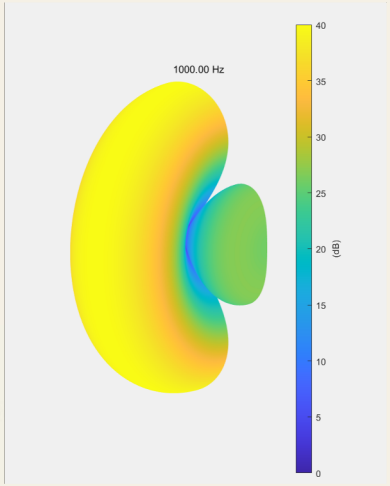
UC San Diego

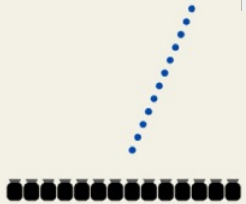
Adaptive Beamforming Algorithm

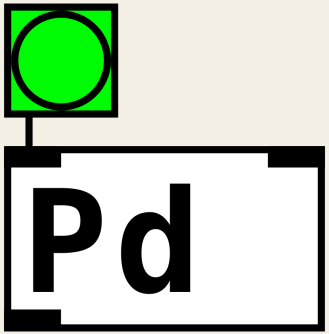
for 14-channels speakers

By Xiaoxuan(Andrina) Zhang, Eduard Shkulipa, Helen Lin, Gabriel Diaz

Beamforming Team | September 15, 2023







Overview

Started With:

- MATLAB
- Max
- Fixed Beamforming

Goal:

- C++
- Pure Data
- Real Time Adaptive Beamforming

Beamforming Class

Class setup

- Created a class that encloses beamforming algorithm.
- Class consists of 3 major components - constructor, assignment of parameters of the system, filter calculator
- By using Eigen library, we could streamline translation from Matlab

```
public:
    // constructor
    Beamforming(int nSpeakers=14, int nCtrlPts=1);
    // destructor
    ~Beamforming();


    bool control_point_geometry(); //set up control point geometry  TODO: maybe remove
    MatrixComplex solve_filters(); //main entry point for solving filters


    // Getters
    vector<t_float> get_pf();
    t_float get_fs();
    t_float get_c();
    unsigned int get_nfft();
    unsigned int get_nf();
    VectorReal get_ff();
    t_float get_beta();
    int get_n();
    t_float get_dx();
    bool get_flatten_filters();
    bool get_normalize_filters();
    bool get_crop_filters();
```

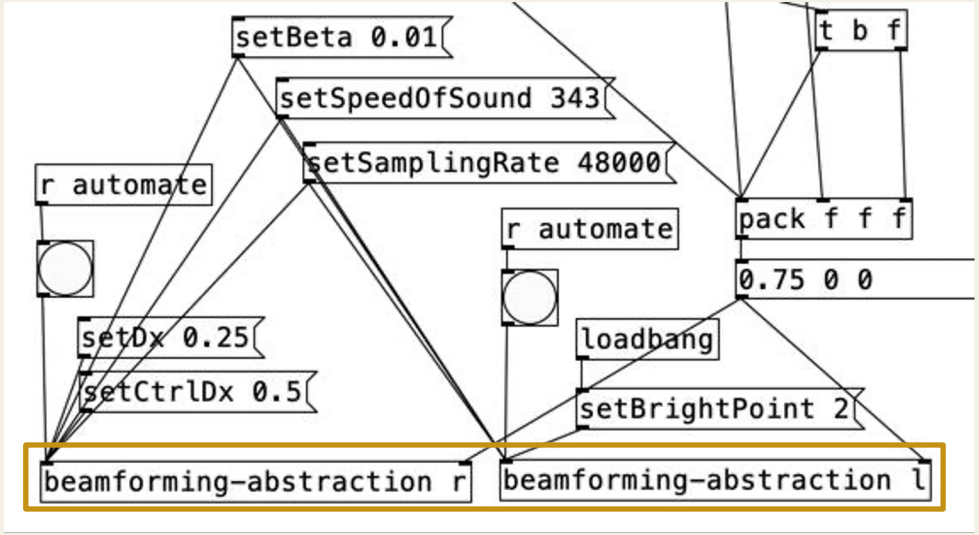
Beamforming Class

```
public:
    // constructor
    Beamforming(int nSpeakers=14, int nCtrlPts=1);
    // destructor
    ~Beamforming();


    bool control_point_geometry(); //set up control point geometry. TODO: maybe remove
    MatrixComplex solve_filters(); //


    // Getters
    vector<t_float> get_pf();
    t_float get_fs();
    t_float get_c();
    unsigned int get_nfft();
    unsigned int get_nf();
    VectorReal get_ff();
    t_float get_beta();
    int get_n();
    t_float get_dx();
    bool get_flatten_filters();
    bool get_normalize_filters()
    bool get_crop_filters();
```

```
42  MatrixComplex Beamforming::solve_filters() {
43      configControlPoints();
44      return a.reshaped<Eigen::RowMajor>().eval();
45  }
46
47  // constructor
48  Beamforming::Beamforming(int nSpeakers, int nCtrlPts) {
49      // Plot type Single or Octave: Which Frequencies to plot for
50      pf = {1000}; //Single
51      // pf = {31.5, 63, 125, 250, 500, 1000, 2000, 4000, 8000, 16000}; //Octave
52
53      // Sampling rate
54      fs = 48000;
55      // speed of sound
56      c = 343;
57      // fft size (samples)
58      nfft = 512;
59      // number of frequencies
60      nf = nfft/2+1; //Not used else except below
61      // frequency vector
62      ff = linspace(0, fs/2, nf);
63      // Regularization Parameter
64      beta = 0.01;              //Limit: 0 to infinity.
65
66      // Speaker setup
67      n = nSpeakers; //num speakers
68      dx = 0.03874; //speaker_spacing
```

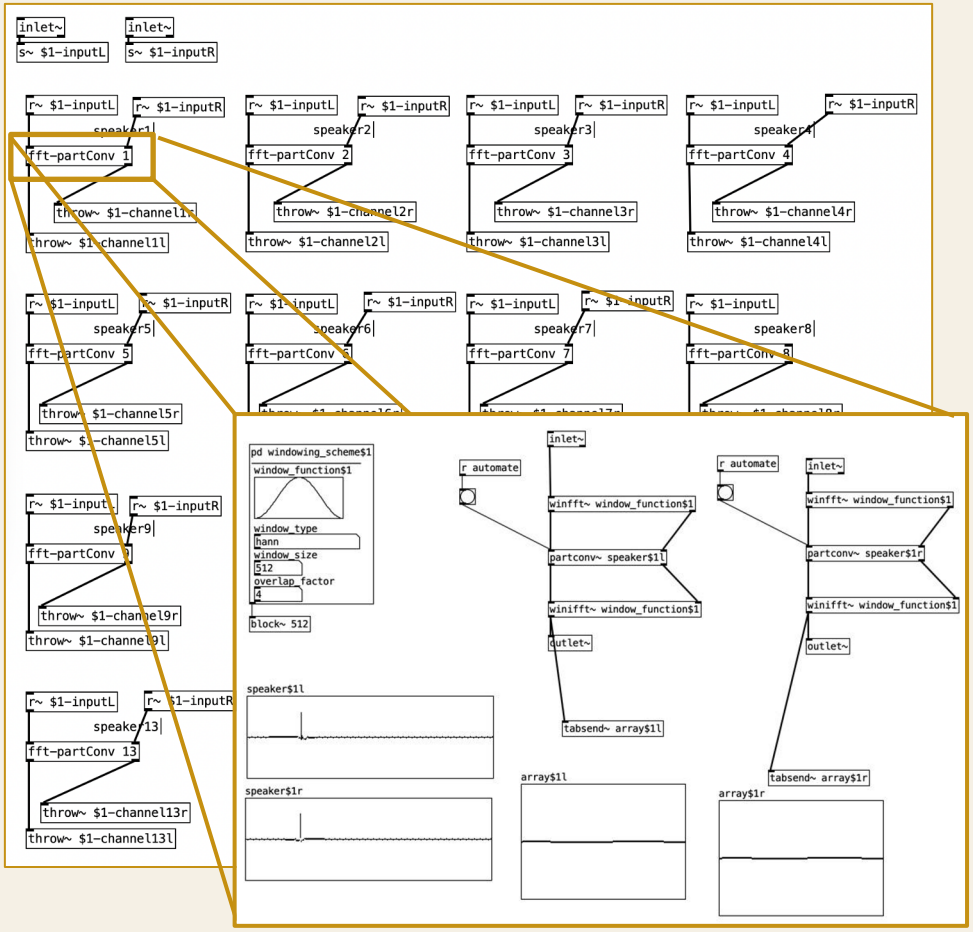
- Created a class that encloses beamforming algorithm.
- By using Eigen library, we could streamline translation from MATLAB
- Class consists of 3 major components
    - Constructor with default parameters
    - Update of parameters
    - Filter calculator

Beamforming External



- Pd external in C++ that interfaces the Beamforming class
- Pdlibbuilder
- Customization and parameter setting
- Positional data is passed in
- Filters are outputted for use in convolution

Real-time convolution

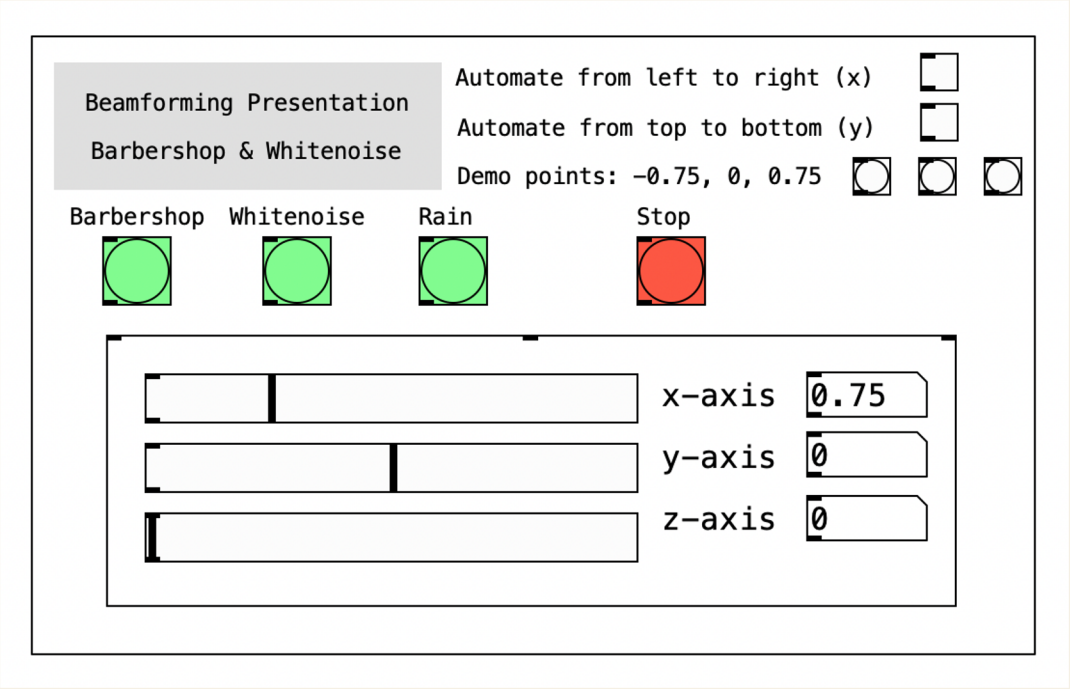


partconv~ object and Pd patches

- Pd Spectral Library and modification
- Implementation of 14-channel streaming in Pure Data
- Pure Data Abstraction – ready for any number of speakers

```
299 static void partconv_analyze_impulse( t_partconv* object )
360     // store parts and spectra_mem_size for use in dsp loop
361     object->parts            = parts;
362     object->spectra_mem_size = spectra_mem_size;
363
364     // report analysis info
365     // post( "partconv~: analyzed %s array", object->impulse_name->s_name );
366
367     object->analyzed_flag = TRUE;
368 }
```

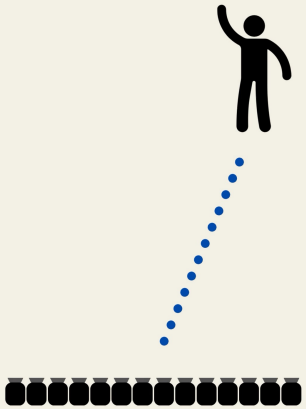

UC San Diego



- Graphical User Interface (GUI)
- Provides Barbershop binaural sample audio and white noise for a diverse experience with adaptive beamforming
- Beam automation allowing users to feel and hear the beam moving

Final Product

UC San Diego



Real Time Adaptive Beamforming

Visualization