Green's Function Method for Creating Accurate Stereo Sound Images: Simple Two-Dimensional Example Rectangular Membrane

David R. Clark, EXE Consulting
P.O. Box 450998 Garland, Texas 75045-0998
11 August 2004, 07 November 2004

Herein is presented an example which shows the Green's function method described in [1] applied to a simple, two-dimensional, pedagogical example. This example could be interpreted as the striking of a rectangular area of a rectangularly shaped membrane, for example the drumhead of a rectangular-shaped drum, and picking up the response at a point location elsewhere on the membrane.

We start from the solution for the three-dimensional (3-D) wave equation derived in [1] and follow the same format as for the one-dimensional example [3] except that we will also adopt the volumetric source as in [4]. The 3-D solution for a volumetric source is:

$$\psi(\mathbf{r},t) = c^2 \frac{8u}{nml} \left(\frac{1}{\pi^3}\right) \left(\frac{1}{x_2 - x_1}\right) \left(\frac{1}{y_2 - y_1}\right) \left(\frac{1}{z_2 - z_1}\right)$$

$$\sum_{nml}{}' \left[\cos\left(\frac{n\pi x_1}{a}\right) - \cos\left(\frac{n\pi x_2}{a}\right)\right] \left[\cos\left(\frac{m\pi y_1}{b}\right) - \cos\left(\frac{m\pi y_2}{b}\right)\right] \left[\cos\left(\frac{l\pi z_1}{c}\right) - \cos\left(\frac{l\pi z_2}{c}\right)\right]$$

$$\left[\sin\left(\frac{n\pi x}{a}\right) \sin\left(\frac{m\pi y}{b}\right) \sin\left(\frac{l\pi z}{c}\right)\right] \int_{t'=-\infty}^{t} s(t') \frac{\sin(\omega_{nml}\tau)}{\omega_{nml}} dt'$$

Modifying it for the case of 2-D:

$$\psi(\mathbf{r},t) = c^2 \frac{4u}{nm} \left(\frac{1}{\pi^2}\right) \left(\frac{1}{x_2 - x_1}\right) \left(\frac{1}{y_2 - y_1}\right)$$

$$\sum_{nm}{}' \left[\cos\left(\frac{n\pi x_1}{a}\right) - \cos\left(\frac{n\pi x_2}{a}\right)\right] \left[\cos\left(\frac{m\pi y_1}{b}\right) - \cos\left(\frac{m\pi y_2}{b}\right)\right]$$

$$\left[\sin\left(\frac{n\pi x}{a}\right) \sin\left(\frac{m\pi y}{b}\right)\right] \int_{t'=-\infty}^{t} s(t') \frac{\sin(\omega_{nm}\tau)}{\omega_{nm}} dt'$$

For a fixed reception point at (x,y), an impulse response function can be precomputed for a source impulse signal $\delta(t')$.

Let

$$A_{nm} = c^2 \frac{4u}{nm} \left(\frac{1}{\pi^2}\right) \left(\frac{1}{x_2 - x_1}\right) \left(\frac{1}{y_2 - y_1}\right)$$

$$\left[\cos\left(\frac{n\pi x_1}{a}\right) - \cos\left(\frac{n\pi x_2}{a}\right)\right] \left[\cos\left(\frac{m\pi y_1}{b}\right) - \cos\left(\frac{m\pi y_2}{b}\right)\right] \sin\left(\frac{n\pi x}{a}\right) \sin\left(\frac{m\pi y}{b}\right)$$

which can be computed for all n,m if the input data are known, and let

$$f(t) = \int_{t'=-\infty}^{t} \sum_{nm}{}' \left[A_{nm} \frac{\sin(\omega_{nm}\tau)}{\omega_{nm}} \right] \delta(t')\, dt'$$

$$= \sum_{nm}{}' \frac{A_{nm}}{\omega_{nm}} \sin(\omega_{nm} t)$$

Now we have the impulse response function for this problem, namely the discrete sin transform of the quantity $\frac{A_{nm}}{\omega_{nm}}$. In any real computation, the number of frequencies is finite and f(t) is periodic.

As described in [1], but adapted to this 2-D example, to compute the resulting sound received at point (x,y) over time from a volumetric (*i.e.* area) source extending from x_1 to x_2 and y_1 to y_2, one could compute the convolution:

$$f(t) * s(t) \equiv \int_{-\infty}^{\infty} f(t')\, s(t - t')\, dt' = \int_{-\infty}^{\infty} f(t - t')\, s(t')\, dt' = \int_{-\infty}^{\infty} f(\tau)\, s(t')\, dt' = \psi(x,y;t)$$

In order to compute $\frac{A_{nm}}{\omega_{nm}}$ it is first necessary to choose a cutoff frequency so that n and m are finite.

$\omega_{nm} = \sqrt{\left(\frac{n\pi c}{a}\right)^2 + \left(\frac{m\pi c}{b}\right)^2}$ resulting in $f_{nm} = \frac{\omega_{nm}}{2\pi} = \frac{c}{2} \sqrt{\left(\frac{n}{a}\right)^2 + \left(\frac{m}{b}\right)^2}$.

The fundamental frequency is given by:

$$f_{11} = \frac{c}{2} \sqrt{\left(\frac{1}{a}\right)^2 + \left(\frac{1}{b}\right)^2}$$

Letting $R = \frac{b}{a}$:

$$f_{11} = \frac{c\sqrt{R^2 + 1}}{2aR}$$

But we also have:

$$f_{n1} = \frac{c\sqrt{n^2R^2+1}}{2aR} \quad \text{and} \quad f_{1m} = \frac{c\sqrt{R^2+m^2}}{2aR}$$

Choosing $f_{11} = 110Hz$, the maximum value of n can be specified by considering:

$$\frac{f_{n1}}{f_{11}} = \frac{\sqrt{n^2R^2+1}}{\sqrt{R^2+1}} = \frac{f_{max}}{110}$$

Solving for n:

$$n_{max} = \left(\frac{1}{R}\right)\sqrt{\left(\frac{f_{max}}{110}\right)^2\left(R^2+1\right) - 1}$$

For any given m:

$$n_{max} = \left(\frac{1}{R}\right)\sqrt{\left(\frac{f_{max}}{110}\right)^2\left(R^2+1\right) - m^2}$$

Solving the corresponding expression for m_{max} gives:

$$m_{max} = \sqrt{\left(\frac{f_{max}}{110}\right)^2\left(R^2+1\right) - R^2}$$

and for any particular n:

$$m_{max} = \sqrt{\left(\frac{f_{max}}{110}\right)^2\left(R^2+1\right) - n^2R^2}$$

The coefficients are now:

$$\frac{A_{nm}}{\omega_{nm}} = \frac{4uc}{nm}\left(\frac{1}{\pi^3}\right)\left(\frac{1}{\sqrt{\frac{n^2}{a^2}+\frac{m^2}{b^2}}}\right)\left(\frac{1}{x_2-x_1}\right)\left(\frac{1}{y_2-y_1}\right)$$

$$\left[\cos\left(\frac{n\pi x_1}{a}\right)-\cos\left(\frac{n\pi x_2}{a}\right)\right]\left[\cos\left(\frac{m\pi y_1}{b}\right)-\cos\left(\frac{m\pi y_2}{b}\right)\right]\sin\left(\frac{n\pi x}{a}\right)\sin\left(\frac{m\pi y}{b}\right)$$

Normalizing this expression for n = m = 1, and once again utilizing R, the ratio of b to a:

$$\frac{A_{nm}}{\omega_{nm}} = \left(\frac{\sqrt{R^2+1}}{nm\sqrt{R^2n^2+m^2}}\right)$$

$$\left[\cos\left(\frac{n\pi x_1}{a}\right)-\cos\left(\frac{n\pi x_2}{a}\right)\right]\left[\cos\left(\frac{m\pi y_1}{b}\right)-\cos\left(\frac{m\pi y_2}{b}\right)\right]\sin\left(\frac{n\pi x}{a}\right)\sin\left(\frac{m\pi y}{b}\right)$$

If we now specify the area determined by the normalized dimensions $\left(\frac{x_1}{a}\right)$, $\left(\frac{x_2}{a}\right)$, $\left(\frac{y_1}{b}\right)$, and $\left(\frac{y_2}{b}\right)$ and if also we specify the normalized pickup locations $\left(\frac{x}{a}\right)$ and $\left(\frac{y}{b}\right)$ and the ratio R of b to a, we can determine the coefficients $\frac{A_{nm}}{\omega_{nm}}$ and, by means of the sin transform, solve the 2-D wave equation for a rectangular membrane. The program listed in the Appendix accomplishes this.

References and Notes

[1] David R. Clark. Green's function method for creating accurate stereo sound images. EXE Consulting, P.O. Box 450998, Garland, Texas 75045-0998, July 2004.

[2] Gabriel Barton. *Elements of Green's Functions and Propagation: Potentials, Diffusion, and Waves.* Oxford, 1989. Reprinted 1991.

[3] David R. Clark. Green's function method for creating accurate stereo sound images: Simple one-dimensional example. EXE Consulting, P.O. Box 450998, Garland, Texas 75045-0998, July-August 2004.

[4] David R. Clark. Green's function method for creating accurate stereo sound images: Volumetric Sources. EXE Consulting, P.O. Box 450998, Garland, Texas 75045-0998, July 2004.

© Copyright EXE Consulting 2004. All rights reserved.

Appendix

A simple C++ demonstration program follows. Output is to standard out. The sound of this particular example resembles timpani.

```
// Two-dimensional pedagogical example for demonstrating application of
//    Green's function formulation.  No windowing or filtering is performed.
//    No FFT's.  (Exercise for intrepid readers.)
//    Rectangular membrane --- Volumetric

// Similar to hammering drumhead at one position, pickup at another

// Maximum frequency for spectra is one-half of sample rate
//    (e.g. 44.1 ksps -> 22.050 kHz)

// Dirac delta-function source applied at areal segment xpp1 to xpp2
// Pickup at x = (x / length)
// Response convolved with half-sinusoid waveform to model hammering

// Compile:
//    $ g++ rect_membrane.cc -o rect_membrane

// WARNING: Be sure to REDIRECT OUTPUT:
//    $ rect_membrane > rect_membrane.out
// Use sox to convert text output to WAV file:
//    $ sox -t .dat rect_membrane.out rect_membrane.wav

// Be sure to try out different impulse and pickup positions because
//    these significantly change the timbre, just as with a real drum.
//    When creating a drum, try to use slightly different impulse and
```

```
//    pickup locations for each note to create more a more realistic sound.

// After generating WAV files, I recommend SPECIMEN for uploading into
//    a soundbank, then you should be able to play your instrument with
//    keyboard or MIDI file.  Although pitch-shifting in SPECIMEN will
//    work, I recommend creating individual notes or small groups for
//    more realistic and interesting sounds, varying the hammer and
//    pickup locations slightly.

// REDUCE amplitudes for full instruments.  Suggested values are HIGH.

// Placed in public domain.  No warranties expressed or implied.
//    By compiling this program, you agree to assume all risks associated
//    with this program and agree to indemnify author for all claims arising
//    from your use or misuse of this program, including derivatives, or
//    arising from your use or misuse of information contained herein.

#include <iomanip>
#include <iostream>
#include <cmath>

int main(int argc, char * argv[])   // One big, bad main...
{

// Get inputs:

   int i_sample_rate = 44100;
   std::cerr << "Sample rate (integer, suggest 48000 or 44100): ";
   std::cin >> i_sample_rate;

   double freq = 110.0;
   std::cerr << "Frequency (Hz, double, suggest ~110 for A): ";
   std::cin >> freq;
   double freq_max = double(i_sample_rate)/2.0;

   double R=3.141592653589793;
   std::cerr << "Ratio of dimensions b to a (double, positive, suggest irrational "
      << std::endl <<  "   such as pi = 3.141592653589793 : ";
   std::cin >> R;

   int N = (int)(rint((1.0/R) * sqrt(
      (freq_max / freq)*(freq_max / freq) * (R*R + 1.0) - 1.0
   )));   // close enough...
   int M = (int)(sqrt(
      (freq_max / freq)*(freq_max / freq) * (R*R + 1.0) - R*R
   )));   // close enough...
   int Nmax = 0, Mmax = 0;

   int T = 1;
   std::cerr << "Length (seconds, integer, suggest 1): ";
   std::cin >> T;
   T = (int)(rint(2 * T * freq_max));


   double x = 0.2;
   std::cerr << "Pickup position (double, 0.0 < x < 1.0, suggest 0.1 to 0.2): ";
   std::cin >> x;
   double xpp1 = 0.20;
   std::cerr
      << "Impulse left position (double, 0.0 < xpp1 < 1.0, suggest 0.20): ";
   std::cin >> xpp1;
   double xpp2 = 0.25;
   std::cerr
      << "Impulse right position (double, 0.0 < xpp2 < 1.0, suggest 0.25): ";
   std::cin >> xpp2;
   if (fabs(xpp1 - xpp2) < 1e-12)   {
      std::cerr << "ERROR: xpp1 and xpp2 must be at least 1e-12 apart!"
                << std::endl;
      exit(0);
```

```
   }

   double y = 0.2;
   std::cerr << "Pickup position (double, 0.0 < y < 1.0, suggest 0.1 to 0.2): ";
   std::cin >> y;
   double ypp1 = 0.20;
   std::cerr
      << "Impulse left position (double, 0.0 < ypp1 < 1.0, suggest 0.20): ";
   std::cin >> ypp1;
   double ypp2 = 0.25;
   std::cerr
      << "Impulse right position (double, 0.0 < ypp2 < 1.0, suggest 0.25): ";
   std::cin >> ypp2;
   if (fabs(ypp1 - ypp2) < 1e-12)   {
      std::cerr << "ERROR: ypp1 and ypp2 must be at least 1e-12 apart!"
                << std::endl;
      exit(0);
   }

   double Amplitude = 4.5;
   std::cerr << "Amplitude (double, suggest 90.0): ";
   std::cin >> Amplitude;
   Amplitude = Amplitude * freq;

   int resolution = 16;   // Accuracy is actually double-precision, normalized
   double normalization = pow(2.0, double(resolution));

   double pi = 3.141592653589793;
   double * * coeff = new double*[N+1];
   for (int i=0; i<N+1; i++)   {
      coeff[i] = new double[M+1];
   }

   double * f = new double[T+1];


// Sinusoid parameters:

   double sin_freq = 120.0;
   const double sin_amplitude = 1.0;
   double sample_rate = double(i_sample_rate);
   int sin_N = int(rint(sample_rate / sin_freq));
   double * sinusoid = new double[T+1];
   double * convolved = new double[T+1];


// A_n_m/omega_n_m:

   double cos_sin_n = 0.0;
   for (int n=1; n<=N; n++)   {
      Mmax = (int)(rint(sqrt(
         (freq_max / freq)*(freq_max / freq) * (R*R + 1.0) - double(n*n)*(R*R)
      )));
      cos_sin_n = sqrt(R*R + 1.0) *
         (cos(double(n) * pi * xpp1) - cos(double(n) * pi * xpp2)) *
        sin(double(n) * pi * x);
      for (int m=1; m<=M; m++)   {
         Nmax = (int)(rint((1.0/R) * sqrt(
            (freq_max / freq)*(freq_max / freq) * (R*R + 1.0) - double(m*m)
         )));
         if (n < Nmax && m < Mmax)   {
            coeff[n][m] = (cos_sin_n/(double(n*m) * sqrt(R*R * double(n*n) + double(m*m)))) *
               (cos(double(m) * pi * ypp1) - cos(double(m) * pi * ypp2)) *
               sin(double(m) * pi * y);
         }
      }
   }

// f(t) with decay:
```

```
      double alpha = log(1e-3) / T;   // t60 for half-period (T)
      for (int t=1; t<=T; t++)   {
         f[t] = 0.0;
         for (int n=1; n<=N; n++)   {
            for (int m=1; m<=M; m++)   {
               f[t] = f[t] + coeff[n][m] *
                  sin((sqrt(double(n*n) * R*R + double(m*m))/sqrt(R*R + 1.0)) *
                     pi * (2 * freq)  *  double(t)/sample_rate);
            }
         }
         // decay:
         f[t] = f[t] * exp(alpha * double(t));
      }

   // Create half-sinusoid:

      for (int n=1; n<(sin_N/2); n++)   {
         sinusoid[n] = sin_amplitude * sin(2 * pi * (double(n) / double(sin_N)));
      }
      for (int n=sin_N; n<=T; n++)   {
         sinusoid[n] = 0.0;   // Ensure zero
      }

   // Convolve:

      std::cout << "; Sample Rate " << i_sample_rate << std::endl;
      std::cout << std::setw(20) << std::setprecision(20);
      std::cout << "0.0   0.0" << std::endl;
      for (int m=1; m<=T; m++)   {
         convolved[m] = 0;
         for (int t=1; t<=m; t++)   {
            convolved[m] = convolved[m] + (f[t] * sinusoid[m-t+1]);
         }
         std::cout << (double(m) / (sample_rate)) << "   "
            << Amplitude * convolved[m] / normalization << std::endl;
      }

      delete[] coeff;
      delete[] f;
      delete[] sinusoid;
      delete[] convolved;

      return(0);
   }

   /* Reference:

   [1] Gabriel Barton.  Elements of Green's Functions and Propagation:
       Potentials, Diffusion, and Waves.  Oxford,  1989.
*/
```