



# REPLACING PERLIN NOISE WITH AI

---

Hyper-Realistic, Fast,  
and Infinite

# What is Perlin Noise?

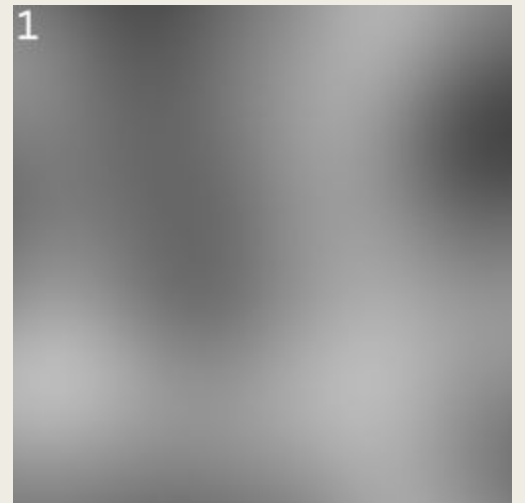
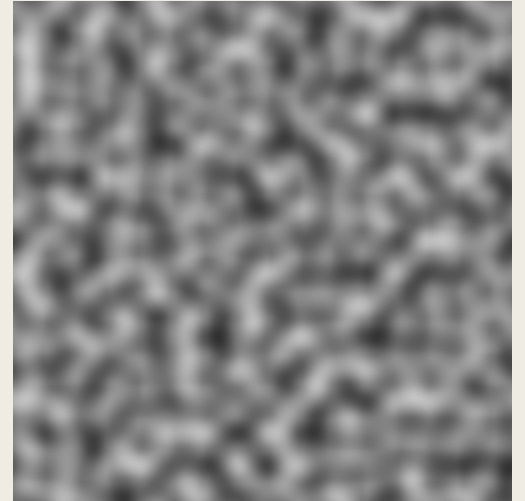
## 1. A Classic Procedural Tool

Developed in the 1980s by Ken Perlin, Perlin noise is a smooth, pseudo-random function used to generate natural-looking textures and terrain in computer graphics and games.

## 2. Why It's Popular

- Simple and fast to compute
- Seamless and tileable
- Looks “organic”
- Often used for procedural terrain

Often stacked to create  
variety



# Why Procedural Terrain Matters

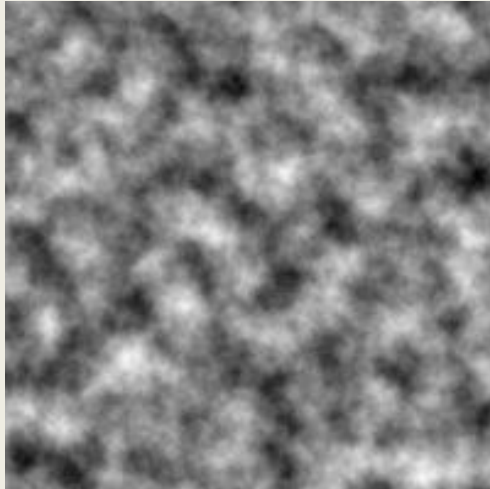
- **Games:** Infinite, procedural worlds (e.g Minecraft, Terraria, No Man's Sky, etc.)
- **Simulations:** Terrain needed for physics, pathfinding, climate, etc.
- **GIS / Digital Twins:** Scalable Earth Models for research or urban planning



# The Problem with Perlin Noise

## Traditional Noise

- Bloppy, repetitive patterns
- Unscaled (values between 0 and 1)
- **Boring, even with post-processing**



## Real Terrain

- Varied, multi-scale details
- Realistic, wide-ranging elevations
- **Awesome out of the box**



# My Goal

**Generate infinite, realistic terrain in real time.**

**Requirements:**

- Fast enough for interactive use
- Infinitely tileable
- Consistent: Given a seed and position, the elevation is always the same, no matter what order generation occurs
- Realistic

# Key Challenges

- Precision: Real terrain ranges in elevation from -10,000m to 10,000m. Diffusion models and GANs will struggle to obtain ~1m accuracy.
- Scalability: Tiling diffusion models has been done before, but extending that to infinite domains is complex.
- Efficiency: Diffusion models are slow. They become *massively* slower when tiling.
- Quality: Most terrain looks boring. We want to bias our model to prefer generating interesting terrain.

# Precision Issues

- Elevation data spans from -10,000m to 10,000m
- When normalized to have mean=0 and std=1, a 0.01 error is magnified to a 30m elevation error.
- Small features like ~50m hills are visually important to humans, but represent <1% of elevation range, and are largely treated as noise by models.
- A shoreline is the difference between -1m and 1m. Imperceptible to models, but critical for humans.

# Solution (Part 1)

## Laplacian Encodings

- Decompose elevation into:
  - *Low frequency: Global shape, larger stdev*
  - *High frequency: Local details, smaller stdev*
- Solves error magnification issues:
  - *Noise in low frequency -> Removed by deleting high frequency component*
  - *Noise in high frequency -> Smaller value range, not amplified as much*
- How?
  - *Low frequency map generated by resizing image and blurring*
    - Resize from 512x512 to 64x64
    - Blur image. I use sigma=5.
  - *High frequency map generated by taking difference between original heightmap and low frequency map*
    - Low-freq map is resized back to 512x512 first. Bilinear interpolation works fine.

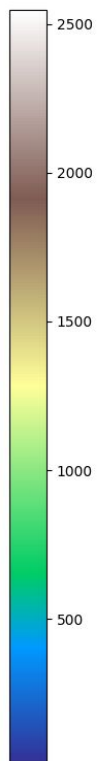
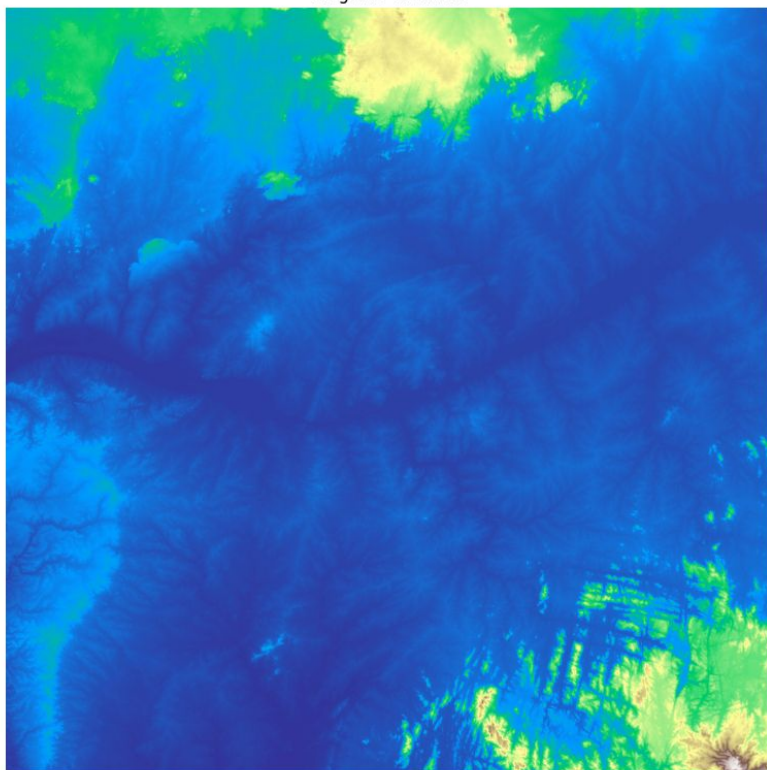


# Solution (Part 2)

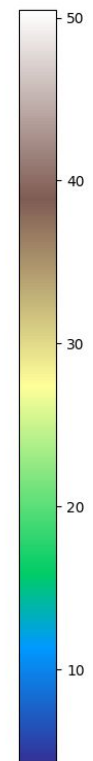
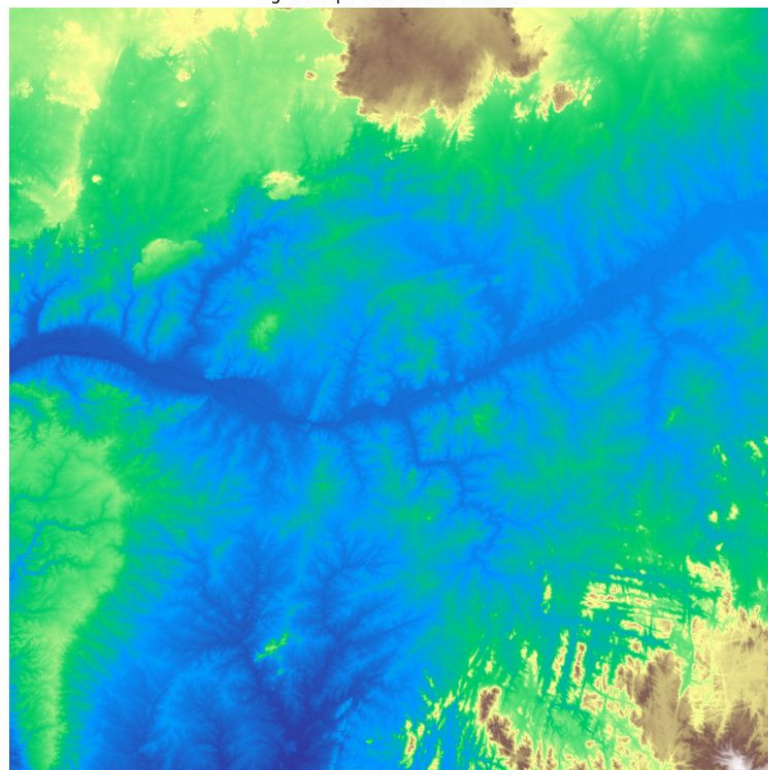
## Signed-Sqrt Transform

- Apply  $f(x) = \text{sign}(x) * \sqrt{\text{abs}(x)}$ 
  - *Can invert with  $f(x) = \text{sign}(x) * x^2$*
- Expands low-lying features
  - *0-100m now covers ~10% of elevation range, so the model can dedicate more capacity to subtle but visually important terrain.*
- Decouples mean and variance
  - *Before, variance in a crop was strongly tied to its mean elevation. Now, variance is more uniformly distributed.*
- Improves shorelines
  - *Before, -1m to 1m was imperceptible. Now, its ~2% of the total elevation range. Enough for models to represent clean coastlines.*

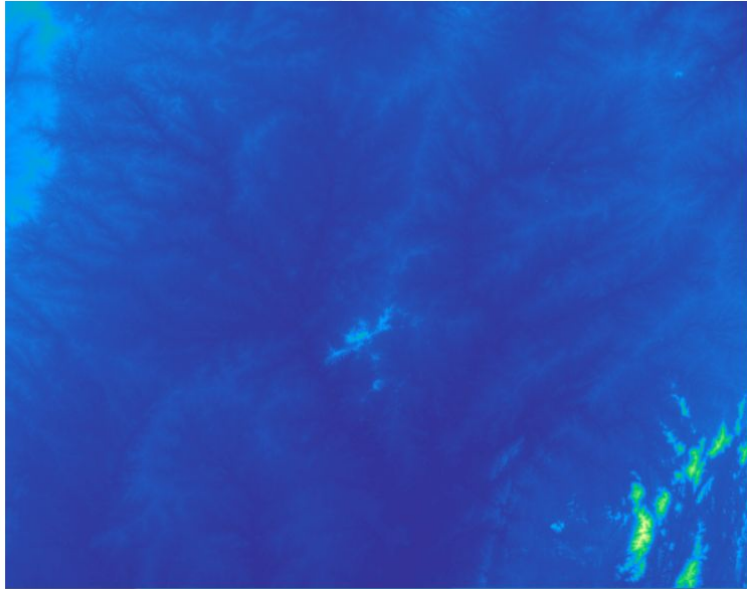
Original - 1881.tif



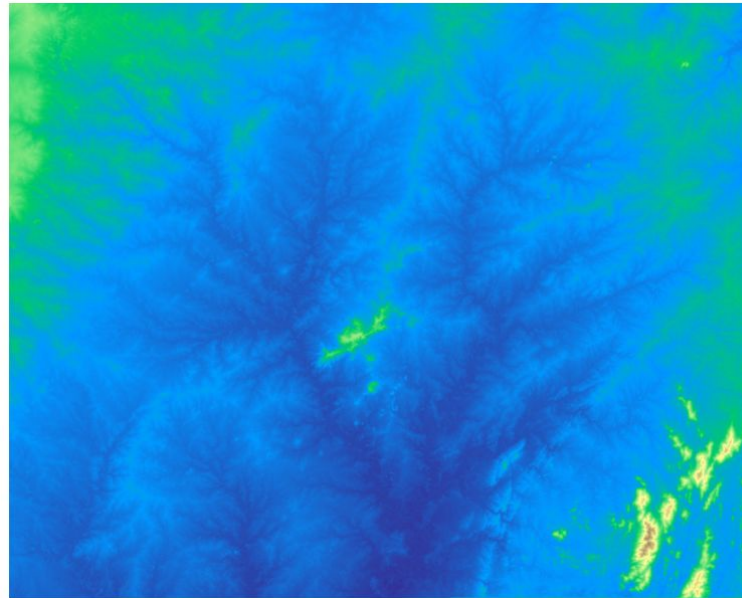
Signed Sqrt Transform - 1881.tif



Original - 1881.tif



Signed Sqrt Transform - 1881.tif



# Scalability

Problem: We don't want one patch of land, we want an entire planet.

Solution: **Multi-Diffusion**

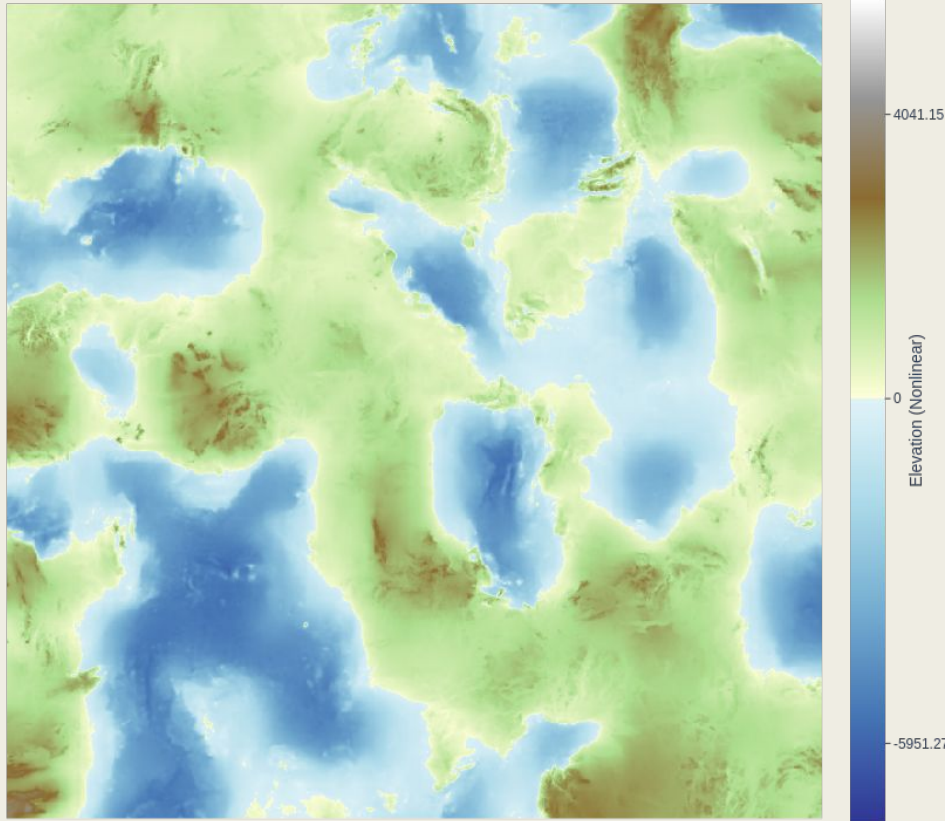
- Key Idea: Train a model to predict the noise of patches of images. During inference, combine all predictions into one large prediction, covering the full image.
  - Overlap in tiles is necessary: This is how the models "collaborate"
  - Mismatched predictions in overlapping regions are solved by taking the average
    - Even better: Take a weighted average. Predictions close to the center of a region weighted > predictions near the edge.
- Can we extend this to infinity? **Yes. With difficulty.** To denoise a tile to step  $x$ , need all surrounding tiles to have been denoised to step  $x-1$ .
  - Has a cascading effect: To generate one tile in 30 steps, need to call diffusion model **10,000 times**. We will come back to this problem.

Getting Multi-Diffusion to work in unbounded spaces was complicated. I wrote a small library to help.

## **Infinite Tensors:**

- Divides infinite dimensions into chunks (tiles)
  - Works just like a normal tensor, but some dimensions can be 'None' (Infinite)
  - Tiles are stored in memory or on disk
- Processes data in windows using any provided function (e.g diffusion model)
- Manages dependencies between tensors
  - Can generate new infinite tensors as products of any number of other infinite tensors (and other arguments)
- Automatically cleans up unused tiles to conserve memory/disk





Early prototype of diffusion tiling

Problem: When tiling with diffusion model, terrain statistics are off.

Real terrain range: -10000m to 10000m  
Generated terrain (Left): -6000m to 4000m

**Extreme values extraordinarily rare:  
Why?**

Core Idea: When generating one tile, the mean elevation of the output is highly correlated with the mean of the initial noise (call this mean noise for short)

# What about two adjacent tiles?

Can be a mismatch:

- One tile has very positive mean noise
- One tile has a very negative mean noise

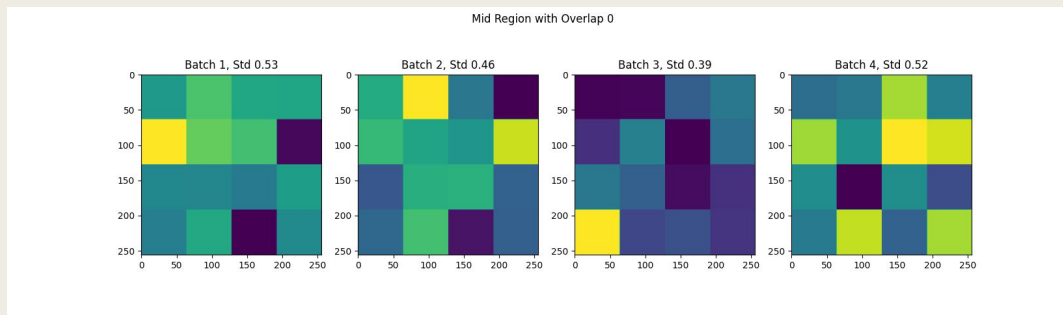
One tile tries to generate Mt Everest, the other wants to generate the Mariana Trench.

What about the overlapping region?

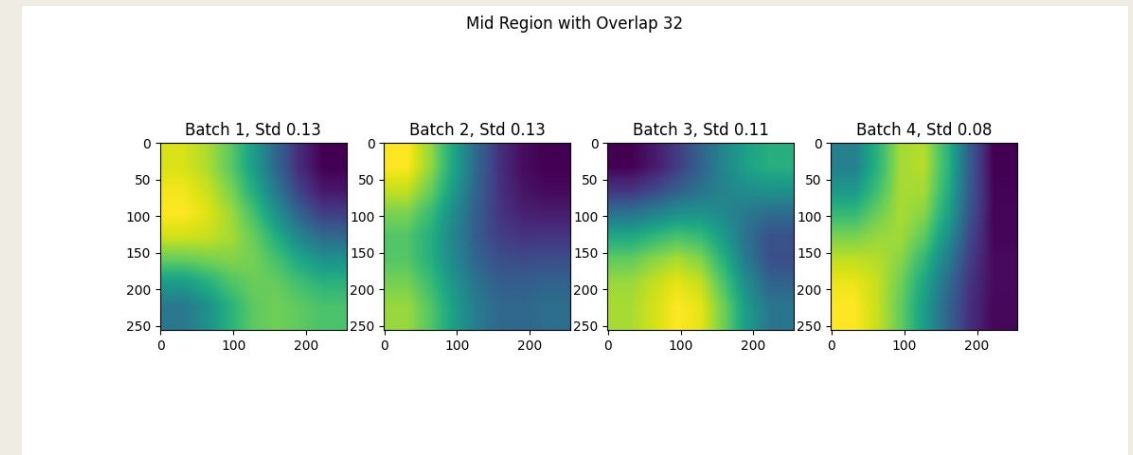
- Has to be a transition somewhere (e.g a shoreline)
- But you can't have Mt Everest that close to a shore; nor can you have the Mariana Trench next to a shore
- Eventually: Diffusion paths compromise. Both features are muted, and we end up with a small mountain and small trench, or nothing at all.
- In practice: Extremely unlikely a high-elevation tile will be surrounded by other high-elevation tiles. So extreme tiles tend to be muted.

# Toy Example

Generating without  
overlap: Std = 0.5



Generating with  
overlap: Std = 0.1





# Solution:

## Condition diffusion models on mean elevation

Now adjacent tiles are aligned.

But how do we get the mean elevation of a tile before generation?

- Train a GAN on small (e.g 12x12), ultra-low resolution (~16km pixels) patches of the world
  - *Small resolution makes the model incredibly fast*
- Remove padding from the model to make it translation invariant
  - *Allows for native infinite generation*
- Use R3GAN (recent work) to maximize quality and stability.
- Negligible performance impact.

# Efficiency

Problem: 10,000 diffusion steps to generate one tile is way too much.

Solution: Consistency models

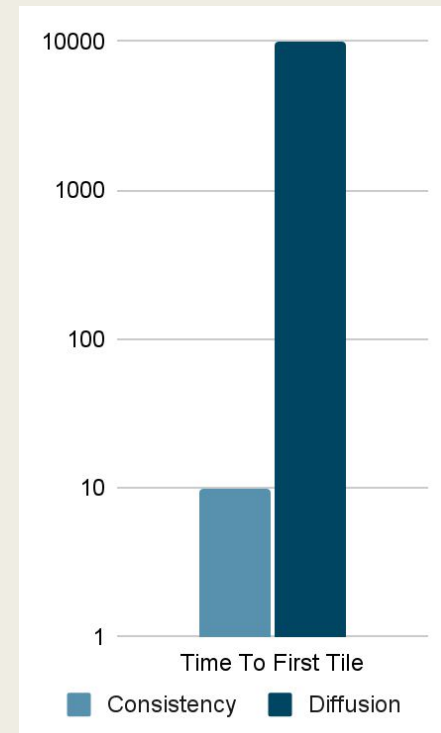
What is a consistency model?

- A recent family of models for fast generative sampling
  - *Same architecture and data as diffusion models*
- Requires only 1 or 2 steps to generate images, not 30.
- Learns to generate images directly by minimizing the difference in the predicted image between two timesteps

Why do we need this?

- Generating 1 tile with diffusion models: 10,000 calls
- Generating 1 tile with 2-step consistency models: 10 calls
- When generation is unbounded, a 15x speedup becomes more like 1000x.

**Issue: Discrete CMs magnify FID by 2-3x**



# Continuous Consistency Models (sCM)

- Consistency models can be made continuous
  - Avoids discretization errors
  - Prone to training instability - **not good in practice**
- **Until *Lu et al.* proposes a fix for training instability (The sCM paper)**
  - FID now increases by just 10% for 2-step CMs

I wrote the **first public implementation** of the sCM paper

- Adapted it for use in terrain diffusion
- Terrain is now **high quality** and **generates in real-time**.

# Quality

Problem: Most terrain is boring

Solution: Bias the model towards generating high-quality terrain

- Want to rate terrain on 1-5 scale
  - *1 indicates terrain is boring – flat, not a lot going on*
  - *5 indicates terrain is crazy cool – grand canyon, the Andes, etc.*
- Terrain data is stored in ~4096x4096 tiles at 90m resolution, or 368x368km.
  - *Very large scale. Not worried about individual mountains or lakes, but entire mountain ranges, canyons, and small countries.*
- Start by rating ~100 tiles manually
- Gather some simple features: Take FFT, bin by distance from center, use mean power as feature
  - *power = log(abs(fft))*
  - *Good metric for how varied terrain is*
- Add in stdev, mean elev, 1/stdev, log(stdev), etc.
- Linear regression on manual ratings is surprisingly accurate!  $R^2 = 0.8$
- **During training, add extra weight to samples with high ratings**
  - *I chose to pick a random rating, then pick a random tile with that rating*