notes 2/11/2015

- maybe discuss UI a little more
- how do I make the most of the underlying model
- is the strength of the model
- simply that it's realistic
- or are there additional control mechanisms that I can make available
- we have to see how the project develops, and then think about the best way of actually doing it
- another direction is to make something that's not necessarily optimal
  - to compare the element methods etc.
- state the *main* research questions in bullets as well as the auxiliary
- in what ways will I have developed additional skills or new ways of problem solving, by the end of the year?
- UI is one area
- on a technical level, then solving the problems here might also work!
- but maybe have a plan B
- e.g. comparing the element methods, looking for realism
- also consider branching
- a 'real-time' branch vs a super-realistic branch
- or a 'high-quality' render
- sonically, papers are not very useful
- better off listening to stuff myself
- therefore, actually building the stuff
- try to open up several directions I *could* go in
- read!
- concentrate on branching
- rather than having 'one empirical version'
- try working faster, maybe at the expense of code quality etc.

The current problem:

- The mesh sounds weird at high frequencies
  - cause: dispersion error
    - plane waves travel at slightly different speeds depending on their frequency and direction relative to the mesh orientation (Duyne & Smith, 1996)
  - solutions:
    - find usable bandwidth of simulation for which error is within some tolerable limit, only use this bandwidth
      - will require analysis of dispersion error
      - analysis by Hacıhabiboglu, Günel, & Cvetkovic (2010) suggests that directional error for the tetrahedral mesh is significantly greater than the cubic mesh (but under what circumstances?)
      - therefore *may not be appropriate* for mic modelling
      - the same paper mentions that the magnitude error of the tetrahedral is lowest for the same spatial sampling period
      - but the referenced paper (**???**) seems to say otherwise?
    - try to correct the error somehow to increase the usable bandwidth

        · investigate frequency warping to mitigate frequency-dependent error

Dispersion analysis

- dispersion analysis is achieved by applying a von neumann analysis directly on the FDS. (Duyne & Smith, 1995)
  - try to find out how to do a dispresion analysis
    * (**???**) might be useful
    * take the difference equation for the system
    * consider it to be in continuous space by replacing sample points in space with generalized impulse functions
      · I don't understand this
    * take the spatial fourier transform of the difference equation, replacing function points with corresponding linear phase terms
      · I don't understand this either
    * this gives us a filter equation with a coefficient
    * we can find the coefficient in terms of the linear phase at the waveguide mesh points (I think)
    * we can find the phase distance travelled in one time sample using this coefficient
  - do the thing
    * I wrote a python script that replicates the measurements in Duyne & Smith (1995)
    * but somehow it's not quite right
      · read (**???**) to get a better idea of how dispersion analysis works
      · *discuss with Alex!*
    * using the equations for dispersion factor in Campos & Howard (2005) I get results that mirror those *in this paper* but the mesh orientation is different to that of Duyne & Smith (1995) so I can't compare very easily
  - write a program which finds the maximum allowable bandwidth for a given maximum dispersion speed error
    * done, but the python script still doesn't seem perfect (doesn't mirror exactly the diagrams in Duyne & Smith (1995)

** TODO ** * I've read about speed error - now I need to read about other kinds of dispersion error

Dispersion error reduction

- (**???**)
  - Shows that dispersion error can be reduced by frequency warping
    * because error in the *interpolated rectangular mesh* is almost independent of propagation direction
      · may not extend to the tetrahedral mesh because error here is not particularly uniform
    * error might be presented as a function of *spatial* or *temporal* frequency
      · I'm not sure I understand this distinction, maybe ask *Alex*?
      · I think it might be trivial to convert from one to the other as the spacing of the mesh is governed by temporal sampling frequency (or vice versa) anyway
  - based on Campos & Howard (2005), it might be most efficient to just oversample the dwm mesh to a point where the error is within acceptable limits
    * this is definitely the most *memory* efficient way, not necessarily the most *time* efficient
- yeah let's just oversample if this becomes a problem

Justification for the Tetrahedral Mesh

- according to Campos & Howard (2005)
  - the main drawbacks of DWM are

        * dispersion error
        * boundary discretization error

- ideal wave propagation speed is sqrt(1/N) where N is the number of dimensions (Campos & Howard, 2005)
- Campos & Howard (2005) the tetrahedral mesh:
    - has the lowest grid sampling efficiency (grid density required to obtain a given bandwidth) **BAD**
    - has a lower max dispersion error than the rectilinear mesh at the maximum theoretical frequency of the mesh
    - has the lowest grid bandwidth but also the lowest dispersion error at the bandwidth
    - is relatively the most efficient method as dispersion requirements become more stringent - at least twice as fast as other meshes at 5% dispersion error (and I'm aiming for 1% so I guess it'll be even faster there)
- frequency warping - requires constant magnitude error at every angle
    - frequency warping therefore only works on interpolated cubic mesh
    - tetrahedral mesh can't be frequency warped as the magnitude error varies
    - the point of frequency warping was to try to reduce the computational load by reducing the number of nodes required (by increasing the viable bandwidth).
        * it will only be viable if the extra cost per-node followed by correction is lower than the cost of just oversampling the mesh

** TODO **

Melding of the DWM and geometric models

- (**???**)
    - (**???**)
    - (**???**)
    - Southern, Siltanen, & Savioja (2011)

Modelling of ambisonic receivers in the dwm mesh

- Southern & Murphy (2007)
    - presents
        * a process to encode the rir into second-order spherical harmonics
            · using the blumlein difference technique
        * also processing of the receiver array to enhance usable bandwidth
    - requires a very small grid spacing (around 3mm) so that multiple 'pressure-sensing' nodes can be placed with the necessary precision
        * probably not practical unless longer processing times are allowed
        * even with this grid spacing the 'frequency resonpose is not ideal'
            · could be combatted with even smaller grid spacing, but this is probably not possible within a reasonable amount of time
- Hacıhabiboglu et al. (2010)
    - modelling of directional point-like receivers
        * doesn't require oversampling for extra receivers
        * does require oversampling for directional accuracy
            · although the amount over oversampling required still needs investigation and I don't understand the maths
            · or maybe I just get 'close enough' with the directional modelling as directional low-frequency cues won't be that important anyway
        * seems like a pretty straight-forward method once I have the maths worked out

** TODO ** * see whether it's possible to optimize for time efficiency by using local memory on the GPU

16/11/2015

- can I do 2D reverb tail estimation?
- can I do variable grid spacings for microphone placement
  - I don't think so
- is it worth doing a bit more estimation, and aiming for a real-time model?

More Modelling of directional receivers

- need a good way of modelling microphone diffuse-field response
  - (**???**) uses a tenth-order minimum-phase iir filter
  - ** TODO ** can I do this too?
    - I mean, probably
  - I have a test-case up and running, but my integrator is nonsense
  - I checked the integrator, seems to work now
  - I generated some graphs demonstrating directionality for the cardioid mic
    - the good news is, it definitely works to some extent
    - the bad news is, the error seems quite large
  - test some other polar patterns
  - check the actual error between the actual and desired polar patterns
  - checked the directional error
  - I implemented an hrtf receiver for the waveguide mesh today
    - ** TODO ** tests and whatnot

7/12/2015

Done

- send Alex a minimal set of documents to read over christmas
  - plus the github repo
- have a look for existing software packages that I can test/compare with
  - doesn't look like there's anything
- look into rotating model to find minimum bounding box
  - it's a cubic algorithm, may not be worth it
  - let's ignore this for now
  - (**???**) is the paper to read
- move grid based on receiver position (supposing gaussian pulses are used as input)

TODO

- go through code with Alex
- check error calculations, try to replicate
  - max error of 19 vs 15 degrees
- look into validation
  - how do I validate questionnable bits of the project?
  - get in contact with damian murphy at York?
    - he might have time to look at my results etc.
- think about unit tests
  - have some way of quickly generating, verifying output

- models, scripts, focused towards testing certain parts of the engine * it's ok if they need human verification, better than no verification
    - very simple models that facilitate certain reflection patterns
    - cube to test reflection times etc.
    - cube to test a few different materials
      - coefficients above, below 1, 0
      - do I get the reverb times I expect?
    - comb filtering, flutter echo in larger spaces
    - test the error
    - how do my two models differ?
  - start simple, with no ambiguity about the expected result
- it would be interesting to actually TEST the frequency-dependent error
  - what's good enough?
- revisit filters?
- might allow me to optimise for different cases
  - speed
  - accuracy
  - other… (?)
- validate the two models against one another
  - same surfaces, sources, recievers - how do the outputs differ
- work on **boundary conditions** next
  - try to get stuff working soonish after Christmas
  - spend lots of time checking variables, performance
  - re-read, re-read, check, etc.
- think about where to cutoff between each model
  - can I derive it from the model etc?
- ** TODO ** chessboard decomposition method - halve the number of grid points!
- ** TODO ** check validity of gaussian pulses as input!

Viewer

Done

- show config info - source and receiver positions
- show mesh node positions
- show mesh node pressure, as the simulation progresses
- fix node positions so that the entire mesh is covered!
- show rays
  - with volume, as the simulation progresses
  - fix image-source in new raytracer formulation
- scroll to zoom?
- raytracer octree implementation
- test raytracer, try to find why disparity between implementations ocurrs
- streamline the boundary node code, make sure it finds corner (three-or-more boundaries) cases
- find some way of fixing the inside/outside bug at least for the echo tunnel example

- which papers cite the boundary modelling paper?
- can I design my filters the other way up?
  - no
- CRITICAL: check how boundary filter coefficients should be calculated
- fix bug in boundary finder code
- MOST CRITICAL: prove that supplying different coefficients to boundaries has the *correct* effect on the resulting impulse responses
  - this seems to be somewhat working now

TODO

- write up the boundary modelling stuff
- add 0.196 stability limit to waveguide config somehow
- check that loading surfaces from file works as expected
- try to find cause of insane memory usage
- CRITICAL: check mic modelling write-up now that the mesh update equations have been fixed, boundaries added
- boundary modelling
  - try to predict the frequency response of each boundary
  - is there a way of mapping between the raytracer and this boundary model?
- fast-mode tetrahedral mesh and a slow-mode, more accurate, rectangular mesh
- think about comparing the two models
- nicer json parsing

[* how should I model perfectly reflective walls? * reflection filters with unity gain turn into impedance filters with infinite gain

- search for arbitrary filter response IIR design (??)]

digital impedance filters

- measure wall reflectance at a certain angle of incidence
  - e.g. normal to the wall
- average several readings per octave band
  - giving a parameter *alpha* per octave band
- convert to reflection coefficients per octave band
  - |R| = sqrt(1 - alpha)
- use these coefficients to create a high-order normal-incidence digital reflectance filter R0(z)
- ensure R0(z) represents a passive boundary
  - |R0(z)| <= 1
  - a complex function of a complex variable is positive real if
    - z real ==> f(z) real
    - |z| >= 1 ==> re{f(z)} >= 0
- convert the reflectance filter to an impedance filter
  - Ew(z) = (1 + R0(z)) / (1 - R0(z))
  - oh look, another filter

testing scratchpad

```
                    I THINK I FIXED THIS BIT
                       (BUT I'M NOT SURE)
             ~ a trilogy in eleventyzillion parts ~
```

filters appear to not be used - try looking at filter inputs
filter inputs sometimes go to NaN
reading bad memory somewhere, not sure how to find where
try viewing preprocessor output?
filter memory goes to nan filter step is the only function which interracts with filter memory filter inputs: float input filter memory array filter coefficients array
filter memory goes NaN on same step as filter input!!!
first 1d filter memory belongs to node 84053
something strange about 83847?
get_filter_weighting
it looks like something is changing the filter memory to nan, but that something is definitely not the filter input I might be looking at the wrong memory something else might be affecting the values although I can't see this anywhere

---

```
    IGNORE THE HEADING UP THERE
```

(I MISSED SOME STUFF AND SOMEHOW IT'S WORSE NOW) ~ in somehow more parts than the prequel ~

2D boundaries seem to be unstable/blow up over time although 1D ones seem fineish

boundary nodes

```
        nodes: [int boundary, int boundary, int boundary]
                |             |             |
                |             |             +----------------------------------------------+
                |             |                                                            |
                +---------+   +---------------+                                            |
                          |                   |                                            |
        boundary data 1: [[int coefficients], [int coefficients], [int coefficients]]     |
                           |                                                               |
+--------------------------+                                     +-------------------------+
|                                                                |
|    boundary data 2: [[int coefficients, int coefficients], [int coefficients, int coefficients]]
|                                                                 |                 |
|+----------------------------------------------------------------+                 |
||+---------------------------------------------------------------------------------+
|||
|||     boundary data 3: [[int coefficients, int coefficients, int coefficients]]
|||
||+-----------------------------+
|+---------------------------+  |
+-------------------------+  |  |
                          |  |  |
        coefficients: [a, b, c, d]
```

talking points

what's the best way of generating boundary filters?

Campos, G. R., & Howard, D. M. (2005). On the computational efficiency of different waveguide mesh topologies for room acoustic simulation. *Ieee Transactions on Speech and Audio Processing*, *13*(5).

Duyne, S. A. V., & Smith, J. O. (1995, October). The tetrahedral digital waveguide mesh. Proceedings of the IEEE Workshop on Application of Signal Processing to Audio and Acoustics.

Duyne, S. A. V., & Smith, J. O. (1996). The 3D tetrahedral digital waveguide mesh with musical applications. Proceedings of the International Computer Music Conference.

Hacıhabiboglu, H., Günel, B., & Cvetkovic, Z. (2010). Simulation of directional microphones in digital waveguide mesh-based models of room acoustics. *IEEE Transactions on Audio, Speech, and Language Processing*, *18*(2).

Southern, A., & Murphy, D. (2007, October). Methods for 2nd order spherical harmonic spatial encoding in digital waveguide mesh virtual acoustic simulations. Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.

Southern, A., Siltanen, S., & Savioja, L. (2011, May). Spatial room impulse responses with a hybrid modelling method. presented at the 130th convention of the Audio Engineering Society.