Appendix B:
Perlin Noise

By mapping 3D coordinates to colors, we can create *volumetric texture*. The input to the texture is local model coordinates; the output is color and surface characteristics.

For example, to produce wood-grain texture, trees grow rings, with darker wood from earlier in the year and lighter wood from later in the year.

- Choose shades of early and late wood
- $f(P) = (X_P^{\,2}+Z_P^{\,2}) \bmod 1$
- $color(P) = earlyWood + f(P) * (lateWood - earlyWood)$

Adding realism

The teapot on the previous slide doesn't look very wooden, because it's perfectly uniform. One way to make the surface look more natural is to add a randomized noise field to f(P):

$$f(P) = (X_P^2 + Z_P^2 + noise(P)) \bmod 1$$

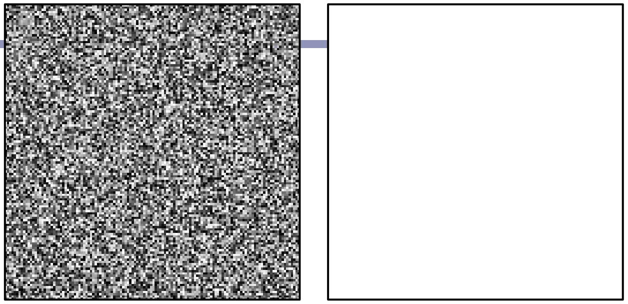
where *noise(P)* is a function that maps 3D coordinates in space to scalar values chosen at random.

For natural-looking results, use *Perlin noise*, which interpolates smoothly between noise values.

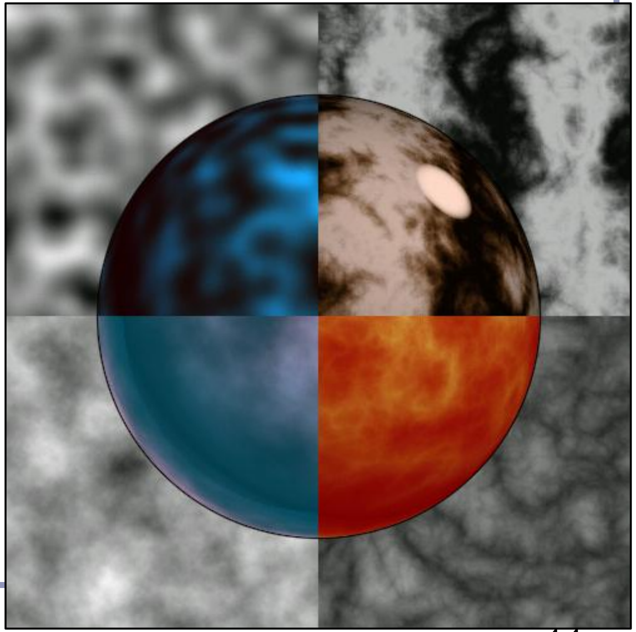
Perlin noise

Perlin noise (invented by Ken Perlin) is a method for generating noise which has some useful traits:

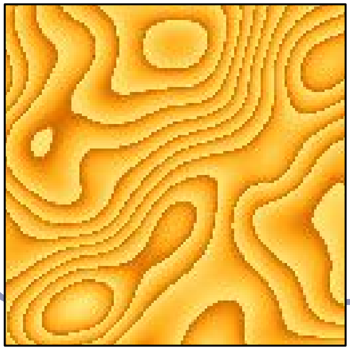
- It is a *band-limited repeatable pseudorandom* function (in the words of its author, Ken Perlin)
- It is bounded within a range close [-1, 1]
- It varies continuously, without discontinuity
- It has regions of relative stability
- It can be initialized with random values, extended arbitrarily in space, yet cached deterministically
  - Perlin's talk: http://www.noisemachine.com/talk1/



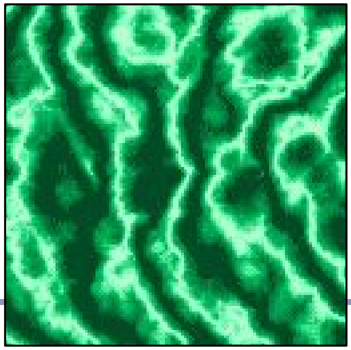
Non-coherent noise (left) and Perlin noise (right)
Image credit: Matt Zucker



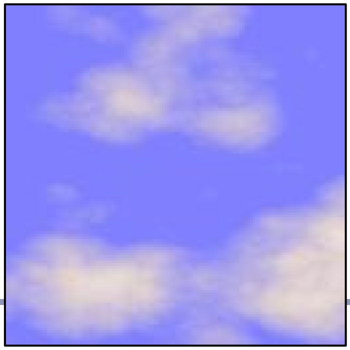
Ken Perlin



Matt Zucker



Matt Zucker



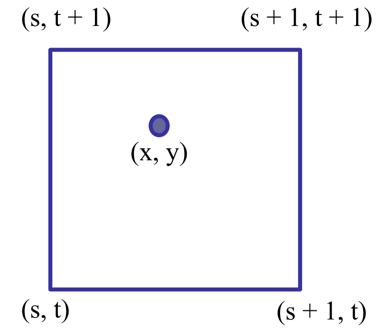
Matt Zucker

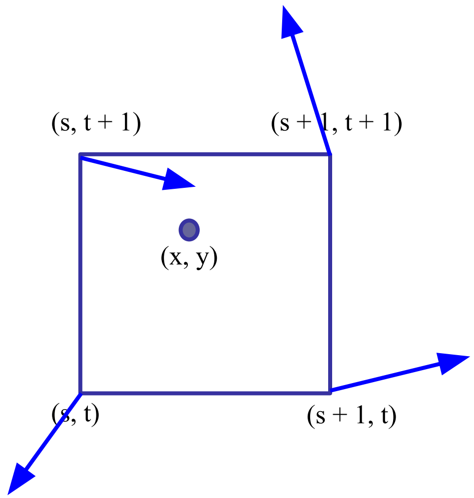
Perlin noise 1

Perlin noise caches 'seed' random values on a grid at integer intervals. You'll look up noise values at arbitrary points in the plane, and they'll be determined by the four nearest seed randoms on the grid.

Given point *(x, y)*, let *(s, t) = (floor(x), floor(y))*.

For each grid vertex in $\{(s, t), (s+1, t), (s+1, t+1), (s, t+1)\}$ choose and cache a random vector of length one.



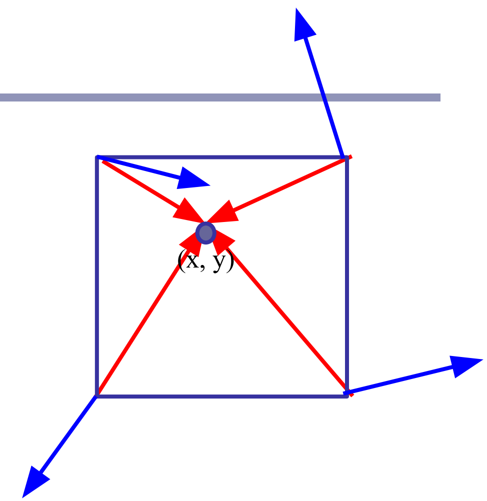


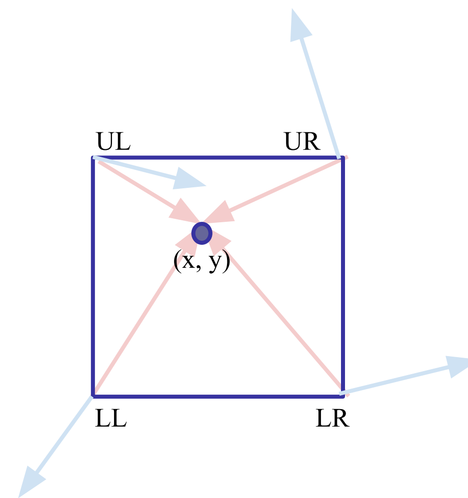
These slides borrow heavily from Mark Zucker's excellent page on Perlin noise at http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html

Perlin noise 2

For each of the four corners, take the dot product of the random seed vector with the vector from that corner to *(x, y)*. This gives you a unique scalar value per corner.

- As *(x, y)* moves across this cell of the grid, the values of the dot products will change smoothly, with no discontinuity.
- As *(x, y)* approaches a grid point, the contribution from that point will approach zero.
- The values of *LL, LR, UL, UR* are clamped to a range close to [-1, 1].





These slides borrow heavily from Mark Zucker's excellent page on Perlin noise at http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html

Perlin noise 3

Now we take a weighted average of *LL, LR, UL, UR*. Perlin noise uses a weighted averaging function chosen such that values close to zero and one are moved closer to zero and one, called the *ease curve*:

$$S(t) = 3t^2 - 2t^3$$

We interpolate along one axis first:

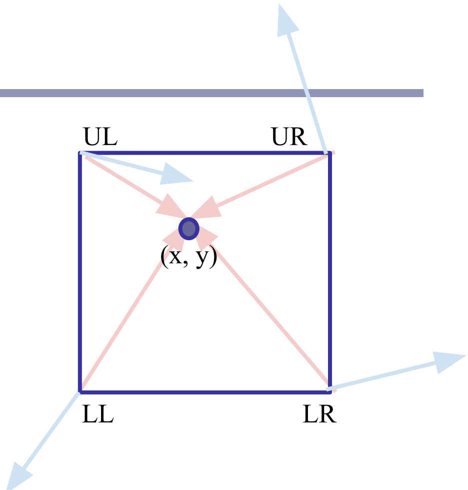
$$L(x, y) = LL + S(x - \mathit{floor}(x))(LR-LL)$$

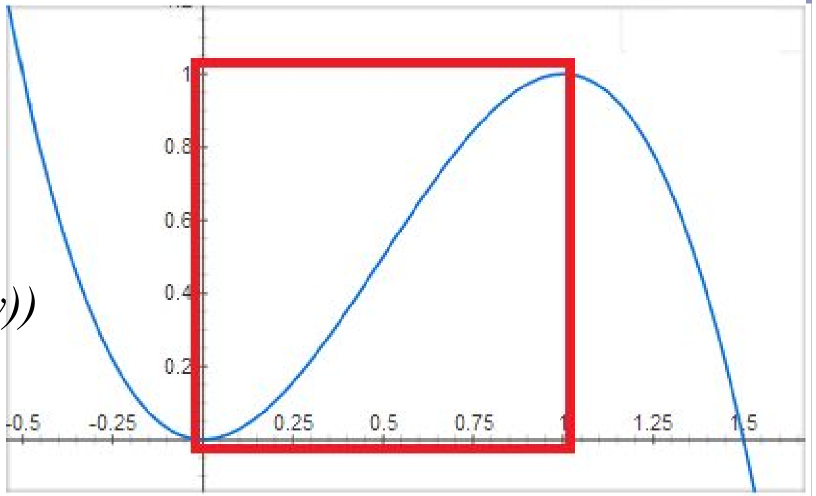
$$U(x, y) = UL + S(x - \mathit{floor}(x))(UR-UL)$$

Then we interpolate again to merge the two upper and lower functions:

$$\mathit{noise}(x, y) = L(x, y) + S(y - \mathit{floor}(y))(U(x, y) - L(x, y))$$

Voila!





The 'ease curve'

Perlin Noise - References

- https://web.archive.org/web/20160303232627/http://www.noisemachine.com/talk1/
- http://webstaff.itn.liu.se/~stegu/TNM022-2005/perlinnoiselinks/perlin-noise-math-faq.html