CS440 - Advanced Computer Graphics - Homework_3

Baris Sevilmis

March 18, 2022

1 Introduction

Purpose of this homework is to implement Monte Carlo Sampling in addition to the two new rendering algorithms using Nori framework in C++. Provided sampling methods will provide random samples from necessary distributions. One of our rendering algorithms Ambient Occlusion integrator, will ensure squareToCosineHemisphere warp functionality and rely on it to work on intended.

2 Monte Carlo Sampling

2.1 squareToTent and squareToTentpdf

$$u = \sqrt{1 - x} \tag{1}$$

$$v = y * (1 - (1 - u)) \tag{2}$$

$$Return : ((-1, 0) + (1 - u) * (2, 0) + v * (1, 1))) \tag{3}$$

$$Pdf(Point2f\&p) : (s, t) : (v1, v2), (v2, v3), (v3, v1); res\_i = (p.x - t.x) * (s.y - t.y) - (s.x - t.x) * (p.y - t.y) \tag{4}$$

$$For\_all\_i : if((res\_i > 0)or(res\_i + 1 > 0)or..)((res\_i < 0)or(res\_i + 1 < 0)) : return(0); else : return(1); \tag{5}$$

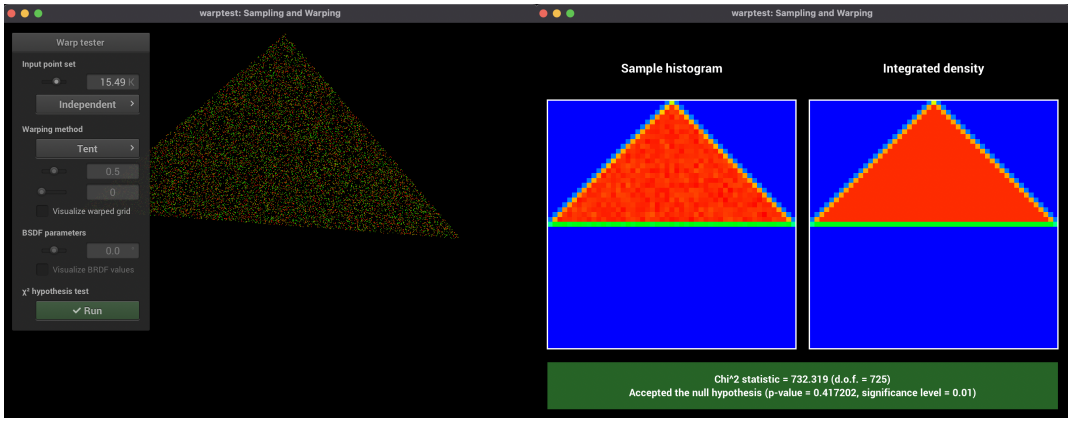

Figure 1: Tent sampling and Chisquare test

2.2 squareToUniformDisk and squareToUniformDiskpdf

$$\theta = 2 * \pi * x, r = \sqrt{y} \tag{6}$$

$$Return(r, \theta) \tag{7}$$

$$Pdf(Point2f\&p) : If(p.x^2 + p.y^2 \leq 1) : Return(\pi); Else : Return(0); \tag{8}$$

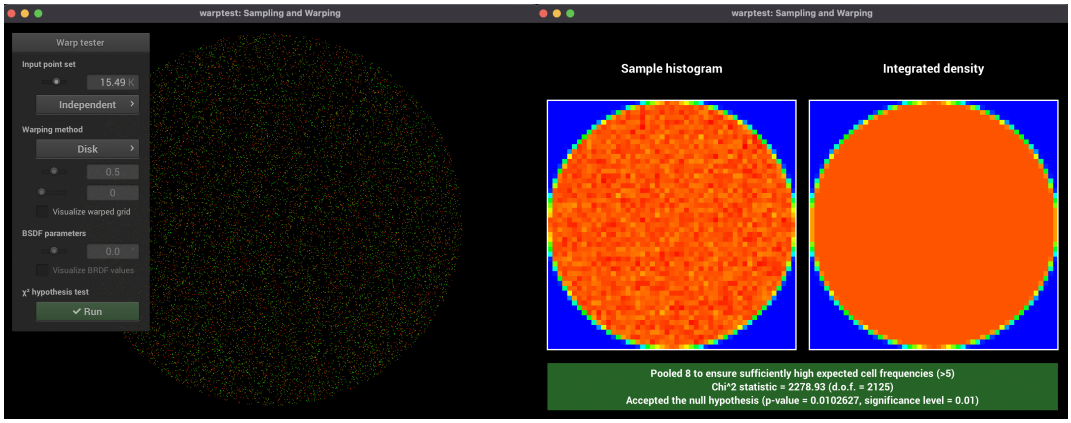

Figure 2: Uniform Disk sampling and Chisquare test

2.3 squareToUniformSphere and squareToUniformDiskpdf

$$\theta = acos(1 - 2 * x), \phi = 2 * \pi * y \tag{9}$$

$$Return(sin(\theta) * cos(\phi), sin(\theta) * sin(\phi), cos(\theta)) \tag{10}$$

$$Pdf(Vector3f\&v) : If(v.x^2 + v.y^2 + v.z^2 - 1 \leq \epsilon) : Return(\pi/4); Else : Return(0); \tag{11}$$

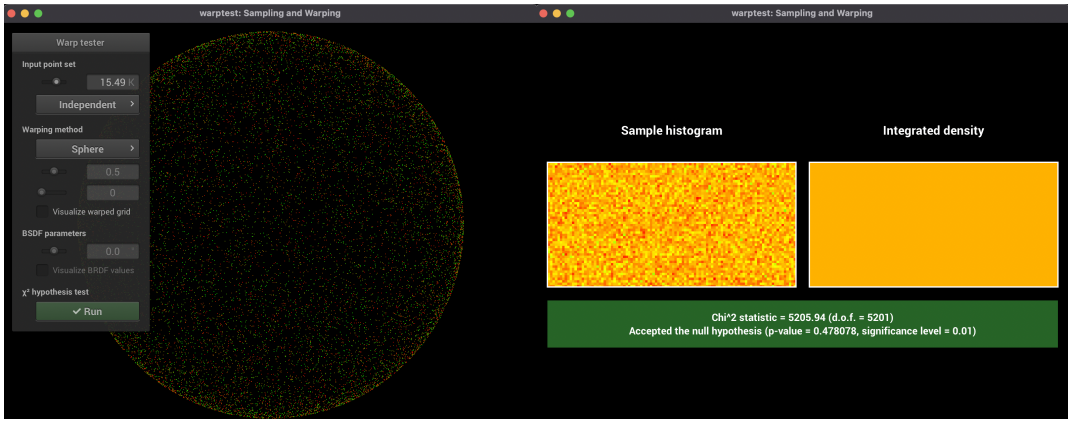

Figure 3: Uniform Sphere sampling and Chisquare test

2.4 squareToUniformHemisphere and squareToUniformHemispherepdf

$$\theta = acos(1 - x), \phi = 2 * \pi * y \tag{12}$$

$$Return(sin(\theta) * cos(\phi), sin(\theta) * sin(\phi), cos(\theta)) \tag{13}$$

$$Pdf(Vector3f\&v) : If(v.x^2 + v.y^2 + v.z^2 - 1 \leq \epsilon) : Return(\pi/2); Else : Return(0); \tag{14}$$

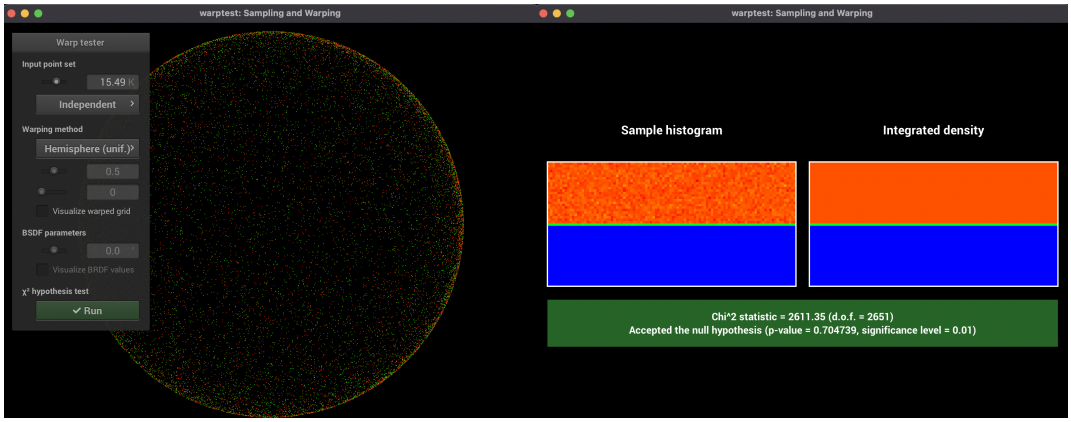

Figure 4: Uniform Hemisphere sampling and Chisquare test

2.5 squareToCosineHemisphere and squareToCosineHemispherepdf

$$\theta = acos(\sqrt{1-x}), \phi = 2 * \pi * y \tag{15}$$

$$Return(sin(\theta) * cos(\phi), sin(\theta) * sin(\phi), cos(\theta)) \tag{16}$$

$$Pdf(Vector3f\&v) : If(v.x^2 + v.y^2 + v.z^2 - 1 \leq \epsilon \&\& v.z > 0) : Return(\pi * v.z); Else : Return(0); \tag{17}$$

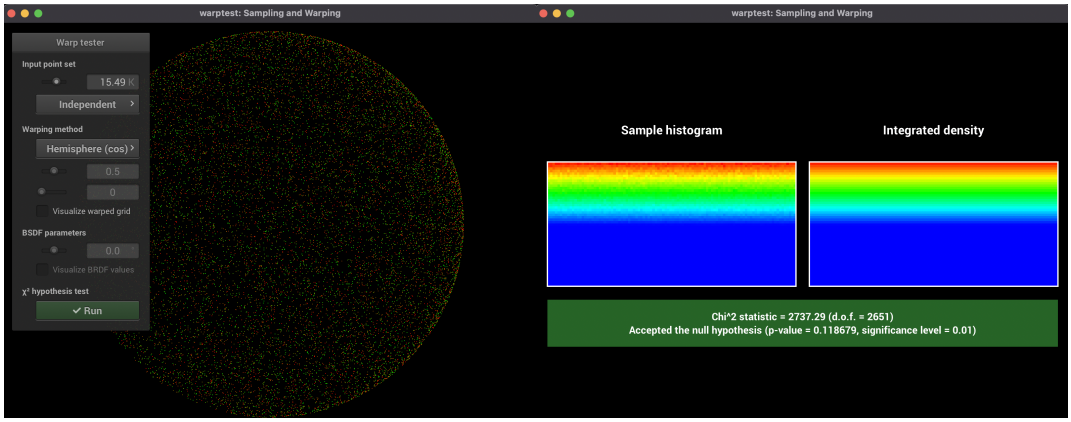

Figure 5: Cosine Hemisphere sampling and Chisquare test

2.6 Beckmann and Beckmannpdf

$$\theta = atan(\sqrt{\alpha^2 * log(1-y)}), \phi = 2 * \pi * x \tag{18}$$

$$Return(sin(\theta) * cos(\phi), sin(\theta) * sin(\phi), cos(\theta)) \tag{19}$$

$$Pdf(Vector3f\&v, \alpha) : Return(\pi * (exp(\frac{tan(acos(v.x))^2}{\alpha^2})/\alpha^2 * m.z^3) \tag{20}$$

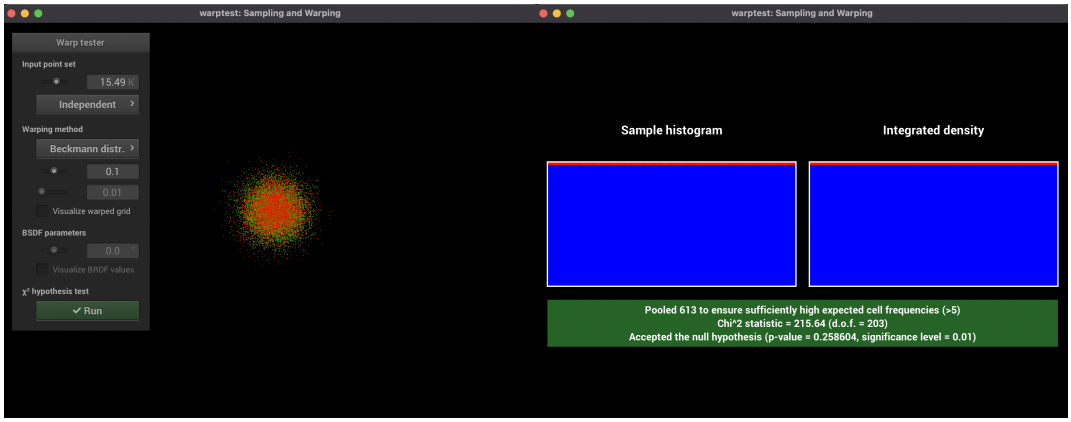

Figure 6: Beckmann sampling and Chisquare test

3 Simple Integrator

Figure 7 contains newly rendered by Simple algorithm bust image by the implemented in comparison to the reference image. As it can be seen easily that the rendering result is same as the reference. For the implementation, provided formula was utilized. In case of given vector intersecting with our scene, we simply subtract intersection point from the saved position. Squared norm of this vector corresponds the norm in our equation. Normal vector of the current frame and normalization of our difference vector are processed through a dot product results in our $cos(\theta)$ angle. In case of existing shadow rays, we simply return 0, and if not provided equation is returned.Rendering time of simple integrator is around 14-15s on release mode.

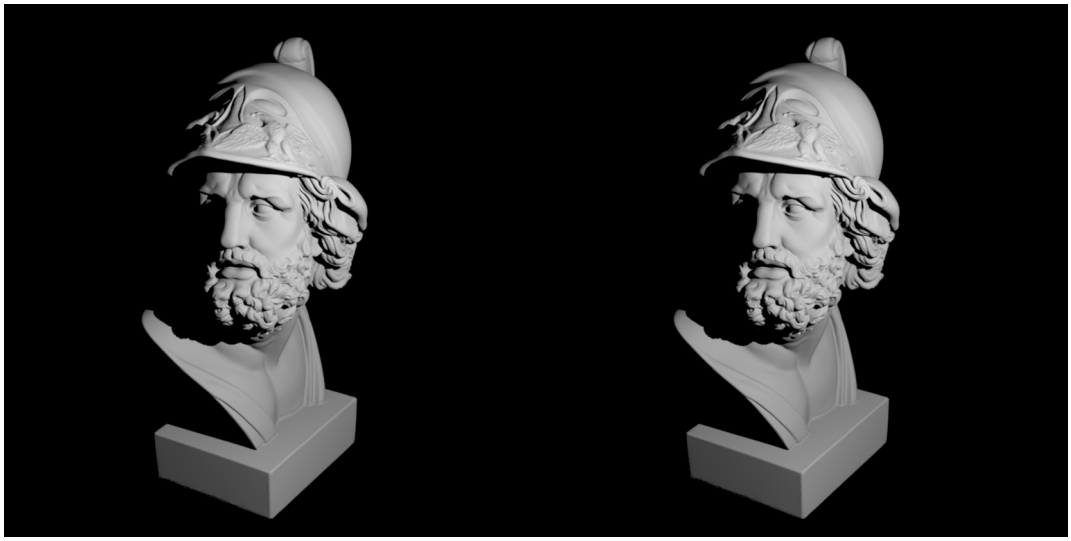

Figure 7: Rendered vs Reference

4 Ambient Occlusion Integrator

Figure 8 contains newly rendered by Ambient Occlusion algorithm bust image by the implemented in comparison to the reference image. As it can be seen easily that the rendering result is same as the reference as in Simple algorithm case. For the implementation, we take sample points and warp them into the cosine hemisphere distribution. When this coordinate is transformed to global world coordinates, we simply check its shadow ray. If shadow ray does indeed intersect with our scene , we return 0 and otherwise, we simply return 1. Rendering takes around 6 minutes and therefore can be stated as slow in comparison to simple algorithm.

Figure 8: Rendered vs Reference