HuygensGraph Program

R. Steven Turley

March 15, 2014

Contents

1. Introduction

The HuygensGraph program computes non-specular reflection from a rough surface using Huygen's Principle. Radation from the surface is approximated by putting circular radiators along the surface whose phase is determined by the phase of an incident plane wave. The contributions from each Huygens generator is then added to together to get the resultant reflected wave.

Section 2 discusses the theory behind the calculations in the program. Details about how the theory is implemented is discussed in Section 3. Section 4 has details about how to install and run the program.

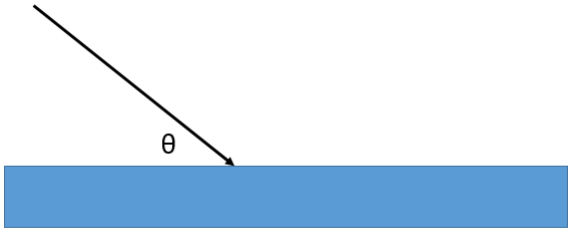

Figure 1: Geometry for incident wave

2. Theory

2.1. General Development

Assume that the surface is illuminated from above by a plane wave making a grazing angle θ with the surface as illustrated in Figure 1 . The wave has a wave vector

$$\begin{aligned} \vec{k} &= k_x \hat{\mathbf{x}} + k_y \hat{\mathbf{y}} && (1) \\ k &= \left|\vec{k}\right| && (2) \\ &= \frac{2\pi}{\lambda} && (3) \\ k_x &= k\cos\theta && (4) \\ k_y &= -k\sin\theta && (5) \end{aligned}$$

an amplitude of 1 and a phase given by

$$e^{i\vec{k}\cdot\vec{r}} = e^{i(k_x x + k_y y)} \ . \qquad (6)$$

In the far field at an angle ϕ, the waves originating on a Huygen scatterer at point (x, y) will have a phase which lags behind a source at the origin. Using the geometry from Figure 2 , the phase δ is

$$\begin{aligned} c &= \sqrt{x^2 + y^2} && (7) \\ \psi &= \arctan\frac{y}{x} && (8) \\ \eta &= \phi - \psi && (9) \\ \delta &= -2\pi\frac{c\cos\eta}{\lambda} && (10) \\ &= -kc\cos\eta && (11) \end{aligned}$$

Therefore, the net phase of the reflected wave hitting the surface point (x_j, y_j) is the sum of the phase from the incident wave and the reflected wave. The complex amplitude of the wave is then

$$A_j = \exp\left[i(k_x x_j + k_y y_j - k c_j \cos\eta_j)\right] \ . \qquad (12)$$

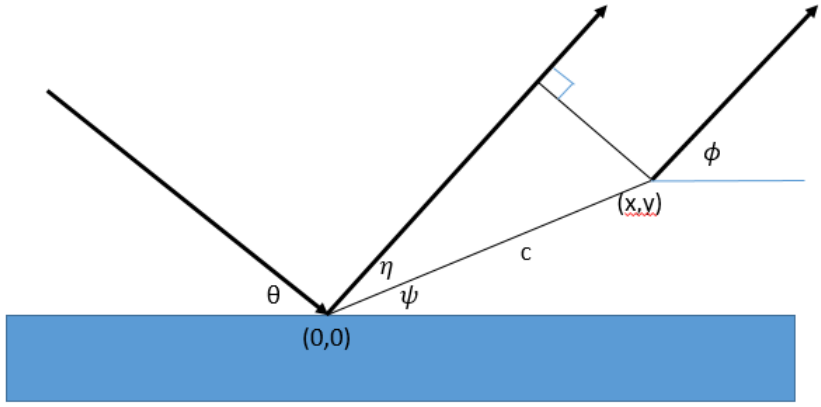

Figure 2: Geometry for reflected wave

The average of all n Huygen waves is the far field

$$E(\phi) = \frac{1}{n}\sum_{j=1}^{n} A_j \tag{13}$$

$$= \frac{1}{n}\sum_{j=1}^{n} \exp\left[i(k_x x_j + k_y y_j - k c_j \cos\eta_j)\right] \ . \tag{14}$$

The intensity is the absolute value squared of this field

$$I(\phi) = |E(\phi)|^2 \ . \tag{15}$$

2.2. Limiting Values

Consider the limiting case of reflection from a flat surface ($y = 0$) where the distance between the Huygen's sources becomes negligible. Let δ be the distance between sources and let the beam have a width L. In this limit,

$$c = x \tag{16}$$

$$\psi = 0 \tag{17}$$

$$\eta = \phi \tag{18}$$

$$E = \frac{1}{n}\sum_{j=1}^{n} \exp[ikx(\cos\theta - \cos\phi)] \tag{19}$$

$$= \frac{1}{n\delta}\sum_{j=1}^{n} e^{ikx\Delta}\delta \tag{20}$$

where

$$\Delta = \cos\theta - \cos\phi \ . \tag{21}$$

$$n\delta = L \tag{22}$$

In the limit as $n \to \infty$, this sum can be replaced by the integral

$$\begin{aligned} E &= \frac{1}{L}\int_0^L e^{ikx\Delta}\,dx && (23)\\ &= \frac{1}{ik\Delta L}\left(e^{ik\Delta L}-1\right) && (24)\\ &= e^{ik\Delta L/2}\frac{2}{k\Delta L}\frac{e^{ik\Delta L/2}-e^{ik\Delta L/2}}{2i} && (25)\\ &= e^{ik\Delta L/2}\frac{2}{k\Delta L}\sin\frac{k\Delta L}{2} && (26)\\ I &= 4\mathrm{sinc}\xi^2 && (27) \end{aligned}$$

where

$$\xi = \frac{k\Delta L}{2}\ . \qquad (28)$$

This is just the physical optics expression for the reflection from a flat plate.

3. Implementation

This section explains how the reflection formula in Equations 14 and 15 was implemented in C# code.

3.1. Surface Generation

The rough surface was created with random Fourier components using a half-Gaussian cutoff. First the amplitude of each Fourier component was computed in k-space as a uniform random number between -0.5 and 0.5 multiplied by the half gaussian attentuation factor

$$\alpha = e^{-k^2/\sigma_k^2} \qquad (29)$$

where σ_k is the cut-off frequency. The real and imaginary parts were chosen to be the Fourier transform of a real sequence. (The negative frequency components are the complex conjugate of the positive frequency components and the lowest and highest frequencies are real.) The real and imaginary component were selected to be independent random numbers.

The FFT routine used required the number of points to be a power of two. Since the user could specify the length of the surface and spacing of the points independently, the surface was padded to the required length and then truncated to the specified length after the FFT was completed.

After transforming the surface to the spatial domain, the surface was forced to have a specified rms roughness σ_x. This was done by computing the actual rms roughness σ_a and then multiplying each surface point by σ_x/σ_a.

In the section below I will relate the discussion above to the actual code in the routine fftRoughness which constructs this surface. Here is header for the function call.

parameter	description
npts	number of points on surface
length	length of surface in wavelengths
rms	desired rms deviation in surface height
cutoff	cutoff frequency for gaussian attentuation in spatial frequency domain

Table 1: Call parameters for the fftRoughness routine

```
private double[] fftRoughness(int npts, double length, double rms, double cuto
```

The call parameters are defined in Table 1 . The next section of the code traps the special case of a smooth surface where the rms height is zero. It skips the relatively slow random number generation and FFT for this special case.

```
{
    double[] y = new double[npts];
    if (rms == 0)
    {
        for (int i = 0; i < npts; i++)
            y[i] = 0;
        return y;
    }
```

The next step is code to generate the random attenuated amplitudes in the frequency domain. Care needs to be taken with the highest and lowest frequency points in the array which are the first two elements, stored as real numbers. The rest of the elements are the alternating real and imaginary coefficients at each spatial frequency.

```
    double[] x = xpts;
    // Create FFT array which is next size bigger in factors of two
    int fpts = (int)Math.Pow(2.0, Math.Truncate(Math.Log(npts)
                / Math.Log(2.0))+1.0);
    double[] fy = new double[fpts];
    Random rand = new Random();
    fy[0] = 0; // No zero frequency component
    double k = 0;
    for (int j = 2; j < fpts; j+=2)
    {
        k = j * Math.PI / length;
        double atten = Math.Exp(-k * k / (cutoff * cutoff));
        fy[j] = (rand.NextDouble()-0.5)*atten;
        fy[j+1] = (rand.NextDouble() - 0.5) * atten;
    }
    k = fpts * Math.PI / length;
    fy[1] = (rand.NextDouble() - 0.5) *
            Math.Exp(-k * k / (cutoff * cutoff));
```

```
            // highest frequency point
```

Next, the fast Fourier transform is used to convert from the spatial frequency domain to the spatial domain. Since the array was padded to get a power of two for the number of points, the first part of the array is then copied to the output array.

```
    LomontFFT fft = new LomontFFT();
    fft.RealFFT(fy, false); // inverse transform
    // Copy beginning of padded sequence
    for (int i = 0; i < npts; i++)
    {
        y[i] = fy[i];
    }
```

Lastly, the rms height of the surface is calculated so that the y values can be renormalized.

```
    // Normalize to fix rms
    double sum = 0;
    double sum2 = 0;
    for (int i = 0; i < npts; i++)
    {
        sum += y[i];
        sum2 += y[i] * y[i];
    }
    sum /= npts;
    sum2 /= npts;
    double sigma = Math.Sqrt(sum2 - sum * sum);
    for (int i = 0; i < npts; i++)
    {
        y[i] *= rms / sigma;
    }
    return y;
}
```

3.2. Reflection Computation

Here is the routine ophase which computes the phase shift in the output relative to the origin. It takes as its arguments the output angle phi in radians and the x and y positions of the point in wavelength units.

```
private double ophase(double phi, double x, double y)
{
    double c = Math.Sqrt(x * x + y * y);
    double psi = Math.PI / 2 * Math.Sign(y);
    if (x > 0)
    {
        psi = Math.Atan2(y, x);
```

```
    }
    double eta = phi - psi;
    return -c * Math.Cos(eta) * 2 * Math.PI;
}
```

It is used in the following routine which the intensity of the reflected light per unit angle normalized to one at specular reflectance. The constructor which does the calculation is called with the incident angle in degrees, the array of x and y surface points. Thetar is the incident angle converted into radians.

```
private double[] HuygensRefl(double thetai, double[] xpts, double[] ypts)
{
    double[] refl = new double[deg.Length];
    double thetar = thetai * Degree;
    int npts = xpts.Length;
```

The next part of the routine finds the phase of the incident wave at the surface iphase.

```
    double kx = 2 * Math.PI * Math.Cos(thetar);
    double ky = -2 * Math.PI * Math.Sin(thetar);
    double[] iphase = new double[npts];
    for (int i = 0; i < npts; i++)
    {
        iphase[i] = kx * xpts[i] + ky * ypts[i];
    }
```

Next, the routine adds in the phase lag for the radiated light computed by ophase and adds up the contribution from each Huygen point in the far field.

```
    for (int j = 0; j < deg.Length; j++)
    {
        double phi = deg[j] * Degree;
        double rtotal = 0;
        double itotal = 0;
        for (int i = 0; i < npts; i++)
        {
            double pprime = ophase(phi, xpts[i], ypts[i]);
            double tphase = pprime + iphase[i];
            rtotal += Math.Cos(tphase);
            itotal += Math.Sin(tphase);
        }
```

Finally, each sum is normalized by the number of points and the absolute value squared of the field is found.

```
        rtotal /= npts;
        itotal /= npts;
        refl[j] = rtotal * rtotal + itotal * itotal;
```

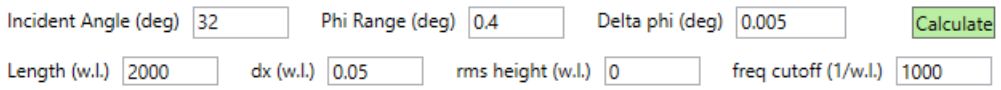

Figure 3: input parameters

```
    }
    return refl;
}
```

4. Program Operation

This section explains the inputs and outputs of the HuygensGraph program. The general use is to change any of the inputs to the desired values and then click on the Calculate button. Each calculation results in a fresh set of values from a new random surface.

4.1. Input Parameters

The input parameters appear at the top of the program window as shown in Figure 3. Table 2 describes each of the inputs in the program. After changing an input, the Calculate button must be clicked in order for the change to be reflected in the output.

4.2. Graphical Output

The output of the program consists of two graphs, the reflected intensity and the surface. These graphs were generated using a Microsoft library that has a bug in it. If the scale on the graphs don't show up correctly, just hit calculated again and the scales will be okay. Figure 4 is an example of a reflectance output available in the reflecance tab. The horiztonal axis is in degrees and the vertical axis is intensity per unit angle. The surface tab plots the surface used in the corresponding calculation as shown in the example in Figure 5 . Note that both the y and x axis are in units of wavelengths but with vastly different scales.

A. Graphics Details

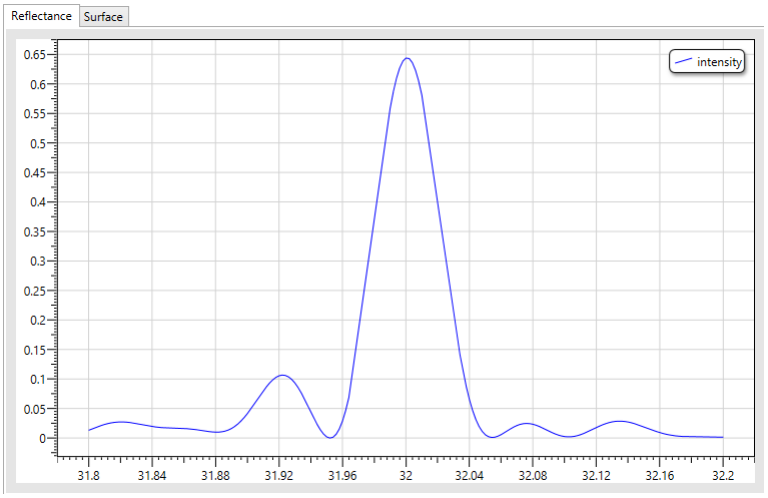

Figure 4: sample intensity graph

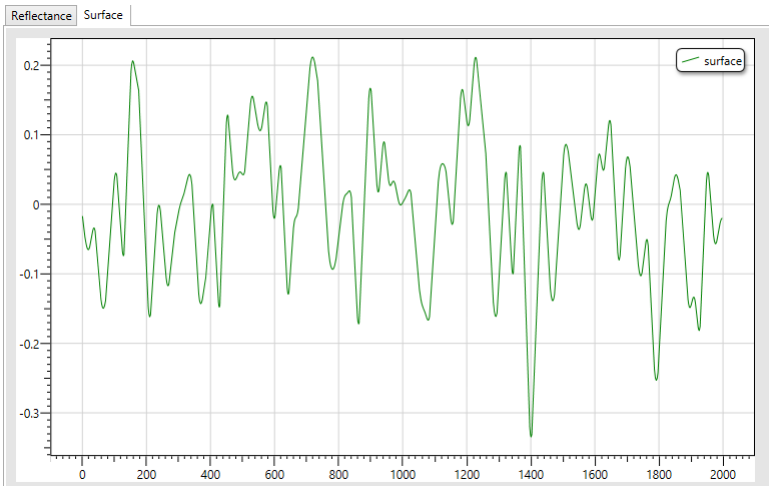

Figure 5: sample surface plot

parameter	**description**
Incident Angle (deg)	angle of incident light in degrees measured relative to grazing
Phi Range (deg)	range of output angles; output will range from the incident angle to plus or minus half this value
Delta phi (deg)	spacing of output angles in degrees
Length (w.l.)	length of beam (or mirror) in wavelengths
dx (w.l.)	spacing between x points in wavelenths
rms height (w.l.)	rms height variation in the surface in wavelengths
freq cutoff (1/w.l.)	σ: cutoff frequency of gaussian filter in k space; filter is $\exp[-(k/\sigma)^2]$

Table 2: input descriptions