EasyChair Preprint

№ 4934

Quantum Generators: Pattern Analysis for 3D Model Reconstruction from the Structured Compute Units.

Poondru Prithvinath Reddy

January 26, 2021

Quantum Generators: Pattern Analysis for 3D Model Reconstruction from the Structured Compute Units.

Poondru Prithvinath Reddy

ABSTRACT

Quantum Generators is a means of achieving mass food production with short production cycles, and when and where required by means of machines rather than land based farming which has serious limitations. The process for agricultural practices for plant growth in different stages is simulated in a machine with a capacity to produce multiple seeds from one seed input using computational models of multiplication (generating multiple copies of kernel in repetition). In this paper, we present different methods for generating 3D pattern reconstruction from the structured Compute Units resulted by the computational models of multiplication and also present a method related to 3D reconstruction so that they can be linked to tissues of the kernel which mimic the real cell structure that grows into full-fledged natural tissue. We use simulation to show that we achieve 3D structure with respect to the size of the input space and realize good pattern. The results suggest that it is possible to achieve relevant cell structure for quantum generation.

INTRODUCTION

A **Quantum** (plural quanta) is the minimum amount of any physical entity (physical property) involved in an interaction. On the other hand, **Generators** don't actually create anything instead, they generate quantity prescribed by physical property through multiplication to produce high quality products on a mass scale. The aim of Quantum Generators is to produce multiple seeds from one seed at high seed rate to produce a particular class of food grains from specific class of **seed** on mass scale by means of machine rather than land farming.

The process for agricultural practices include preparation of soil, seed sowing, watering, adding manure and fertilizers, irrigation and harvesting. However, if we create same conditions as soil germination, special watering, fertilizers addition and plant growth in different stages in a machine with a capacity to produce multiple seeds from one seed input using computational models of multiplication(generating multiple

copies of kernel in repetition) then we will be closure to achieving mass food production by means of quantum generators(machine generated) rather than traditional land based farming which has very serious limitations such as large space requirements, uncontrolled contaminants, etc. The development of Quantum Generators requires specialized knowledge in many fields including Cell Biology, Nanotechnology, 3D Cellprinting, Computing, Soil germination and initially they may be big occupying significantly large space and subsequently small enough to be placed on roof-tops.

The Quantum Generators help world meet the food needs of a growing population while simultaneously providing opportunities and revenue streams for farmers. This is crucial in order to grow enough food for growing populations without needing to expand farmland into wetlands, forests, or other important natural ecosystems. The Quantum Generators use significantly less space compared to farmland and also results in increased yield per square foot with short production cycles, reduced cost of cultivation besides easing storage and transportation requirements.

In addition, Quantum Generators Could Eliminate Agricultural Losses arising out of Cyclones, Floods, Insects, Pests, Droughts, Poor Harvest, Soil Contamination, Land Degradation, Wild Animals, Hailstorms, etc.

Quantum generators could be used to produce most important *food* crop *like* rice, wheat and maize on a mass scale and on-demand when and where required.

Computers and Smartphones have become part of our lives and Quantum Generators could also become very much part of our routine due to its potential benefits in enhancing food production and generating food on-demand wherever required by bringing critical advanced technologies into the farmland practices.

3D Bioprinting

3D Bioprinting is a form of additive manufacturing that uses cells and other biocompatible materials known as bioinks, to print living structures layer-by-layer which mimic the behavior of natural living systems. Three dimensional bioprinting is the utilization of 3D printing–like techniques to combine cells, growth factors, and biomaterials to fabricate biomedical parts that maximally imitate natural tissue characteristics.

Bioprinting (also known as **3D bioprinting**) is combination of **3D printing** with biomaterials to replicate parts that imitate natural tissues, bones, and blood vessels in the body. It is mainly used in connection with drug research and most recently as cell scaffolds to help repair damaged ligaments and joints. In this paper, we are looking at natural tissues related to food crops like rice, wheat or maize.

METHODOLOGY and THE ARCHITECTURE

A Quantum Generator device has one or more Compute Units. A work-group executes on a single Compute unit. A Compute Unit is composed of one Processing Element and Seed Object. A Compute Unit may also include filter Units that can be accessed by its processing elements.

A Device is a collection of Compute Units. Quantum Generator device typically corresponds to a collection of multiple Compute Units generated by the seed of a number.

A Seed is a function declared in a program and executed on a quantum generating device. A seed is identified by the Seed Qualifier applied to any function defined in any program.

A Seed Object encapsulates a specific seed function declared in a program and the argument values to be used when executing this Seed Function.

A Synchronization refers to mechanisms that define the order of execution and the visibility of operations between two or more units of execution. The Operations are that define order controls in a program. They play a special role in controlling how operations of in one unit of execution (such as work-items) are made visible to another. Synchronization essentially involves establishing a relation between operations in two different units of execution that define an order control in a device.

Seed Objects

A seed is a function declared in a program. A seed is identified by the seed qualifier applied to any function in a program. A Seed Object encapsulates the specific seed function declared in a program and the argument values to be used when executing this seed function.

Seed Objects are created for any seed functions in program that have the same function definition across all Compute Units for which a program has been built successfully in a device.

Kernel Methods

The aim of every classifier is to predict the classes correctly. For that, the data should be separable. In a fairly simple case, we normally see that all points above the cut-off line belong to the first class and the other points to the second class. However, it is extremely rare to have a dataset that simple. In most case, the data are not separable.

In the case of plot of a dataset which is not linearly separable and If we draw a straight line, most of the points will be not be classified in the correct class.

One way to tackle this problem is to take the dataset and transform the data in another feature map. It means, you will use a function to transform the data in another plan, which should be linearable.

The data from the Compute Units is in a two-dimension plan which is not separable and we try to transform these data in a three-dimension, it means, you create a figure with 3 axes.

In our case, we will apply a polynomial mapping to bring our data to a 3D dimension. The formula to transform the data is as follow.

$$\emptyset(x, y) = (x^2, \sqrt{(2)}\, xy, y^2)$$

We define a function for the above formula to create the new feature maps and the new mapping is with 3 dimensions with 3 axis, x, y and z respectively. We see an improvement compared to cut-off line method, it is clear that the dataset is now separable on changing the orientation of the plot.

To manipulate a large dataset and we may have to create more than 2 dimensions, and we will face a big problem using the above method. In fact, you need to transform all data points, which is clearly not sustainable and It will take lot of computational time and memory

The most common way to overcome this issue is to use a **kernel**.

What is a Kernel?

kernel refer to: a non-parametric way to estimate a probability density, the set of vectors **v** for which a linear transformation T maps to the zero vector — i.e. T(**v**) = 0, the set of elements in a group G that are mapped to the identity element by a homomorphism between groups (group theory), the core of a computer operating system (computer science), or something to do with the seeds of nuts or fruit.

The idea is to use a higher-dimension feature space to make the data almost linearly separable.

There are plenty of higher dimensional spaces to make the data points separable. For instance, we have seen that the polynomial mapping is a great start. We have also demonstrated that with lots of data, these transformation is not efficient. Instead, we can use a kernel function to modify the data without changing to a new feature plan.

It is highly important to understand how kernels are used in vector classification. For practical reasons, it is important to understand importance of specifying a kernel

function, and there are not established, general rules to know what kernel will work best for particular data.

Kernel Definition

- A function that takes as its inputs in the original space and returns the dot product of the vectors in the feature space is called a ***kernel function***
- More formally, if we have data X, Z ∈ X and a map $\emptyset : X \to \Re^N$ then

$$K(X, Z) = \langle \emptyset(\boldsymbol{X}), \emptyset(Z) \rangle$$

Is a kernel function.

Our kernel function accepts inputs in the original lower dimensional space and returns the dot product of the transformed vectors in the higher dimensional space.

It can somewhat help to understand how the kernel function is equal to the dot product of the transformed vectors by considering that each coordinate of the transformed vector $\boldsymbol{\phi}$(**x**) is just some function of the coordinates in the corresponding lower dimensional vector **x**.

The magic of the kernel is to find a function that avoids all the trouble implied by the high-dimensional computation. The result of a kernel is a scalar.

The objective of kernel is to create a higher dimension by using a polynomial mapping and the output is equal to the dot product of the new feature map. For this, we have to Transform the vectors into a new dimension, Compute the dot product common to all kernels and Transform the vectors into a new dimension. However, there is a problem, we need to store in memory a new feature map to compute the dot product. If you have a dataset with millions of records, it is computationally ineffective.

Instead, we can use the **polynomial kernel** to compute the dot product without transforming the vector. This function computes the dot product of vectors as if these vectors have been transformed into the higher dimension. Said differently, a kernel function computes the results of the dot product from another feature space.

You can write the polynomial kernel function as the power of the dot product of vectors. The output is equal to the other method with intended degree of the polynomial kernel. This is the magic of the kernel.

Type of Kernel Methods

There are lots of different kernels available. The simplest is the linear kernel. This function works pretty well for text classification. The other kernel is:

- Polynomial kernel
- Gaussian Kernel

Polynomial kernel

The **polynomial kernel** is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors in a feature space over polynomials of the original variables.

Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. In the context of regression analysis, such combinations are known as interaction features. The (implicit) feature space of a polynomial kernel is equivalent to that of polynomial regression.

Definition

For degree-*d* polynomials, the polynomial kernel is defined as

$$K(\mathrm{x}, \mathrm{y}) = (\mathrm{x}^{\mathrm{T}}\ \mathrm{y} + \mathrm{c})^{\mathrm{d}}$$

where *x* and *y* are vectors in the *input space*, i.e. vectors of features computed from test samples and $c \geq 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. When $c = 0$, the kernel is called homogeneous.

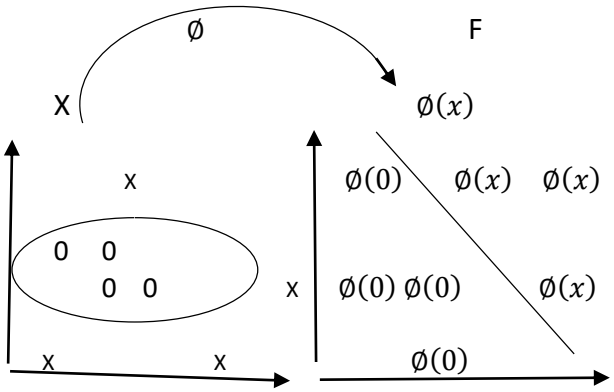
As a kernel, *K* corresponds to an inner product in a feature space based on some mapping φ:

$$K(\mathrm{x}, \mathrm{y}) = \langle \emptyset(x), \emptyset(y) \rangle$$

Let $d = 2$, so we get the special case of the quadratic kernel. After using the multinomial theorem (twice—the outermost application is the binomial theorem) and regrouping,

$$K(\mathrm{x}, \mathrm{y}) = (\sum_{i=1}^{n} \mathrm{xi}\ \mathrm{yi}\ +\ \mathrm{c})^2 = \sum_{i=1}^{n} (x_i^2)(y_i^2) + \sum_{i=2}^{n} \sum_{j=1}^{i-1} \left(\sqrt{2}\ x_i x_j\right) (\sqrt{2}\ y_i y_j) + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\sqrt{\boldsymbol{2c}}\boldsymbol{x_i}) (\sqrt{2c} y_i) + c^2$$

From the above it follows that the feature map is given by:



It can be seen from the mapping ∅ that On the left a set of samples in the input space, on the right the same samples in the feature space where the polynomial kernel *K*(x, y) (for some values of the parameters ***c*** and ***d***) is the inner product. The hyperplane learned in feature space by an SVM is an ellipse in the input space.

In this paper, we have dealt with simulation of sequence of data generated by the Compute Units to show that we achieve 3D mapping with respect to the input space and not about synchronizing Compute Units to tissues of the kernel which mimic the real cell structure that grows into full-fledged natural tissue.

Unlike the simulation results which are based on few parameters, In natural or real tissues which are 3D Bioprinted there are number of parameters to be considered for pattern analysis.

The QG System

Our objective is to build a target system, we need to generate the cell for the device by running synthesis and implementation on the design. The cell includes custom logic for every Compute unit in the cell container. The generation of custom compute units uses the High-Level synthesis tool, which is the computer unit generator in the application compilation flow. Therefore, it is normal for this step to run for longer period of time than the other steps in the system build flow.

After all compute units have been generated, these units are connected to the infrastructure elements provided by the target device in the solution. The infrastructure elements in a device are all of the memory, control and output data planes which the device is formulated to support an application. The environment combines the custom compute units and the base device infrastructure to generate a cell binary which is used to program the QG device during application execution.

The processing flow of application execution is given as below:-

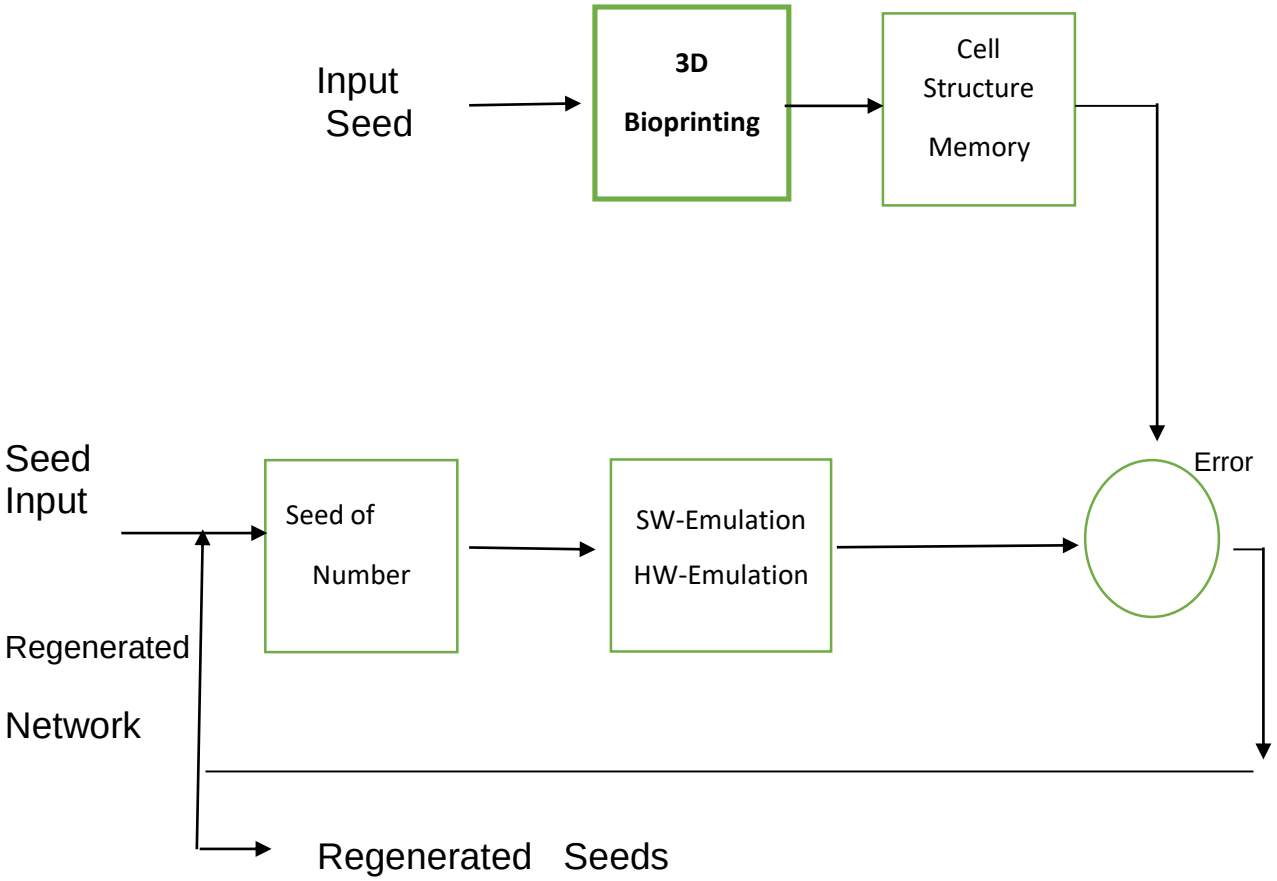


Fig. 1 Process Flow in a Quantum Generator.

The different steps in application are as below:-

1. 3D print a seed and copy its cell structure to memory.
2. Input seed with a seed of a number required.
3. Generate a seed kernel once.
4. Compare the kernel with 3d printed cell
5. If error in seed structure, generate the kernel again.
6. Repeat many times till the seed number is met.

TEST RESULTS

The objective of the Polynomial Kernel is to classify the sequence of data generated based on function parameters of Compute Units to evaluate a logistic regression to have 3D benchmark model. Although, we have generated seed structure with good 3D pattern but this is not mapped to original tissues of seed kernel which are in 3D plane to test the deviation.

CONCLUSION

Quantum Generators (QG) creates new seeds iteratively using the single input seed and the process leads to a phenomenon of generating multiple copies of kernels in repetition. We presented different methods of generating 3D pattern reconstruction from the structured Compute Units and also presented a method for 3D reconstruction which can be used to mimic the tissues of real kernel. The results suggest that it is possible to achieve suitable cell structure for quantum generation.

REFERENCE

1. Poondru Prithvinath Reddy: “Quantum Generators: A Formulation of Computational Models of Multiplication”, Google Scholar.
2. Poondru Prithvinath Reddy: “Quantum Generators: Foundations of the Compute Units in Pattern Reconstruction”, Google Scholar.
3. “Polynomial Kernel – Wikipedia”, https://en.wikipedia.org/wiki/Polynomial_kernel