Monte Carlo Methods in Practice

This project contains the following files (right-click files you'd like to download):

mcsim.cpp mcintegration.cpp

Monte Carlo simulation of light transport

Instructions to compile this program:
Download the mcsim.cpp file to a folder. Open a shell/terminal, and run the following command where the files is saved:
c++ -O3 -o mcsim mcsim.cpp -std=c++11
Run with: ./mcsim. Open the file ./out.png in Photoshop or any program reading PPM files.

```
#include <cstdlib>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <fstream>

double getCosTheta(const double &g) // sampling the H-G scattering phase function
{
    if (g == 0) return 2 * drand48() - 1;
    double mu = (1 - g * g) / (1 - g + 2 * g * drand48());
    return (1 + g * g - mu * mu) / (2.0 * g);
}
```

Combpute the new photon direction (due to scattering event)

```
void spin(double &mu_x, double &mu_y, double &mu_z, const double &g)
{
    double costheta = getCosTheta(g);
    double phi = 2 * M_PI * drand48();
    double sintheta = sqrt(1.0 - costheta * costheta); // sin(theta)
    double sinphi = sin(phi);
    double cosphi = cos(phi);
    if (mu_z == 1.0) {
        mu_x = sintheta * cosphi;
        mu_y = sintheta * sinphi;
        mu_z = costheta;
    }
    else if (mu_z == -1.0) {
        mu_x = sintheta * cosphi;
        mu_y = -sintheta * sinphi;
        mu_z = -costheta;
    }
    else {
        double denom = sqrt(1.0 - mu_z * mu_z);
        double muzcosphi = mu_z * cosphi;
        double ux = sintheta * (mu_x * muzcosphi - mu_y * sinphi) / denom + mu_x * costheta;
        double uy = sintheta * (mu_y * muzcosphi + mu_x * sinphi) / denom + mu_y * costheta;
        double uz = -denom * sintheta * cosphi + mu_z * costheta;
        mu_x = ux, mu_y = uy, mu_z = uz;
    }
}
```

Simulate the transport of light in a thin translucent slab

```
void MCSimulation(double *&records, const uint32_t &size)
{
```

Total number of photon packets

```
    uint32_t nphotons = 100000;
    double scale = 1.0 / nphotons;
    double sigma_a = 1, sigma_s = 2, sigma_t = sigma_a + sigma_s;
    double d = 0.5, slabsize = 0.5, g = 0.75;
    static const short m = 10;
    double Rd = 0, Tt = 0;
    for (int n = 0; n < nphotons; ++n) {
        double w = 1;
        double x = 0, y = 0, z = 0, mux = 0, muy = 0, muz = 1;
        while (w != 0) {
            double s = -log(drand48()) / sigma_t;
            double distToBoundary = 0;
            if (muz > 0) distToBoundary = (d - z) / muz;
            else if (muz < 0) distToBoundary = -z / muz;
```

Did the pack leave the slab?

```
            if (s > distToBoundary) {
#ifdef ONED
                // compute diffuse reflectance and transmittance
                if (muz > 0) Tt += w; else Rd += w;
#else
                int xi = (int)((x + slabsize / 2) / slabsize * size);
                int yi = (int)((y + slabsize / 2) / slabsize * size);
                if (muz > 0 && xi >= 0 && x < size && yi >= 0 && yi < size) {
                    records[yi * size + xi] += w;
                }
#endif
                break;
            }
```

Move photon packet

```
            x += s * mux;
            y += s * muy;
            z += s * muz;
```

The photon packet looses energy (absorption)

```
            double dw = sigma_a / sigma_t;
            w -= dw; w = std::max(0.0, w);
            if (w < 0.001) { // russian roulette test
                if (drand48() > 1.0 / m) break;
                else w *= m;
            }
```

Scatter

```
            spin(mux, muy, muz, g);
        }
    }
#ifdef ONED
    printf("Rd %f Tt %f\n", Rd * scale, Tt * scale);
#endif
}

int main(int argc, char **argv)
{
    double *records = NULL;
    const uint32_t size = 512;
    records = new double[size * size * 3];
    memset(records, 0x0, sizeof(double) * size * size * 3);
    uint32_t npasses = 1;

    float *pixels = new float[size * size]; // image
    while (npasses < 64) {
        MCSimulation(records, size);
        for (int i = 0; i < size * size; ++i) pixels[i] = records[i] / npasses;
        //display(pixels);
        npasses++;
        printf("num passes: %d\n", npasses);
    }

    // save image to file
    std::ofstream ofs;
    ofs.open("./out.ppm", std::ios::out | std::ios::binary);
    ofs << "P6\n" << size << " " << size << "\n255\n";
    for (uint32_t i = 0; i < size * size; ++i) {
        unsigned char val = (unsigned char)(255 * std::min(1.0f, pixels[i]));
        ofs << val << val << val;
    }

    ofs.close();

    delete [] records;
    delete [] pixels;

    return 0;
}
```