{dec}_{\mu,10^5}(z). \quad (5)$$

[1]We hypothesize that this is the effect of random CPU allocations during the simulation. After giving it our best efforts, we were not able to force Baumgarten's agent to be deterministic.



(a) Coherent levels **(b)** Metric volume **(c)** Discrete approximation **(d)** Levels in latent space

Fig. 4: Defining a geometry in latent space. Here we illustrate how our method works using a VAE trained on Zelda levels. First, in Fig. 4a we decode a coarse grid in latent space and identify the regions where non-functional content is (blue corresponds to functional). Using this knowledge, we then calibrate a decoder to have high metric volume in these non-functional regions (see Fig. 4b; we show log-volumes). From these metric values, we construct an approximation by thresholding, arriving at the discrete graph presented in Fig. 4c. Notice how there is a wider band around non-functional content, helping us to avoid it. Finally, we verify by decoding a grid of levels in latent space in Fig. 4d.

This semaphore function $\alpha(z)$ is implemented using a translated sigmoid function. Namely, consider minDist(z) to be the minimum distance to a non-functional training code, then we can define

$$\alpha(z) = \text{Sigmoid}\left(\frac{\text{minDist}(z) - \beta k}{\beta}\right), \tag{6}$$

where β is a hyperparameter that governs how quickly we transform from 0 to 1 (we settled for $\beta = 5.5$ in our experiments), and $k \approx 6.9$.

In summary, we decode a coarse grid in latent space to identify where possible non-playable regions are, we use it to calibrate our decoder, making it locally noisy around non-playable regions. We then compute the pullback metric of the calibrated decoder, resulting on a notion of distance in which non-playable levels are far away. Figures 4a and 4b show exactly this: non-functional regions (showed as white) render high log-volume.

C. Approximating the manifold with a graph

Once we have a calibrated decoder $\text{dec}(z)$, we can equip our latent space with the metric $M(z) = J_{\text{dec}}(z)^\top J_{\text{dec}}(z)$. This metric, as discussed above, has high volume close to non-playable levels. Empirically, we found it useful to approximate this manifold using a finite graph. At this point we can easily compute $M(z)$ for many training codes by approximating the Jacobian using first order finite differences (see Eq. (4)), knowing that $\det(M(z))$ is a high number for points that are close to non-playable regions of space (according to a coarse grid). We can then sample an arbitrary grid, compute the metric $M(z)$ for each point and construct a finite graph by considering only the points whose volume $\log(\det(M(z))$ is below some threshold v (which we chose to be the mean volume, but can be thought of as a hyperparameter).

Fig. 4c shows how we can transform a coarse grid into a continuous manifold, and then approximate it using a finite graph as discussed above. Notice that this finite approximation can be of any resolution. In this graph, interpolation and random walks can be easily implemented. We discuss these algorithms next.

a) Shortest paths on graphs through A:* Once we have a finite graph embedded on $\mathbb{R}^{\dim(\mathcal{Z})}$, computing the shortest path between two latent codes z and z' in the graph can be done using the A* algorithm [19]. It must be noted that, for simplicity in the implementation, we consider a connected graph in which the points we are trying to avoid are at infinite distance.

b) Random walks on graphs: Starting at a node z_0, our random walk algorithm samples uniformly at random from the set of neighbours of z_0, arriving at an intermediate point z_0^1. Since the grid is usually very fine (thus making local distances small), we continue this process of sampling uniformly at random from neighbors on z_0^1 arriving at z_0^2. This process continues for m "inner" steps. After these, we assign $z_1 = z_0^m$. For our experiments we decided on $m = 25$, but it can be considered a hyperparameter.

IV. Experiments & Results

A. Training results for the VAE

We trained our one-layer hierarchical VAEs on data from two tile-based games: Super Mario Bros (SMB) levels and The Legend of Zelda. Both of these are common benchmarks for testing generation in the game AI community. These levels, usually extracted from the Video Game Level Corpus (VGLC)[2], are processed into token-based text files of height and width 14 in the case of SMB, and levels of shape 11×16 in the case of Zelda. This processing involves sliding a 14×14 window across the original SMB levels, and by splitting the original Zelda levels into rooms. We arrive at a total of 2264 playable levels for SMB; and of the 237 original levels for Zelda, we keep 213 that are solvable according to a definition we discuss in the next section.

We trained 10 different VAEs for SMBs and 10 for Zelda. All these networks have MLP encoders with 3 hidden layers of sizes 512, 256 and 128 respectively, then mapping to a latent space of size 2; the decoders were symmetric, and the last hierarchical layer is of size D, the data's size. For the SMB

[2]https://github.com/TheVGLC/TheVGLC

and Zelda networks, we used learning rates of 10^{-3} and 10^{-4} respectively; for both we used a batch size of 64, and an Adam optimizer [20] and early stopping with at most 250 epochs. All these neural networks were trained using PyTorch [21].

We noticed an ELBO loss of 53.4 ± 3.1 for the training runs on SMB, and of 31.8 ± 10.9 for Zelda. As can be seen from the variance, the training processes for Zelda were surprisingly unstable, with some networks converging to flat representations, failing to capture the structure of the dataset. Thus, for our experiments we used 4 out of the 10 VAEs trained on Zelda, keeping only the networks in which the latent space had more than flat, constant representations.

B. Finding playable regions using a coarse grid

To find the non-playable regions in the latent space of Super Mario Bros, we ran Robin Baumgarten's A* agent [17] on the decoded levels of a 50×50 grid in all latent spaces using the simulator provided by the MarioGAN repository[3]. In the case of Zelda, we implemented a "grammar check" which makes sure that (i) the level has either stairs or doors, (ii) if there are more than two doors, they must be connected by a walkable path (with e.g. no lava/water tiles blocking all possible paths), (iii) doors must be complete and in the right places, and (iv) the level must be surrounded by walls. Figs. 3 and 4 show examples of these two latent spaces.

C. Approximating the manifold using graphs

After training these VAEs, we can regularize the decoder, forcing it to be expensive in non-playable regions (see Sec. III-B). When the decoder is calibrated, the log-volume $\log(\det(J_{\text{dec}}(z)^T J_{\text{dec}}(z))) = \log(\det(M(z)))$ is high for latent codes that correspond to non-playable levels.

We can, then, consider a finer grid of any resolution (we chose 100×100 levels), cutting off the nodes for which $\log(\det(M(z))$ is higher than the average over all the grid. This process was used to approximate the playability manifold for both SMB and Zelda. For example, Fig. 4c shows the resulting 100×100 grid in the Zelda.

To argue why a discrete approximation was necessary, consider the following random walk algorithm for the continuous case: at any step z_n in the random walk, compute the pullback metric $M(z_n)$ and sample from the Gaussian $N(z_n, M(z_n)^{-1})$. We found, empirically, that these random walks tended to "get stuck" close to non-training codes. Indeed, in those regions the inverse of the metric can be thought of as infinitesimally small.

D. Comparing interpolations and random walks

To test whether our proposed geometry improved on reliably interpolating/randomly walking between functional content from the latent space we considered, for each VAE, 20 interpolations and 10 random walks. 10 equally spaced points were then selected from each interpolation, and each random walk was ran for 50 steps.

[3]https://github.com/CIGbalance/DagstuhlGAN

We considered linear interpolation to be a suitable baseline for interpolation, since it is common practice in the deep generative models community [22]. For random walks, we considered two baselines: first a naïve one, based on randomly sampling at each step z_n from a Gaussian $N(z_n, I_2)$ where I_2 is the identity matrix of size 2; secondly a baseline that randomly samples playable levels from the graph and takes a step of fixed size in that direction. This second baseline then tends to stay within playable levels, converging to its center of mass in the limit, and this first baseline is closer to common methods for latent space exploration [2]. Example interpolations and random walks for all methods (ours, plus the two baselines) can be found in Figs. 5c, 5e, and 5d.

Table I shows the average playabilities of these 10 interpolations and 20 random diffusions for our proposed geometry, plus the two baselines discussed above. In the case of Super Mario Bros (SMB), we see a slight increase in the reliability of interpolations and random walks: our methods indeed stay more within the playable regions, showing an expected playability (i.e. functionality) of close to 100%. This is to be contrasted with e.g. the playability of performing Normal random walks, which results only in about 77% of playable levels. In the case of Zelda, we see a similar increase in the probability of presenting a playable level when using our geodesic interpolation; the baselines for random walks, however, seem to perform relatively poorly in these latent spaces. Our method is able to avoid non-playable regions, presenting an estimated 99.5% probability of sampling a random level. Fig. 6 shows a violin plot of the mean playabilities for these interpolations and random walks, providing more than a point uncertainty estimate of these results.

E. Measuring diversity in decoded levels

Another metric that is particularly relevant for games is the diversity present in a corpus of levels [23]. We leverage the literature that discusses similarities between Categorical data [24] to implement the following similarity metric: given two levels l_1 and l_2, both of size (h, w), we define

$$\text{sim}(l_1, l_2) = \frac{1}{wh} \sum_{i,j}^{w,h} [l_1[i,j] = l_2[i,j]] \, , \tag{7}$$

where $[l_1[i,j] = l_2[i,j]]$ is one when the boolean condition is satisfied, and 0 otherwise. In other words, we measure how many times the two levels l_1 and l_2 agree on average. We define the **diversity** of a corpus of levels $\mathcal{L} = \{l_m\}_{m=1}^{M}$ as

$$\text{diversity}(\mathcal{L}) = 1 - \frac{2}{M(M-1)} \sum_{m<m'} \text{sim}(l_m, l_{m'}), \tag{8}$$

that is, as one minus the average similarity of different pairs of levels in the corpus, counted only once. With this measure, the entire Zelda dataset used for training has a diversity of 0.23, and the SMB dataset has a diversity of 0.17.

Table I also shows the mean diversity of the interpolations and random walks. We see that our improvements on reliablity

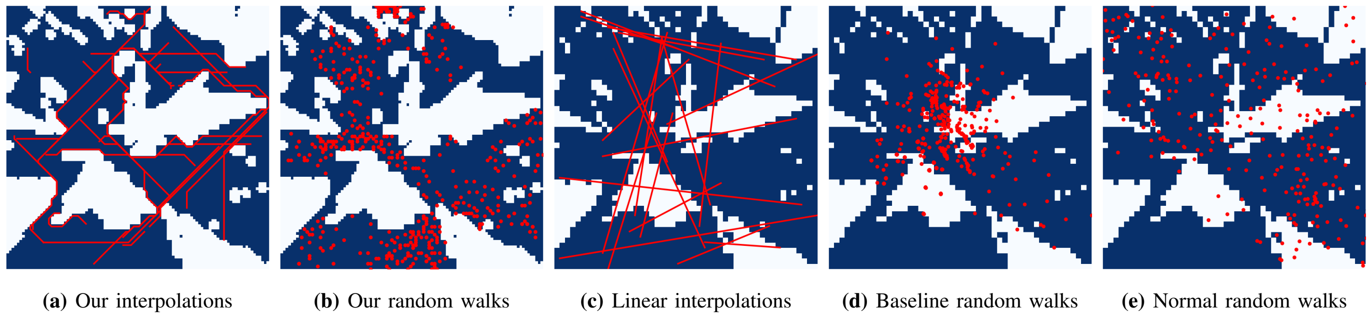

(a) Our interpolations **(b)** Our random walks **(c)** Linear interpolations **(d)** Baseline random walks **(e)** Normal random walks

Fig. 5: Interpolations and diffusions in the jumping submanifold. We showcase examples of the different interpolations and diffusions used in all comparison experiments (see Secs. IV-D and IV-F) for the submanifold given by solvable levels in which Mario jumps. Notice how this graph is a subset of Fig. 3b. Figs. 5a and 5b show our interpolations and diffusions respectively. Fig. 5c shows example linear interpolations (used in both baselines) and Figs. 5d and 5e show the random walks of our baselines. We manage to staw away from non-functional levels (shown in white) in both our interpolation and random walks, sometimes at the cost of getting stuck bottlenecks (see the upper part of 5b).

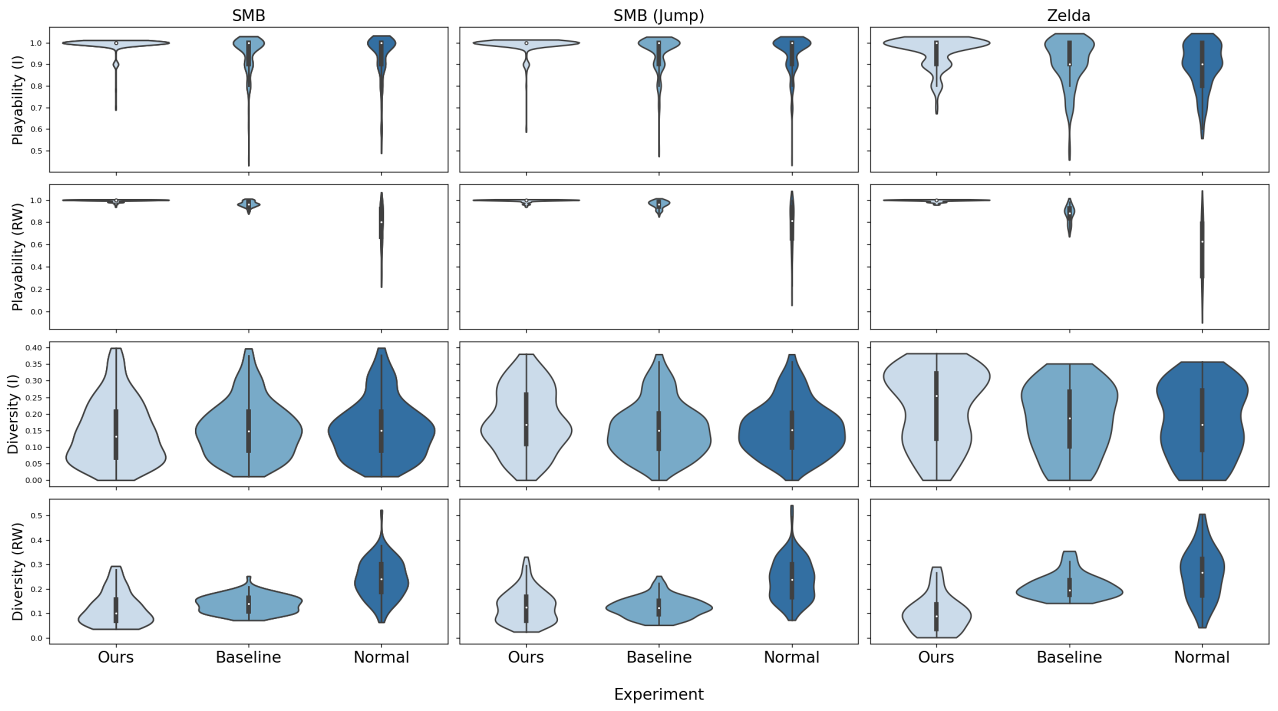


Fig. 6: Comparing our geometry against baselines. This figure shows the distributions of playability and diversity, which are summarized on Table I, where (I) stands for interpolations and (RW) stands for random walks. For each VAE we performed 20 interpolations and 10 random walks, selecting the starting points at random. These quantities were measured in each interpolation/random walk. Our geometry has most of its playability mass closer to 1.0 than the baselines; however, this comes at a slight cost on diversity: the mass for estimated diversities is lower than the baselines.

come at a small cost on the diversity of the decoded levels. This diversity cost is stark in the case of geometric random walks for Zelda, and we hypothesize that it is because of the “bottlenecks” between the playable regions, as well as the fact that some playable regions may be disconnected (see Fig. 5b for a showcase of this behavior on similar manifolds).

F. A different definition of functionality

So far, we have dealt with presenting levels that are functional in the sense that they are solvable, or playable by users. We can, however, modify this definition to encompass a more restricted set. We experiment with restricting the SMB playability graph even further to the subset of levels in which Mario has to jump.

This manifold is presented in Fig. 5, which is to be compared with Fig. 3. Table I shows the proportion of levels in which Mario jumps in 20 interpolations and 10 random walks, just like in Sec. IV-D. First, we notice that the expected playability is not hindered by restricting to this submanifold. Second, since the area of levels without jump is wider than that of playabiliy (leading to a non-convex manifold), we decode to levels that include jumps more reliably than the two discussed baseline (with an estimated increase of e.g. 10% for interpolations).

Geometry	$\mathbb{E}$[playability] ↑ Interpolation	$\mathbb{E}$[playability] ↑ Random Walks	$\mathbb{E}$[diversity] ↑ Interpolation	$\mathbb{E}$[diversity] ↑ Random Walks
Super Mario Bros				
Ours	**0.993**±0.033	**0.996**±0.010	0.146±0.034	0.121±0.024
Baseline	0.953±0.084	0.963±0.026	0.154±0.028	0.138±0.026
Normal	0.949±0.093	0.773±0.169	**0.155**±0.029	**0.240**±0.026
The Legend of Zelda				
Ours	**0.961**±0.068	**0.995**±0.011	**0.222**±0.112	0.099±0.072
Baseline	0.916±0.104	0.874±0.073	0.182±0.104	0.213±0.051
Normal	0.896±0.105	0.567±0.257	0.178±0.107	**0.261**±0.103
Super Mario Bros (Jump)				
	$\mathbb{E}$[playability] ↑		$\mathbb{E}$[jumps > 0] ↑	
Ours	**0.990**±0.040	**0.995**±0.013	**0.99**±0.01	**1.00**±0.00
Baseline	0.957±0.078	0.960±0.034	0.90±0.03	0.75±0.08
Normal	0.952±0.083	0.768±0.200	0.90±0.02	0.94±0.02

TABLE I: Comparison between the discretized geometry and baselines for SMB, Zelda, and the jump submanifold. This table shows the expected functionality (i.e. playability) and diversity of the decoded levels after running interpolations and random walks according to our proposed discretized geometry (see Sec. III), as well as baselines composed of linear interpolation, Gaussian random walks and a custom center-of-mass seeking random walk. We report a mean and standard deviation after running the experiments on 10 different VAE runs for SMB, and 4 selected VAE runs for Zelda, . These results show that our proposed geometry is avoiding non-playable regions of the latent space more reliably than the baselines when performing both interpolations and random walks. However, it must be noted that this increase in reliability comes at a cost on the diversity of the decoded levels, especially when it comes to performing random walks on Zelda. This results hold even when we consider a submanifold of the original manifold, given by the levels in which Mario jumps at least once.

V. Related work

Our work is situated among the applications of Machine Learning (ML) to Procedural Content Generation (PCG) [8]. More precisely, recent research has focused on applying Deep Generative Modeling to the problem of creating content for videogames. From levels in VizDoom [1], graphic assets in MOBAs [25] to levels in several of the games available in the Video Game Level Corpus [26] for applications such as latent space exploration and evolution [2], [5], [23], or game blending [4]–[6], [27]. This research spans applications of Generative Adversarial Networks [28], autorregresive models [29], Variational Autoencoders [4] and Neural Cellular Automata [23]. The method we propose here differs from most of this research, given that our focus is to understand parts of the latent space that obey certain functionality criteria, and use them to define geometric algorithms for interpolation and diffusion. There exist, however, approaches in which functional content creation is addressed: [30] bootstrap the creation of playable content; and in [31] authors create content and then repair it using linear programming; both approaches are complementary to ours.

We apply techniques from the Uncertainty Quantification community. Namely, the geometric quantities we discussed (e.g. manifolds and metrics) provide invariants to reparametrization, and thus constitute sensible tools for understanding representations [32]. These geometric methods have been studied for latent variable methods like Gaussian Process Latent Variable Models (GPLVMs) [11], Generative Adversarial Networks [33] and Variational Autoencoders [10], [18], [34], [35]. Our contribution to this corpus consists of a different way to ensure high volume in the latent space of a Categorical VAE by calibrating a hierarchical layer in the decoding process, as well as a discretized approximation that allows for sensible random walks. These differential geometry tools have also been applied e.g. in robot control [36]–[38]. This line of research is also similar in spirit to learning functions in the latent space of e.g. proteins, modeled as tokens using VAEs [18], [39].

VI. Conclusion

In this paper we defined a geometry in the latent space of two tile-based games: Super Mario Bros and The Legend of Zelda. After learning a latent representation using a one-layer-hierarchical Variational Autoencoder (VAE), we explore the latent space for non-functional regions (e.g. unsolvable or incoherent levels) and place high metric volume there. This geometry is then approximated with a finite graph by thresholding volume, and is subsequently used for interpolations and random walks.

When compared to simpler baselines (e.g. linear interpolation and Gaussian random walks), our method presents functional levels more reliably. We also tested the ability to define and explore submanifolds of the SMB playability manifold, rendering a system for computing interpolations and random walks that stay not only within playable levels, but also levels in which Mario has to jump.

There are multiple avenues for future work. First of all, many of our constructions are sensitive to the hyperparameters specified (e.g. the bandwith β in Eq. (6), or the threshold cut for the metric volume in Sec. III); further work is required to generalize these methods beyond the need for these hyperparameters. Moreover, the induced geometries in latent spaces have mostly been studied when the dimension of the latent space is 2. Testing these tools and improving on them for dimensions bigger than 2 should be addressed in future work. We also rely on the assumption that functionality is distributed smoothly in the latent space, for which we have no theoretical guarantees. Finally, learning these discrete approximations of manifolds using graphs may allow for regression methods therein [40], opening the doors for e.g. optimizing certain metrics while reliably presenting functional content.

Acknowledgements

This work was funded by a Sapere Aude: DFF-Starting Grant (9063-00046B), the Danish Ministry of Education and Science, Digital Pilot Hub and Skylab Digital. SH was supported by research grants (15334, 42062) from VILLUM FONDEN. This project has also received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement Nr. 757360). This work was funded in part by the Novo Nordisk

Foundation through the Center for Basic Machine Learning Research in Life Science (NNF20OC0062606).

References

[1] E. Giacomello, P. L. Lanzi, and D. Loiacono, "Doom level generation using generative adversarial networks," 2018.

[2] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 221–228. [Online]. Available: https://doi.org/10.1145/3205455.3205517

[3] J. Schrum, V. Volz, and S. Risi, "Cppn2gan: Combining compositional pattern producing networks and gans for large-scale pattern generation," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 139–147. [Online]. Available: https://doi.org/10.1145/3377930.3389822

[4] A. Sarkar, Z. Yang, and S. Cooper, "Conditional level generation and game blending," in *Proceedings of the Experimental AI in Games (EXAG) Workshop at AIIDE*, 2020.

[5] A. Sarkar and S. Cooper, "Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder," in *Proceedings of the Foundations of Digital Games*, 2021.

[6] ——, "Dungeon and platformer level blending and generation using conditional vaes," in *Proceedings of the IEEE Conference on Games (CoG)*, 2021.

[7] Z. Yang, A. Sarkar, and S. Cooper, "Game level clustering and generation using gaussian mixture vaes," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020.

[8] A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, "Procedural content generation via machine learning (pcgml)," 2018.

[9] J. Schrum, J. Gutierrez, V. Volz, J. Liu, S. Lucas, and S. Risi, "Interactive evolution and exploration within latent level-design space of generative adversarial networks," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 148–156.

[10] G. Arvanitidis, L. K. Hansen, and S. Hauberg, "Latent space oddity: on the curvature of deep generative models," in *International Conference on Learning Representations (ICLR)*, 2018.

[11] A. Tosi, S. Hauberg, A. Vellido, and N. D. Lawrence, "Metrics for probabilistic geometries," in *The Conference on Uncertainty in Artificial Intelligence (UAI)*, Quebec, Canada, Jul. 2014.

[12] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[13] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1278–1286. [Online]. Available: https://proceedings.mlr.press/v32/rezende14.html

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[15] J. M. Lee, *Introduction to Smooth Manifolds*. Springer, 2000.

[16] ——, *Introduction to Riemannian Manifolds*. Springer, 2018.

[17] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[18] N. S. Detlefsen, S. Hauberg, and W. Boomsma, "What is a meaningful representation of protein sequences?" 2020.

[19] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[21] A. Paszke et al, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[22] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4396–4405.

[23] S. Earle, J. Snider, M. C. Fontaine, S. Nikolaidis, and J. Togelius, "Illuminating diverse neural cellular automata for level generation," *arXiv:2109.05489 [cs]*, Sep 2021, arXiv: 2109.05489. [Online]. Available: http://arxiv.org/abs/2109.05489

[24] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *Proceedings of the 2008 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr 2008, p. 243–254. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972788.22

[25] R. Karp and Z. Swiderska-Chadaj, "Automatic generation of graphical game assets using gan," in *2021 7th International Conference on Computer Technology Applications*. ACM, Jul 2021, p. 7–12. [Online]. Available: https://dl.acm.org/doi/10.1145/3477911.3477913

[26] A. J. Summerville, S. Snodgrass, M. Mateas, and S. O. n'on Villar, "The vglc: The video game level corpus," *Proceedings of the 7th Workshop on Procedural Content Generation*, 2016.

[27] A. Sarkar and S. Cooper, "Sequential segment-based level generation and blending using variational autoencoders," in *Proceedings of the 11th Workshop on Procedural Content Generation in Games*, 2020.

[28] M. Awiszus, F. Schubert, and B. Rosenhahn, "Toad-gan: Coherent style level generation from a single example," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.

[29] A. Summerville and M. Mateas, "Super mario as a string: Platformer level generation via lstms," *arXiv:1603.00930 [cs]*, Mar 2016, arXiv: 1603.00930. [Online]. Available: http://arxiv.org/abs/1603.00930

[30] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius, "Bootstrapping conditional gans for video game level generation," in *2019 IEEE Conference on Games (CoG)*, 2019.

[31] H. Zhang, M. C. Fontaine, A. K. Hoover, J. Togelius, B. Dilkina, and S. Nikolaidis, "Video game level repair via mixed integer linear programming," in *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'20. AAAI Press, 2020.

[32] S. Hauberg, "Only bayes should learn a manifold," 2018.

[33] B. Wang and C. R. Ponce, "The geometry of deep generative image models and its applications," p. 25, 2021.

[34] G. Arvanitidis, S. Hauberg, P. Hennig, and M. Schober, "Fast and robust shortest paths on manifolds learned from data," in *Proceedings of the 19th international Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[35] G. Arvantidis, M. González-Duque, A. Pouplin, D. Kalatzis, and S. Hauberg, "Pulling back information geometry," 2021.

[36] N. Chen, A. Klushyn, R. Kurle, X. Jiang, J. Bayer, and P. van der Smagt, "Metrics for deep generative models," *International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, vol. 7, p. 1540–1550, 2018.

[37] H. Beik-Mohammadi, S. Hauberg, G. Arvanitidis, G. Neumann, and L. Rozo, "Learning riemannian manifolds for geodesic motion skills," in *Robotics: Science and Systems (RSS)*, 2021.

[38] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon, "Geometry-aware manipulability learning, tracking and transfer," vol. 20, no. 2-3, pp. 624–650, 2020.

[39] D. Schwalbe-Koda and R. Gómez-Bombarelli, "Generative models for automatic chemical design," *arXiv:1907.01632 [physics, stat]*, vol. 968, p. 445–467, 2020, arXiv: 1907.01632.

[40] V. Borovitskiy, I. Azangulov, A. Terenin, P. Mostowsky, M. P. Deisenroth, and N. Durrande, "Matérn gaussian processes on graphs," 2021.