Procedural Textures with Perlin Noise

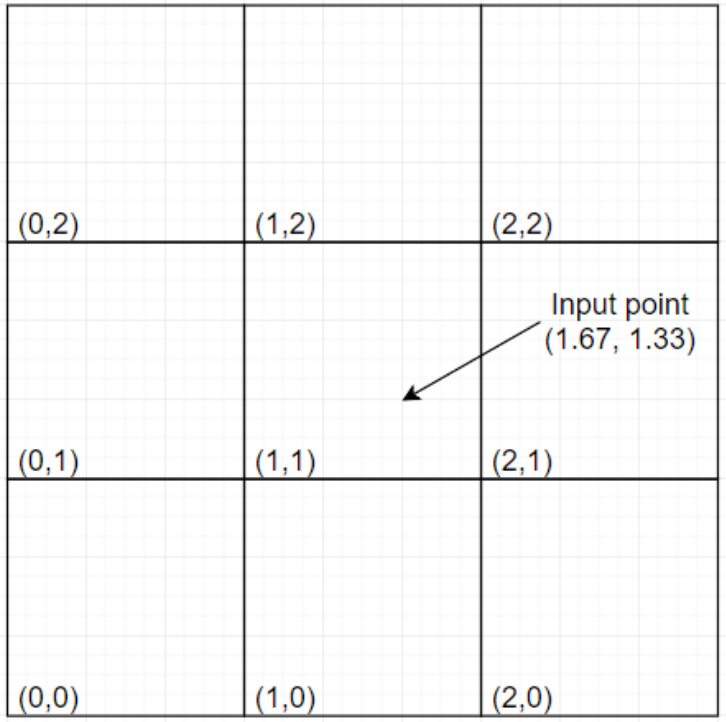
Manjot Bal

UNIVERSITY OF CALGARY

Perlin Noise

- Input an n-dimension vector
- Outputs a pseudo-random number.
- Same input always produces the same output.

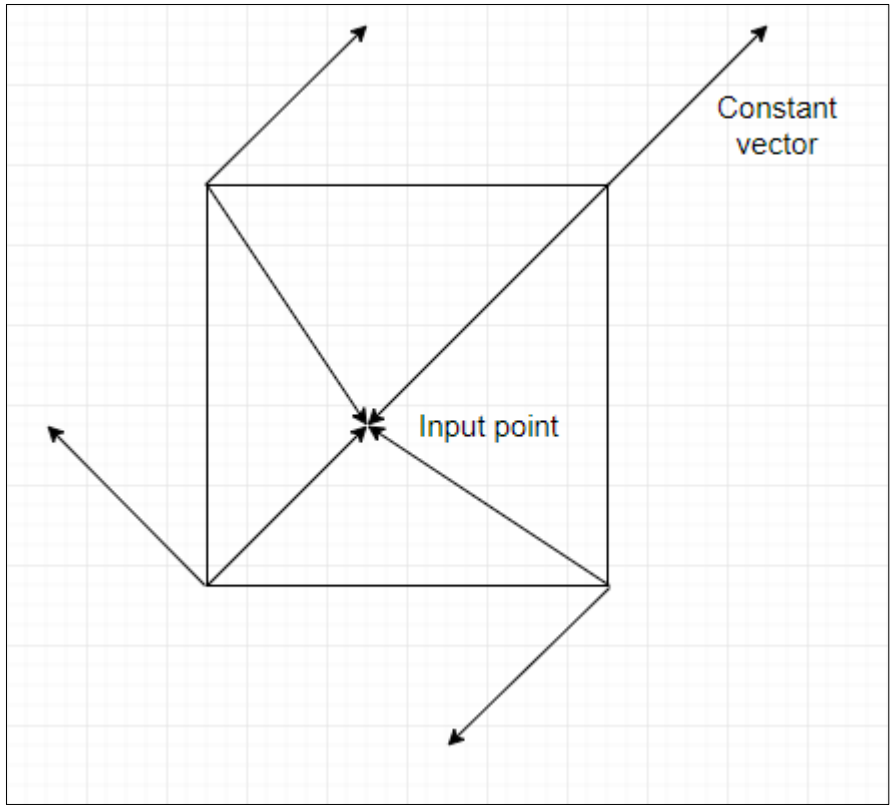
Perlin Noise



https://rtouti.github.io/graphics/perlin-noise-algorithm

- Object is placed inside a integer lattice grid.
- Each corner has hash value which is used to get a constant vector for the corner.

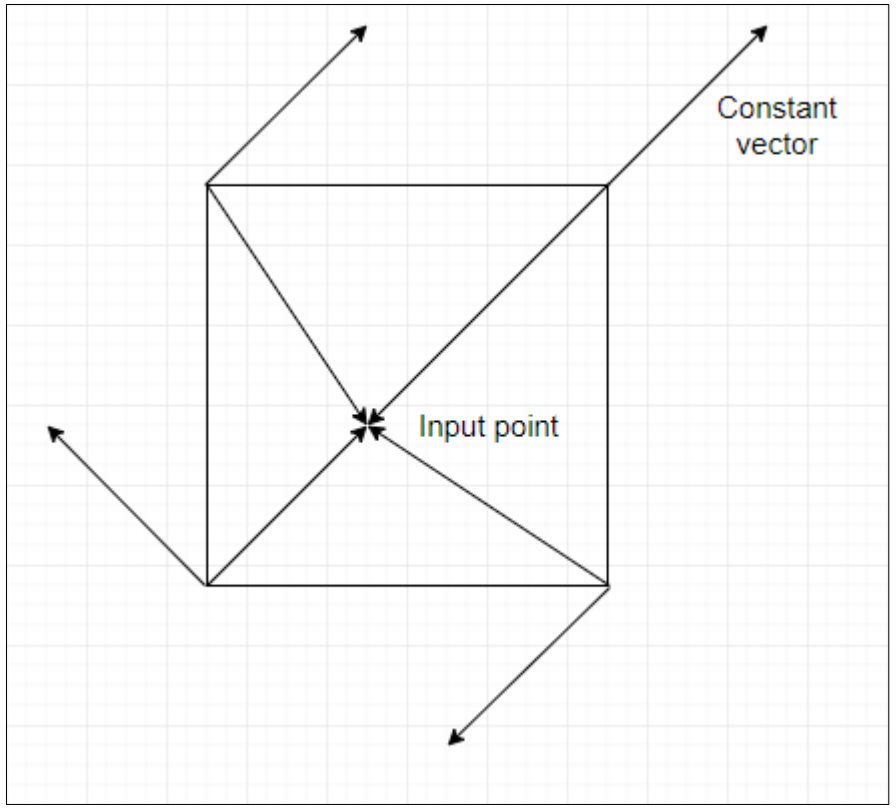
Perlin Noise



https://rtouti.github.io/graphics/perlin-noise-algorithm

- In 2d we get 4 vectors from each corner to the input point.
- Then take the dot product of the constant vector with the vector from the corner to the point.
- Corner shared between two grid cells must have the same constant vector.

Perlin Noise

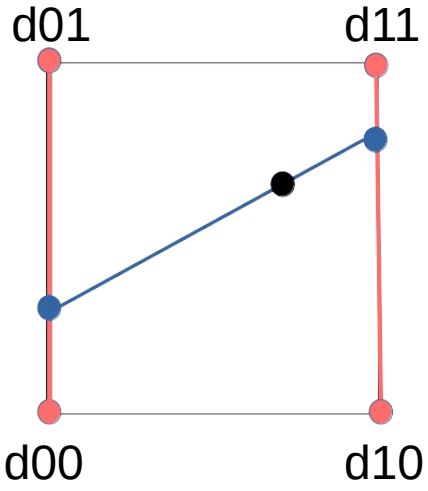


https://rtouti.github.io/graphics/perlin-noise-algorithm

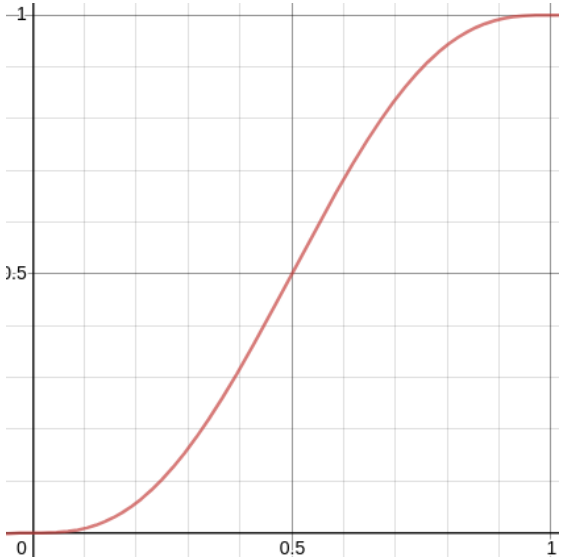
- Randomness comes from a permutation matrix.
- Suppose grid is 256x256 and input point is p=(x,y). We then create an array:
  - int[256] perm = {0,1,..,255}
- Shuffle the array.
- Bottom left has hash value:
  - perm[perm[xi] + yi].
- Bottom right has hash value:
  - perm[perm[xi+1] + yi].

Perlin Noise - Interpolation

- Linear Interpolation
  - p=(x,y), pfrac=frac(p)
  - y0=lerp(d00, d01, pfrac.y)
  - y1=lerp(d10, d11, pfrac.y)
  - x0=lerp(y0, y1, pfrac.x)



- First ease then lerp.
  - u=ease(pfrac.x), v=ease(pfrac.y)
  - https://easings.net/

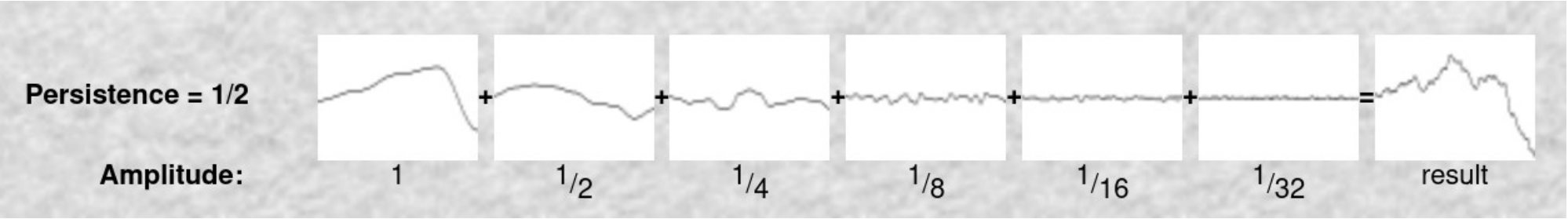


$y=6t^5-15t^4+10t^3$

UNIVERSITY OF CALGARY

Perlin Noise - Turbulence

- Add together several noise functions of varying frequencies and amplitudes.
  - frequency = 2^i
  - amplitude = persistence^i

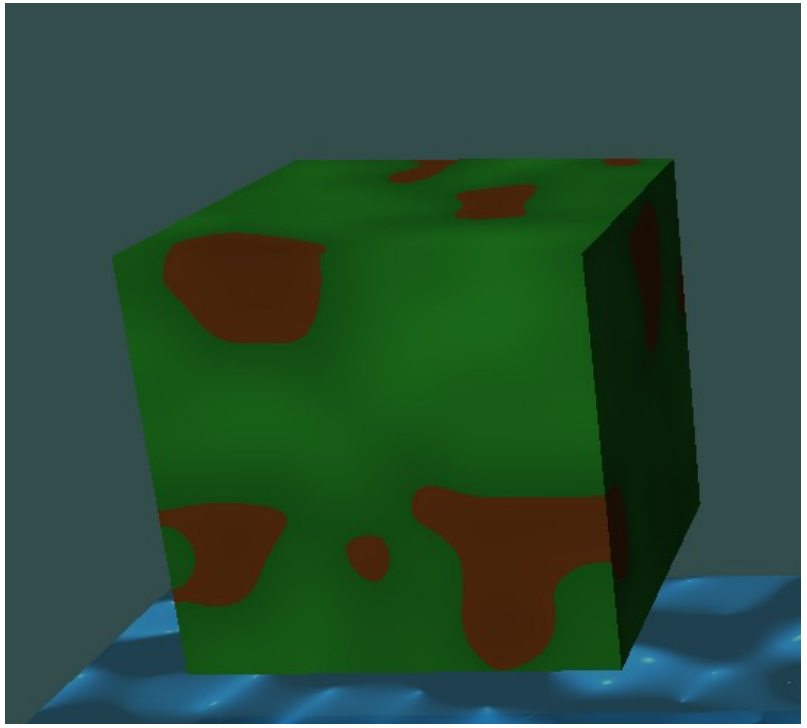


http://www.arendpeter.com/Perlin_Noise.html

Grass/Terrain Texture

- Use Perlin noise to create a grass texture with some dirt.

```
vec3 green, vec3 brown
turb = turbulence(point, persistence)
if (turb < persistence * cutoff)
        return brown * turb
 else
         return green * turb
```



Wood

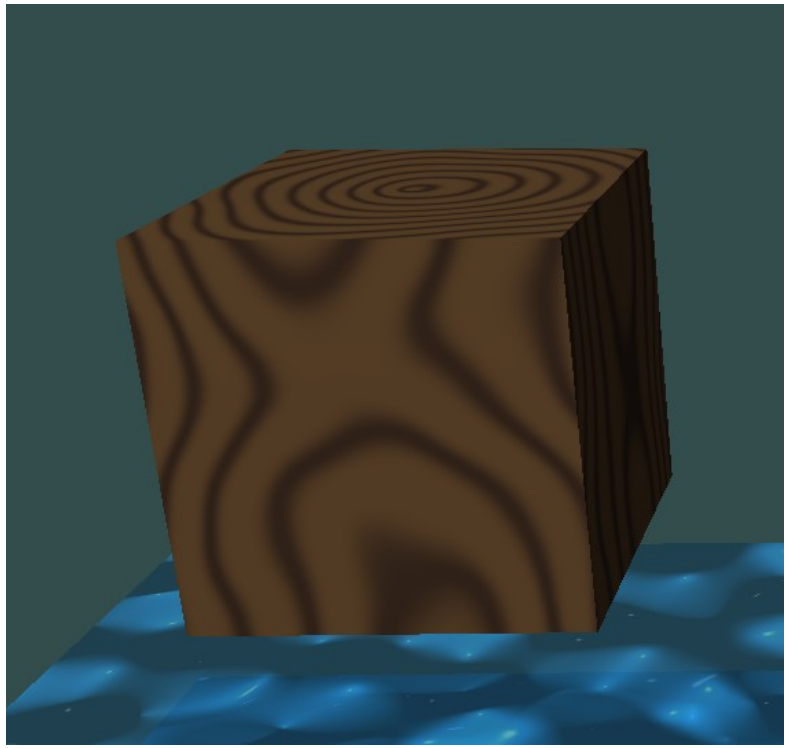
Create rings for 2 dimensions

turb = turbulence(point, persistence)

dist = length(point.xz) + turb

value = (cos(dist * freq) + 1) * 0.5

return mix(light, dark, value)



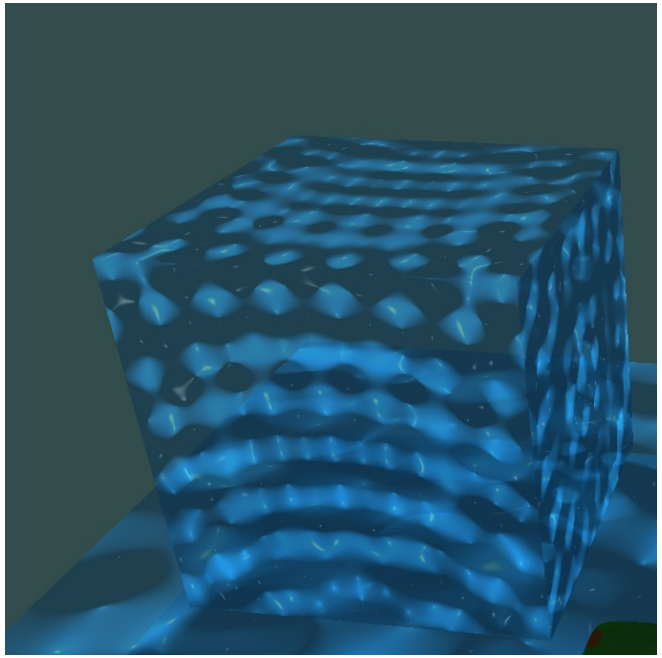
Water

- Add noise to the normals in a cyclic pattern.

  For each center

  ```
  vec3 center = normalize(diffNoise(i * vec3(100,0,0)))
  vec3 toPoint = point – center
  disp = normalize(toPoint) * cos(length(toPoint)*freq – time*speed)
  normal += disp
  ```

- Add several centers around a unit sphere to get wave interference



References

(1) Ken Perlin. 1985. An image synthesizer. SIGGRAPH Comput. Graph. 19, 3 (Jul. 1985), 287–296. DOI: 10.1145/325165.325247

(2) Ken Perlin. 2002. Improving noise. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques (SIGGRAPH '02). 681–682. DOI:10.1145/566570.566636

- Source Code: https://mrl.cs.nyu.edu/~perlin/noise/

(3) https://rtouti.github.io/graphics/perlin-noise-algorithm

(4) http://www.arendpeter.com/Perlin_Noise.html

(5) http://luthuli.cs.uiuc.edu/~daf/courses/computergraphics/week8/shading.pdf

(6) http://www.sci.utah.edu/~leenak/IndStudy_reportfall/Perlin%20Noise%20on%20GPU.html

(7) https://lodev.org/cgtutor/randomnoise.html#Wood