Poisson Disc Sampling

1 Introduction

Poisson Disc Sampling, broadly speaking, is a strategy to position points or objects in a plane or space. This algorithm has 2 objectives: the first, that the points are uniformly positioned and the second, that it seems natural as well as that there is a minimum distance between the points.

To see the differences, the first image is the result of positioning the points randomly. It is clear that can occur that two points can be very close together or even overlap.

On the other hand, the second image is the result of applying a Jittered Grid strategy. This strategy is based on dividing the plane like a table and randomly positioning a single point in each cell. Although in this case the points will be better distributed along the plane, there may still be cases in which one point is too close to another, for example, a point is located in the upper part of its cell, on the edge, and the point of the upper cell is touching the lower edge of its square.

Finally, we have the Poisson Disk Sampling strategy, which, like the Jittered Grid approach, partitions the plane like a table and positions a point in each cell. The difference is that Poisson Disk Sampling takes into account a minimum radius at each point so that the points are not too close to each other.

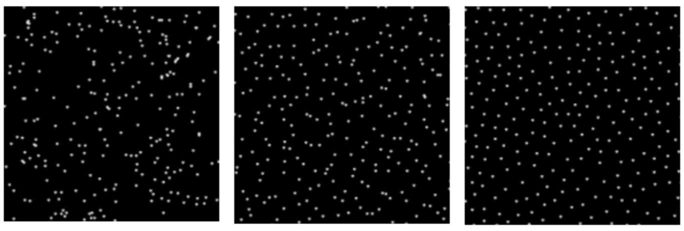

Figure 1: Comparison: Random Points - Jittered Grid - Poisson Disc Sampling

2 Algorithm

The implementation of the algorithm is based on placing the first point randomly and from that point placing the next one and so on until no more can be placed.

To do this, we must first define the radius r that indicates the distance between points and a constant k that allows us to know how many iterations pass until we consider that no more points can be placed. This constant is usually 30. From here, the first step is to define our table. To do this, we must first understand that the radius will be the same as the diagonal of the cell, since in this way, if a point is attached to a corner, we make sure that no point can be in the same cell.

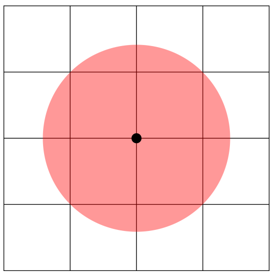

Figure 2: Point positioned in the center of the grid

This way, to know the size of the cells, we can perform the operation: $s = \frac{r}{\sqrt{2}}$ (where s is the side of the cell). For the 3-dimensional case, the formula will be: $s = \frac{r}{\sqrt{3}}$.

The next step will be to add the first point (chosen randomly) to a list of active points.

The third and last step is to start generating points. To do this, as long as our list of active points is not empty, we will take a random point from it. We then generate k random points within the annulus from r to $2r$.

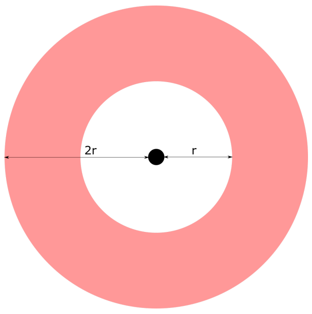

Figure 3: Annulus in which points are placed

Now, for each point generated, we examine whether within the radius r of the new point there are any points already placed. To do this, we check the distance of the new point from the others, but with the help of the table we can reduce the check to the points located in the cells of the 5x5 table which has the new cell of the new point in the center.

In the image above, the 4 black dots represent 4 possible points within the cell and the red areas the radius of each point. As you can see in the image, there is no need to check the points beyond the 5x5 table around the cell of the new point.

If after the k points we don't find any new point, we remove the point from the list of active points. Otherwise, we add the new point to the list of active points.

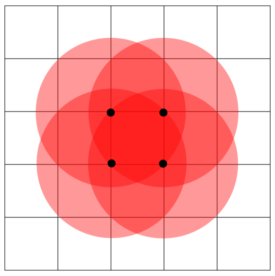

Figure 4: Radius of the 4 possible points in one cell

3 Examples

With these examples we can see how the different parameters affect the distribution of points.

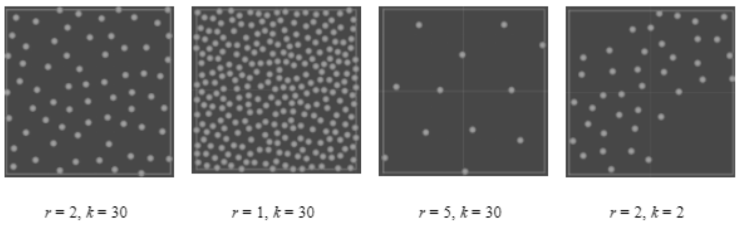

Figure 5: Algorithm applied in a 2-dimensional space

It can be seen that the greater the radius of minimum distance, the fewer spheres there will be and the smaller this radius, the more spheres will fit in our plane.In addition, we can also see that if our k (parameter that tells us how many iterations to perform until we decide that a point is not valid) is very small, there will be regions of the map that will not be populated since we do not generate any point in that region with few iterations.

We can also apply the algorithm to 3 dimensions. You just have to change all the 2-dimensional vectors to 3-dimensional vectors and the table that stores the points must be changed to a 3-dimensional table so that instead of a plane we have a space.

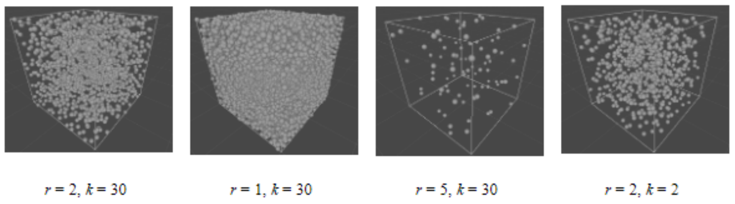

Figure 6: Algorithm applied in a 3-dimensional space