

Realtime Ocean water Simulation using FFT

**NAME Shoaib zafar Taha Malik**

**REGISTRATION 2112340, 2112119**

**INSTRUCTOR Saqib Sadiq**

**COURSE Design and Analysis of Algorithm**

**FAST FOURIER TRANSFORMATION**

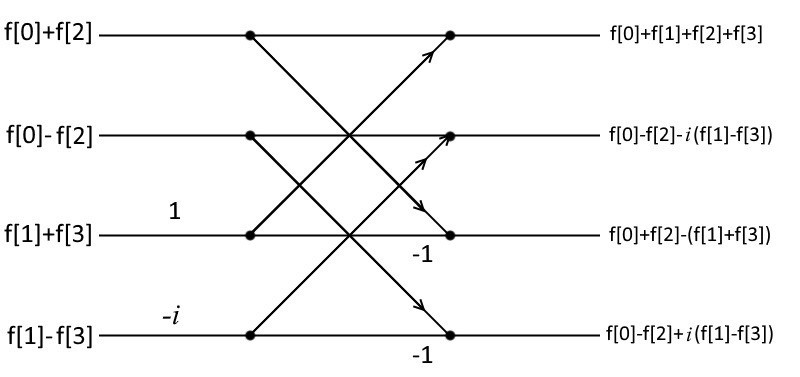
The Collee-Tukey FFT algorithm also known as the Butter fly algorithm is one most ambiguous technique in world of computation. The Fast Fourier transformation known as is the most ingenious algorithm when it comes to the real time computation of the such kind of data which has application of a large-scale distribution. The core components for the FFT algorithm which tells the entire compute personality and its implementation in the world of Realtime - computation like in digital signal processing the compute personality of FFT is distributive and it also resembles linear wave theory that’s why the FFT is essential tool in signal processing.

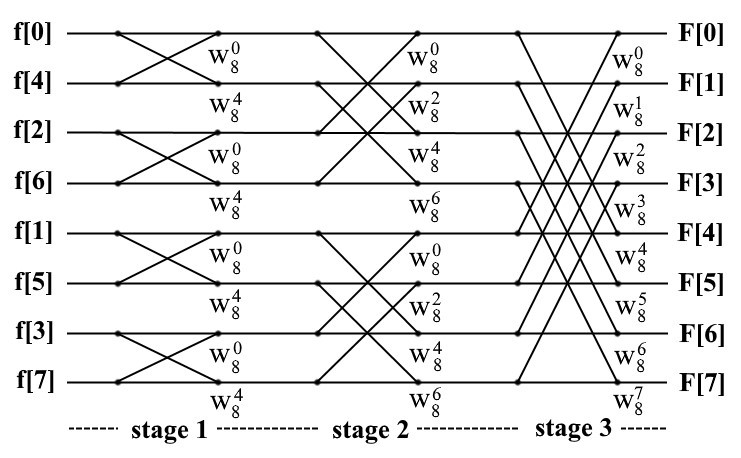
**FFT IN DIGITAL SIGNAL SIMUALTION AND PROCESSING**

The way FFT solve the incomes raw data from waves by creating a Realtime -wave simulation of incoming flux of data and then converting those wave simulation into the clock cycles through which it can tell the CPU the simultaneous occurrence and the number of oscillations of each wave so that a programmer can allocate the amount of DRAM CPU threads and GPGPU cores to each stage of decode large influx of data by analyzing them through Realtime wave simulation.

**TIME COMPLEXITY OF FFT**

The basic concept of the FFT is to exploit the structure of the DFT by smartly rearranging the products. Before the general procedure of the FFT algorithm in computation of the DFT with *N* = 4. Computing **F** of a discrete signal **f** with *N* samples in the naive way would require *N* complex multiplications and *N −* 1 complex additions for each element of **F**, which total time is proportional to *O*(*n*2). The FFT algorithm reduces the complexity to *O*(n *l*og n). For large *N* this leads to a massive performance advantage.

****

In above example we saw the indices holding the data of each Fourier transform and these Fourier transforms are forming distributive diagonal transform forming a butterfly called N number of points which equal to four the size of twiddle indices consist of size 8 which means it would distribute eight Fourier transform in each FFT stage.

. The overall time complexity and performance of an FFT algorithm would depends on the amount of butterfly operation in each stage. Further the number of butterfly stages can be generically determined to log2 *n* stages, since the butterfly-span is doubled after each stage to a maximum span of *N/*2 in the last stage. Thus, the total execution time of the FFT with log2 *n* stages and *O*(*n*) for each stage is *O*(*n* log *n*). Due to the reason that the butterfly span is doubled after each stage, the Cooley-Tukey FFT algorithm is a divide and conquer algorithm. Since the algorithm can only be applied on DFT’s with *N* as a power of two, it is also called Radix-2 FFT algorithm.

**PSEUDOCODE**

Initial Spectrum *h*˜0(**k**) and *h*˜0(*−***k**) shade pass; **while** *is Running* **do**

Fourier components *h*˜(**k***,t*) shade pass; pingpong := 0; **for** *i=0 to i<log*2*N* **do**

horizontal butterfly shaderpass for stage *i*; pingpong := pingpong++ mod 2;

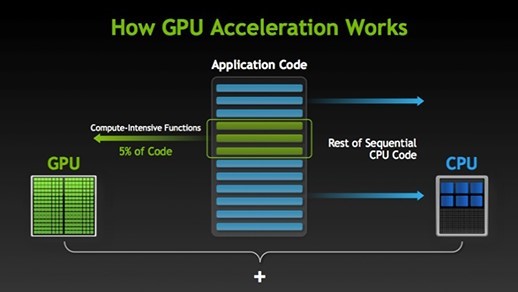
**end for** *i=0 to i<log*2*N* **do**

vertical butterfly shaderpass for stage *i*; pingpong := pingpong++ mod 2;

The following lines code is actually an abstract of implementation of an FFT algorithm where the shader passes are holding the data and for loop is iterating the twiddle indices holding these Fourier transform and the ping pong is performing the butterfly operation on each stage.

**REALTIME RENDERING IN CGI**

In Realtime 3d application and modern game engines the one most challenging part that we have to achieve is actually the realistic and physically accurate photo realistic CGI Computer generated imaginary for pretty much every industry which uses Realtime simulation for the analysis of their problems in 3d. the CGI let us create a fake digital reality where we can simulate our past predicted events in front of our eye and we can perform the specific events whether from past and future prediction . we can create colliding bodies like cars, water waves and subatomic particles like DNA wave simulation for drug synthesis. The FFT algorithm contains the simultaneous occurrence of float point operation in each stage horizontally and vertically which makes it obvious that if we want crunch data out form FFT we need a parallel or Discrete processing unit if we want large of amount in each stage of FFT and distribute that data in Realtime with very less latency. The FFT is expensive on performance so we need to apply the implementation of this algorithm using compute shader. the Compute shader is actually shader script of shader program which actually the core of our graphics API like OPENGL, VULKAN, DIRECTX. The compute shader is actually a line of functional code that tells the data that what specs and configuration and it would utilize for the processing of this influx of data. the speed of the processing of the compute shader depends upon that how much time each is being spend by each GPGPU core on each GPU stage if we want execute our shader in less time and latency then we have to optimize the GPGPU cores and sometimes we have to use the state of they art GPU computing tech for high performance Realtime rendering.

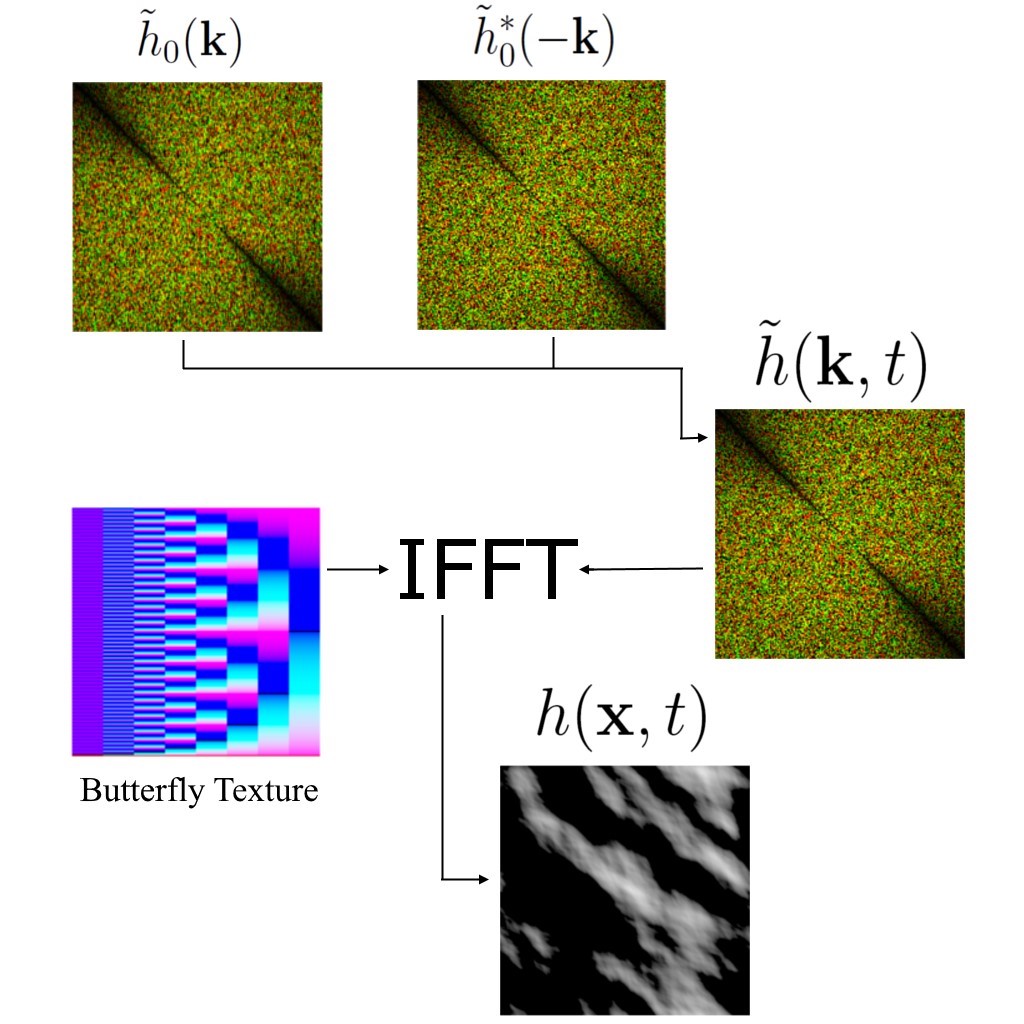


**REALTIME OCEAN THE SIMULATION USING FFT**

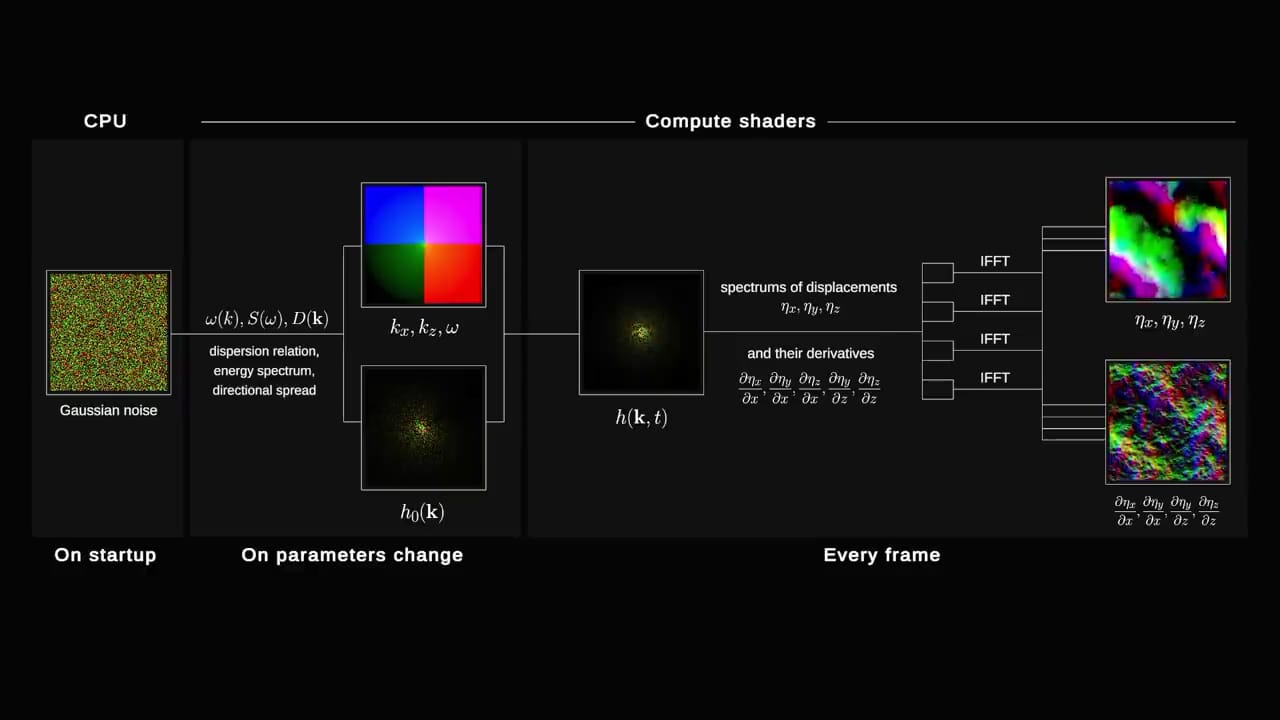
In Realtime CGI rendering water is most difficult to achieve in computer to its physicality which is actually described the linear wave theory. In Graphics pipeline we use shader programs which contains five to six fundamental compute shader scripts which tells our Realtime 3D renderer the geometry, animated normal maps, number of polygons or vertex in each CPU stage and post processing effects for the denoising of each rendered frame. The physically of water is basically a wave which we have to create a dynamic mesh which would perform number oscillations. We can create these number of oscillations using FFT which contains the property of butterfly we can Geometry computer shader to distribute these number of vertices while maintain large scaling and projection could give us pretty high detailed water waves.

### *h*˜0(k) and *h*˜0(*−*k) Spectrum Textures

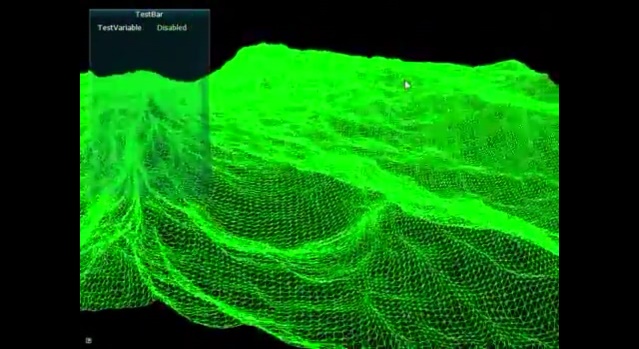
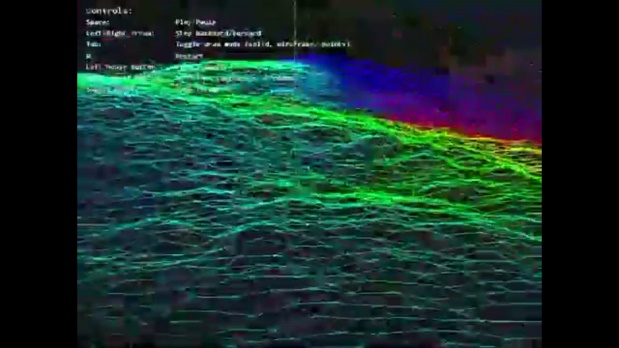
One texture is needed for storing the initial spectrum from (3.6). The gaussian random numbers *ξr* and *ξi* are generated from two independent noise textures with the Box-Muller method. The real part of *h*˜0(**k**) is stored in the red channel while the imaginary part is stored in the green channel. A further texture is used to store *h*˜ with gaussian random numbers *ξr* and *ξi* generated from two further noise textures. shows the rendered *h*˜0(**k**) spectrum texture. These textures would give an butterfly normal map texture which would provide the color value of each Fourier transform and the final texture *h*˜(x,t) would be an animated map in black and white form called the animated displacement map .

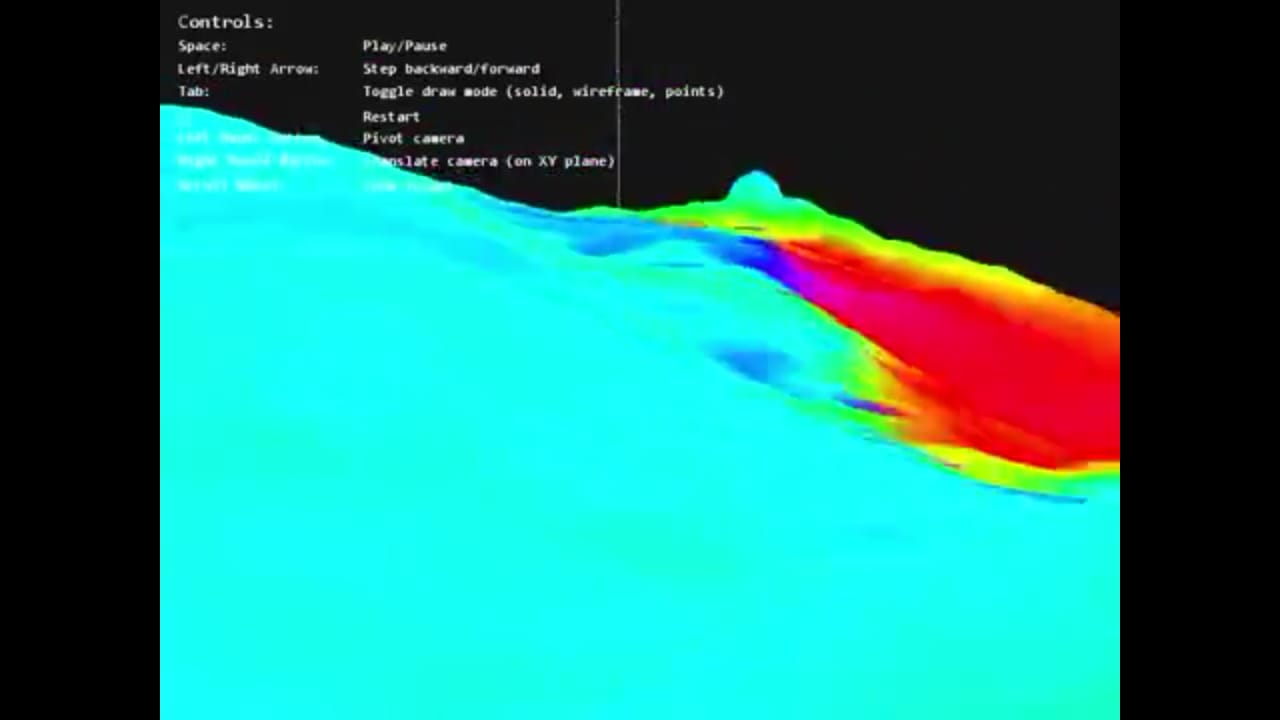


**THE FINAL RENDER PIPELINE**

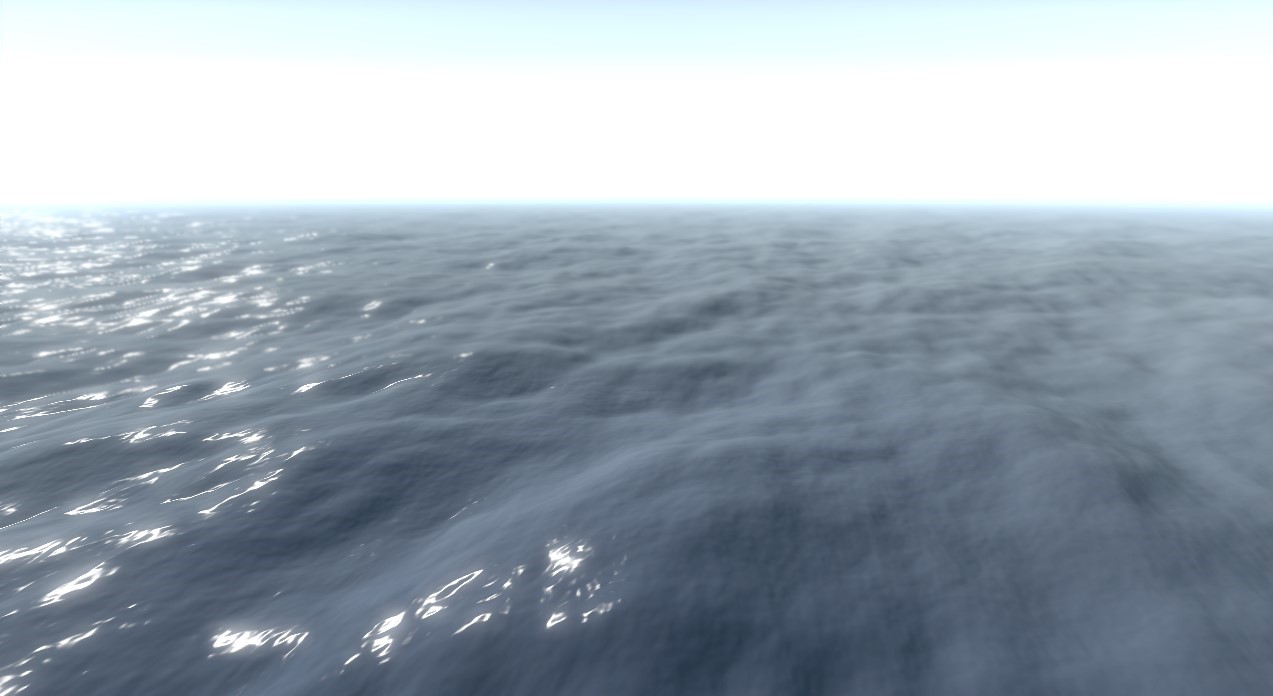
****

**RENDERING PROCESSING IN GRAPHICS PIPELINE**

****

****

**FINAL OUTPUT**

****