Code Examples from Stereo Position Simulator

Written by Ross Penniman

SPSB_pcg.m (Matlab)

```
function panCurves=SPSB_pcg(levelDif, timeDif);
% Supporting function for SPS_Basic
% takes in values levelDif and timeDif and outputs
% set of pan curves with the appropriate gain and time
% values (in seconds).
%
% levelDif is in dB and timeDif is in ms

t=linspace(0, 1, 201);

% Starting point for curves
l=-levelDif*t;
r=levelDif*t - levelDif;

% Convert to gain
gainL=10.^(l./20);
gainR=10.^(r./20);

% Convert to power
powL=gainL.^2;
powR=gainR.^2;
pSum=powL+powR;
pSumDB=10*log10(pSum);

% Apply correction
l=l-pSumDB;
r=r-pSumDB;

% Convert back to gain
gainL=10.^(l./20);
gainR=10.^(r./20);

% Convert time to seconds
timeDif=timeDif/1000;

% Calculate sinusoidal time delay curves
timeL=(timeDif/2)*-cos(pi*t)+(timeDif/2);
timeR=(timeDif/2)*cos(pi*t)+(timeDif/2);

% Compile gain and time values in pan curves
panCurves=[gainL; gainR; timeL; timeR];
```

panCurveGen (C++, function from SPS_Orch.cpp)

```
void SPS_Orch::panCurveGen(float width, float distance, float pattern, float angle)
{
	// panCurveGen
	// Takes in parameters width, distance, pattern, and angle and calculates
	// pan curves based on the defined microphone pair
	//
	// width: width between microphones in meters (not to exceed 1)
	// distance: distance between source and center of microphone pair in meters
	//		(not to be less than 1)
	// pattern: polar pattern of microphone
	//		1: Omnidirectional
	//		2: Wide-cardioid
	//		3: Cardioid
	//		4: Hyper-cardioid
	//		5: Figure of 8

	float speedSound, theta, t_Inc, m, n;
	float distL, distR, incAngL, incAngR;
	float gainMax, timeMin, temp;
	long i=0; // Vector index	(from 0	to 200)
	distL=distR=incAngL=incAngR=0;
	gainMax=timeMin=temp=0;

	// Source angle vector
	theta=PI;
	t_Inc=PI/200;

	// Convert input parameters
	width=width/2;
	angle=angle*PI/180;	// convert to radians

	// Speed of sound (meters/second)
	speedSound=344.3; // At about 70	F
	// THIS VALUE WILL AFFECT MAX DELAY TIME
	// BE CAUTIOUS!!!

	while(i < 201)
	{
		// Calculation of distance between source and microphones
		m=(distance*distance) + (width*width);
		n=2*distance*width;
		distL=sqrt(m+(n*cos(theta)));
		distR=sqrt(m-(n*cos(theta)));

		// Triangulation of source and microphone positions
		temp=(-(width*width) + (distL*distL) +
			(distance*distance))/(2*distL*distance);
		if(temp > 1) temp=1;
		else if(temp < -1) temp=-1;
		incAngL=acos(temp);

		temp=(-(width*width) + (distR*distR) +
			(distance*distance))/(2*distR*distance);
		if(temp > 1) temp=1;
		else if(temp < -1) temp=-1;
		incAngR=acos(temp);

		incAngL=theta-incAngL-angle-(PI/2);
		incAngR=(PI/2)-theta-incAngR-angle;
```

```
            // Apply incident angle to mic polar pattern
            switch ((long)pattern)
            {
                  case 1:
                        panCurves[0][i]=1;
                        panCurves[1][i]=1;
                        break;
                  case 2:
                        panCurves[0][i]=0.7+0.3*cos(incAngL);
                        panCurves[1][i]=0.7+0.3*cos(incAngR);
                        break;
                  case 3:
                        panCurves[0][i]=0.5+0.5*cos(incAngL);
                        panCurves[1][i]=0.5+0.5*cos(incAngR);
                        break;
                  case 4:
                        panCurves[0][i]=0.3+0.7*cos(incAngL);
                        panCurves[1][i]=0.3+0.7*cos(incAngR);
                        break;
                  case 5:
                        panCurves[0][i]=cos(incAngL);
                        panCurves[1][i]=cos(incAngR);
                        break;
            }

            // Account for inverse square law
            panCurves[0][i]*=(distance/distL);
            panCurves[1][i]*=(distance/distR);

            // Initital time  calculation
            panCurves[2][i]=distL-distance;
            panCurves[3][i]=distR-distance;

            // Update theta and i
            theta-=t_Inc;
            i++;
      }

      // Find gainMax and timeMin values
      for   (i=0; i<201; i++)
      {
            if(panCurves[0][i] > gainMax) gainMax=panCurves[0][i];
            if(panCurves[2][i] < timeMin) timeMin=panCurves[2][i];
      }

      timeMin=fabs(timeMin);

      // Apply    gain and time normalization
      for   (i=0; i<201; i++)
      {
            panCurves[0][i]=panCurves[0][i]/gainMax;
            panCurves[1][i]=panCurves[1][i]/gainMax;
            panCurves[2][i]=(panCurves[2][i]+timeMin)/speedSound;
            panCurves[3][i]=(panCurves[3][i]+timeMin)/speedSound;
      }

      return;
}
```