Neural augmentation for voxel raytracing

Cedric Goujon Maxime ?

ESGI ESGI

Paris, France Paris, France

[Cedric.goujon78@gmail.com](mailto:Cedric.goujon78@gmail.com) ?

Abstract

One of the major issues to voxel raytracing is the fact that we can see the discretization of space if the resolution of the voxel grid isn’t small enough. Thus this paper proposes three techniques to improve voxel raytracing.

Keywords

HyperNEAT, neural network, voxel, raytracing, neuroevolution

Introduction

One of the major problems of rendering voxels with raytracing is that if the resolution of the grid isn’t high enough, we can see the blocks forming our voxel scene. And even if we can’t see blocks forming the scene a cube can only have six normals making the lighting less precise than polygons that can have an infinite number of normal. Thus this paper will present a deep learning technique to estimate what the normal of the surface hit would be if the surface was smooth. This paper will also present a technique to estimate the depth map if the scene was made of smooth surfaces, and a technique to upscale a voxel scene that could potentially be used in real-time. All those techniques will be based on the indirect encoding of hyperNEAT.

Normal estimation

The normal is estimated using 3 generic phenotype neural networks that takes a 3d kernel as an input and an output node at the position hit by the ray, the center of the 3d input kernel is the position of the voxel that was hit. Three phenotype neural networks are generated with one CPPN, each of the phenotypes corresponds to a 3d axis. They compute the value of the normal for their respecting axis. The encoding of the nodes is the relative position to the position of output node on the axis that we are currently computing. As the encoding of the input node is computed based on the encoding of the output, it is not necessary to give the position of the output node to the CPPN. A constant bias input of one is added to the CPPN inputs. The value of an input is either 0 or 1 for respectively an empty voxel or filled voxel. Alternatively to avoid generating a neural network each a ray hits a voxel, we propose to take the center of the last empty voxel the ray traveled through as the position of the output node. Making a total of eighteen (6 position x 3 axis) possible neural networks, or you can have one neural network by rotating value depending on the theoretical kernel center.

Upscaling

The upscaling method is very similar to the normal estimation method, the difference here is that position of the output node is the center of new voxel that we to create or the position for which we want to estimate if a voxel would be there if the resolution of our grid. The center of the 3d kernel is the center of the voxel that contains our new voxel. And this time we give to our CPPN the Euclidean distance between the input and output node. This time only one phenotype is needed, as we just need to know if the output value of our phenotype neural network is equal or above one. Once we know if a voxel would be in this position, we need to determine what the color of that voxel should be. The position of the nodes of the phenotype network remains the same, and we still give same value to the CPPN. The phenotype network is used as many times as there is channel in the color type used. It takes in input the value of the channel being computed for each input voxel, the output is then divided by the number of voxels non-empty inside the kernel. When the camera is very close to the camera even with normal estimation the users will be able to see the voxels. Upscaling the part of the voxel grid very close to the camera where the rays passes through could prevent or reduce this problem.

Depth map estimation

To decide whether or not a point of a surface is hit by a light, we need to trace a ray from the position that was hit by our initial ray, to the position of the light. More precise hit position could also mean more precise normal estimation. The first step here is to generate the depth map before generating the final image. The depth map should contain the exact values of the distance traveled by the rays. This time we are using the a 2D kernel as an input of our phenotype neural network, the center of the kernel and the output node would have the position of the pixel that we want estimate the actual value. The inputs fed into the phenotype network are zero if the value of the pixel is too far from the value of the pixel of the center. Then once the output is computed we divide it by the number of pixel from which we have taken the real value. The CPPN is once again fed the Euclidean distance between the input node and the output node.

Conclusion and future works

TBD