\chapter{Update for Cochlear Model Implementation }

\label{sec:cochlear\_model\_optimization}

\citealt{Saboddd2013682} show that the cochlear model equations can be approximated by using the NVidia GPU. An updated version of the hardware requires some optimizations to work with the updated version of CUDA-supporting devices.

\phantomsection

\section{ Updates for Single-Precision Computations }

\label{sec:single-precision-computations-updates}

\phantomsection

\subsection{Serial solution of boundary conditions}

\label{sec:cochlear\_model\_solution\_boundary\_condition}

\begin{figure}

\centering

\includegraphics[width=0.5\textwidth,keepaspectratio=true]{figs/tikztdelta}

\vspace{.2in}

\caption{Flowchart to replace $t+\deltat$.}

\label{fig:tikztdelta}

\end{figure}

\citealt{Saboddd2013682} obtain the boundary conditions from partial difference equation \cref{eq:pressure-relate-to-speed}

by substituting \cref{eq:pressure-equation,eq:ocillates-properties, which yields

\begin{align}

\label{eq:regular-differential-equation-pressure}

&\frac{\partial P(x,t)}{\partial x^2} & = & Q(x) [ P(x,t) - G(x,t) ] \\

&G(x,t) & = & -[R\_{bm}(x)\xi\_{bm}(x,t) + K\_{bm}(x)\dot \xi\_{bm}(x,t) +P\_{tm}(x,t)] \\

&Q(x) & = & \frac{2 \rho \beta}{A M\_{bm}(x)},

\end{align}

with $t$ as a parameter.

\cref{eq:regular-differential-equation-pressure} is a second-order regular differential equation and depends on $x$.

It can be solved by applying the boundary conditions from \cref{eq:pressure-initial-conditions,eq:ow-equations}.

The boundary condition problem is solved at every time point by converting the problem from the continuous to the discrete domain. The cochlear length is divided into $N$ equal sections where $x=n \deltax$ for $0 \le n \le N$ and

The first and second derivatives of $P$ are approximated by a Taylor series in the discrete domain:

\begin{align}

\frac{\partial P(x\_n,t)}{\partial x} & \approx \frac{P(x\_{n+1},t) - P(x\_n,t)}{\deltax} \label{eq:pressure-diff-eq-on-x} , \\

\frac{\partial^2 P(x\_n,t)}{\partial x} & \approx \frac{P(x\_{n+1},t) - 2 P(x\_n,t) + P(x\_{n-1},t)}{\deltax^2} \label{eq:pressure-diff-eq-on-x2}.

\end{align}

We substitute \cref{eq:pressure-diff-eq-on-x} with the first boundary condition into \cref{eq:pressure-initial-conditions}

for $x=0$ to obtain

\begin{equation}

\label{eq:pressure-depends-on-time}

P(\deltax,t) = P(0,t) +\deltax 2\rho C\_{ow}\frac{\partial^2 \xi\_{ow}(t)}{\partial t^2}.

\end{equation}

$\frac{\partial^2 \xi\_{ow}(t)}{\partial t^2}$ is obtained from Eq.~\eqref{eq:ow-equations} and modifies Eq.~\eqref{eq:pressure-depends-on-time} to

\begin{equation}

\label{eq:pressure-depends-on-time-isolate}

P(\deltax,t) = \left(1 +\deltax \frac{2\rho C\_{ow}}{\sigma\_{ow}}\right)P(0,t) + Y\_0,

\end{equation}

where

\begin{equation}

\label{eq:input-equtaion}

Y\_0 = 2\rho C\_{ow}\left[\frac{\Gamma\_{ME}}{\sigma\_{ow}} P\_{in}(t) - \gamma\_{ow} \dxi\_{ow}(t) - \omega^2\_{ow} \xi\_{ow}(t)\right] \deltax.

\end{equation}

The second boundary condition of \cref{eq:pressure-initial-conditions} for $x=L\_{co}$ yields

\begin{equation}

\label{eq:discrete-pressure-zero}

P(x\_N,t) = 0.

\end{equation}

Substituting \cref{eq:pressure-diff-eq-on-x2} into \cref{eq:regular-differential-equation-pressure} for $1 \le n \le N-1$ yields

\begin{align}

\label{eq:pressure-discrete-transfer-equation}

P(x\_{n+1},t) – [2 + \deltax^2 Q(x\_n)] P(x\_n,t) + P(x\_{n-1},t) = \deltax^2 Q(x\_n) G(x\_n,t).

\end{align}

\Cref{eq:pressure-depends-on-time-isolate,eq:discrete-pressure-zero,eq:pressure-discrete-transfer-equation} can be expressed in matrix form as

\begin{equation}

\label{eq:pressure-matrix}

\Lambda \underline{P}(t) = \underline{Y}(t),

\end{equation}

where $\Lambda$ is a tridiagonal matrix with a main diagonal of

\begin{equation}

\left[-\left(1 +\deltax \frac{2\rho C\_{ow}}{\sigma\_{ow}}\right),-(2+\deltax^2 Q(x\_1)),\cdots,-(2+\deltax^2 Q(x\_{N-1})),1\right],

\end{equation}

and the two other nonzero diagonals are unity, except for the (N,N1) element of the matrix, which is zero. The quantities $\underline{P}(t)$ and $\underline{Y}(t)$ are

\begin{align}

\underline{P}(t) & = [P(0,t),P(x\_1,t),\cdots,P(x\_{N-1},t),P(x\_N,t)]^T ,\\

\underline{Y}(t) & = [Y\_0,\deltax^2 Q(x\_1)G(x\_1,t),\cdots,\deltax^2 Q(x\_{N-1}) G(x\_{N-1},t),0]^T

\end{align}

given that $\underline{Y}(t)$ is the solution of Eq.~\eqref{eq:pressure-matrix} and is the pressure of the cochlear partition:

\begin{align}

\label{eq:pressure-discrete-matrix-solution}

\underline{P}(t) = \Lambda^{-1}\underline{Y}(t).

\end{align}

The inversion of the matrix $\Lambda$ was obtained by lower-upper (LU) decomposition.

\subsection{Serial solution of initial condition equations}

\label{sec:cochlear\_model\_solution\_initial\_condition}

A set of initial conditions of second-order ordinary differential equations for $\xi\_{bm}(x,t)$, $\xi\_{tm}(x,t)$, and $\xi\_{ow}(t)$ and a first-order differential equation of $\psi(x,t)$ as a function of $t$ is obtained by \cref{eq:membrane-voltage,eq:ow-equations} along with the substitution of \cref{eq:pressure-equation,eq:ohc-pressure} and \cref{eq:delta-lohc} into \cref{eq:ocillates-properties}.This set is solved by using the modified Euler method for first-order initial condition equations with adaptive step size \cite{nagle1989fundamentals}. The second-order differential equations are solved by defining the first derivative of $\xi\_{bm}(x,t)$, $\xi\_{tm}(x,t)$, and $\xi\_{ow}(t)$ as additional unknown variables $\dxi\_{bm}(x,t)$, $\dxi\_{tm}(x,t)$, and $\dxi\_{ow}(t)$, respectively. Thus, five first-order differential equations must be solved for every point $x$ along the cochlear partition and two additional first-order differential equations must be solved for $\xi\_{ow}(t)$ and $\dxi\_{ow}(t)$. If $N$ is the number of samples along the cochlear partition, a total $5N+2$ initial conditions must be found.

The solution algorithm is briefly explained as follows: Define a typical initial condition problem as

\begin{align}

\label{eq:y-initial-conditions}

\left . \begin{array}{rr}

\frac{dy}{dt} = f(t,y(t)) ,\\

y(0) = Y\_0.

\end{array} \right\}

\end{align}

The solution is found by using the Euler method and yields for step size $\deltat$

\begin{align}

\label{eq:y-step-equation}

y(t+\deltat) = y(t) + \deltat f(t,y(t)).

\end{align}

In particular, for $t = 0$, $y(\deltat) = y\_0 + \deltat f(0,y\_0)$.

However, when the Euler method is used the convergence is ensured only when $t$ is very small (on the order of $10^{-15}$~s ). Such a step size is time consuming and unpractical.

To increase the step size, we used the modified Euler method. An iterative series \{$\omega\_n$\}

is defined for deriving $y(t+\deltat)$ for a known $y(t)$:

\begin{align}

\label{eq:cochlear-section-force-equtaion}

\left . \begin{array}{rl}

\omega\_0 =& y(t) + \deltat f(t,y(t)), \\

\omega\_n =& y(t) + \frac{\deltat}{2} [f(t,y(t)) + f(t+\deltat,\omega\_{n-1})], \quad n \ge 1.

\end{array} \right\}

\end{align}

The first element in the series is obtained by using the Euler method. If the iterative series converges, then $\omega\_n \rightarrow y(t+\deltat)$.

The condition for convergence can be determined by using the Lipschitz constant for a function $f(t,y(t))$ that obeys the condition

\begin{align}

\label{eq:lifchitz-condition}

|f(t+\deltat,y(t+\deltat)) - f(t,y(t))| \le L|y(t+\deltat) - y(t)|

\end{align}

where $L$ is the Lipschitz constant. By substituting \cref{eq:lifchitz-condition,eq:y-step-equation} and replacing $\omega\_{n-1}$ and $\omega\_n$ by $y(t+\deltat)$ in \cref{eq:cochlear-section-force-equtaion} \citealt{Saboddd2013682} get the constraint

\begin{align}

\label{eq:lifchitz-constraint}

\frac{L \deltat}{2} < 1.

\end{align}

Thus, when $\deltat < 2/L$, $\omega\_n$ converges to $y(t+\deltat)$.

Because it is rather complicated to evaluate the Lipschitz constant for the set of the differential equations described above,

an estimate for the Lipschitz constant was derived from \cref{eq:pressure-diff-eq-on-x} as follows \cite{nagle1989fundamentals}:

\begin{align}

\label{eq:lifchitz-estimated-constraint}

\hat{L} = \frac{|f(t+\deltat,\omega\_n) - f(t,\omega\_{n-1})|}{|\omega\_n - \omega\_{n-1}|},\quad n \ge 1.

\end{align}

To solve $5N + 2$ initial conditions, the time step $\deltat$ must satisfy the Lipschitz constraint \eqref{eq:lifchitz-constraint} for every equation. To lower the computation load, \citealt{Saboddd2013682} chose to verify the most sensitive variable, $\dxi\_{bm}$, such that $\deltat < 2/L$ for $N$ equations.

This means that the existing time step $\deltat$ is sufficient for convergence, so the

resulting derivation is retained. The size of the time step is doubled if $max\_N\{\frac{\hat{L}\deltat}{2}\} < 0.25$. When $\hat{L}\deltat \ge 1$, the time step $\deltat$ is too large for convergence and is divided by 2, and the procedure is repeated with the smaller time step.

\subsection{Run-time estimation}

\label{sec:input-interpolation}

The algorithm of \citealt{Saboddd2013682} starts by setting time $t$ and variables $\xi\_{bm}$, $\xi\_{tm}$, $\xi\_{ow}$,and $\psi$ to zero to fulfill the initial conditions \eqref{eq:ow-initial-conditions}, then computes $\underline{Y}(t)$. The next phase is to derive $\underline{P}(t)$ by solving equations system~\eqref{eq:pressure-discrete-matrix-solution} by using LU decomposition. The time $t$ is then updated to $t+ \deltat$ and the next iteration is started. The input of the algorithm is the acoustic stimulus of $P\_{in}(t)$. $F\_s$ denotes the sample frequency of the system, so the interval time between two consecutive samples is $T\_s=\inv{F\_s}$. Because the algorithm convergence requirements

yield greater time resolution then the input signal, a linear interpolation is applied to compute the input pressure for

time $t$ as follows:

\begin{align}

\label{eq:input-sampling}

\begin{array}{ll}

n\_s = \floor{\frac{t}{T\_s}},\quad \Delta \tau = \frac{t-n\_s T\_s}{T\_s}, \\

P\_{in}(t) = (1-\dtau) P\_{in}(n\_s T\_s) + \Delta \tau P\_{in}((n\_s+1) T\_s).

\end{array}

\end{align}

For convergence of the partial differential equation and according to the Courant—Friedrichs--Lewy condition \citealt{CFL\_condition} and given $N = 256$, the condition $\frac{c\deltat}{\deltax} < 1$ is imposed where $c$ is the speed of sound inside the perilymph. Given a cochlea length of 0.035~m, the condition $c = 1500 $~m/s is met when $\deltat < 9.11 \times 10^{-8}$~s. In the simulations, the typical time step was in the range of $10^{-7}$ to $10^{-7}$~s and the maximum time step was $ 10^{-6}$~s, with 6 iterations guaranteeing convergence. \cite{Saboddd2013682} calculates 226 instructions per iteration with $2.5 \times 10^{-7} $~s as the average calculated for $N = 256$ cochlear sections. The estimated execution time for 1 s of acoustic simulation is $\frac{10^7}{2.5}\times 6 \times 226 \times 256 = 1.388 \times 10^{12}$ clock cycles, given a 3 to 4 GHz CPU, and about 350 to 500 s of execution time for 1 s of stimulus.

\section{Parallelizing the algorithm}

\label{sec:parrallelizing-the-algorithm}

The serial solution of the algorithm depends on the LU decomposition in the longitudinal dimension and on the iterative steps in the time dimension. This method is several orders of magnitude longer than a real-time voice signal. \citealt{Saboddd2013682} developed a massive parallel-algorithm solution that efficiently uses massive the parallelism of the GPU.

\phantomsection

\subsection{Parallelism in the time dimension}

\label{sec:parrallelizm-in-time-dimension}

The model uses a one-dimensional description of the cochlear partition, so it was natural to choose a one-dimensional

grid in the parallel implementation. The cochlear partition was divided into $N\_x$ sections along the $x$ axis, where each section is

processed by one thread. The parallel algorithm uses large amounts of constant data in its computations. The data are read

from the CPU host to the global memory of the GPU during the initialization phase of the launch of the CUDA kernel. At

the end of a time step, when that time step converges, the algorithm stores the current results and advances to the next

time step. The memory traffic can dramatically slow the execution time. To prevent excessive memory accesses the

equation is solved for a long time interval in one kernel launch, so that the traffic to and from the global memory occurs

once for a long segment of input processing. Each thread handles one cochlear section and resolves the BMV

for a block time interval of $t\_{bti}$. The thread stores results to global memory once each $\deltat\_{OUT}$ seconds, where $\deltat\_{OUT}$ is the desired temporal resolution of the output.

\begin{figure}

\centering

\includegraphics[width=0.9\textwidth,keepaspectratio=true]{figs/cochlea\_pic1}

\vspace{.2in}

\caption{Partitioning in the time dimension. Each bar represents a time interval handled by one CUDA block. Overlapping sections are discarded.}

\label{fig:time-divisions}

\end{figure}

To achieve parallelization in the time dimension \citealt{Saboddd2013682} partitioned the processing into $N\_t$ CUDA blocks, such that each

block solves the equations for all the cochlear sections for a different time interval. Thus, the number of time intervals

equals the number of CUDA blocks. The time intervals are handled by blocks that are not completely independent, and

two consecutive time intervals, which are mapped to two consecutive CUDA blocks, have an overlapping interval of $t\_{BOP}$ seconds. This

requires each block to run the algorithm for a total of $t\_{BTI} + t\_{BOP}$ seconds. That is, a specific block BL solves the model for

the interval $T^{(BL)} = [(BL-1) t\_{BTI} \le t \le BL (t\_{BTI} + t\_{BOP})]$. Because the output is required to have a temporal resolution of $\deltat\_{OUT}$, for each kernel launch each thread returns $\frac{t\_{BTI} + t\_{BOP}}{\deltat\_{OUT}} + 1$ results of a specific cochlear section. The results computed in the first $t\_{BOP}$ seconds for each block are discarded.

\figref{fig:time-divisions} graphically illustrates the partitioning of the time dimension and shows the time interval processed by each block, including

the discarded (overlapping) interval. The nonoverlapping intervals processed by each block are connected together to form the final result.

\subsection{Parallelism in longitudinal dimension}

\label{sec:parrallelizm-in-longitudinal-dimension}

\citealt{Saboddd2013682} found that the boundary conditions described by \cref{eq:pressure-discrete-transfer-equation} are the main obstacle for parallelism because of the interdependency between the sections. The serial solution is obtained by using LU decomposition to solve the equation. However, this is not a good solution for a massive parallel algorithm and is thus incompatible with the use of a GPU.

Basic smoothers (numerical algorithms used to solve a system of linear equations) such as the Jacobi method and

successive over relaxation (SOR) can be used to parallelize a linear system. The purpose of smoothers is to reduce, or smooth,

in an efficient way the approximation error \cite{Yavneh94onred}. More advanced smoothers are difficult to solve in parallel \cite{GoSt11CR}.

Taking advantage of the tridiagonal shape of the equation set, we used Jacobi relaxation and implemented it with only

a few floating point operations for each Jacobi iteration. Manipulating \cref{eq:pressure-discrete-transfer-equation} yields for $1 \le n \le N-1$

\begin{equation}

\label{eq:pressure-parallel-estimation}

\left .

\begin{array}{lll}

\hat{Q}(x\_n) &= & \inv{-[2+\deltax^2 Q(x\_n)]}, \\

P(x\_n,t) & = & (Y\_n(t) - 1 P(x\_{n-1},t) - P(x\_{n+1},t)) \hat{Q}(x\_n)

\end{array} \right\}

\end{equation}

by denoting

\begin{equation\*}

\begin{array}{lll}

\hat{A}\_l & = & [0,1,...,1,0]^T,\\

\hat{A}\_u & = & [1,1,...,1,0]^T,

\end{array}

\end{equation\*}

and

\begin{equation\*}

\hat{A}\_m = [\inv{-(1 +\deltax \frac{2\rho C\_{ow}}{\sigma\_{ow}})},\inv{-(2+\deltax^2 Q(x\_1))},\cdots,\inv{-(2+\deltax^2 Q(x\_{N-1}))},1]^T .

\end{equation\*}

Combining \Cref{eq:pressure-parallel-estimation,eq:pressure-depends-on-time-isolate,eq:discrete-pressure-zero} yields

\begin{equation}

\label{eq:pressure-parallel-estimation-discretization}

P(x\_n,t) = (Y\_n(t) - A\_l[n] P(x\_{n-1},t) - A\_u[n] P(x\_{n+1},t)) A\_m[n].

\end{equation}

$\underline{P}(t)$ was computed by running a Jacobi relaxation for $J$ iterations. \citealt{Saboddd2013682} assumes that the convergence tests catch the cases in which the computation of $\underline{P}(t)$ does not converge after $J$ iterations. The equation of iteration $j$ is

\begin{equation}

\label{eq:pressure-parallel-estimation-discretization-iteration-j}

P^j(x\_n,t) = (Y\_n(t) - A\_l[n] P^{j-1}(x\_{n-1},t) - A\_u[n] P^{j-1}(x\_{n+1},t)) A\_m[n] ; 0 \le n \le N,

\end{equation}

where $\underline{A}\_l$, $\underline{A}\_m$, and $\underline{A}\_u$ are constants solved in advance and stored in shared memory. Division by mass is replaced by multiplication by $\inv{M\_n}$. The pressure $P(x\_n,t)$ is computed per cochlear section $x\_n$ on different threads. Thus, each thread performs three multiplications, two add or subtract

operations, and seven shared-memory accesses for one Jacobi iteration. However, interthread synchronization is required

after each Jacobi iteration because each thread uses the result of its neighbors for the next iteration. Because the synchronization

between threads that belong to the same block is much more efficient in CUDA, \citealt{Saboddd2013682} limited $N\_x$ to the size of each CUDA block. Whereas the CUDA block can contain up to 1024 threads, the main limitation for resources were the shared memory available per block, so $N\_x = 256$. This resolution ensures convergence of the algorithm and meets the resource-limitation demands on block size. The block starts solving the model for time $t = t\_{BTI} (BL - 1)$.

\section{Updating the Computing Algorithm to Reduce Errors}

\label{sec:updating-BMV-model}

In \cref{sec:input-interpolation}, the signal pressure $P\_{in}(t)$ is used in the $Y\_0$ initial condition because of the pressure transmitted through the oval window [\cref{eq:input-equtaion}], so we need to evaluate $P\_{in}(t+\deltat)$, since $P\_{in}$ is defined for

\begin{equation}

P\_{in}(T\_s k) ,\quad k \in \mathbb{N}.

\end{equation}

To evaluate $P\_{in}(t)$ for every time $t$ that does not satisfy the integer part of $t/T\_s$, a linear time interpolation was implemented [\cref{eq:input-sampling}].

% here was pin T\_s discretization formula

This was implemented in the code by using a single-precision variable to indicate time from the start of the signal denoted $t$. Another indicator for the start of the block time interval, $t\_{bti}$, offset from the start, is denoted $nearest\_s$. The formula for $\Delta\tau$ is

\begin{align}

nearest\_s = \floor{\frac{t-t\_{bti}}{T\_s}} ,\\

\dtau = \frac{t-t\_{bti} - nearest\_s T\_s}{T\_s}.

\end{align}

$\dtau$ is a single-precision variable, and all arithmetic binary floating point calculations on CUDA-compatible devices are done following the IEEE-754 2008 standard \cite[Section~G.2]{NVIDIA\_Programming\_guide}. The IEEE 754 floating point register is divided to 23 mantissa, eight exponents, and one sign bit. The decimal precision is

\begin{align}

d\_{precision} = \frac{23 \log(2)}{\log(10)} = \frac{23}{\log\_{2}10} =7.2.

\end{align}

If $\dtau \leq 10^{-d\_{precision}} t$, then

\begin{align} \label{eq:undetermine-input}

P\_{in}(t+\dtau) == P\_{in}(t).

\end{align}

\citealt{Saboddoron20131215} used 0.32 input signals, which means that, for $t> 0.1 $~s, and for \cref{eq:undetermine-input} for $\dtau < 10^{-8} $~s, the convergence for this interval is not guaranteed. Two methods are considered to solve this time-interpolation domain:

\begin{enumerate}

\item By using double-precision calculations we lose 87.5\% and 96.88\% for throughput for computing capabilities 2.0 and 6.0, respectively. With four units of add, multiply, and fused add-multiply, we use double precision for every 32 or 128 single-precision variables, respectively.

\item By using single precision, \cref{eq:input-sampling} represents time as a combination of $n\_s$ and $\tdel$, so each time we need $t$ we substitute $t =n\_s T\_s + \tdel$. The time step was set to its maximum value $\dtau=10^(-6) $~s (see \cref{sec:input-interpolation}). With these parameters, the algorithm ensures

\begin{equation}

\tdel \leq T\_s + \dtau,

\end{equation}

with $\dtau \ll T\_s$ we claim that, effectively, $\tdel \leq T\_s$ \cite{odedst2017}. With $T\_s=50\ \mu $s, a linear approximation of $P\_{in}(t)$ is updated for

\begin{equation}

\dtau > 10^{\lg\_{10}(T\_s)+d\_{prescision}} \approx 3\times10^{-11}\text{ s},

\end{equation}

regardless of $t$.

\end{enumerate}

\subsection{Error measurements}

\paragraph{Basilar Membrane Velocity}

The difference between the output of the old interpolation method and that of the proposed interpolation method can be shown for both the BMV and the nerve response.

\begin{figure}[ht]

\setlength{\subfigcapmargin}{.1in}

\centering

\includegraphics[width=0.95\textwidth,keepaspectratio=true]{figs/SignalsEnvelope}

\vspace{.2in}

\caption{Envelope of Basilar Membrane Velocity section in response to 4 kHz tone 36 dB with old (red) and new (blue) interpolation methods.}

\label{fig:cochlea-interpolation-effect-on-bmv}

\end{figure}

\cref{fig:cochlea-interpolation-effect-on-bmv} shows the difference between the peak envelopes of BMV in a section 0.64 cm from \ac{ow} for both the old and new algorithms. The response at relatively low power does not have feedback ripples, so an expected envelope needs to be approximately constant (except for the transition state). With the old interpolation method we see rippling and a peak jump at 0.025 and 0.25~s, because precision drops as a direct function of $log\_{10}(t)$.

\begin{figure\*}[ht]

\setlength{\subfigcapmargin}{.1in}

\centering

\includegraphics[width=0.95\textwidth,keepaspectratio=true]{figs/LHigh\_4000HZ\_SignalPowerInDB\_36\_mode\_17}

\caption{Result yielded by new interpolation method for $P\_{in}(t)$. At 4 kHz the nerves cannot follow the membrane phase and gives a constant yield proportional to $|P\_{in}(t)|$, as observed experimentally.}

\label{fig:cochlea-interpolation-effect-on-hsr-fibers-correct}

\vspace{.2in}

\includegraphics[width=0.95\textwidth,keepaspectratio=true]{figs/LHigh\_4000HZ\_SignalPowerInDB\_36\_mode\_2}

\caption{Result for old method of interpolation. We see the nerve response change phase both temporarily and spatially in response to the degraded $d\_{precision}$.}

\label{fig:cochlea-interpolation-effect-on-hsr-fibers-incorrect}

\end{figure\*}

HSR fibers excitation $\lambda\_{high}$ measured in spikes per second, in response to 4 kHz tone 40 dB with the old (\cref{fig:cochlea-interpolation-effect-on-hsr-fibers-incorrect}) and new (\cref{fig:cochlea-interpolation-effect-on-hsr-fibers-correct}) interpolation methods. For a low-amplitude input and a single-tone signal, the ANR is expected to be cyclic (except in the transient stage).

As \citealt[Ch~6]{Saboddd2013682} showed, we need to ensure that our mismatches from the reference software are less than those for the previous version because this work intends to calculate the ANR difference measured both for the BMV and $\lambda\_{high}(x,t)$ from \cref{eq:an-response}.

Thus, both velocity and auditory nerve errors are measured.

For $\dot{\xi}\_{bm}$, the old and new interpolation errors are tested and compared.

\begin{enumerate}

\item We define the average speed for the membrane section with a normalized input of power $P$:

\begin{equation\*}

|\dot{\xi}\_{bm}[i,P]| = \inv{t\_{end}-t\_{start}+1}\sum\limits\_{t=t\_{start}}^{t\_{end}}|\dot{\xi}\_{bm}[i,P,t]|,

\end{equation\*}

which defines the average speed of the signal from $t\_{start}$ to $t\_{end}$ in longitudinal section $i$.

\item To measure error for signals with range of amplitudes, the energy of the membrane measured by the reference program is

\begin{equation\*}

E\_{ReferenceMean}[P]= \sum\limits\_{i=0}^{N}|\dot{\xi}\_{bm}[i,P]|\_{referenceSoftware}^2.

\end{equation\*}

\item We then measure the energy difference between the CUDA and reference program for both interpolation methods:

\begin{equation\*}

E\_{InterpolationError}[P]= \sum\limits\_{i=0}^{N}[|\dot{\xi}\_{bm}[i,P]|\_{CUDA}-|\dot{\xi}\_{bm}[i,P]|\_{referenceSoftware}]^2.

\end{equation\*}

\item The normalized error is a function of the input signal power $P$:

\begin{equation\*}

Err[P]= \frac{E\_{InterpolationError}[P]}{E\_{ReferenceMean}[P]}.

\end{equation\*}

\end{enumerate}

Because we want to calculate the JND from the results, which requires testing for multiple pure tone inputs, we choose three frequencies to examine the effectiveness. With an oval window self-frequency of 1 kHz and the AC filter as described in \cref{tab:Lambda-parameters}, 0.5, 1, 2, and 4 kHz are examined as below cutoff and in the AC range, on the oval window cutoff and on slope of the AC filter, slightly after slope of AC filter and significantly DC response,

\begin{figure}[ht]

\setlength{\subfigcapmargin}{.1in}

\centering

\includegraphics[width=0.95\textwidth]{figs/diffrential\_Velocity\_145msec}

\caption{ Mean-error difference improves for all frequencies and powers of relative error of $0.4\times10^{-3}$. For lower frequencies, the error decreases to less than half of its former value.}

\label{fig:cochlea-interpolation-difference}

\end{figure}

\paragraph{Auditory Nerve Response}

The AN errors measured as a function of the departure from $\lambda\_{spont}$ to ensure the rate reflects the JND measurements. We use HSR fibers because they are dominant when searching for the JND.

\begin{enumerate}

\item Define the average activity above the static level for cochlea section $i$ with signal power $P$ and interval $[t\_{start},t\_{end}]$:

\begin{equation\*}

\Delta\lambda\_{AN}[i,P] = \inv{t\_{end}-t\_{start}+1}\sum\limits\_{t=t\_{start}}^{t\_{end}}(\lambda\_{AN}[i,P,t]-\lambda\_{spont}).

\end{equation\*}

\item Create reference software energy equivalent from the square of $\Delta\lambda$ along the cochlea:

\begin{equation\*}

E\_{\lambda~ReferenceMean}[P]= \sum\limits\_{i=0}^{N}(\Delta\lambda\_{AN}[i,P])\_{referenceSoftware}^2.

\end{equation\*}

\item Measure the equivalent error for cochlear sections for both interpolation methods:

\begin{equation\*}

E\_{\lambda~InterpolationError}[P]= \sum\limits\_{i=0}^{N}[(\lambda\_{AN}[i,P])\_{CUDA}-(\lambda\_{AN}[i,P])\_{referenceSoftware}]^2.

\end{equation\*}

\item Finally, normalize the error:

\begin{equation\*}

Err\_{\lambda}[P]= \frac{E\_{\lambda~InterpolationError}[P]}{E\_{\lambda~ReferenceMean}[P]}.

\end{equation\*}

\end{enumerate}

\begin{figure}[ht]

\setlength{\subfigcapmargin}{.1in}

\centering

\includegraphics[width=0.95\textwidth]{figs/diffrential\_AN\_145msec}

\caption{ Mean-error difference improves for all frequencies. However, the effect is most significant for 4~kHz, where the error is minimized from 16\% to less than 1\% for 40 dB, which allows the new interpolation fit to calculate the JND whereas the old version errs significantly.}

\label{fig:cochlea--an-interpolation-difference}

\end{figure}

%\paragraph{Time Measurements}

%Here, to measure reference to original program improvements, $SPLRef$ will be used as $1.5\cdot10^{-8} Pa$ To measure speed improvement with the fixed algorithm, 2 parameters were check

%\begin{enumerate}

% \item number of $t+\deltat$ operation needed per $T\_s$ interval, to assess average of necessary compute $\dot{\xi}\_{bm}(\underline{x},t),\xi\_{bm}(\underline{x},t),\xi\_{tm}(\underline{x},t),\xi\_{ow}(t),\dot{\xi}\_{ow}(t)$ and $\psi(\underline{x},t)$ from \cite[Fig~3.1]{Saboddoron20131215}

% \item number of Jacoby relaxations

%\end{enumerate}