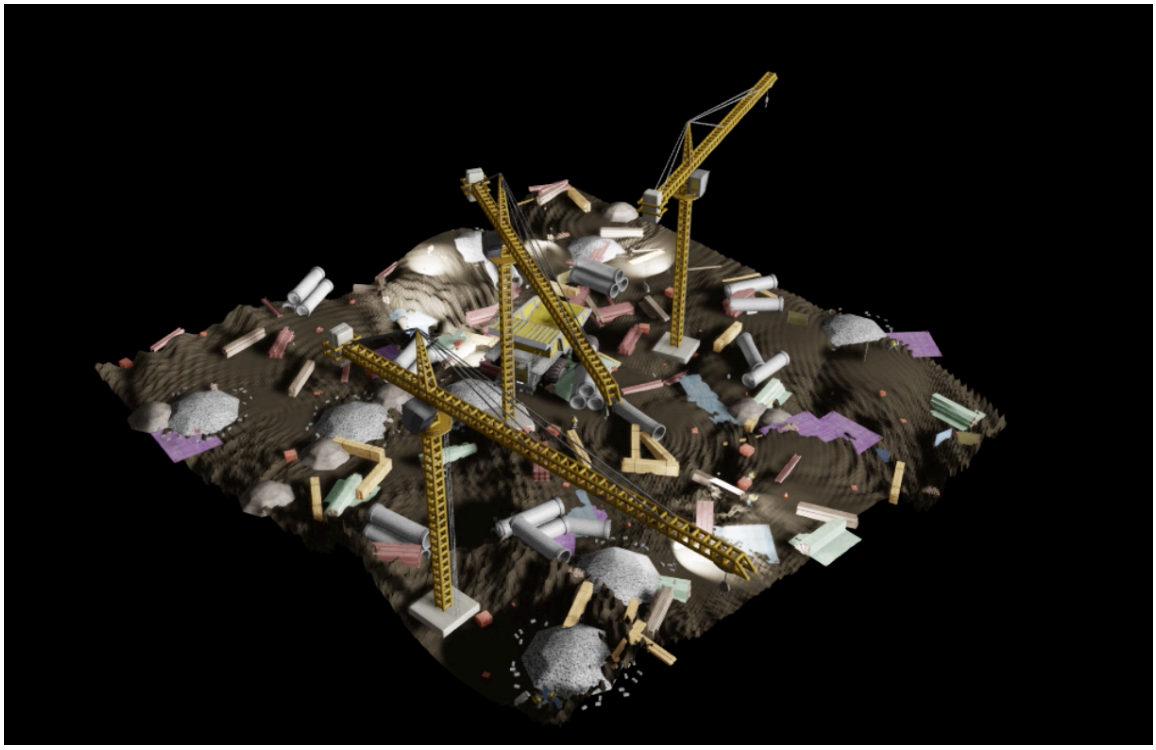
Basic Scene Generation for Construction Site Environments

Sean Brynjólfsson



Introduction

For our robot to predict how difficult it is to traverse a region from the sensors at its disposal, we implement a machine learned algorithm that uses point cloud data to make a prediction. For this writeup I will assume the details of our machine learning model. One way to train this model is to have pairs of LiDAR data and a corresponding traversability score for the region sampled by LiDAR.

Using a live robot to collect this data is undesirable—environments made to make the robot get stuck, run into things, or fall large distances involve a risk to the robot and also a large amount of supervision. Instead, an easy and efficient solution is to generate these data points via simulation. This requires that our simulation can produce generalizable results to the real world, so a simulation environment that generates a convincing scene and solves for the robot's ability to cross it is needed.

Our target application involves construction sites, so we will focus on them.

The Environment

We would like our robot able to navigate safely a construction site. We considered multiple ways to set up our simulation geared towards these environments. Browsing asset stores, it is possible to find individual paid models of construction sites, but these had many downsides, not limited to: flat terrain, no clutter, not enough variety, and cost.

Another option was to use real life scans, but these environments were unappealing because of their one-off nature, nondiversified challenges, lack of clutter, and incompatible formats.

We opted instead to purchase an asset pack of construction-site objects and code up a way to assemble them into a coherent scene. This way we would be able to parametrize our environments and leave room to extend the work.

Assets

We decided to try and create random scenes from real world assets for our research experiments. The 3D assets were purchased for around $10 from the unity asset store and converted to .usd files via the Omniverse Unity plugin which we accessed through Omni Developer—this conversion also is only able to be done on Windows through that application.

We also had to manually reassign textures to the 3D assets, but this was mainly for aesthetic purposes because in practice (I believe this was mentioned in our last call with David), the 3D point clouds we will have in practice do not contain color information. However, if RGB does become important at some point, we are ready for that. The framework we designed is set up to be able to import any .usd file with a valid physics material, allowing for further extension.

Most of the assets at our disposal. Not pictured: *Material Pile, Bucket, Concrete Bag*

Terrain

Our desired terrain should have the following characteristic to meet our needs:

- The terrain difficulty varies locally, containing areas of easy and challenging navigation to properly test our robot's capabilities.
- There is a safe place for the robot to easily spawn and navigate initially before moving to more challenging terrain.
- The terrain is represented by a continuous function which can be sampled to make the mesh and determine the height and orientation of assets.
- Can be tweaked procedurally to product a variety of terrains.

With these in mind, we salvaged code from IsaacGymEnvs that included a method for generating a mesh from a terrain. The implementation at the time had several fixed implementations of the terrain, but at its core utilized a height field which the mesh was built off of. We stripped all else away until we had a simple interface where all that was needed as input was height as a function of position.

Terrain Generation

Our terrain is generated by taking the product of a few simple components. The following are the broad definitions of these functions, glossing over the math itself.

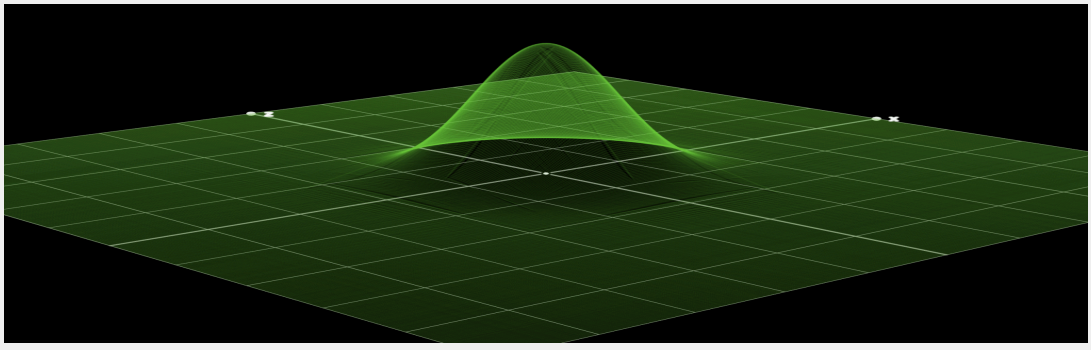
$P = x, y \in R^2$ *A variable point in space where the function is sampled.*

roughen$(P,\ (C, f)_{list}, r)$ *Function that roughens a region using a wavelet.*

- $C \in R^2$ *Centers to place roughing nodes.*
- $f \in [low,\ high]$ *Frequency of the wavelet (band limited)*
- $r \in R^+$ *Radius to roughen.*

wavelet$(P,\ C,\ f,\ r)$ *Function of a wavelet centered at C.*

- $C \in R^2$ *Center of the wavelet*
- $f \in [low,\ high]$ *Frequency of the wavelet (band limited)*
- $r \in R^+$ *Radius of the wavelet*

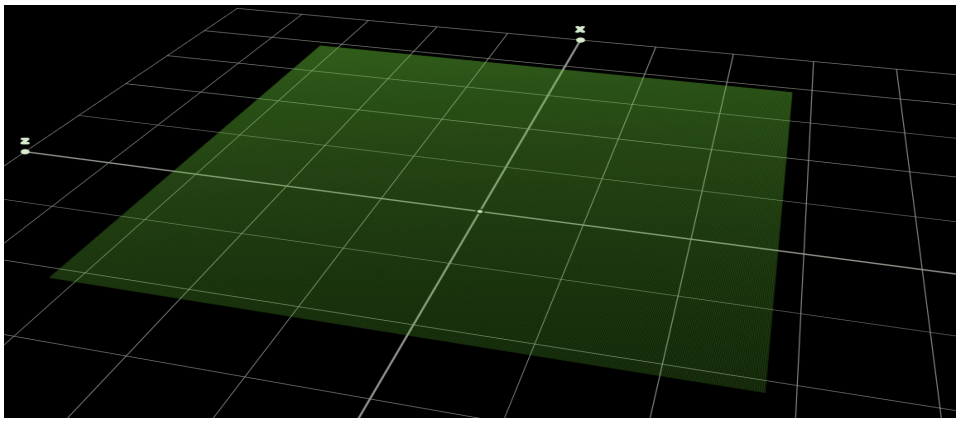


Section view of the **wavelet** function while varying the *frequency* parameter.

smoothen$(P,\ C_{list},\ r)$ *Function that smooths out a region*

$C_{list} \in R^{n\times 2}$ *Centers to place smoothing nodes*

$r \in R^{+}$ *Radius to smooth.*

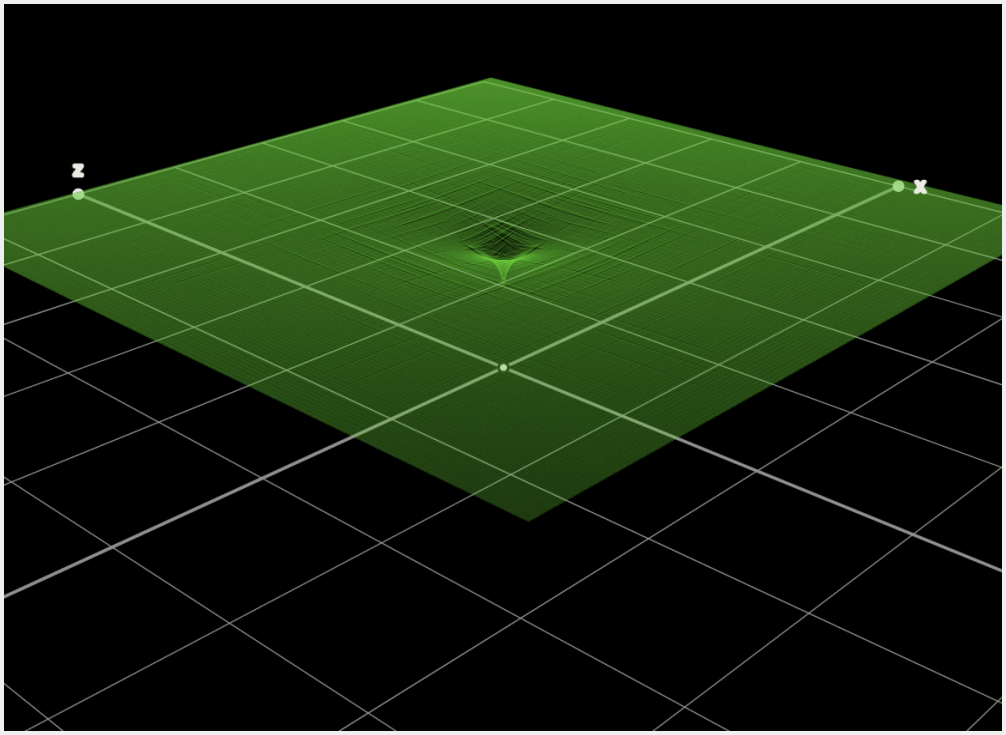


Varying the *radius* parameter in the **smoothen** function.

protect(P, r, d) *Function that protects the origin*

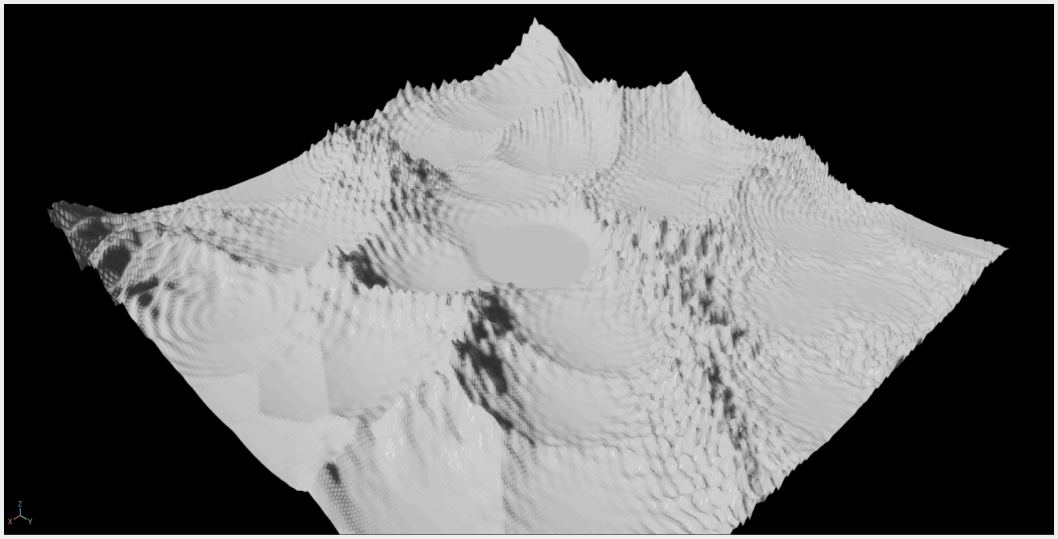
$r \in R^{+}$ *Radius to protect around the origin*

$d \in R^{+}$ *Rate at which protection decays (exponentially)*

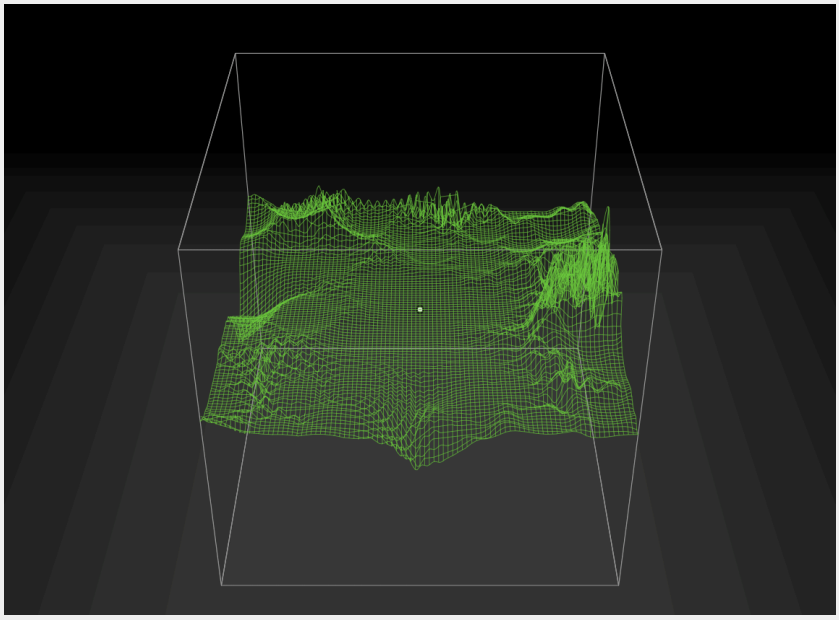


First varying the *radius* parameter in the **protect** function
and then varying the *decay* parameter.

terrain(P) *Function that samples the terrain at point P.*



A mesh of the terrain created by sampling the terrain function.



A wireframe from the Desmos twin of the terrain generator.

In order to rapid prototype and view the terrain, we coded up a twin on Desmos. Here's the link to the online version https://www.desmos.com/calculator/dyfjrj8fhk.

The Desmos version does not strictly follow the notation laid out above but the implementation is identical. Whatever is seen in the browser is what can be expected when generating the terrain. The corresponding parameters in the Desmos version can be found in the ▷ *Scene Parameters* folder.

Below is the method spec for the terrain function that is currently implemented.

`[wavelet_terrain_fn(...)(x,y)]` Function of terrain height at (x,y).

- `[xdim,ydim]` : Intended domain for the terrain's generation.
- `[x,y]` : Position on the plane. $x \in \left[-\frac{xdim}{2}, \frac{xdim}{2}\right]$, $y \in \left[-\frac{ydim}{2}, \frac{ydim}{2}\right]$
- `[amp]` : Amplitude multiplier.
- `[low, high]` : The range of frequencies the wavelets may take.
- `[num_rough, num_smooth]` : The number of rough/smooth patches.
- `[roughness, smoothness]` : Prominence (radius) of each rough/smooth patch.
- `[protect]` : Where to place the protected patch.
- `[protect_radius]` : The radius to leave flat around the protected patch.
- `[protect_decay]` : The protection decays more gradually as protect_decay increases.

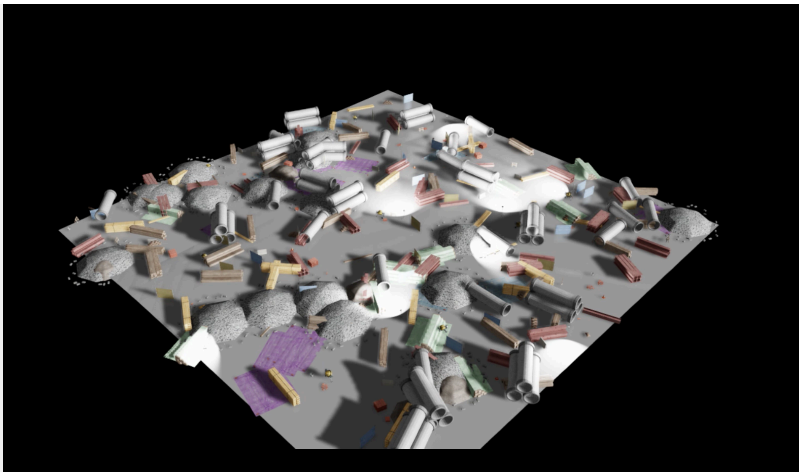
Asset Placement

Given that we have a directory of .usd assets, we first calculate the area of the bounding box of all available assets within the asset directory. We do this because when we are placing assets, we keep adding them until the total area of the bounding boxes exceeds a set fraction of the total scene area—this allows us to parametrize the density of our environment with human units.

Once we have calculated all of the asset areas, we create a distribution over our assets with probabilities inversely proportional to their area. It is important to weight the assets by their area in order to avoid sampling many large objects and quickly exhausting the scene's budget. We arbitrarily created three classes—small, medium, and large. Assets between 0 and 3 units were small; 3 and 70, medium; greater than 70, large. We then use a softmax to make probabilities. We found weighting the small and medium asset probabilities by 5x (pre soft-max) created a good balance.

After an asset is sampled for placement, we randomly select a point in the scene and set its height to the average height of the function estimated by random sampling within a square of equal area. This allows for small objects to be placed on top of high frequency features while large objects settle closer to the average height. The objects are also assigned a random rotation about the vertical axis, but no alignment to the normal has been implemented yet.

Demonstration of asset placement with the density parameter set to 35% and terrain effects removed.



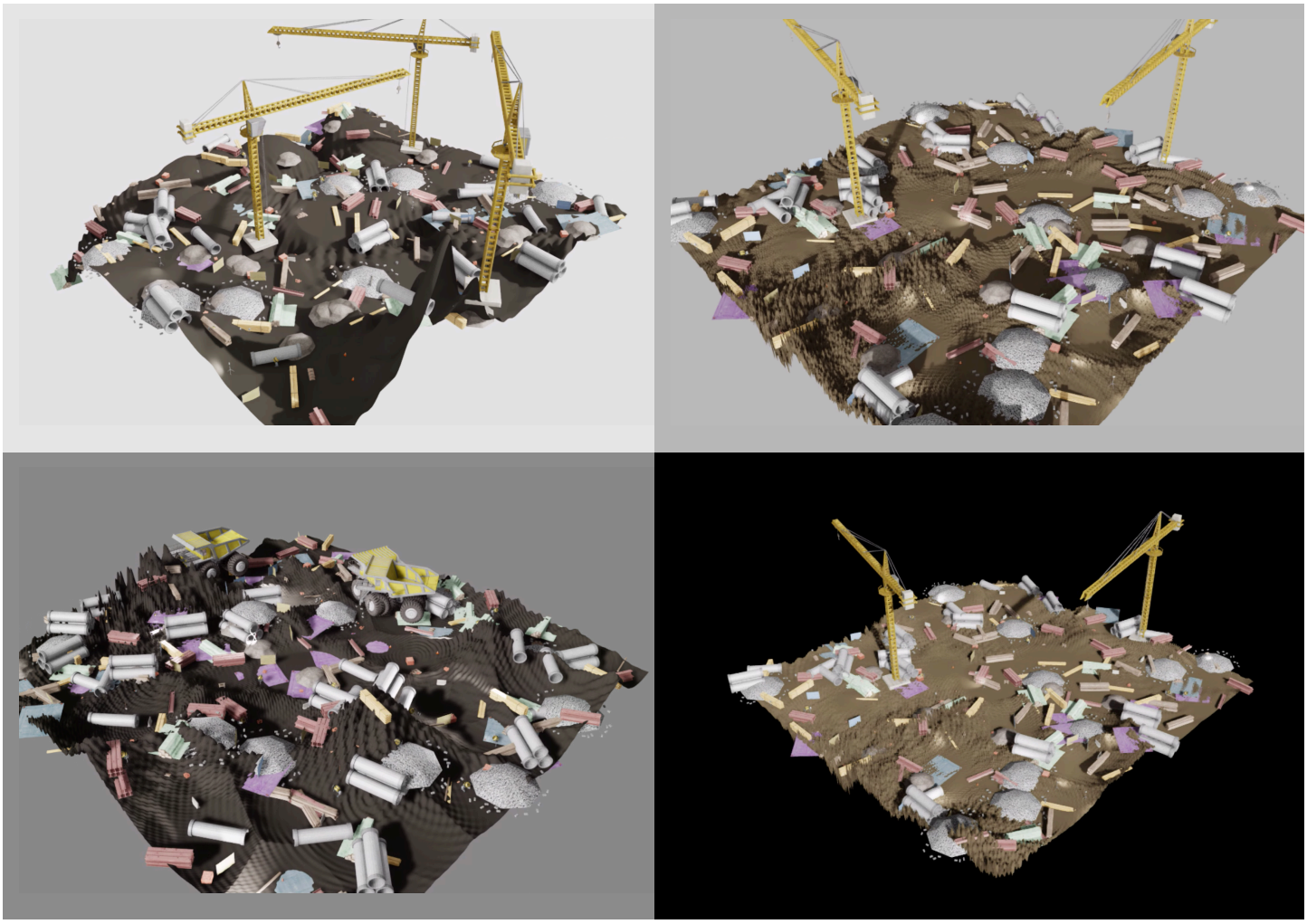
Current Thoughts

In our current traversability learning model, we have not yet trained on our construction environments. Consequently, we are unable to evaluate the level of difficulty these environments may pose or how well we can traverse an unseen arrangement—and much less a real world scene.

Furthermore, we identify a fundamental difference between these environments and those found in real life. Specifically, the terrain is often highly periodic and limited in roughness by mesh resolution. The assets within these environments are also not always spawned in realistic poses, for example the occasional protrusion of long objects like planks from hills due to the absence of proper consideration of normals or solid objects intersecting in impossible ways.

We also lack many common construction site occurrences, but most importantly for traversability concerns, stairs, ledges, and interior spaces.

Gallery



Next Steps

There are several good directions to work towards. Currently, from highest to least priority, they are:

- Finish codebase refactor (already in progress).
- Generate mesh and apply physics texture headlessly (currently manually applying/saving the meshes).
- Implement mesh concatenation to prepare for IsaacGym.
- Consider whether or not creating a construction site model is necessary for training or if it could be a testing target while the training is done in an environment which could be much more unstructured.
- Improve Environment (not in any order)
    - Do a literature review on what features are important to traversability in the real world, more emphasis on smaller environments with more details for rapid batching and realism.
    - Do a literature review on structured domain randomization and improve the tools for parametrizing the environment and making consistently diverse challenges.
    - Design a new terrain function. The current function is slow, needlessly complicated, and derived from nothing more than a smorgasbord of functions that produce an undulating terrain which is in some places difficult to cross. More care should be taken here.