**Supplement for "Acoustic Micromachining of Three-Dimensional Surfaces for Biological Applications"**

The MATLAB LP simulator code (below) will generate 'patches' in 3D space that represent the walls of a groove produced by the record cutter. The simulation is capable of outputting infinitely sharp curves which are not possible to cut due to physical limitations from the finite size of the cutting stylus. We have chosen to simulate these limitations by employing a simple Butterworth low-pass filter with a 15kHz cut-off based on the reported frequency response of the cutting stylus.

Figure 2B (in the manuscript) presents a 3D view of the LP simulation with a 500Hz square wave input sampled at 96kHz. A digital Butterworth low-pass filter was employed to mimic the physical limitations of the cutting stylus. A 12" 33 1/3 rpm record disc with an inter-groove distance of 150 microns and unmodulated groove depth of 50 microns were used for the simulation.

Note that most standard recording studios will apply the Recording Industry Association of America (RIAA) equalization standard to the input signal, emphasizing the higher frequencies while suppressing lower frequencies. If desired, the RIAA equalization can be turned off during the cutting process.

**`LPsim.m`**

```
function [vertex, faces, vertex_color] = LPsim(t, sig_left, sig_right,
track_skip, groove_depth, Rmax, rpm)
% LPsim  Vinyl LP groove track simulator
% 2004-09-01      hb
%   [vertex, faces] = LPsim(t, signal) outputs the vertices and faces of
%   a virtual cutting tip's motion given a monoaural signal (signal)
%   sampled in time (t). The sampling rate for the time vector (t) must
%   be constant, i.e. there can be no skipped data points, otherwise the
%   faces outputted will be erroneously connected when using the 'patch'
%   command for visualization (see below).
%
%   [vertex, faces] = LPsim(t, sig_left, sig_right)
%
%   Returns 'patch' style data for the etched groove for times given in
%   vector 't', a signal in 'sig_left' and 'sig_right' for dual-channel
%   stereo input (respectively).
%
%   [vertex, faces] = LPsim(t, sig_left, sig_right, track_skip)
%
%   Same as above except the distance between successive grooves without
%   any lateral modulation is given as 'track_skip' in unit distance
%   (default is 150 microns for a 12" disc).
%
%   [vertex, faces] = LPsim(t, sig_left, sig_right, track_skip,
%                           groove_depth)
%
%   Same as above with the unmodulated groove depth specified in unit
%   distances. Default is 50 microns.
%
%   [vertex, faces] = LPsim(t, sig_left, sig_right, track_skip,
```

```
%                           groove_depth, Rmax)
%
%   The radius at time 0 ('Rmax') is assumed to be the radius of the
%   cutting stylus for a standard record going from outside to in. However,
%   a negative 'track_skip' value can simulate a backwards record that
%   moves away from the center of rotation. The default value is 0.152e6
%   microns, representing a 12" disc (6" radius).
%
%   [vertex, faces]= LPsim(t, sig_left, sig_right, track_skip,
%                          groove_depth, Rmax, rpm)
%
%   The rotational speed of the record in revolutions per minute (RPM) is
%   specified by 'rpm'. A standard 12" record was recorded for playback
%   at 33 1/3 rpm.
%
%   [vertex, faces, vertex_colors]=LPsim(...)
%
%   Also returns a depth-encoded color (vertex_colors) for the grooves.
%
%   To visualize the output, use:
%   patch('Vertices', vertex, 'Faces', faces, 'FaceVertexCData',
%   vertex_color, 'FaceColor', 'interp', 'EdgeColor', 'none');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check for default arguments
if(nargin<7)
    rpm=33.3333;            % 33 1/3 rpm industry standard
end

if(nargin<6)
    Rmax=0.1524e6;          % Maximum radius (at time t=0), in microns
(6"=0.1524m)
end

if(nargin<5)
    groove_depth=50;        % Groove depth (in microns)
end

if(nargin<4)
    track_skip=150;         % Distance between tracks (in microns)
end

if(nargin<3)
    sig_right=sig_left;     % Assume monoaural input
end

RPS=rpm/60;                 % Rotational speed of record (revolutions per
second)

% Some pre-computed constants
cos45=sqrt(2)/2;
sin45=sqrt(2)/2;

% Pre-allocate memory for large vectors (speeds performance)
nVerts=length(t)*3;
vertex=zeros(nVerts, 3);
vertex_color=zeros(nVerts, 1);
```

```
faces=zeros((length(t)-1)*2, 4);

% Construct verticies

% We are using the following conventions:
%   x: lateral position, from center of rotation
%   y: position on the lacquer, from center of rotation
%   z: depth, z=0 at the surface
%
% The record cutter revolves around on the (x-y) plane.
%
% Compute neutral stylus position
x0=(Rmax-RPS*t*track_skip).*cos(RPS*t*2*pi);
y=(Rmax-RPS*t*track_skip).*sin(RPS*t*2*pi);
% Mix in the signal
x=cos45*(sig_left+sig_right)+x0;
z=sin45*(sig_left-sig_right)-groove_depth;
% Transpose to place them in row order
x=x';
y=y';
z=z';
% Construct 3 vertices: one at the upper left corner of the V-groove,
% another in the bottom apex of the V-groove, and lastly, one for the
% upper right corner. Both upper corners have a height (z) of '0' by
% definition (where z=0 at the surface of the lacquer).
vertex(1:3:end-2, :)=[x-z, y, zeros(length(z), 1)];
vertex(2:3:end-1, :)=[x, y, z];
vertex(3:3:end, :)=[x+z, y, zeros(length(z), 1)];
% Color-code groove based on depth (leave all other vertices at '0'
% since they are set to be at the surface (hence the zeros(length(z))...)
vertex_color(2:3:end-1)=z;

% Connect verticies to form faces
% Each time point generates a triangle, and every 2 triangles generates a
face
% Therefore, given 'n' time points, you will have (n-1)*2 faces
% It turns out that since the verticies are numbered in order, for a 3
% triangle setup you will have (where the verticies are numbered from left
% to right, front to back)
% [1 2 5 4]
% [2 3 6 5]
% [4 5 8 7]
% [5 6 9 8]
% [ . . . ] and so on

% Compute the number of faces
nFaces=(length(t)-1)*2;

% Determine the index for the first vertex of next-to-last triangle
ntl_vertex=(length(t)-1)*3-1;

% Vectorized output
faces(1:2:nFaces, :)=[1:3:ntl_vertex; 2:3:ntl_vertex+1; 5:3:ntl_vertex+4;
4:3:ntl_vertex+3]';
faces(2:2:nFaces, :)=faces(1:2:nFaces-1, :)+1;
```