RSO Report #4
Path Tracing

Jakub Profota

January 4, 2026

Source Code

The implemented path tracing algorithm casts a single shadow ray (obtained by light sampling) and a single bounce (obtained by BRDF sampling) on each hit according to the instructions. The tracing of a given ray may only be terminated by either the Russian roulette or missing all objects after the bounce. However, in contrast to the assignment instructions, the emission of a light directly hit after a bounce is not discarded, but rather contributes to the final collected radiance of the traced ray, to better support specular materials and reduce noise. Multiple importance sampling is utilized to prevent double counting the emission of the same light by both a shadow ray and a direct hit after a bounce, and thus keep the path tracing unbiased.

```
// inside for loops in render()
vec3 color(0,0,0);
if (method == PATH_TRACING) {
  double invTotalSamples = 1.0 / nTotalSamples;
  for (int s = 0; s < nTotalSamples; s++)
    color = color + pathTrace(camera.getRay(
      X + drandom() - 0.5, Y + drandom() - 0.5)) * invTotalSamples;
} else
  color = trace(camera.getRay(X, Y));
```

```
// pathTrace()
  vec3 pathTrace(const Ray &r) {
    vec3 radiance(0, 0, 0);
    vec3 throughput(1, 1, 1);
    Ray ray = r;
    Intersectable *lastObject = NULL;

    double prevPdfBRDF = 1.0;
    bool isPrimaryRay = true;

    while (true) {
      Hit hit = firstIntersect(ray, lastObject);
      if (hit.t < 0)
        break;

      // Light source hit
      if (hit.material->Le.average() > epsilon) {
        // Primary ray, light source directly visible from camera
        if (isPrimaryRay) {
          radiance = radiance + throughput * hit.material->Le;
        }

        // Light source hit after a bounce
        else {
          double dist2 = hit.t * hit.t;
```

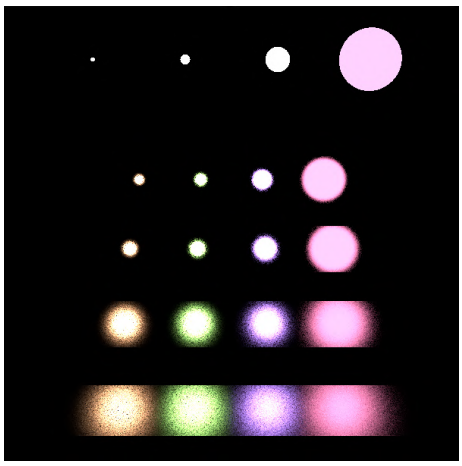
```
    double cosThetaLight = dot(hit.normal, ray.dir * (-1));

    double pdfLight = 0.0;
    if (cosThetaLight > epsilon)
      pdfLight = hit.object->pointSampleProb(totalPower) *
        dist2 / cosThetaLight;
    double weight = (prevPdfBRDF + pdfLight > epsilon)
                      ? prevPdfBRDF / (prevPdfBRDF + pdfLight)
                      : 0.0;
    radiance = radiance + throughput * hit.material->Le * weight;
  }

  break;
}

vec3 V = ray.dir * (-1);
vec3 N = hit.normal;

// Shadow ray by light sampling
auto lightSample = sampleLightSource(hit.position);
if (lightSample) {
  vec3 L_dir;
  vec3 L_radiance = lightSample->Sample_Li(L_dir, hit);
  double pdfLight = lightSample->Pdf_Li(L_dir, totalPower);

  if (pdfLight > epsilon && L_radiance.average() > epsilon) {
    double dist = L_dir.length();
    vec3 L_norm = L_dir.normalize();
    Ray shadowRay(hit.position + N * 1e-4, L_norm);
    Hit shadowHit = firstIntersect(shadowRay, hit.object);

    bool visible = false;
    if (shadowHit.t > 0 &&
        shadowHit.material->Le.average() > epsilon)
      if (shadowHit.t < dist + 1e-3)
        visible = true;

    if (visible) {
      double cosTheta = dot(N, L_norm);
      if (cosTheta > epsilon) {
        vec3 brdf = hit.material->BRDF(N, V, L_norm);
        double pdfBRDF = hit.material->sampleProb(N, V, L_norm);

        double weight = (pdfLight + pdfBRDF > epsilon)
                          ? pdfLight / (pdfLight + pdfBRDF)
                          : 0.0;
        radiance = radiance + throughput * L_radiance *
          brdf * cosTheta * weight / pdfLight;
      }
    }
  }
}

// Russian roulette
double p = hit.material->diffuseAlbedo.average() +
           hit.material->specularAlbedo.average();
if (drandom() > p)
  break;
throughput = throughput * (1.0 / p);
```

```
      // Bounce by BRDF sampling
      vec3 nextDir;
      if (hit.material->sampleDirection(N, V, nextDir)) {
        double pdfBRDF = hit.material->sampleProb(N, V, nextDir);
        if (pdfBRDF < epsilon)
          break;

        vec3 brdf = hit.material->BRDF(N, V, nextDir);
        double cosTheta = dot(N, nextDir);
        if (cosTheta <= 0)
          break;

        throughput = throughput * brdf * cosTheta / pdfBRDF;

        ray = Ray(hit.position + N * 1e-4, nextDir);
        lastObject = hit.object;
        prevPdfBRDF = pdfBRDF;
        isPrimaryRay = false;
      } else
        break;
    }
    return radiance;
  }
```

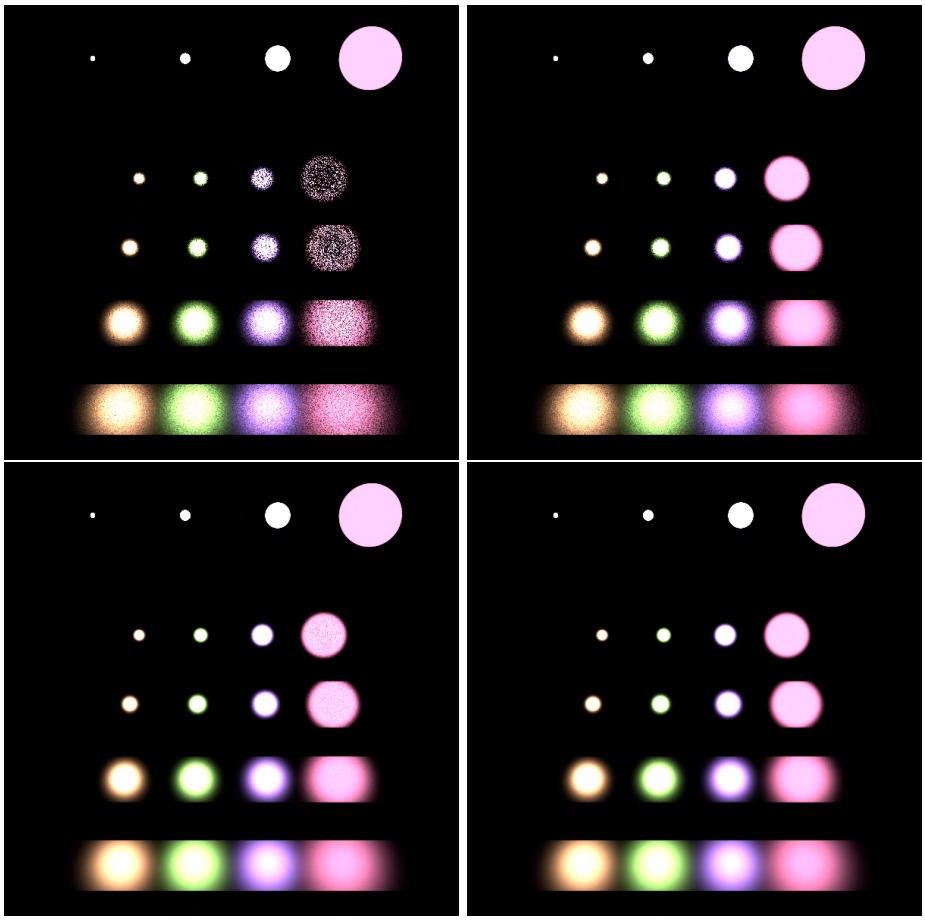
Results

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

The first image serves as a comparison for the following images and shows the output of multiple importance sampling method of balanced BRDF and light sampling from the previous assignments. It was computed using 100 samples per pixel.



The next four images present the results of the plain path tracing (on the left) and multiple importance sampling path tracing (on the right). The top row was computed using 100 samples, the bottom row using 15,000 samples per pixel. It is clearly visible that the MIS path tracing produces less noisy results and is indistinguishable from the MIS sampling method.



The noise difference is more visible on a complex scene: a police snowman in the Cornell box! The first image is plain path tracing, the second is MIS path tracing, both produced using $15,000$ samples per pixel.