# RSO Report #3
# Light Inheritance Hierarchy + Multiple Importance Sampling

Jakub Profota

December 15, 2025

## Light Inheritance Hierarchy

### Source Code

All light sources now share the same interface.

```
struct LightSource {
  virtual ~LightSource() = default;
  virtual vec3 Sample_Li(vec3 &L, const Hit &hit) = 0;
  virtual double Pdf_Li(const vec3 &L, const double totalPower) const = 0;
};
```

The light source represented by a sphere stayed basically the same as in the original *demorso* template. A new light source represented by an environment map was introduced. The functionality stays the same as from the previous report.

```
struct EnvMapLight : public LightSource {
  EnvMap *envMap;
  EnvMapLight(EnvMap *envMap) : envMap(envMap) {}

  vec3 Sample_Li(vec3 &L, const Hit &hit) override {
    // Find sample coordinates
    float fu = envMap->wDist.Sample(drandom());
    int u = static_cast<int>(fu);

    // ...

    // Rotate the vector so we don't look to the ground
    L.x = -L_local.y;
    L.y = -L_local.z;
    L.z = -L_local.x;

    return envMap->image[v * envMap->width + u];
  }

  double Pdf_Li(const vec3 &L, const double totalPower) const override {
    // Rotate the vector back
    vec3 L_local = vec3(-L.z, -L.x, -L.y);

    // ...

    float prob_uv = envMap->normMap[u][v];
    double pdf = (prob_uv * envMap->width * envMap->height) /
                 (2.0 * M_PI * M_PI * sinTheta);
    return pdf;
  }
};
```

A new object representing the environment map that implements the *Intersectable* was added to the codebase. Again, the functionality stays the same as in the previous report, with an exception of a new addition, the *intersect* method.

```
struct EnvMap : public Intersectable {
  EnvMap(const char *envMapFile) {
    if (!loadHDR(envMapFile, image, width, height))
      abort();
    ComputeRadianceMap();
    NormalizeRadianceMap();
    ComputeDistributions();
  }

  // ...

  Hit intersect(const Ray &ray) override {
    Hit hit;
    hit.t = 1.0e20;
    hit.position = ray.start + ray.dir * hit.t;
    hit.normal = ray.dir * -1.0;
    hit.object = this;

    double theta = std::acos(std::max(-1.0, std::min(1.0, ray.dir.y)));
    double phi = std::atan2(ray.dir.z, ray.dir.x);
    phi += M_PI / 2.0;
    if (phi < 0.0)
      phi += 2.0 * M_PI;
    else if (phi >= 2.0 * M_PI)
      phi -= 2.0 * M_PI;

    double u = phi * (1.0 / (2.0 * M_PI));
    double v = theta / M_PI;

    int ui = static_cast<int>(u * width);
    int vi = static_cast<int>(v * height);
    ui = std::max(0, std::min(width - 1, ui));
    vi = std::max(0, std::min(height - 1, vi));

    static thread_local LightMaterial mat(vec3(0, 0, 0));
    mat.Le = image[(height - 1 - vi) * width + ui];
    hit.material = &mat;
    return hit;
  }
};
```

Light source sampling function had to be updated to construct the *EnvMapLight* light source. The function now returns a light source wrapped in a unique pointer, while the exact light source type is determined by dynamic casting of the underlying object.

```
  std::unique_ptr<LightSource>
  sampleLightSource(const vec3 &illuminatedPoint) {
    while (true) {
      double threshold = totalPower * drandom(); double running = 0;
      for (int i = 0; i < objects.size(); i++) {
        running += objects[i]->power;
        if (running > threshold) {
          if (dynamic_cast<Sphere *>(objects[i])) {
            Sphere *sphere = (Sphere *)objects[i];
            vec3 point, normal;
            ((Sphere *)objects[i])
                ->sampleUniformPoint(illuminatedPoint, point, normal);
            return std::make_unique<SphereLight>(sphere, point, normal);
          } else if (dynamic_cast<EnvMap *>(objects[i])) {
            EnvMap *envMap = (EnvMap *)objects[i];
            return std::make_unique<EnvMapLight>(envMap);
          }
          return nullptr;
        }
      }
    }
  }
```

Finally, the light sampling loop in the *trace* method was edited to hide implementation details such as dealing with the geometry of each type of the light sources.

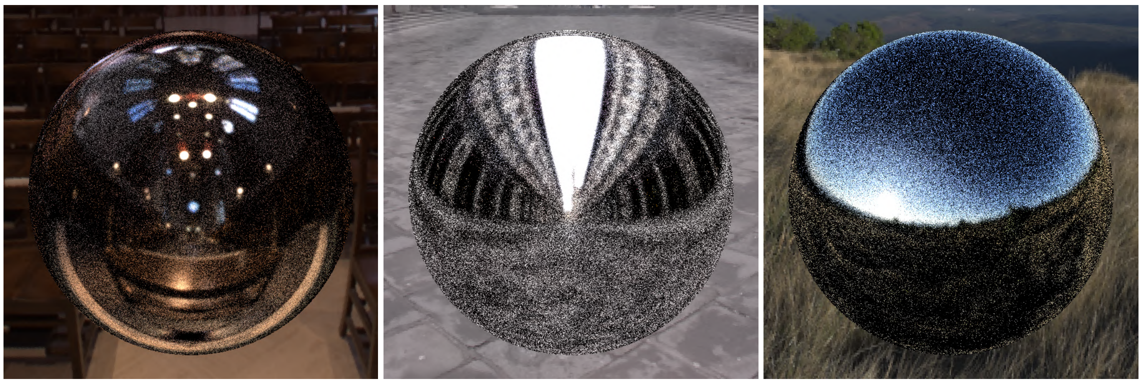
```
double riasX = 0.0, riasY = 0.0, riasZ = 0.0;
#pragma omp parallel for reduction(+ : riasX, riasY, riasZ)
for (int i = 0; i < nLightSamples; i++) {
  auto lightSample = sampleLightSource(hit.position);
  vec3 L;
  vec3 f = lightSample->Sample_Li(L, hit);
  if (L.length() == 0)
    continue;
  double pdf = lightSample->Pdf_Li(L, totalPower);
  L = L.normalize();
  if (pdf == 0.0)
    continue;
  double cosThetaSurface = dot(hit.normal, L);
  if (cosThetaSurface <= 0.0)
    continue;
  f = f * hit.material->BRDF(hit.normal, inDir, L) * cosThetaSurface;
  riasX += f.x / pdf / nTotalSamples;
  riasY += f.y / pdf / nTotalSamples;
  riasZ += f.z / pdf / nTotalSamples;
}
radianceLightSourceSampling.x += riasX;
radianceLightSourceSampling.y += riasY;
radianceLightSourceSampling.z += riasZ;
```

The codebase now seamlessly handles the creation of both light sources in the same vector.

```
objects.push_back(new Sphere(lightCenterPos + vec3(-4.5, 0, 0), 0.07,
                             new LightMaterial(vec3(4, 2, 1))));
objects.push_back(new EnvMap(envMapFile));
```

## Some Images of *EnvMap's intersect* Method

Images of 15,000 light samples. HDR conversion used exposure +0.0, offset +0.0, gamma +2.2.



# Multiple Importance Sampling

## Source Code

The two loops in the *trace* function were edited to utilize multiple importance sampling if a flag is enabled.

```
    // The direct illumination for chosen number of samples
    double riasX = 0.0, riasY = 0.0, riasZ = 0.0;
#pragma omp parallel for reduction(+ : riasX, riasY, riasZ)
    for (int i = 0; i < nLightSamples; i++) {
      auto lightSample = sampleLightSource(hit.position);
      // ...
      double pdf = lightSample->Pdf_Li(L, totalPower);
      // ...
      f = f * hit.material->BRDF(hit.normal, inDir, L) * cosThetaSurface;
#ifdef USE_MIS
      double pdf_brdf = hit.material->sampleProb(hit.normal, inDir, L);
      riasX += f.x * 2.0 / (pdf + pdf_brdf) / nTotalSamples;
      riasY += f.y * 2.0 / (pdf + pdf_brdf) / nTotalSamples;
      riasZ += f.z * 2.0 / (pdf + pdf_brdf) / nTotalSamples;
#else
      riasX += f.x / pdf / nTotalSamples;
      riasY += f.y / pdf / nTotalSamples;
      riasZ += f.z / pdf / nTotalSamples;
#endif
    } // for all the samples from light
    radianceLightSourceSampling.x += riasX;
    radianceLightSourceSampling.y += riasY;
    radianceLightSourceSampling.z += riasZ;


    // The contribution from importance sampling BRDF.cos(theta)
    for (int i = 0; i < nBRDFSamples; i++) {
      vec3 outDir;
      if (hit.material->sampleDirection(hit.normal, inDir, outDir)) {
        double pdfBRDFSampling =
            hit.material->sampleProb(hit.normal, inDir, outDir);
        double cosThetaSurface = dot(hit.normal, outDir);
        if (cosThetaSurface > 0) {
          vec3 brdf = hit.material->BRDF(hit.normal, inDir, outDir);
          Hit lightSource =
              firstIntersect(Ray(hit.position, outDir), hit.object);
          if (lightSource.t > 0 &&
              lightSource.material->Le.average() > 0) {
            double distance2 = lightSource.t * lightSource.t;
            double cosThetaLight = dot(lightSource.normal, outDir * (-1));
            if (cosThetaLight > epsilon) {
              double pdfLightSourceSampling =
                  lightSource.object->pointSampleProb(totalPower) *
                    distance2 / cosThetaLight;
              vec3 f = lightSource.material->Le * brdf * cosThetaSurface;
              double p = pdfBRDFSampling;
#ifdef USE_MIS
              double distance2 = lightSource.t * lightSource.t;
              double cosThetaLight = dot(lightSource.normal,
                outDir * (-1));
              if (cosThetaLight < epsilon)
                continue;
              double pdf_light =
                  lightSource.object->pointSampleProb(totalPower)
```

```
                    * distance2 / cosThetaLight;
              radianceBRDFSampling += f * 2.0 /
                (pdf_light + p) / nTotalSamples;
#else
              radianceBRDFSampling += f / p / nTotalSamples;
#endif
            } else
              printf("ERROR: Sphere hit from back\n");
          }
        }
      }
    } // for i
```

## Images

The following images were all rendered using just 100 samples. The first row shows light sampling on the left and BRDF sampling on the right.

The second row shows multiple importance sampling with half of the samples obtained using light samples, and the other half with BRDF sampling. The left image is just a sum of the light and BRDF radiances, while the right image uses the $2/(p_{light} + p_{brdf})$ formula.

The conversion from HDR again used exposure +0.0, offset +0.0, gamma +2.2.