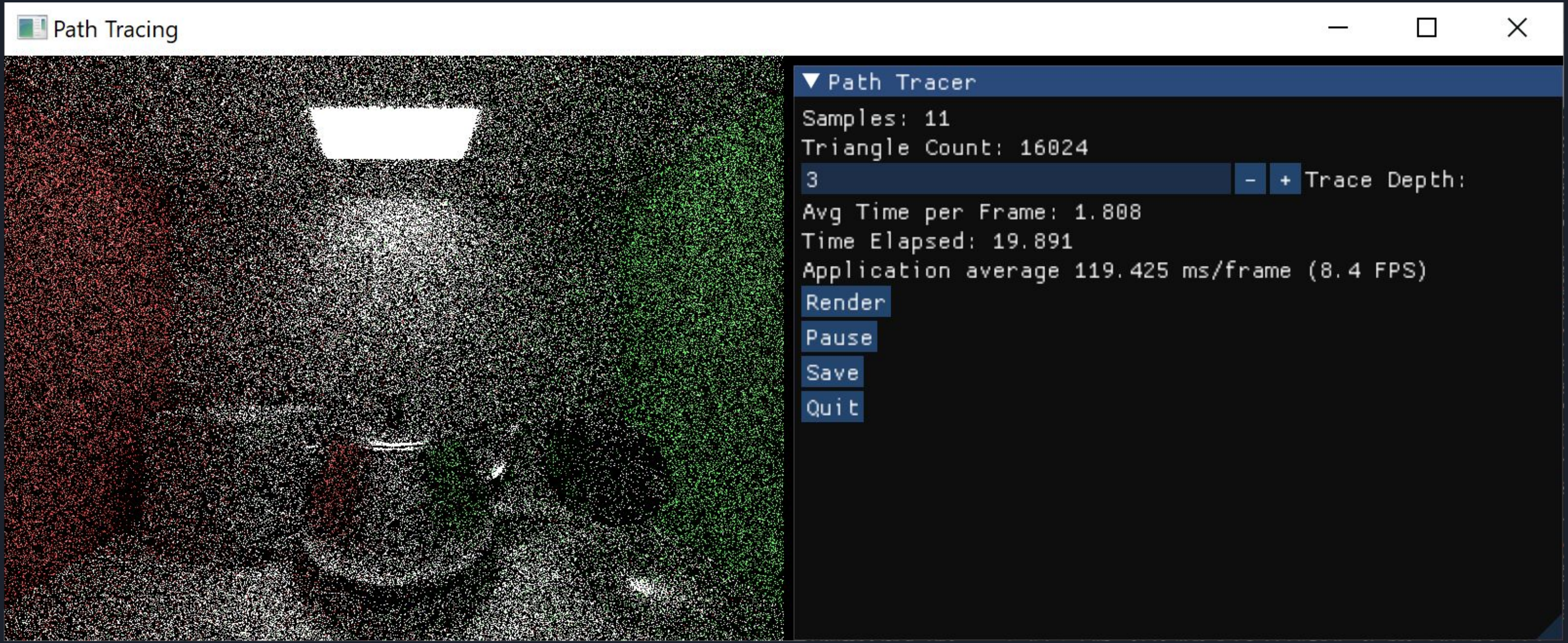
How can we Simulate Light More Realistically?

Junjie and Angel

Monte Carlo Path Tracer

Follows the path of a ray while it bounces in random direction
(more realistic depiction of how light interacts with surfaces)



Samples: 11

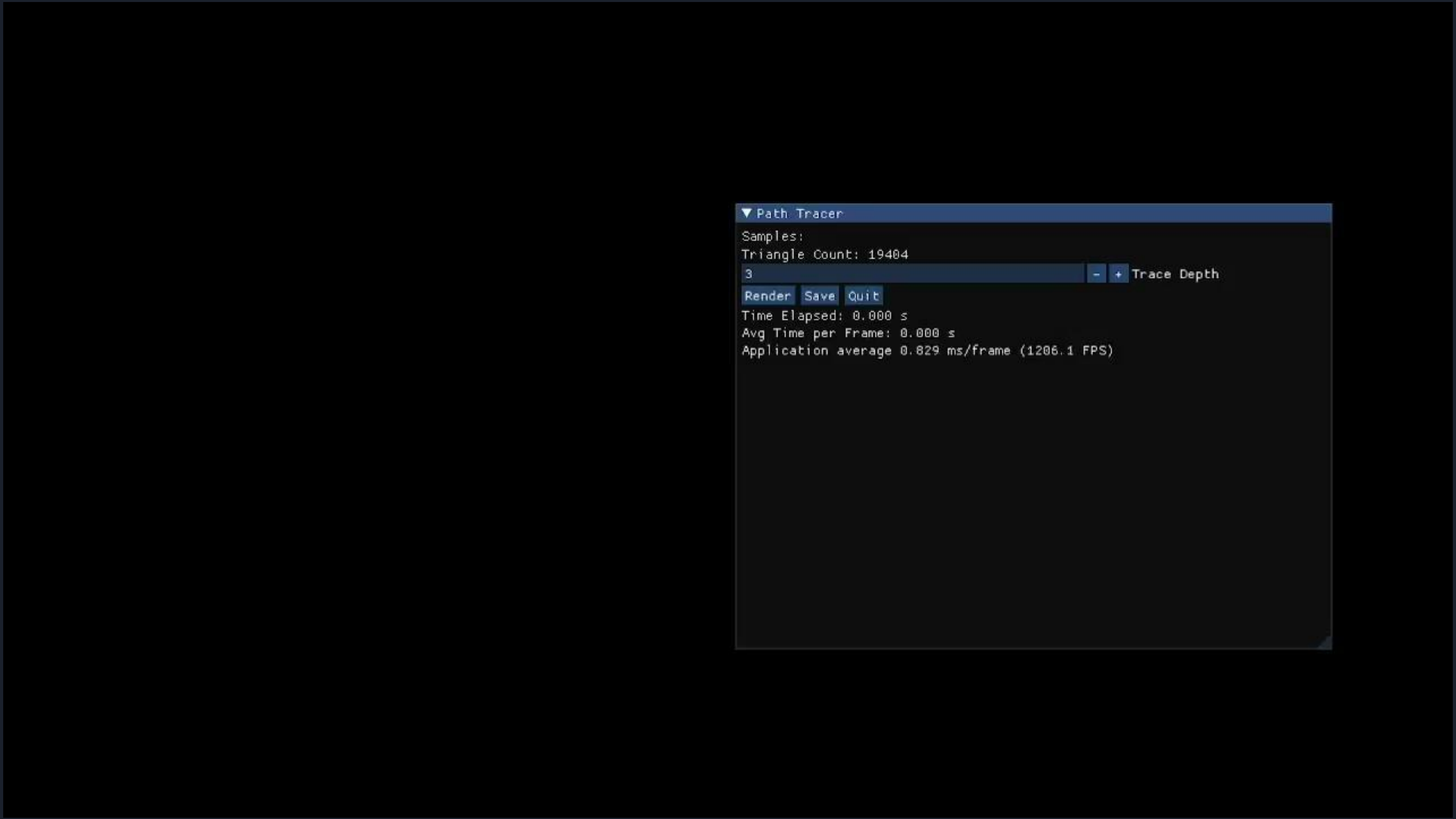
Triangle Count: 16024

3 − + Trace Depth:

Avg Time per Frame: 1.808

Time Elapsed: 19.891

Application average 119.425 ms/frame (8.4 FPS)



```
Samples:
Triangle Count: 19404
3                                   Trace Depth
Time Elapsed: 0.000 s
Avg Time per Frame: 0.000 s
Application average 0.829 ms/frame (1206.1 FPS)
```

Psuedocode

$$L_r(\omega_r) = L_e(\omega_r) + \int f_r(\omega_i, \omega_r) L_i(\omega_i) \cos\theta_i \, d\omega_i$$

```
// For every ray sample
Vec3 Trace(Ray ray, int depth)
        If (depth > MAX_DEPTH) return black;
        depth++;

        Triangle t = ray.Intersect(triangles);
        If (t.found)
                Ray newRay;
                // BRDF part: one sample is one ray direction (path)
                If (t.material == DIFFUSE)
                        newRay.dir = UniformSampling(t.normal);
                Else if (t.material == SPECULAR)
                        newRay.dir = Reflect(ray.dir, t.normal);
                Else if (t.material == GLOSSY)
                        newRay.dir = WeightedSampling(ray.dir, t.normal);

                Return t.emissive + Trace(newRay, depth) * t.color;
        Else Return background;
```

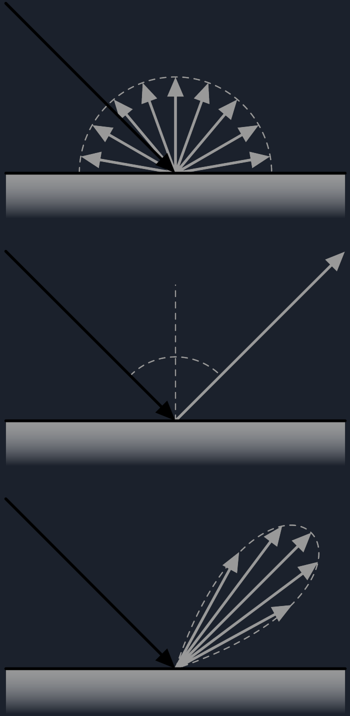


Image credit: Wikipedia

Psuedocode

$$L_r(\omega_r) = L_e(\omega_r) + \int f_r(\omega_i, \omega_r) L_i(\omega_i) \cos\theta_i \, d\omega_i$$

```
int samples = 0;
Image totalImage;
Image currentImage;
// One sample per pixel per frame
void RenderFrame()
    samples++;
    For each pixel
        Ray ray = CreateRay(camera, pixel);
        // Render current sample
        Vec3 color = Trace(ray, 0);
        // Sum up the result of every sample
        totalImage[pixel] += color;
        // Average the results
        currentImage[pixel] = totalImage / samples;
void main()
    While (true)
        RenderFrame();
        Display(currentImage);
```

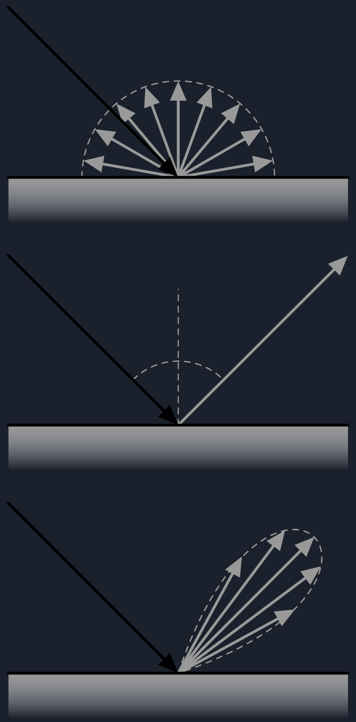


Image credit: Wikipedia

Acceleration Techniques

Bounding Volume Hierarchy (BVH)

BVH is a binary tree, which decreases the traverse (ray triangle intersection) time from O(n) to O(log(n)).

```
Class BVHNode
        BVHNode* left, right; // Children
        AABB aabb; // Stores bounding box if its an internal node
        Triangle triangle; // Stores triangle if its a leaf node

        // Construct
        BVHNode(Triangle* triangles); // Internal node
        BVHNode(Triangle t); // Leaf node

        // Traverse (ray intersection test)
        Bool Hit(Ray ray, Triangle& triangleOut, float& distantOut);
```

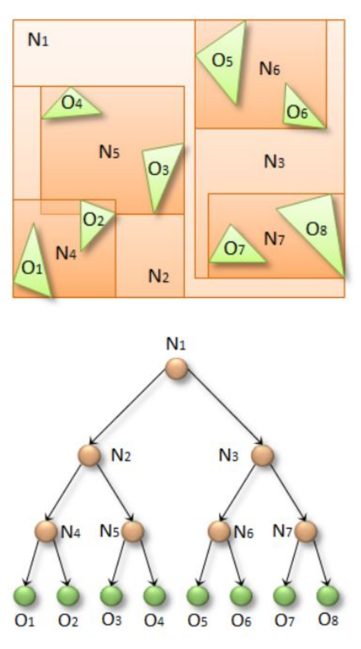


Image credit: Nvidia

Acceleration Techniques

Bounding Volume Hierarchy (BVH)

```
// 1. Construct
BVHNode::BVHNode(Triangle* triangles)
    SortByAxis(triangles, RandomAxis());
    // Insert leaf child nodes
    If  (triangles.size == 1)
        left = right = new BVHNode(triangles[0]);
    Else if (triangles.size == 2)
        left = new BVHNode(triangles[0]);
        right = new BVHNode(triangles[1]);
    // Insert internal child nodes
    Else // Split the triangle list into two for each child
        left = new BVHNode(triangles.leftHalf);
        right = new BVHNode(triangles.rightHalf);
        aabb = BoundingBox(left->aabb, right->aabb);

BVHNode::BVHNode(Triangle t)
    triangle = t;
    left = right = null;
```

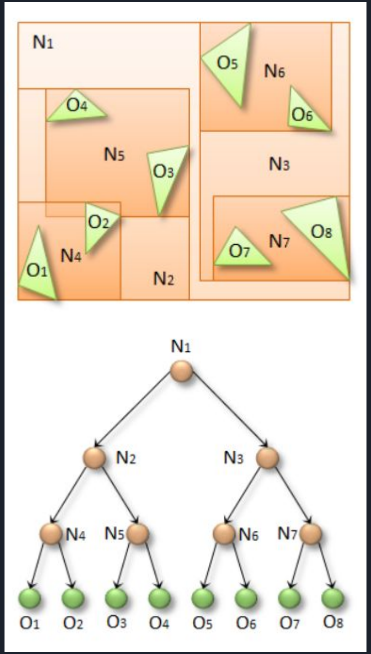


Image credit: Nvidia

Acceleration Techniques

Bounding Volume Hierarchy (BVH)

```
// 2. Traverse (ray intersection test)
Bool BVHNode::Hit(Ray ray, Triangle& tOut, float& dOut)
    If (left && right) // If it's not a leaf node, test the aabb
        If ( ! aabb.Intersect(ray)) Return false;
        Triangle tLeft, tRight; Float dLeft, dRight;  // Test children
        Bool hitLeft = left->Hit(ray, tLeft, dLeft);
        Bool hitRight = right->Hit(ray, tRight, dRight);
        If (hitLeft && hitRight) // If both are hit
            tOut = dLeft < dRight ? tLeft : tRight;
            dOut = dLeft < dRight ? dLeft : dRight;
            Return true; // We return the nearest one
        Else if (hitLeft || hitRight)
            tOut = hitLeft ? tLeft : tRight;
            dOut = hitLeft ? dLeft : dRight;
            Return true; // Else return the one we hit
        Else Return false; // Or else both are missed
    Else // If it's a leaf node, test the triangle
        Return RayTriangleTest(ray, triangle, tOut, dOut);
```

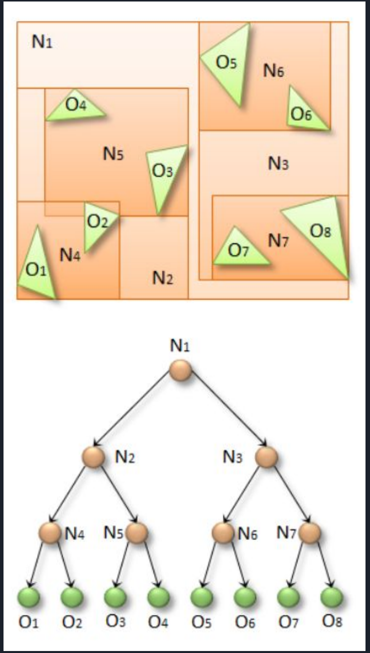
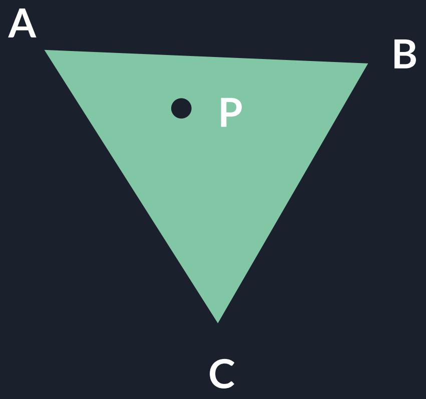


Image credit: Nvidia

Acceleration Techniques

Triangle intersection



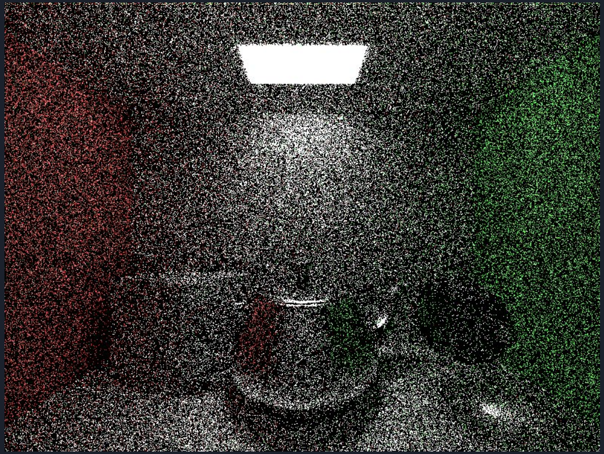
Rather than checking sum of angles with slow arctans

The point is in the triangle ABC:

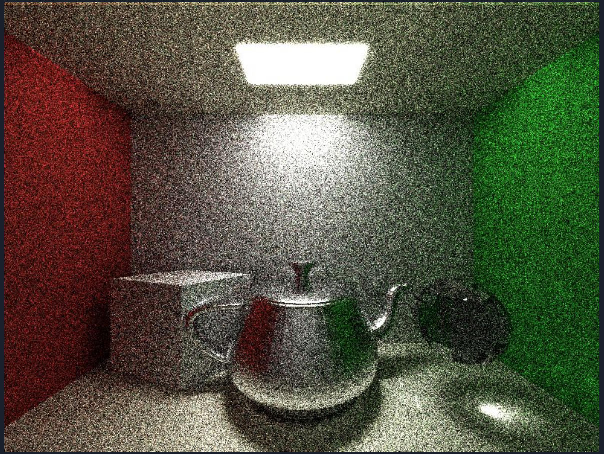
If the point is on the same side of AB as C
And
If the point is on the same side of BC as A
And
If the point is on the same side of CA as B

```
Bool isSameSide (Point p1, Point p2, Point edgeA, Point edgeB):
        glm::vec3 edge = edgeB - edgeA;
        glm::vec3 cp1 = glm::cross(edge, (p1 - edgeA));
        glm::vec3 cp2 = glm::cross(edge, (p2 - edgeA));

        return (glm::dot(cp1, cp2) >= 0);

Bool IsInside (Point p, Point a, Point b, Point c)
        return (
                IsSameSide(p, a, b, c) && IsSameSide(p, b, a, c) && IsSameSide(p, c, a, b)
        )
```

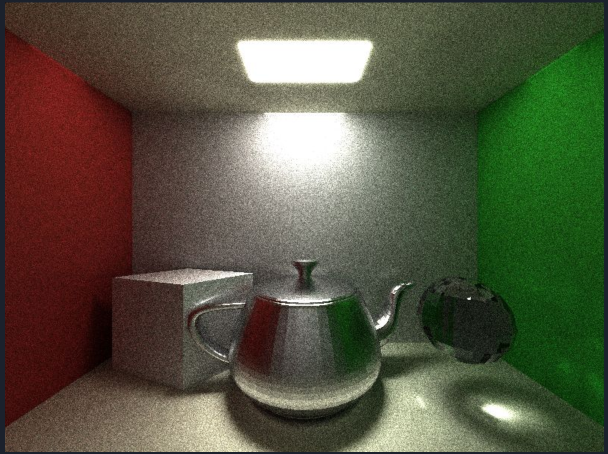
Resolution: 800 x 600
Triangle Count: 16024
Depth: 3



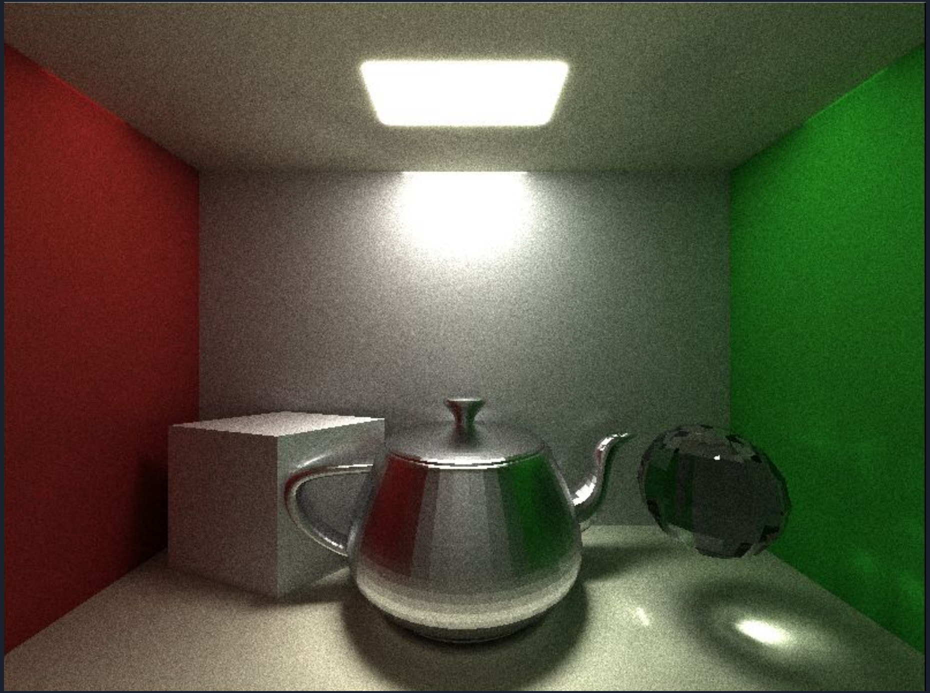
Samples: 10
Time Elapsed: 19.8893 s
Avg Time per Frame: 1.98893 s



Samples: 100
Time Elapsed: 263.466 s
Avg Time per Frame: 2.63466 s

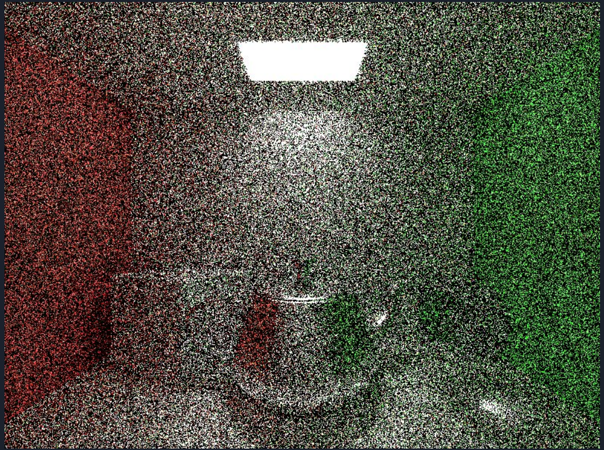


Samples: 1000
Time Elapsed: 2622.65 s
Avg Time per Frame: 2.62265 s

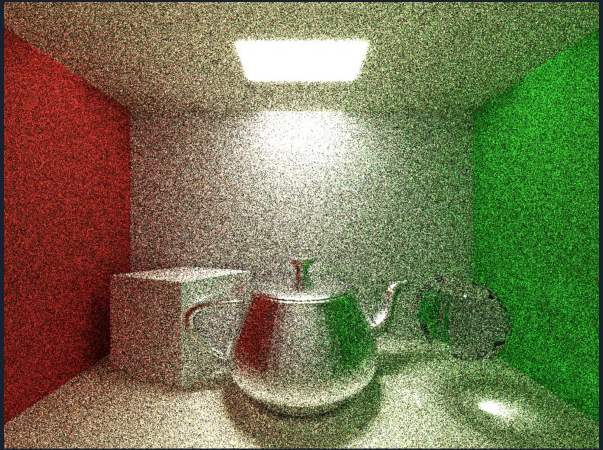


Samples: 2982
Triangle Count: 16024
Depth: 3
Time Elapsed: 7891.675 s
Avg Time per Frame: 2.646 s
Resolution: 800 x 600

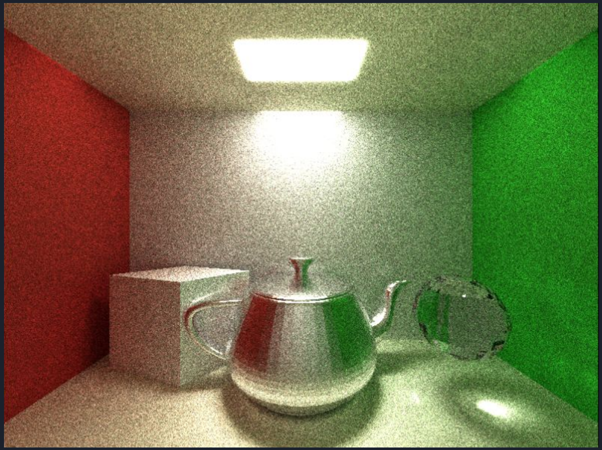
Resolution: 800 x 600
Triangle Count: 16024
Depth: 6



Samples: 10
Time Elapsed: 39.2161 s
Avg Time per Frame: 3.92161 s

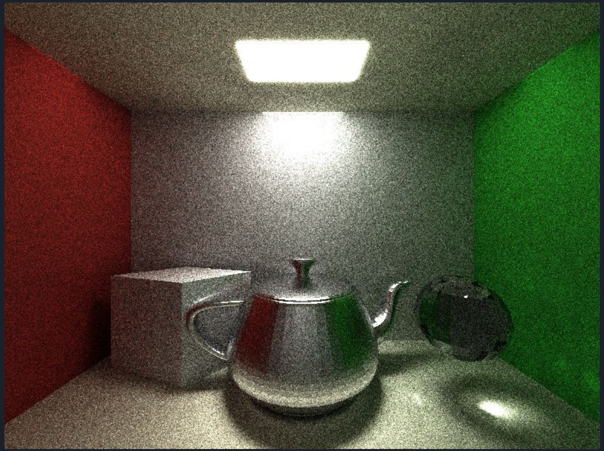


Samples: 100
Time Elapsed: 417.389 s
Avg Time per Frame: 4.17389 s

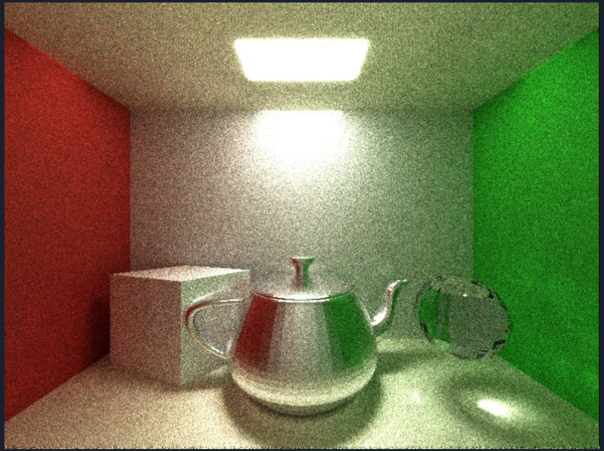


Samples: 500
Time Elapsed: 2214 s
Avg Time per Frame: 4.234 s

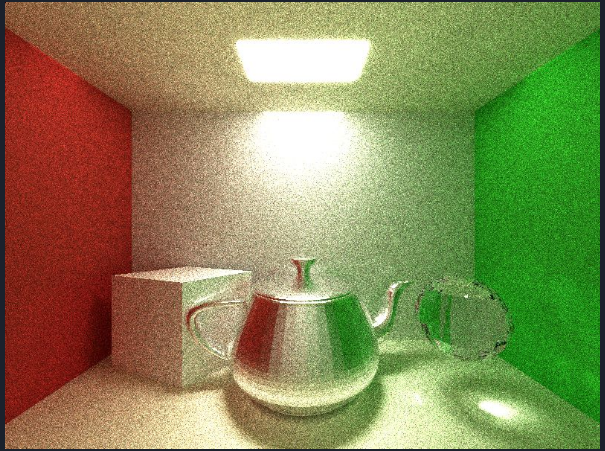
Resolution: 800 x 600
Triangle Count: 16024
Samples: 500



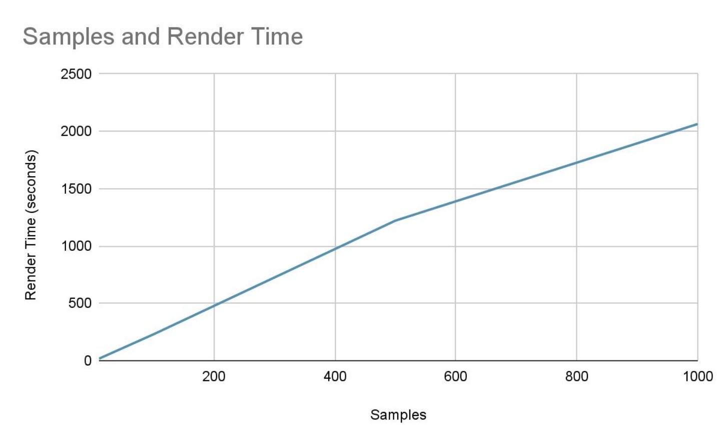
Depth: 3
Time Elapsed: 1220.88 s
Avg Time per Frame: 2.44176 s



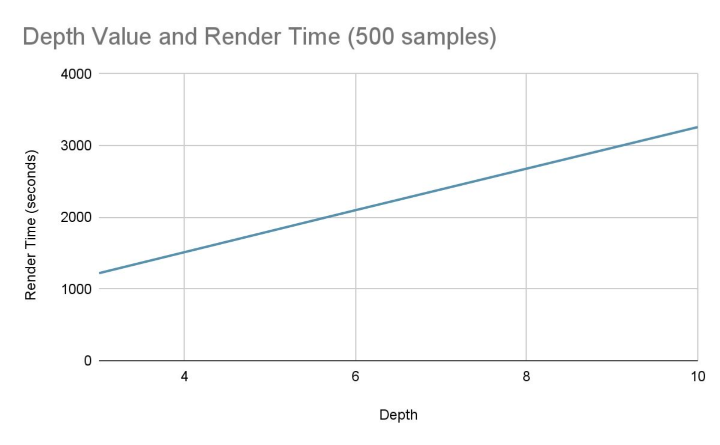
Depth: 6
Time Elapsed: 2214 s
Avg Time per Frame: 4.234 s



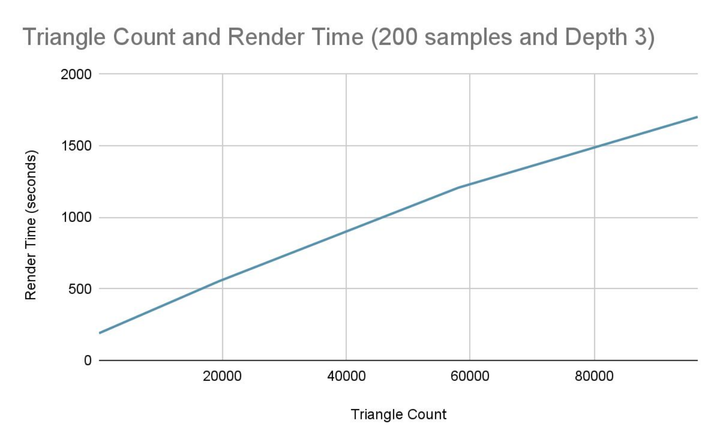
Depth: 10
Time Elapsed: 3258.49 s
Avg Time per Frame: 6.51698 s



Samples and Render Time



Depth Value and Render Time (500 samples)



Triangle Count and Render Time (200 samples and Depth 3)

Limitations and Challenges

- Took us a while to understand BRDF especially how to combine it with the Path Tracer's algorithm

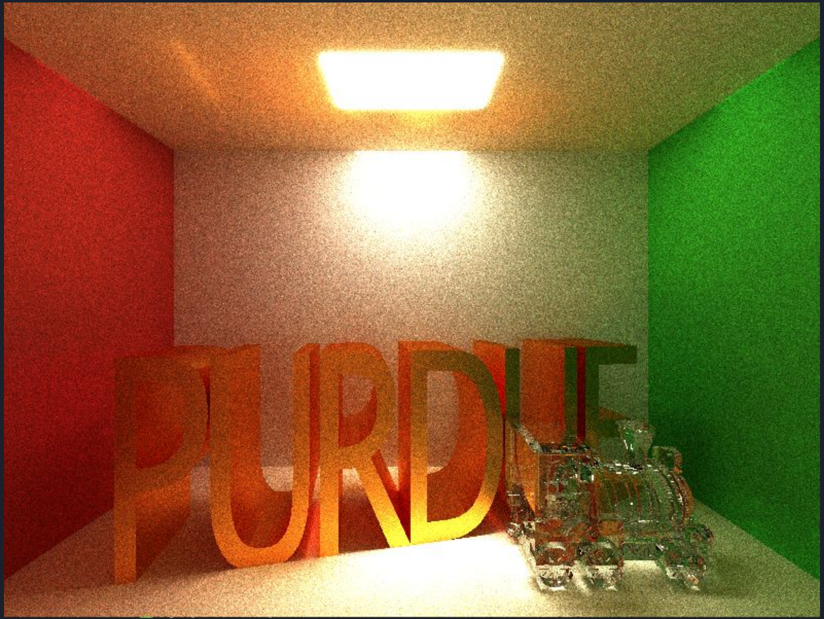
- Long time to converge
  (unlikely for a ray to hit the objects) grows exponentially

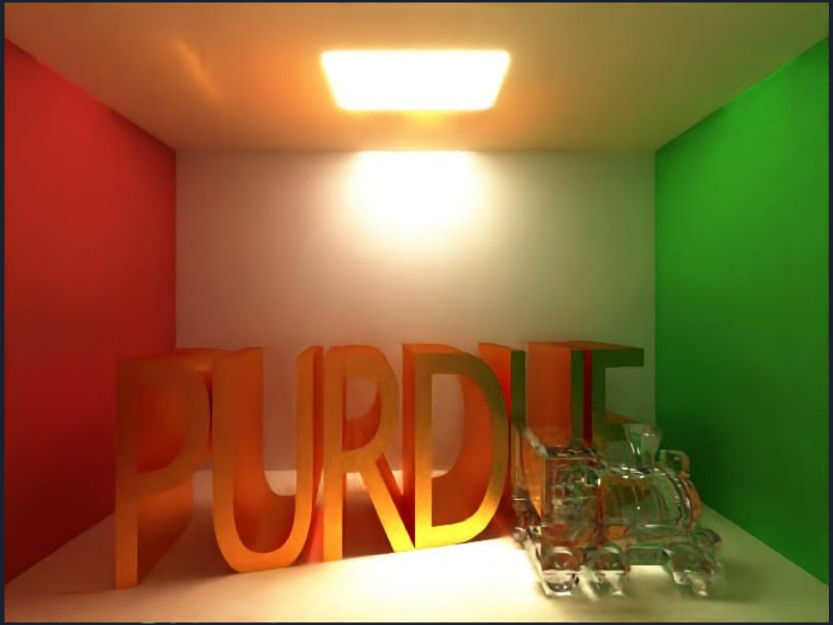
- Noisy

Future Work

- Denoiser
- Importance Sampling
- CUDA implementation

1000 Samples Denoised with an online denoiser Depth 10





Thank You