Projektarbeit: Simulation of Spontaneous Phase Matching in β-barium borate crystals

Sebastian Wagner

January 29, 2026

Contents

List of Figures

1 Introduction to Problem

This document was devoloped as a Project with the aim to build a simulation tool for the spontaneous parametric down conversion (SPDC) outlined in [KAB+20]. The main goal was to produce similar images as shown in Figures 8 and 9 of the previously mentioned paper. Initially a matlab simulation was tested, which mainly failed due to poor performance and resource constraints. Later on a C++ program was developed due to easier control over system resources and due to a higher familiarity with the programming language. As well as showing some of the challenges attributed to this project, this document is also meant to server as a defacto tutorial on how to install and use the simulation program.

To begin: The heart of the [KAB+20] paper is equation 81, which tells us the coincidence rate of a signal and idler pair at two points. The equation is again shown below:

$$R_{si}(\mathbf{p}_s, \mathbf{p}_i, z) = \alpha_s \alpha_i A \left| \int \int e^{i(k_s+k_i)z} V(\mathbf{q}_s + \mathbf{q}_i) \Phi(\mathbf{q}_s, \mathbf{q}_i) \right. \tag{1}$$

$$\left. \exp\left[i \left(\mathbf{q}_s \mathbf{p}_s + \mathbf{q}_i \mathbf{p}_i - \frac{|\mathbf{q}_s|^2 z}{2k_s} - \frac{|\mathbf{q}_i|^2 z}{2k_i} \right) \right] d^2\mathbf{q}_s d^2\mathbf{q}_i \right|^2 \tag{2}$$

Keep in mind that both q_i and q_s are vectors consisting of qix, qiy and qsx, qsy respectively the equation shown above is really a 4 Dimensional Integral. The function denoted by V is the pump field and the function denoted by the capital Phi is called the phase matching function. These are both (especially in the case of the phase matching function), rather large functions. The resulting coincidence rate per position of the idler and signal can then be used to calculate the photon count rates for the signal or idler. Which is expressed in the paper in equation 82. This equation is again written out down below as:

$$R_s(\mathbf{p}_s, z) = \int R_{si}(\mathbf{p}_s, \mathbf{p}_i, z) d^2\mathbf{p}_i \tag{3}$$

In total this would leave us with a 6 fold Integral, which will be hard to compute. To remedy this issue somewhat a Fourier Transform approach was chosen to compute the coincidence rates outlined in equation 81 of the paper. I first saw this approach in the following github repository: `https://github.com/mvchalupnik/spdc-simulator` which goes over the same paper as we do. In general this github has been very helpful in cross-checking errors and confiming results. As the original paper [KAB+20] does

not include how they actually numerically computed the problem. The hardest to calculate function is the phase matching function. You can distinguish between different phase matching types. These are:

- type-I phase matching: Idler and Signal have same polarization
- type-II phase matching: Idler and Signal are orthogonal.
- Collinear Phase matching: Idler and Signal propagation direction: Along the direction of the pump photon

Here we will, however, only look at type-I and type-II phase matching. There are many parameters which you can change in this problem. The main ones are:

- Crystal Length
- Angle between Optical axis and beam propagation direction
- Pump incidence angle on crystal surface
- indeces of refraction of the bbo-crystal
- location of beam waist of gaussian pump field

Why Fourier Transform Why can we use a Fourier Transform to calculate this Integral in the first place. When looking at the Coincidence Rate Equation Eq Nr. 81 again, you can see a exponential function at the end. In this exponential function You have $\rho_s q_s + \rho_i q_i$ if you write this out in long form (without the vectors) you get $\rho_{sx} q_{sx} + \rho_{sy} q_{sy} + \rho_{ix} q_{ix} + \rho_{iy} q_{iy}$. If you then write the integral of the Coincidence Equation again you get:

$$\int\int\int\int V(q_p)\Phi(q_s,q_i)e^{-i\left(\frac{|\mathbf{q_s}|^2 z}{2k_s}+\frac{|\mathbf{q_i}|^2 z}{2k_i}\right)}e^{i(\rho_{sx}q_{sx}+\rho_{sy}q_{sy}+\rho_{ix}q_{ix}+\rho_{iy}q_{iy})}\mathbf{dq}_s\mathbf{dq}_i \tag{4}$$

Which is precicely the Definition for the Fourier Transform, of a Function F($\mathbf{q}_s, \mathbf{q}_i$) -> $\mathfrak{F}(\mathbf{p}_s, \mathbf{p}_i)$.

2 Issues with FFT-approach

The main Issues with the fft-approach that will be outlined below is that you will need to choose a high enough sample count such that aliasing does not occur. And you need to optimize the maximum q value upto which you compute your input data or else you will need to increase your smaple count again. Below here you will see the two main issues with this approach:

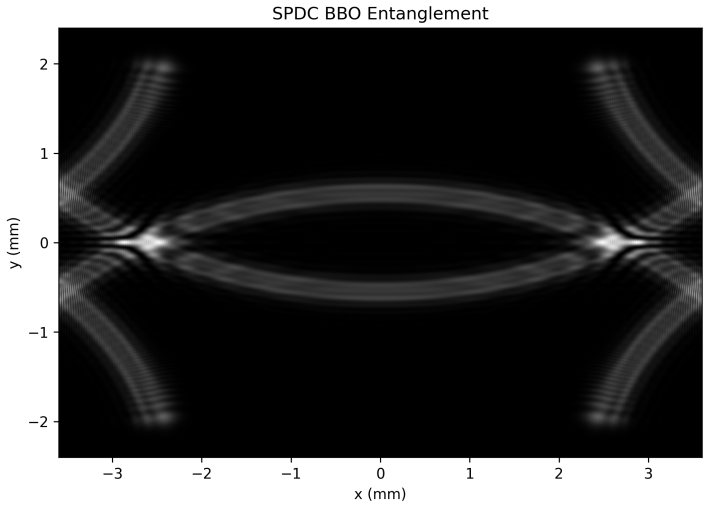


Figure 1: Bad Example of Aliasing in FFT-based SPDC-Simulation

Aliasing Issue The two following images will have the same parameters but different sample counts. Bad Example:

Good Example: As you may see in the bad example the higher *frequency* components of the signal will seem to be mirrored at the edges of the image. This does not occur in the Good example as it has a higher number of samples.

Phasing out of Image-Signal Now unlike the Aliasing mentioned above if you choose to use a maximum q value that is too low, the signal will seem to *phase out* near the edges of the image. Below there are some attempts to calculate a maximum q value such that the signal stays mostly intact, however the approach below seems not without fail. Therefore if you encounter an image that seems similiar to the one below here, you will need to increase the maximum q value inside the code and you will therefore also (most, very very likely) need to increase the sample count.

3 Max frequency and Sampling Points

Since we use an FFT approach to circumvent computing the 4-dimensional Integral in Equation 81 of Paper [KAB$^+$20] we will have to define some

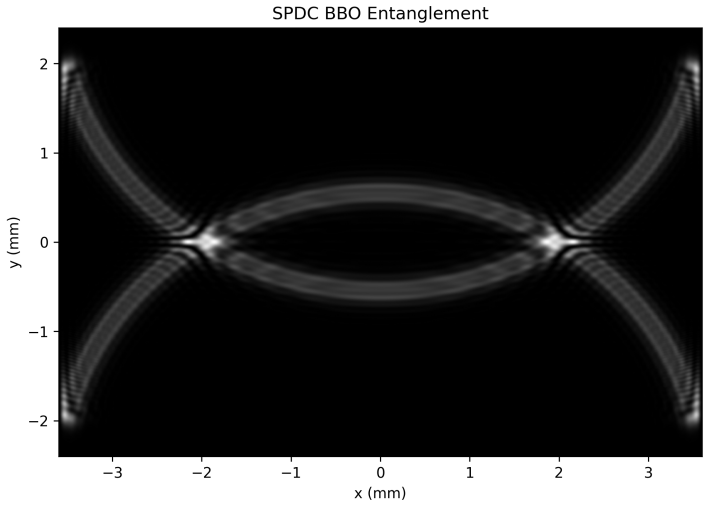


Figure 2: Good Example of no Aliasing in FFT-based SPDC-Simulation

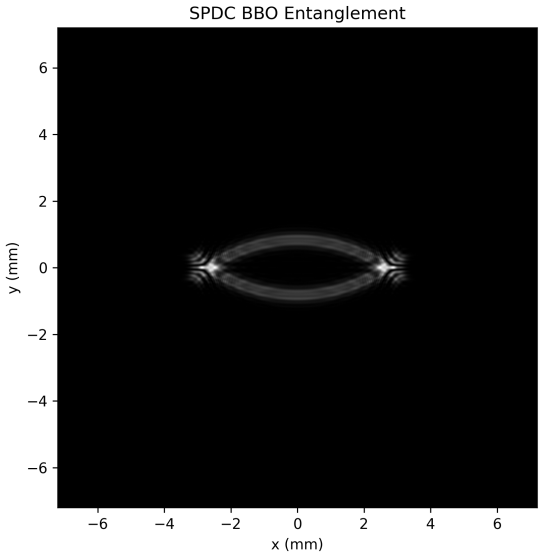


Figure 3: Bad Example of a too low maximum q value for the sampled function. Signal seems to phase out.

maximum frequency and a corresponding number of samples such that no aliasing occurs.

3.1 Input data creation

To compute the FFT we will have to create it's input data. The input tensor is of rank 4, which means that not only will it take a significant amount of time to compute the respective input data, but the memory consumption of our program will increase with the 4th power of the number of sample points. The following approach asks the question **When is the original input function aka the coincidence so insignificantly low that we can cut if off so to speak** As such we will look at the components of the Coincidence Function one at a time and try to make a guess as to what maximum value of q (named q max here) we still need to compute the Pump function and phase matching function of.

Pump-Function V

$$V(q_p) = \exp\left[-\frac{\mid q_p \mid^2 w_0^2}{4}\right] \exp\left[-i\frac{\mid q_p \mid^2 d}{2k_p}\right] \tag{5}$$

When looking at the magnitude of the Pump field we can ignore the second exponential since it will always have magnitude 1. Which will leave us with the following expression:

$$V(q_p) = \exp\left[-\frac{\mid q_p \mid^2 w_0^2}{4}\right] \tag{6}$$

Suppose we want this function to be below some value, let's say e^{-5} we can then reform the equation as follows:

$$5 = \frac{\mid q_p \mid^2 w_0^2}{4} \tag{7}$$

$$\frac{20}{w_0^2} = \mid q_p \mid^2 \tag{8}$$

Qp is defined as qs+qi. Which in turn are again

$$q_p = \begin{pmatrix} qsx + qix \\ qsy + qiy \end{pmatrix} \tag{9}$$

$$\mid q_p \mid^2 = (qsx + qix)^2 + (qsy + qiy)^2 \tag{10}$$

At worst 3 of the 4 variables will be 0 so the worst case scenario is (for which we will achieve a $V(q_p)$ of less than e^{-5}):

$$\mid q_p \mid^2 = (qsx + 0)^2 + (0 + 0)^2 = qsx^2 \tag{11}$$

Inserted into the above equation we get a qsx_{max} of:

$$qsx_{max} = \sqrt{\frac{20}{w_0^2}} \tag{12}$$

With w_0 being 388 μm in our case we get:

$$qsx_{max} = 11527 \tag{13}$$

For $V(q_p)$ below e^{25} and e^{250}, qsx_{max} is 25774 and 81502 respectively.

Phase matching function Looking at the phase-matching function inside the Integral we find the following Expression:

$$\phi = Lsinc\left[\Delta k_z \frac{L}{2}\right] \exp\left[i\Delta k_z \frac{L}{2}\right] \tag{14}$$

Where, with similiar reasoning to above, the exponential part can be ignored as it's magnitude is always one. The sinc function is at maximum 1 and it's envelope decreases linearly with 1 over $\mid \Delta k_z \mid$. If we choose the example of above again we would need to have a $k_z \frac{L}{2}$ above e^5 for the envelope of the sinc function alone to be below e^{-5}. Type-I phase matching $k_z - Term.$ (Eq. 85) [KAB+20]

$$\Delta k_z = n_{os}\frac{\omega_s}{c} + n_{oi}\frac{\omega_i}{c} - \eta_p \frac{\omega_p}{c} \tag{15}$$

$$+\frac{c}{2\eta_p \omega_p}\left[\beta_p^2 q_{px}^2 + \gamma_p^2 q_{py}^2\right] + \alpha_p\left(q_{sx} + q_{ix}\right) \tag{16}$$

$$-\frac{c}{2n_{os}\omega_s} \mid q_s \mid^2 - \frac{c}{2n_{oi}\omega_i} \mid q_i \mid^2 \tag{17}$$

Where both q_s and q_i are vectors consisting of

$$q_s = \begin{pmatrix} qsx \\ qsy \end{pmatrix} \tag{18}$$

$$q_i = \begin{pmatrix} qix \\ qiy \end{pmatrix} \tag{19}$$

The fist part of the equation consists of constants:

$$n_{os}\frac{\omega_s}{c} + n_{oi}\frac{\omega_i}{c} - \eta_p\frac{\omega_p}{c} \tag{20}$$

If you then take the rest of the equation and group them by terms you end up with the following

$$q_{sx}\left(\alpha_p\right) + q_{ix}\left(\alpha_p\right) \tag{21}$$

$$+q_{sx}^2\left(\frac{c}{2\eta_p\omega_p}\beta_p^2 - \frac{c}{2n_{os}\omega_s}\right) \tag{22}$$

$$+q_{sy}^2\left(\frac{c}{2\eta_p\omega_p}\gamma_p^2 - \frac{c}{2n_{os}\omega_s}\right) \tag{23}$$

$$+q_{ix}^2\left(\frac{c}{2\eta_p\omega_p}\beta_p^2 - \frac{c}{2n_{oi}\omega_i}\right) \tag{24}$$

$$+q_{iy}^2\left(\frac{c}{2\eta_p\omega_p}\gamma_p^2 - \frac{c}{2n_{oi}\omega_i}\right) \tag{25}$$

$$+q_{sx}q_{ix}\left(\frac{2c}{2\eta_p\omega_p}\beta_p^2\right) \tag{26}$$

$$+q_{sy}q_{iy}\left(\frac{2c}{2\eta_p\omega_p}\gamma_p^2\right) \tag{27}$$

Unfortunately the constants alpha, beta, gamma and eta depend on $theta_p$ so you cannot decouple the max q from it. Now for all these you will still need to keep the $\frac{L}{2}$ of the original equation in mind. For the parameters given in [KAB+20] (They are also listed in the Calculating sample count tab below). The constant factor is about 8.4. The following image shows Values for factors inside the parenthesis of equations 20 - 26. The ones in 20 are called linear factors. 21 to 24 are called quadratic factors and 25 and 26 are called mixed terms. The Image below shows the program output for these (keep in mind these are with the L/2 factored in) Again we argue that in

Figure 4: Factors for q max calculation. Linear, quadratic and mixed terms.

the worst case only one of the variables is not zero. Therefore if we want to calculate when the argument of sinc becomes larger (as absolute) than some defined value, again lets say e^5, for the case of qsx we would write from Equation 20 and 21 (Not 25 because worst case -> qix =0):

$$\mid e^5 \mid = 6.72 * 10^{-05} qsx - 1.78 * 10^{-11} qsx^2 \tag{28}$$

For - e^5 this gives us two real values of -1575039 and 5350320 for q_max. This value is similar for all other q variables. This is for the sinc alone which is not really fair. As there is a prefactor of L (being the bbo-crystal length) even before the sinc function. If you factor out the the L from the envelope (the envelope being $\frac{1}{\Delta k_z \frac{L}{2}}$) you are left with sinc($\frac{\Delta k_z}{2}$). If you redo the calculations you are still left with a very large q max value of around 4000000. Now this is also not entirely correct, since we somewhat cheat by just equating the sinc to its envelope function. Approached from another direction we get: Suppose you just insert the values you have from V pump above. Namely qmax: 11527, 25774 and 81502. If you insert the highest, where V pump is below e^{-250}, into the phase-matching function you get:

$$sinc(-1.78 * 10^{-11} * 81502^2 + 6.7 * 10^{-5} * 81502) = -0.156 \tag{29}$$

The argument to the sinc function is 5.3. So we know from the envelope at least (since it is after x=1) that it is below 1/5.3 (in magnitude!). Or with L being 0.002 we get 0.000377. The initial approach therefore was to set q max to 81502. In Testing however it turns out that this is not sufficient. So this appraoch is either entirely flawed or there is some mistake that I don't see. The max q value that is being used now is 350000.

4 Aliasing

The main two parameters that can be changed for a fft-based simulation are, in this case, the sample count and the maximum k (or max q called earlier) you use for simulation. Where :

$$\Delta k = \frac{k_{max}}{samplecount} \tag{30}$$

$$"x_s" = \frac{1}{\Delta k} \tag{31}$$

Where x_s refers to a sampling spatial value. Nyquist theorem tells us, that at best the maximum frequency one can obtain, without aliasing of the signal is :

$$f_{max} = \frac{1}{2} f_s \tag{32}$$

So conversely if you wanted to simulate to some value x_{max} you would need a x_s of:

$$x_s = 2x_{max} \tag{33}$$

4.1 Estimating Sample count

Suppose you have a sine function of the following form:

$$sin(ax + b) \tag{34}$$

The parameter a would then be the frequency of the sine function. When looking at the Integral (Eq. 81) we can see four parts:

$$e^{i(k_s+k_i)z} \; k_s \text{ and } k_i \text{ are both constants, so no oscillatory impact} \tag{35}$$

$$V(q_p)\text{Gaussian function -> will act similiarly to a windowing function} \tag{36}$$

$$\phi(q_s, q_i)\text{sinc function -> highly oscillatory} \tag{37}$$

$$e^{-i\left(\frac{|q_s|^2 z}{2k_s}+\frac{|q_i|^2 z}{2k_i}\right)}\text{oscillatory} \tag{38}$$

The main focus will be on the phase matching function and the trailing exponential term. Frequency component of phase matching function Linear Term:

$$q_{sx} : \frac{L\alpha_p}{2} \tag{39}$$

$$q_{ix} : \frac{L\alpha_p}{2} \tag{40}$$

Frequency component quadratic term:

$$q_{sx} : \frac{L}{2} q_{sx} \left(\frac{c}{2\eta_p \omega_p} \beta_p^2 - \frac{c}{2n_{os}\omega_s} \right) \tag{41}$$

$$q_{ix} : \frac{L}{2} q_{ix} \left(\frac{c}{2\eta_p \omega_p} \beta_p^2 - \frac{c}{2n_{oi}\omega_i} \right) \tag{42}$$

$$q_{sy} : \frac{L}{2} q_{sy} \left(\frac{c}{2\eta_p \omega_p} \gamma_p^2 - \frac{c}{2n_{os}\omega_s} \right) \tag{43}$$

$$q_{iy} : \frac{L}{2} q_{iy} \left(\frac{c}{2\eta_p \omega_p} \gamma_p^2 - \frac{c}{2n_{oi}\omega_i} \right) \tag{44}$$

Frequency component mixed term:

$$q_{sx} q_{ix} : \frac{L}{2} \left(\frac{2c}{2\eta_p \omega_p} \beta_p^2 \right) \tag{45}$$

$$q_{sy} q_{sx} : \frac{L}{2} \left(\frac{2c}{2\eta_p \omega_p} \gamma_p^2 \right) \tag{46}$$

The quadratic and mixed terms depend on q so the effective frequency will increase linearly with our variable q. For a θ_p of 28.95 degrees and λ_i/λ_s of 810nm, aswell as a crystal length of 0.002m, we get the following constants: Linear:

$$q_{sx} : 6.72 * 10^{-05} \tag{47}$$
$$q_{ix} : 6.72 * 10^{-05} \tag{48}$$

Quadratic:

$$q_{sx} : -1.78 * 10^{-11} q_{sx} \tag{49}$$
$$q_{ix} : -1.78 * 10^{-11} q_{ix} \tag{50}$$
$$q_{sy} : -1.70 * 10^{-11} q_{sy} \tag{51}$$
$$q_{iy} : -1.70 * 10^{-11} q_{iy} \tag{52}$$

Mixed:

$$q_{sx} q_{ix} : 4.19 * 10^{-11} \tag{53}$$
$$q_{sy} q_{iy} : 4.35 * 10^{-11} \tag{54}$$

Trailing Exponential function:

$$e^{-i\left(\frac{|q_s|^2 z}{2k_s} + \frac{|q_i|^2 z}{2k_i}\right)} \tag{55}$$

With a z value of 35mm and a k_s, k_i of 7757018 you get:

$$\frac{z}{2k_s} = 2.26 * 10^{-9} \tag{56}$$
$$qsx : qsx 2.26 * 10^{-9} \tag{57}$$

From this we can see, that at q values above 29734 (6.72e-5/2.26e-9) the trailing exponential function is dominating the effective frequency of the signal. At a qsx_max of 350000 you get an effective frequency x_{max} factor of 0.000791. (By multiplying qsx_max by Eq. 55)

$$q_{sampling} = \frac{1}{2x_{max}} = 632.1 \tag{58}$$

If we then take our original q_{max} of 350000 and divide by our $q_{sampling}$ we get:

$$Samplecount = \frac{q_{max}}{q_{sampling}} = \frac{350000}{632.1} = 554 \tag{59}$$

Since we go from -qmax to + qmax essentially we need twice the samplecount so 1108. Which is at least close to the values we actually need. For Type-I Phase Matching we actually need 800 samples.

For Type-II Phase Matching we need 1200.

5 Install Instructions

There are two main ways to install the necessary libraries for the simulation program associated with this document. One way is through a cmake file combined with vcpkg, a package manager for c/c++. The other way is the normal gcc way and including the header files and the respective libraries. It is very much recommended to install it using the standard gcc (msys2) way as this one is a lot faster and is tested more. (The file is also more recent, however you would just need to copy the spdc from spdc_gcc)

The libraries that are needed are:

- Eigen 3.4 (When using a different Eigen version you may need to change some included headers)
- fftw 3.3.10

The program also uses some c++20 features so your compile should be able to handle that. (Could be rewritten though, just uses std::format)

5.1 cmake + vcpkg

!Important not spdc is not 100% up to date use gcc one for fastest version. As stated above vcpkg is a package manager for c/c++, which can then be integrated into a cmake build system. Once you have vcpkg setup, installing additional packages is rather straightforward, the usually tedious process of building libraries gets simplified to calling vcpkg install <package>. However since we only use two packages, namely Eigen and fftw (and Eigen is a header only libary), building the libraries from scratch might actually be faster. To use the cmake + vcpkg flow you will need to follow these steps:

- Install vckpkg.
- Set vckpgk toolchain environment variable VCKPG_ROOT to <path-to-vcpkg-root>.
- Install packages Eigen and fftw. With the commands vcpkg install eigen3 –triplet <your-triplet> (x64-windows-dynamic) and vcpkg install fftw3 –triplet <your-triplet>.

The cmake files are already set up to include the libraries of vcpkg, so if the toolchain environment variable is correctly set you will only need to call the following two cmake commands, from within the build directory.

- cmake .. -DCMAKE_BUILD_TYPE=Debug –preset debug (you can also set Release –preset release), and then
- cmake –build .

5.2 Standard gcc way

The other way you can install this program, is by building fftw3 from scratch and then including and linking to it as a libray. For this you will only need a compiler. Gcc is recommended here mainly just because it is the compiler that was used, so it is tested to work, however any compiler should work. If you are on windows you will also need to make sure, that you have msys2 installed. For macos you should have Apple developer tools installed (according to the fftw3 wiki, which is from about 10 years ago, so this might have changed) (if you can compile with gcc or any other compiler you should be fine hopefully) . Make sure gcc is up to date (for c++20 features) gcc version 15.2 was used here.

5.2.1 Eigen

Installing Eigen is very straightforward as you only need to copy the git repository and then later pass the include directory to gcc. The version used is 3.4, the newest version would be 5.0, however due to compatability reasons with the vcpkg version 3.4 is used. (When upgrading to a newer version you should only have to change the includes). Since the correct version of Eigen is already included in the project directory you only need to pass it as a include directory flag to gcc. (For which there is a command below)

5.2.2 FFTW

Installing fftw is more work, as you will need to firstly build the libraries from source. For that download the unix version of fftw3.3.10. Then

- unzip the file to some location
- call a ./configure file in msys2 or just a regular terminal in macos/linux, from within the recently unzip folder. (You need some configure flags which are listed below)
- Then call make
- and lastly make install.

Then you should have a new library in /usr/lib and a new header in /usr/include.

5.2.3 configure fftw

Depending on what operating system you use you may need to change some configure flags. Since we are using the float32 version of fftw3 mainly to save memory space, you will need to compile the correct fftw3 library. This is done through setting a specific ./configure flag. The float32 flag is –enable-float. The following configure command was used to build a fftw3 library on windows:

```
./configure --with-our-malloc16 --with-windows-f77-mangling
--enable-shared --disable-static --enable-threads --with-combined-threads
--enable-portable-binary --enable-float --with-incoming-stack-boundary=2
--disable-alloca
```

When installing on a unix system (and on macos) the following command should suffice:

```
./configure --enable-shared --disable-static --enable-threads --enable-float
```

Afterwards you will need to type:

```
make
```

And then:

```
make install
```

After that you can compile the project. The command for that with gcc is as follows (make sure you are in msys2 if you use windows):

```
gcc spdc.cpp -o spdc_o3 -lfftw3f -Ieigen -std=c++20 -L/usr/lib/
-I/usr/include/ -lstdc++ -O3
```

You can leave out the -O3 flag for a build with no/less optimizations.

5.3 Python Visualization

Included in the github repository is also a python program to visualize the final binary. The C++ program will tell you with what arguments to call it. Alternatively you can also pass in -h or –help for help. The program has 3 Packages that need to be installed. They are

- matplotlib
- numpy
- argparse (Only need to install if you have python 3.2 or lower (somewhere around 2015) otherwise its included in the standard library)

If you have matplotlib and numpy installed in your global python you dont need to do anything else. It is however recommended to more or less always use a virtual environment. (It just Decouples a set of packages from your python. So you can have 5 different Virtual Environments(Venvs) where you can install different packages in each and one of them and they will not interfere with each other). To create a Virtual environment you simply have to call : (Python may be called something else on Mac or Linux (I think its py3 or python on linuz))

```
py -m venv <name_of_venv>
```

After that you can:

```
cd <venv_folder_name>
cd Scripts
activate (may be called a bit different or
you may need to use ./activate on unix)
```

You will then see an indicator on the left with the name of you venv. Then you will need to install the two packages.

```
pip install matplotlib numpy
```

After this you can call the visualizer program. And once you are done you can just type exit and you will quit out of your venv. And the packages will be *deloaded*. Below are step by step pictures on how to setup and use venv:

6 How to use the Program

This section will focus on how to use the program. Once you have compiled the program you can call it by running spdc.exe or spdc_o3.exe. It will then ask you to input some Angle (in Degrees) for theta_p. After that it will ask

```
PS D:\Code\SPDC_gcc> .\spdc_o3.exe
Please type in Theta_p as an angle:
41.78
```

you for an alpha angle value again in degrees. For more Information on this alpha value you can look at Figure 4. of [KAB+20, P. 8] as well as equation 33 on page 9 of the same paper.

After this you are asked to insert a name for the binary file that will be

```
Please type in alpha as an angle (if you don't know just leave as 0.0):
0
```

created.

Then you will need to choose between type-I or type-II phase-matching.

```
Please type in File_name:
test_max_41_78.bin
```

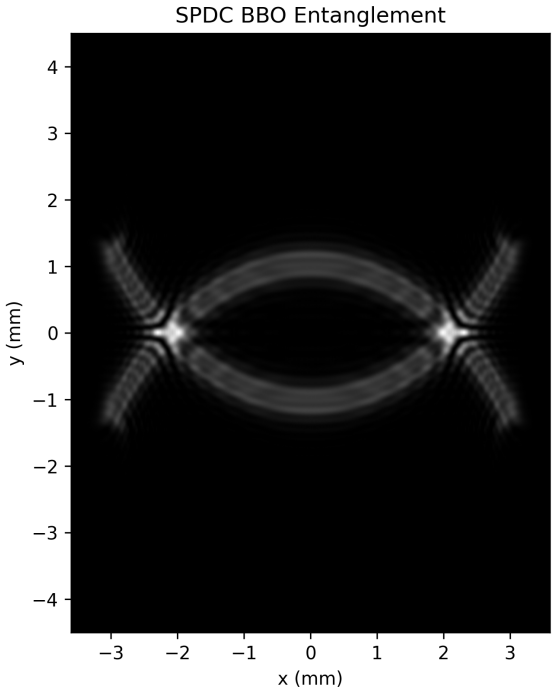
After this point the program starts and will print out some debug information as well as progress bar and an estimate of memory usage. In our case we chose type-II phase-matching, with an estimated memory consumption of 4.8 GB. The values titled *Frequency Constants* here are the values that correspond to the ones used in equations 42-52. After finishing the program will again

```
Please choose the type of phase_matching (Input 1 for type-1; and 2 for type-2):
2
Constants : 0.0763327   1.00569 1.04184 1.63357

omega_i debug: 2325492209395040
k_p: 15514037.795505151  k_i: 7757018.897752576
Printing Frequency constants for Type-1 phase matching (delta_k_z)
qsx_constant (Quadratic Terms): -1.88508e-11     qix_constant: -1.88508e-11      qsy_constant: -1.73903e-11      qiy_constant: -1.73903e-11
qsx_linear: 7.63327e-05  qix_linear: 7.63327e-05
qx_mixed: 3.99082e-11    qy_mixed: 4.28292e-11
Printing Frequency constants for Type-2 phase matching (delta_k_z)
qsx_constant (Quadratic Terms): -2.06021e-11     qix_constant: -1.88508e-11      qsy_constant: -2.19216e-11      qiy_constant: -1.73903e-11
qsx_linear: 4.53456e-06  qix_linear: 7.63327e-05
qx_mixed: 3.99082e-11    qy_mixed: 4.28292e-11
Estimated Memory Consumption (Total): 4.8 GB
theta_p: 0.729199

qx span: 349066
qy span: 418879
starting malloc and input Data creation. May take a while.
Current Progress: Input Data:
```

print out some debug Information. At the end it will give the python program

with its arguments you will need to call in order to Visualize the resulting binary file. For this to work however you will need to either have matplotlib, argparse and numpy installed in your global python. Or you activate a correct python environment.

The python program will then create a png file that looks as follows: The standard parameters such as the length of the bbo-crystal you can set



inside the cpp file. For this the same values as in [KAB+20] were used. The main parameters that will need to be customized if you encounter the edges *phasing* out. Or some sort of aliasing are the max_momentum you use to calculate the input tensor data and the number of samples. In the following figure you can where in the spdc file you can change those values.

7 Program Availability

You can get the program through the following github link `https://github.com/Wagner-Sebastian/SPDC_Simulation`. You can either use:

```
git clone https://github.com/Wagner-Sebastian/SPDC_Simulation.git
```

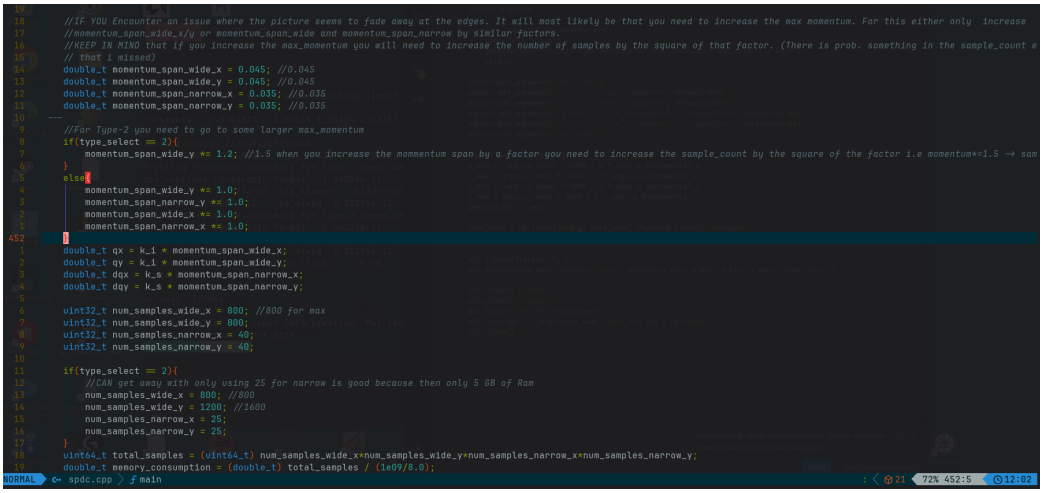


Figure 5: Figure showing where to change the Sample count and q max inside the program

Or simply go to the url above, click on the green Code button and choose download Zip to download an archive file of the repository. This file is included in the repository as well.

References

[KAB+20] Suman Karan, Shaurya Aarav, Homanga Bharadhwaj, Lavanya Taneja, Arinjoy De, Girish Kulkarni, Nilakantha Meher, and Anand K Jha. Phase matching in β-barium borate crystals for spontaneous parametric down-conversion. *Journal of Optics*, 22(8):083501, jun 2020.