Steps in the coding implementation of Perlin Noise:

1. Initialize a temporary table. Usually with entries 0 to 255. Shuffle these entries.
2. Initialize a permutationTable. This table contains double the entries of the temporary table. This is because when you perform table look-ups to get the values to inerpolate between you do this in the following manner for 3D: permutationTable[x + permutationTable[y]] + z; meaning that y could possibly resolve to maximum number 255 and if x is also 255 then you need to lookup a number higher than 255. Even for dimensions higher than 3D however, your permutationTable stays at size double the temporary table.
3. Generating the noise. Now that we have permutationTable set up, we can pass in a point (x,y,z) to get noise from it.
4. All values of x,y,z we ensure that they are between 0 to 255 in value (essentially by converting the doubles to integers with the floor function and then performing x
5. Calculate the distance of each component to the floor of the component
6. compute a fade curve for these distance values
7. obtain the values at the 8 closest points in the hypercube of the 3D shape by doing permutationTable[X + permutationTable[Y]] + Z; for each permutation of (X,Y,Z) and (X + 1, Y + 1, Z + 1);
8. Linearly interpolate between first the sides of the cube, then the squares of the cube, then the entire cube.
9. Return a normalized result.

Features of simplex noise:

1. Faster than Perlin noise. Perlin noise has $O(n \cdot 2^n)$ time complexity with n being the dimension. Simplex noise has $O(n^2)$ time complexity. Simplex noise is much faster than Perlin noise when you generate noise on higher dimensions.
2. Interpolation can only be performed in 1D so if you are performing Perlin noise generation you require $2^n - 1$ interpolations per pixel. Simplex noise requires $n + 1$ radial attenuations per pixel.

Challenges of simplex noise:

1. The largest challenge associated with simplex noise is finding the closest grid points to each pixel. In Perlin noise since you are using a hypercube, you can simply round the pixel up and down to get the closest grid point. Obviously you cannot do this with simplex.
2. In 2D simplex noise, to find the closest point you apply a skew function to the surface to turn the equilateral tilings into right triangles, then find your closest points and unskew.
3. Non-euclidean space has curvature so you cannot find a global skew and unskew transformation. You can perhaps use local skewing (with the Jacobian) or a non-linear global skew function.

The Fibonacci Grid on a Sphere

1. You want a lattice (way of arranging points) on the sphere such that these points cover approximately equal area. On a sphere this is traditionally done with latitude-longitude points.
2. However, latitude-longitude leads to uneven spacing between points (an anisotropic distribution). The Fibonacci lattice can cover approximately equal areas, it has a more uniform distribution over the sphere.

3. It is impossible to arrange more than 20 points on the sphere with even coverage. (This shape is the dodecahedron).

Constructing the Fibonacci Lattice

1. Points are arranged along a tightly wound generative spiral with each point fitted into the largest gap between previous points.

2. The angle between consecutive points along the spiral is based on the golden ratio ϕ. The angle is either $360\phi^{-2}$ or $360\phi^{-1}$. Both produce the same points.

3. Golden ratio is $\phi = 1 + \phi^{-1} = (1 + \sqrt{5}/2) \cong 1.618$ or the Fibonacci ratio can serve as a rational approximate. ϕ is the most irrational number leading to optimum packing efficiency. Periodicities and near periodicities are avoided, clumping of lattice points never occurs.

4. Paper gives pseudocode to generate any odd number of points along a sphere. Returns the coordinates in angle representation.

Delauney Triangulation

1. Let S be a set of points in n-dimensional Euclidean space E^n. Let $\Omega(S)$ represent the convex hull.

2. Triangulation T can be defined as the decomposition of $\Omega(S)$ into the set of simplices $\{s_i\}_{i=1}^{k}$ such that:

   (a) A point $p \in E^n$ is a vertex of a simplex iff $p \in S$

   (b) For any pair of simplices $s_i, s_j \in T$, $s_i \cap s_j = \emptyset$

   (c) The union of all simplices completely covers the domain bounded by $\Omega(S)$

3. We say a triangulation T_D of a set $S \in E^n$ is a Delauney triangulation iff the circumsphere of every simplex $s \in T_D$ contains no other point in S but the vertices that form S.

4. If we're talking spherical Delauney Triangulations then the above criterion needs to be relaxed into: Let $S \in U$ be a set of points $\{p_i\}_{i=1}^{N}$ on the unit sphere then for $N \geq 3$ a Delauney Triangulation T_D is defined if the empty circumsphere criterion is satisfied for all spherical triangles covering the convex hull $\Omega(S)$. $U \subset \mathbb{R}^3$ is all the points on the unit sphere

TODO

1. Assuming we use the Fibonacci grid method to generate evenly spaced points and then take the convex hull, how do we know these faces are simplices? Is there some sort of isomorphic relationship between simplices and convex hull faces?

2. Each point on the Fibonacci grid should have some sort of radial attenuation function that combines to create the noise.

3. How do you determine which simplex a random point on your grid falls into?

4. Assume now that points will be represented with the spherical coordinates (r, ϕ, θ). We only have to store a permutation table with ϕ and θ in the range of 0-360 degrees.