Image Based Lighting

Implementation

Read an HDR image, beware gamma
Sky dome object with
    geometry
    emissive property
Sampled as a light:
    rays may intersect accidentally
    rays may intersect explicitly
    both return a pixel's value (or interpolation among 4)

Geometry

Sphere of large **radius** (10^3 or more)
No real intersection calculation. Just extend ray's direction to sphere radius.
Set normal to -ray's direction

Emissive property

To fit in the path-tracing framework, any light must answer 3 queries:
    SampleLight, **PdfLight**, **Radiance**
Use Cumulative density function binary-searches in each dimension. (some preprocessing needed)

Simple example

As a simple example transform a 1D array of pixel radiance
into a PDF ready for sampling:
    Radiance array [1,1,1,2,8,3]
    Total of array is 16.
    PDF is [1,1,1,2,8,3]/16
To sample according to the PDF, we build a CDF (cumulative density function)
    CDF [1,2,3,5,13,16]/16
To sample, choose a random number uniformly in 0..1 range ξ
    search for ξ in the CDF (A binary search of course!!)
    The index of that search result is the sample pixel.
    For instance if $\xi=1/2$,
        the search will fall between 5/16 and 13/16
        resulting in an index of 4
        and a choice of the pixel with radiance 8.
    Indeed half of all ξ choices will fall between 5/16 and 13/16
        return a choice of pixel 4 with a radiance of 8.
        This makes sense because that one pixel contains ½ of a the radiance.

Strategy. Cumulative Density functions and binary searches

See http://www.igorsklyar.com/system/documents/papers/4/fiscourse.comp.pdf pages 19, 20, and 21.

```
// Pre-processing step: Marginal and conditional CDF
   pBuffer = new float[width*(height+1)];
   pUDist = &pBuffer[width*height];
   float* pSinTheta = new float[height];

   float angleFrac = PI/float(height);
   float theta = angleFrac*0.5f;
   for (unsigned int i=0; i<height; i++, theta+=angleFrac)
       pSinTheta[i] = sin(theta);

   for (unsigned int i=0,m=0; i<width; i++, m+=height)  {
       float *pVDist = &pBuffer[m];
       unsigned int k = i*3;
       pVDist[0] = 0.2126f*image[k+0] + 0.7152f*image[k+1] + 0.0722f*image[k+2];
       pVDist[0] *= pSinTheta[0];

       for (unsigned int j=1,k=(width+i)*3; j<height;  j++, k+=width*3) {
           float lum = 0.2126*image[k+0] + 0.7152*image[k+1] + 0.0722*image[k+2];
           pVDist[j] = pVDist[j-1] + lum*pSinTheta[j]; }

       if (i == 0)
           pUDist[i] = pVDist[height-1];
       else
           pUDist[i] = pUDist[i-1] + pVDist[height-1];
```

```
virtual Intersection SampleAsLight()
{
    Intersection B;
    double u = myrandom();
    double v = myrandom();
    float maxUVal = ibl->pUDist[ibl->width-1];
    float* pUPos = std::lower_bound(ibl->pUDist, ibl->pUDist+ibl->width,
                                    u*maxUVal);
    int iu = pUPos - ibl->pUDist;
    float* pVDist = &ibl->pBuffer[ibl->height*iu];
    float* pVPos = std::lower_bound(pVDist, pVDist+ibl->height,
                                    v*pVDist[ibl->height-1]);
    int iv = pVPos -pVDist;

    double phi =  ibl->angle - 2*PI*iu/ibl->width;
    double theta = PI*iv/ibl->height;
    B.N = vec3(sin(theta)*cos(phi),
                  sin(theta)*sin(phi),
                  cos(theta));
    B.P =  B.N*radius;
    B.obj = this;
    return B;
}
```

```
virtual float PdfAsLight(const Intersection& B) const {
        vec3 P = normalized(B.P);
        double fu = (ibl->angle-atan2(P[1], P[0]))/PI2;
        fu = fu-floor(fu);          // Wrap to be within 0...1
        int u = floor(ibl->width*fu);
        int v = floor(ibl->height*acos(P[2])/PI);
        float angleFrac = PI/float(ibl->height);
        float* pVDist = &ibl->pBuffer[ibl->height*u];
        float pdfU = (u == 0)?(ibl->pUDist[0]):(ibl->pUDist[u]-ibl->pUDist[u-1]);
        pdfU /= ibl->pUDist[ibl->width-1];
        pdfU *= ibl->width/PI2;
        float pdfV = (v == 0)?(pVDist[0]):(pVDist[v]-pVDist[v-1]);
        pdfV /= pVDist[ibl->height-1];
        pdfV *= ibl->height/PI;
        float theta = angleFrac*0.5 + angleFrac*v;
        float pdf = pdfU*pdfV*sin(theta)/(4.0*PI*radius*radius);
        //printf("(%f %f %f) %d %d %g\n", P[0], P[1], P[2], u, v, pdf);
        return pdf;
    }

    virtual vec3 Radiance(const Intersection& A) {
        vec3 P = normalized(A.P);
        double u = (angle-atan2(P[1], P[0]))/PI2;
        u = u-floor(u);          // Wrap to be within 0...1
        double v = acos(P[2])/PI;
        int i0 = floor(u*width);
        int j0 = floor(v*height);
        double uw[2], vw[2];
        uw[1] = u*width -i0;  uw[0] = 1.0-uw[1];
        vw[1] = v*height-j0;  vw[0] = 1.0-vw[1];
        vec3 r(0.0f, 0.0f, 0.0f);
        for (int i=0; i<2;  i++) {
            for (int j=0; j<2;  j++) {
                int k = 3*( ((j0+j)%height)*width + ((i0+i)%width) );
                for (int c=0;  c<3;  c++) {
                    r[c] += uw[i]*vw[j]*image[k+c]; } } }

        return r; }
```