Global Illumination Ray Tracing

A learning focused implementation

Richard Monette April 6th, 2010

Reference Texts

Physically Based Rendering: From Theory to Implementation
M. Pharr, G. Humphreys

Realistic Ray Tracing
P. Shirley, R. K. Morley

Real -Time Rendering
T. Möller, E. Haines

Reference Codebases

smallpt (http://kevinbeason.com/smallpt/)
Keavin Beason

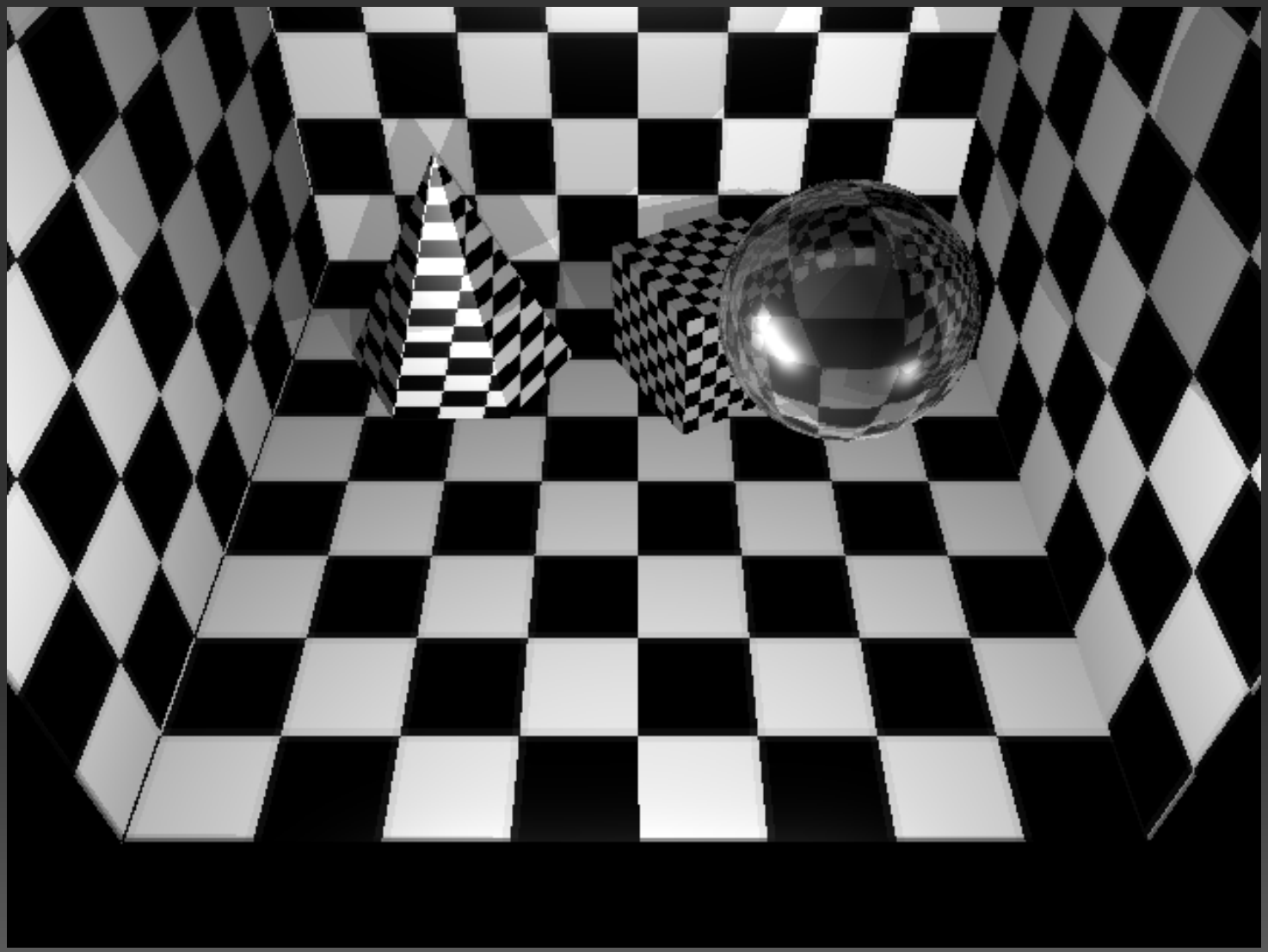
- Compact implementation of Snell / Fresnel Laws

MiniLight (http://www.hxa.name/minilight/)
Harrison Ainsworth

- Has an open source 3D maths library (used for Vector and Octree classes)

DevMaster.net Ray Tracing Tutorial Series (http://www.devmaster.net/articles/raytracing_series/part1.php)
Jacco Bikker

Prior Work



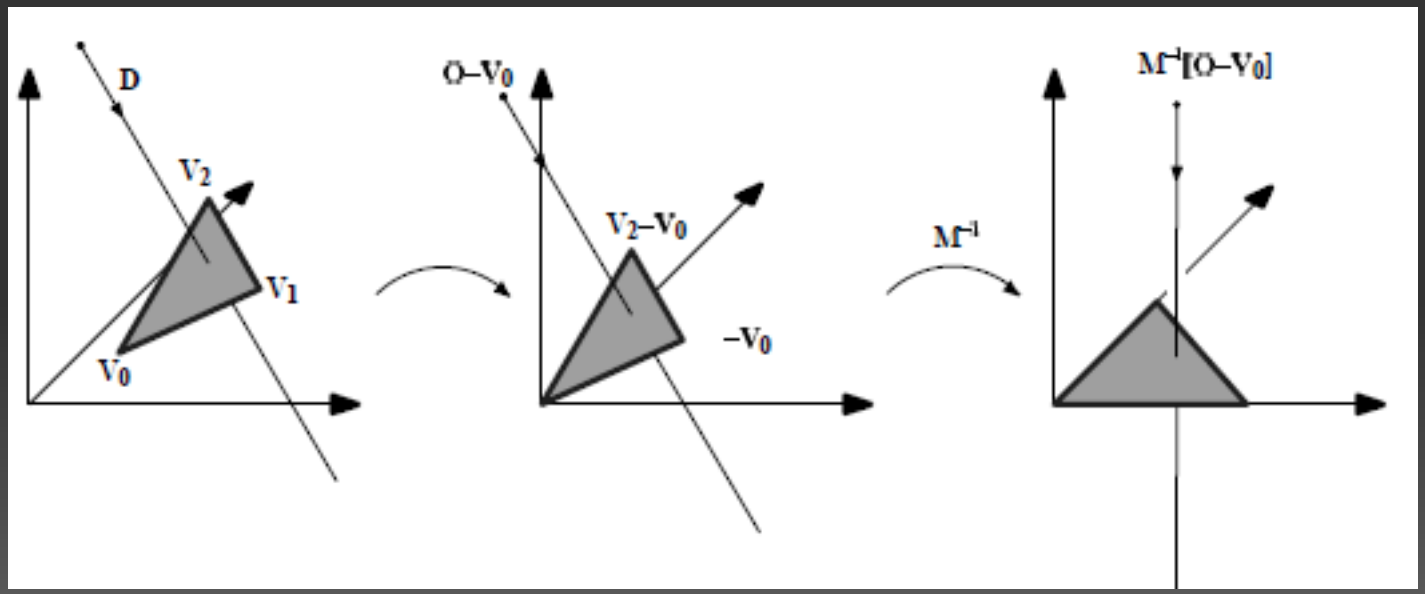
Features

- Monte Carlo based Global Illumination Ray Tracer

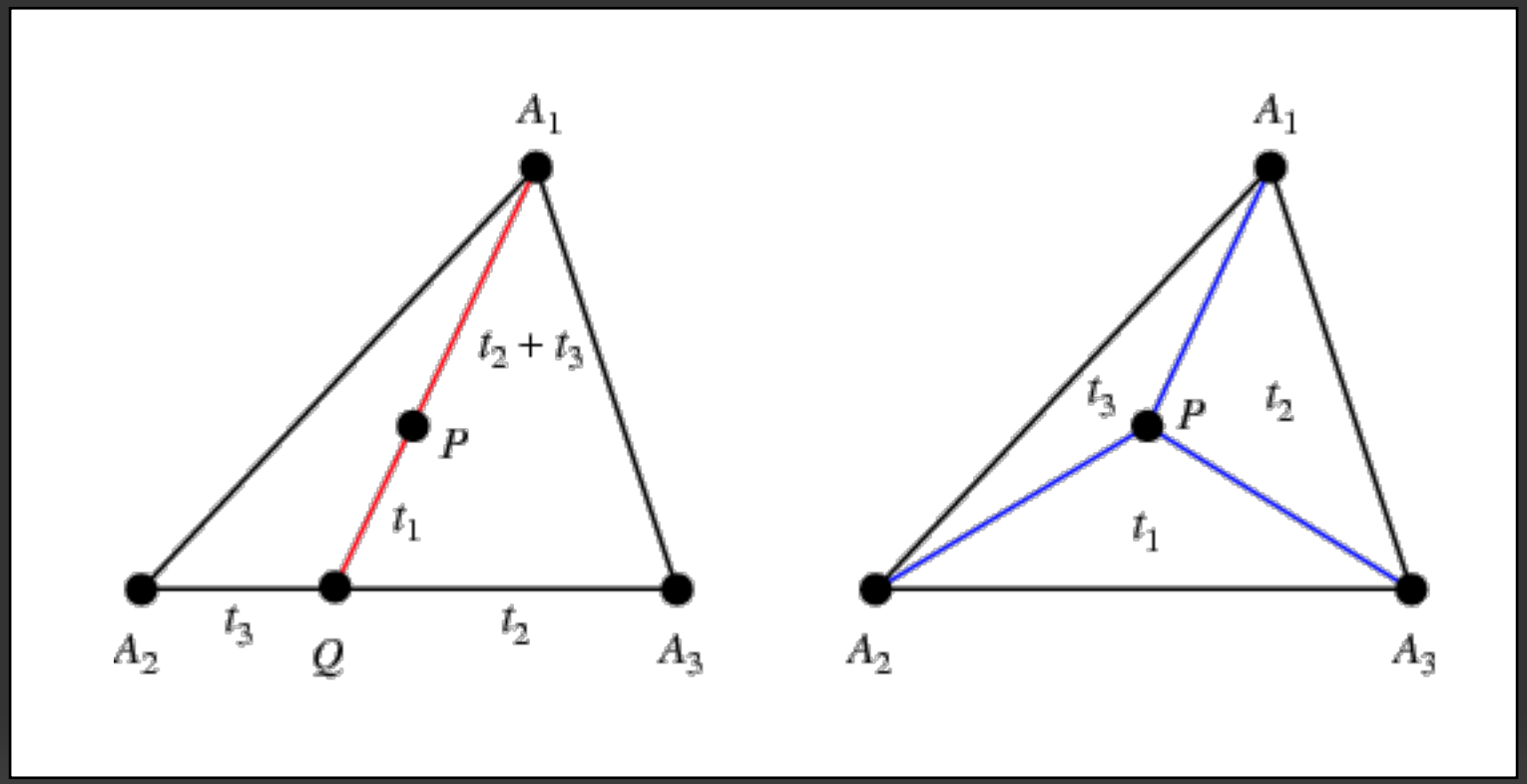
- Moller-Trumbore Ray Triangle Intersection
- Octree Space Partitioning
- Barycentric Normal Interpolation
- .obj Scene File Support
- Lambertian Diffuse BRDF
- Pure Specular (Mirror) BRDF
- Reflection/Refraction BRDF
- Explicit Emitter Sampling
- Texture Mapping

Fast Ray-Triangle Intersection

'Fast, Minimum Storage Ray-Triangle Intersection'
Moller, Trumbore
Journal Of Graphics Tools, v2n1p21, 1997.
http://www.graphics.cornell.edu/pubs/1997/MT97.pdf

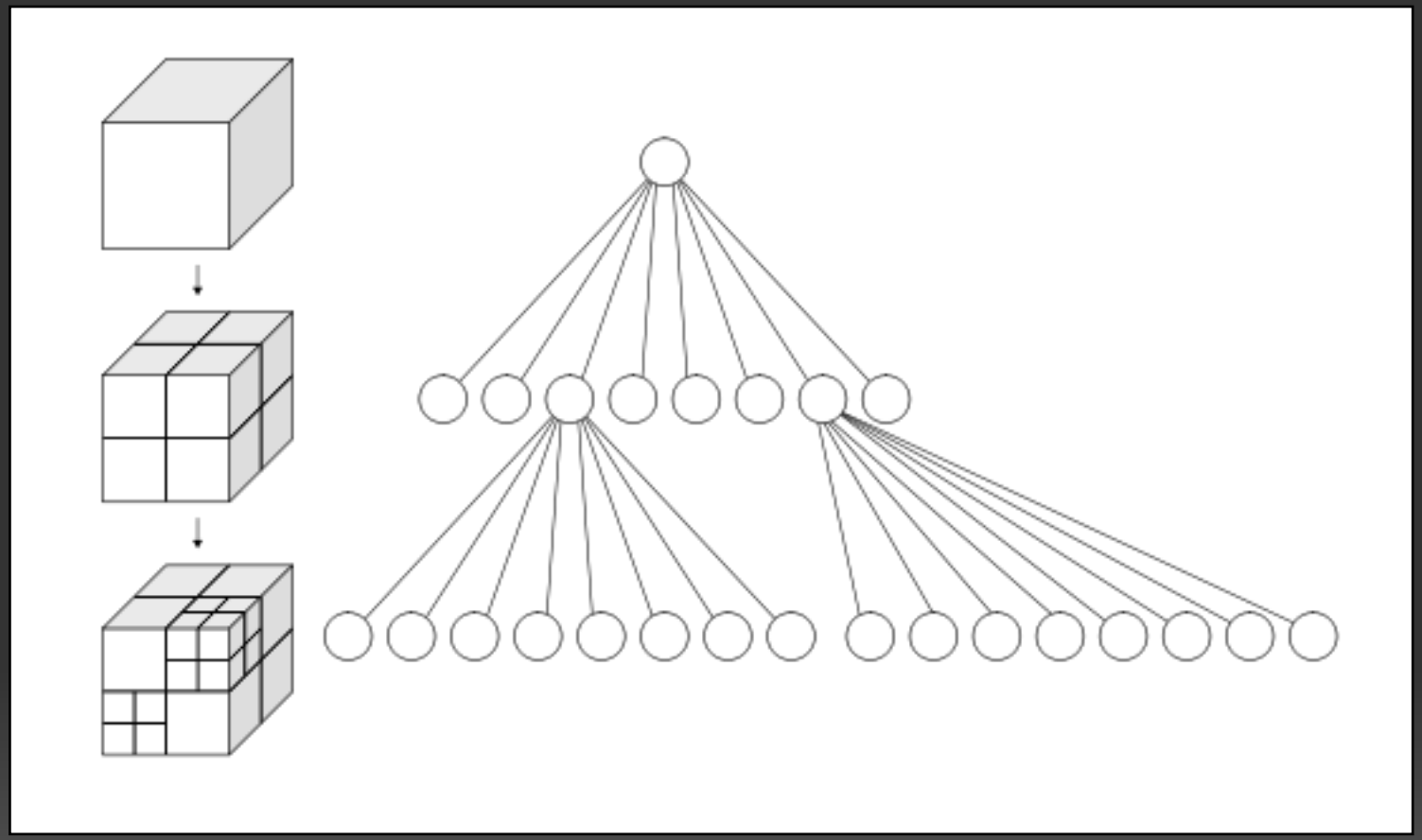


Barycentric Normals



http://mathworld.wolfram.com/BarycentricCoordinates.html

Octree Spatial Partitioning



Explicit Emitter Sampling

- Each time a ray intersects a surface, a check is done to see what, if any, light is coming from a random emitter

- The light contribution then must be weighted appropriately based on the probability distribution
- Greatly improves rendering performance by explicitly sampling the light emitting triangles in the scene (avoid blind search situation)

- Potential issues in terms of bias when working on conjunction with reflection and refraction rays

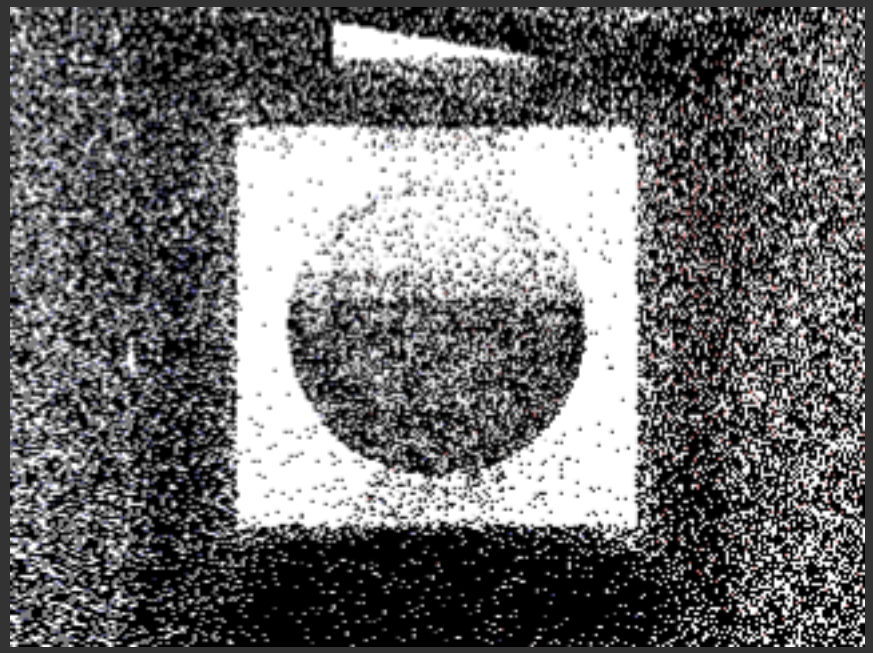
Reflection / Refraction

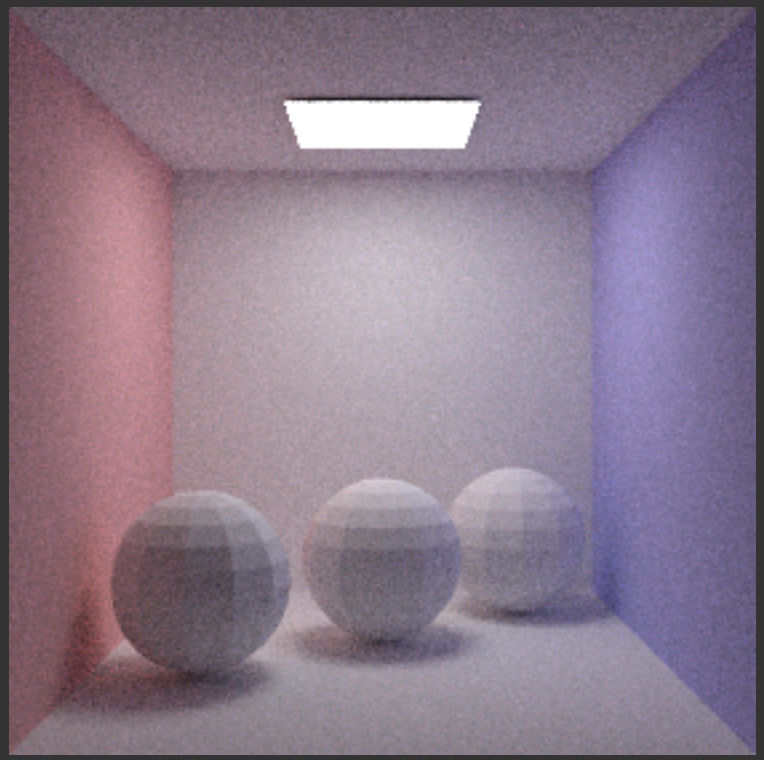
Implemented using Snell's Law(s)

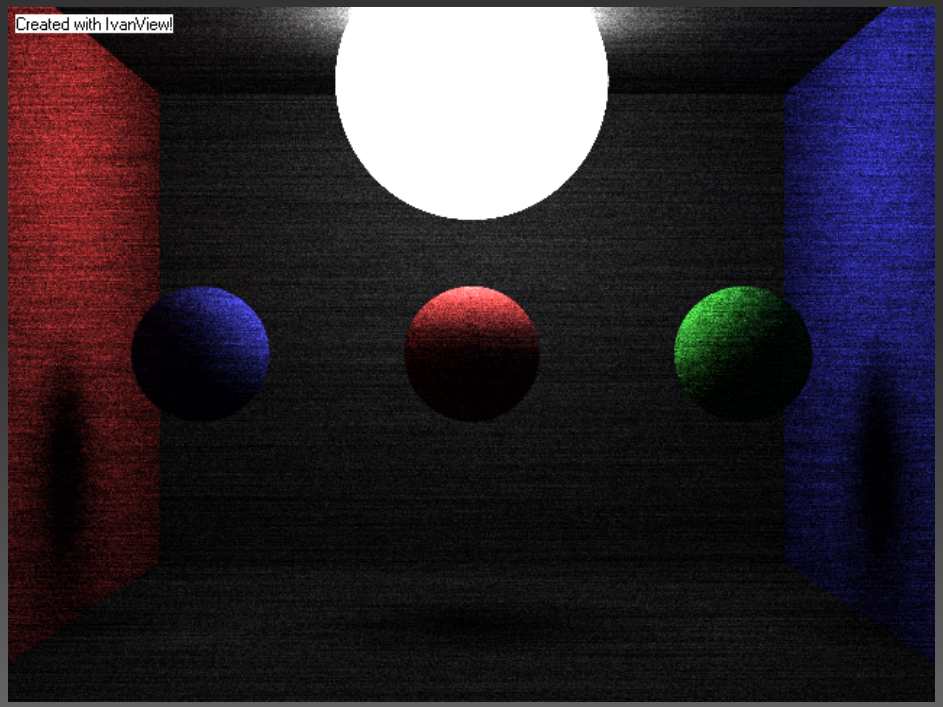
(Beer's Law, Frensel's equation future improvements)

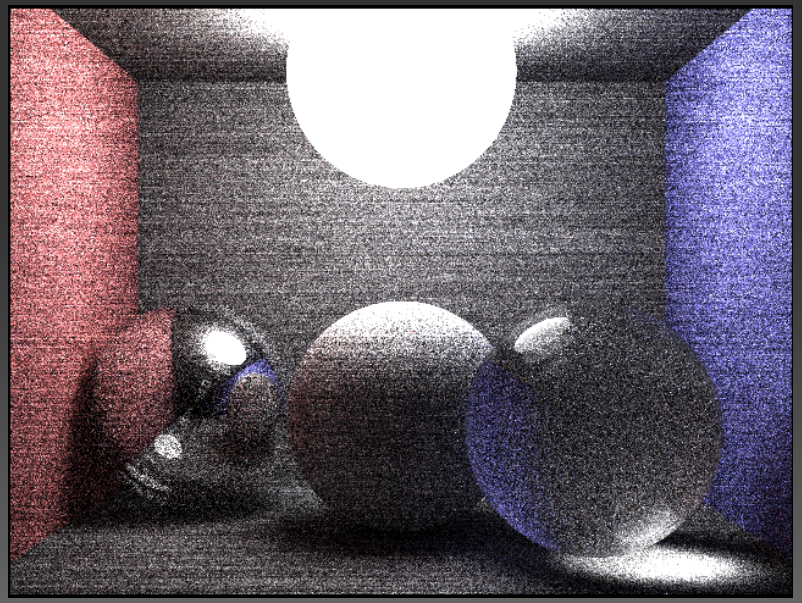
```
const float n1 = 1.0f;
const float n2 = 1.5f;

const bool into = rayDirection.dot( n ) > 0;
const float ni = into ? n2 / n1 : n1 / n2;

const float cosTheta1 = n.dot(-rayDirection);
const float cosTheta2 = 1.0f - ni * ni * ( 1.0f - cosTheta1 * cosTheta1 );

const Vector3f reflectionVector = rayDirection + n * ( 2 * cosTheta1 );

if (cosTheta2 < 0) // Total internal reflection
{
    radiance = radiance + color * getRadiance(hitPosition,
reflectionVector,           random, surfacePoint.getHitId(), true);
}

const Vector3f refractionVector = (rayDirection * ni) + n * ( (cosTheta1 * ni) +
(cosTheta1 > 0.0 ? -sqrtf(cosTheta2) : sqrtf(cosTheta2) ) );
```

Error Images...

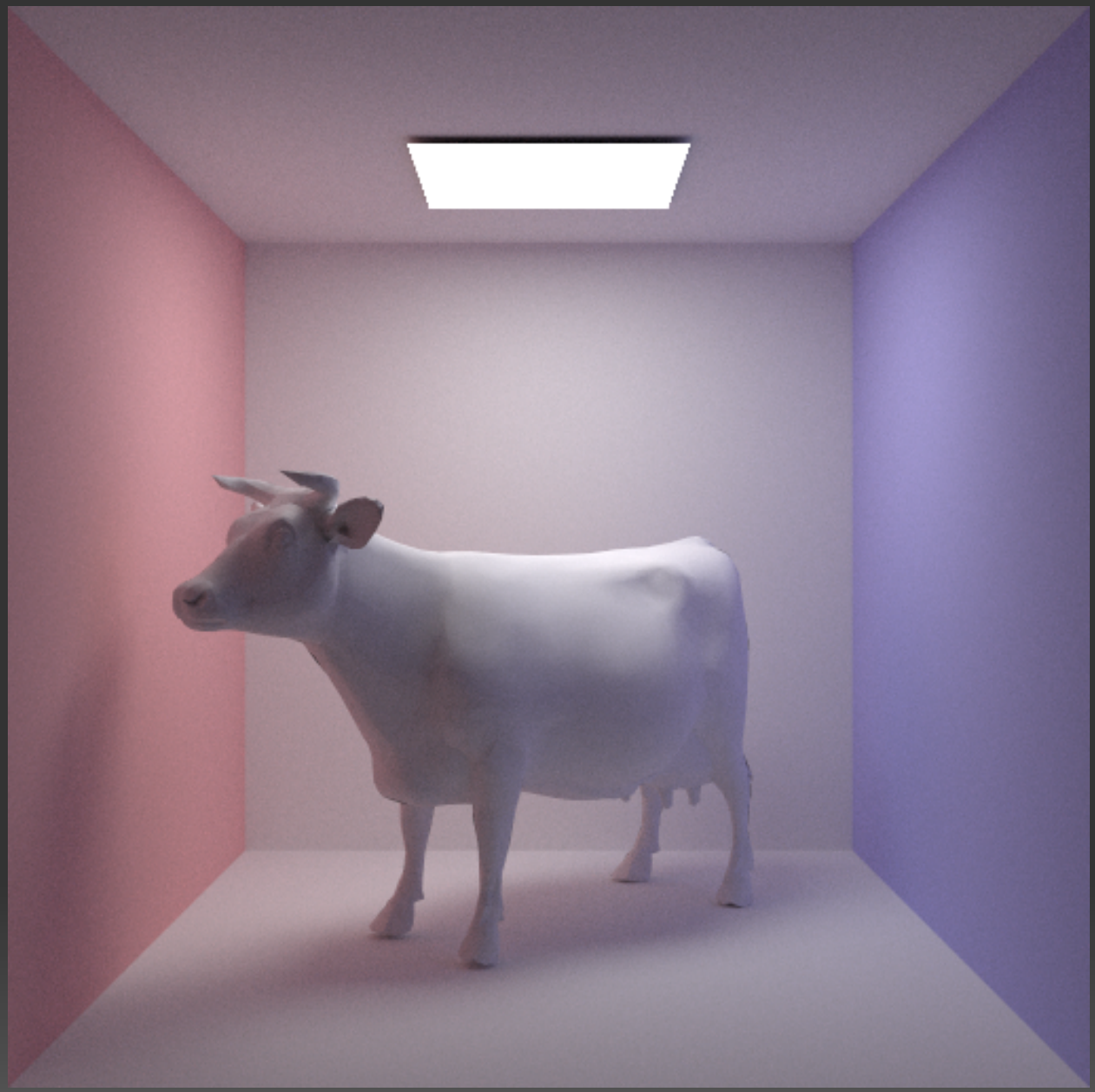





Created with IvanView!



Results...



Arbitrary (relatively) high polygon model with diffuse BRDF