Bidirectional Path-Tracer

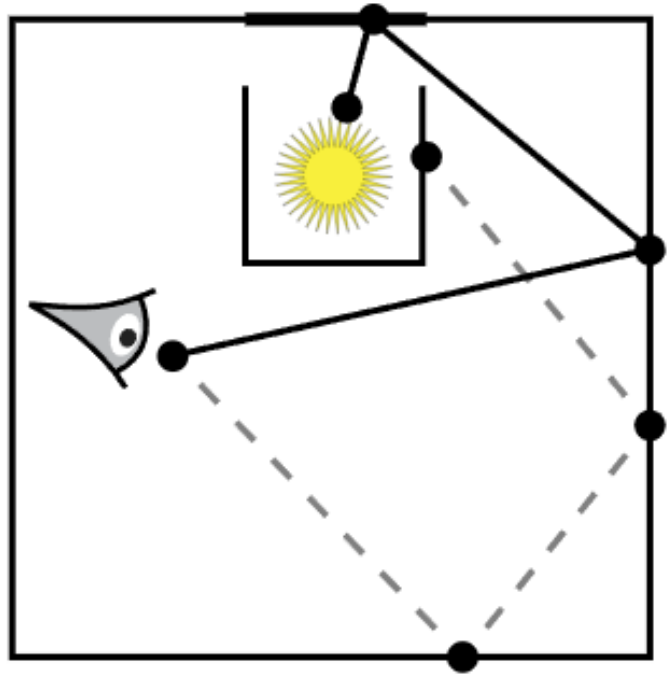
Iñigo Fernández Arenas

CS500

Problem to solve

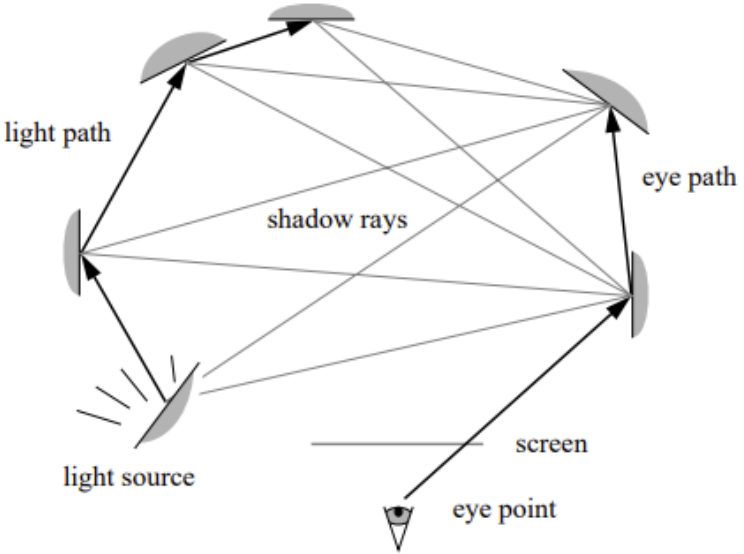
We have been using up until now a traditional path tracing where we sample each pixel and bounce rays in the scene until we reach a light. But this technique has two clear limitations, where the light is very difficult to reach.

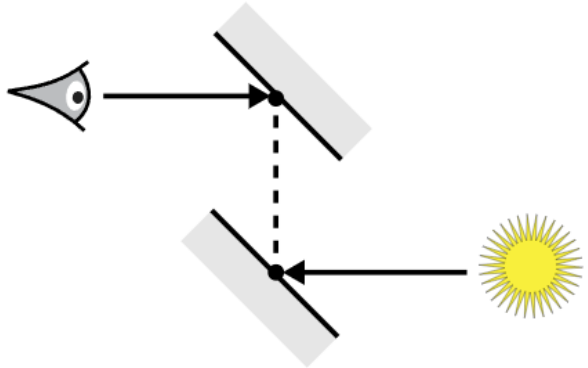
- Light is encapsulated or surrounded by objects as shown in the picture.
- The probability of reaching the light is dependent on the size of the light.



What is Bidirectional Path-Tracing?

Technique that uses two different paths in order to achieve a better solution. An eye path and a light path.



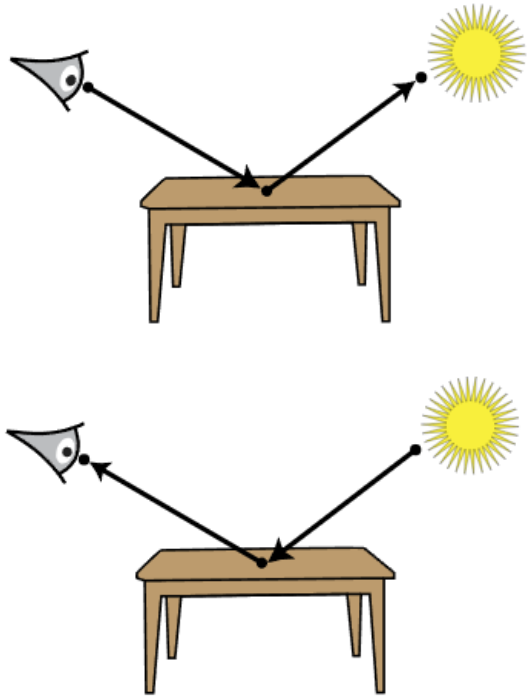


We sample at the same time both paths and connect the different vertices using shadow rays and the appropriate contributions are added to the flux of the pixel in question.

Types of Path-Traces

Eye-Path

- ✓ Sample each pixel in the screen throwing a ray through each of them.
- ✓ Bounce that ray with the scene and store each vertex it collides with.
- ✓ Stop when we reach a light or maximum of bounces.



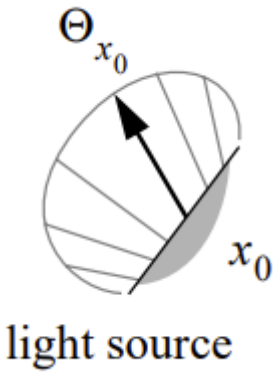
Light-Path

- ✓ Sample the light randomly throwing a ray towards the scene.
- ✓ Bounce that ray with the scene and store all the vertex it collides with.
- ✓ Stop when we reach another light or maximum bounces.

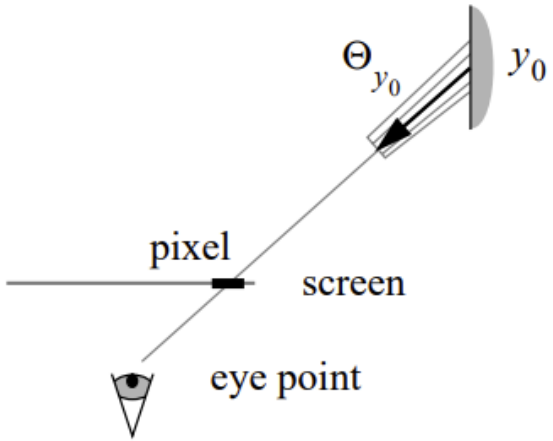
Implementation

- We use **Monte Carlo integration** to sample the full scene.
- We can devide the implementation into three major steps:

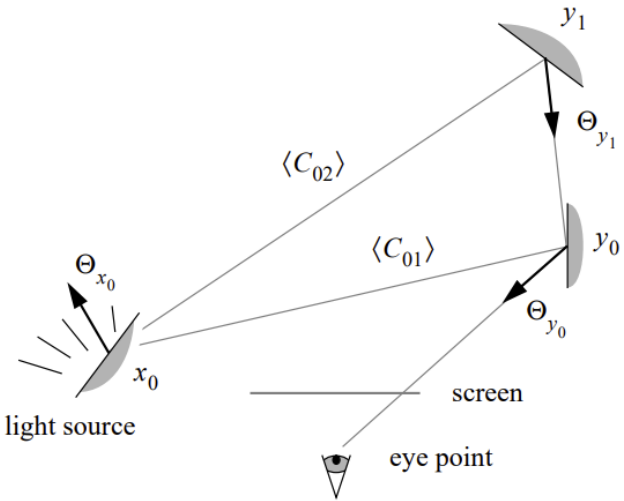
Sample light path



Sample eye path



Compute total flux from paths



Compute Eye Path

- Sample each pixel in the screen.
- Bounce in the scene based on the different materials and surfaces encountered.
- Every time we collide with a diffuse object we store the vertex.
    - Metallic vertices will be stored with a weight factor based on the roughness.
    - Dielectric vertices will be stored in case of total reflection, or transmittance going outside of the dielectric object.

```
//check if a light was reached
if (obj->mbLight)
{
    vtx.color = obj->color * prev_color;
    path.push_back(vtx);
    return true;
}
else
{
    //get bounced vector from the surface
    ray bounced_ray;
    glm::vec3 color = glm::vec3(0.0f);
    if (!obj->mMaterial->GetBounceRay(bounced_ray, color, stats))
        return false;

    //set vtx position
    vtx.pos = bounced_ray.origin;
    vtx.weight = stats.weight;

    //keep the recursive until we run out of bounces or reach a light
    m_ray = bounced_ray;
    prev_obj_collided = obj;
    vtx.color = color * prev_color;

    //check if we are not in a dielectric or if we are exiting it
    if(stats.MatType != MaterialType::dielectric || vtx.weight == 1.0f)
        path.push_back(vtx);
}
```

Compute Light Path

- Sample each light uniformaly.
- Bounce in the scene following the same criteria as the eye path.
- Every time we collide with a surface we store the vertex, in theory.
    - Metallic vertices will be stored with a weight factor based on the roughness.
    - Dielectric vertices will be stored in case of transmittance.

```
//check if a light was reached
if (obj->mbLight)
{
    vtx.color = obj->color * prev_color;
    path.push_back(vtx);
    return false;
}
else
{
    //get bounced vector from the surface
    ray bounced_ray;
    glm::vec3 color = glm::vec3(0.0f);
    if (!obj->mMaterial->GetBounceRay(bounced_ray, color, stats))
        return false;

    //update vertex
    vtx.pos = bounced_ray.origin;
    vtx.color = color * prev_color;
    vtx.weight = stats.weight;

    //update next ray
    m_ray = bounced_ray;
    prev_obj_collided = obj;
    prev_color *= color;

    //check if it was a valid vertex
    if (stats.weight != 0.0f)
        path.push_back(vtx);
}
```

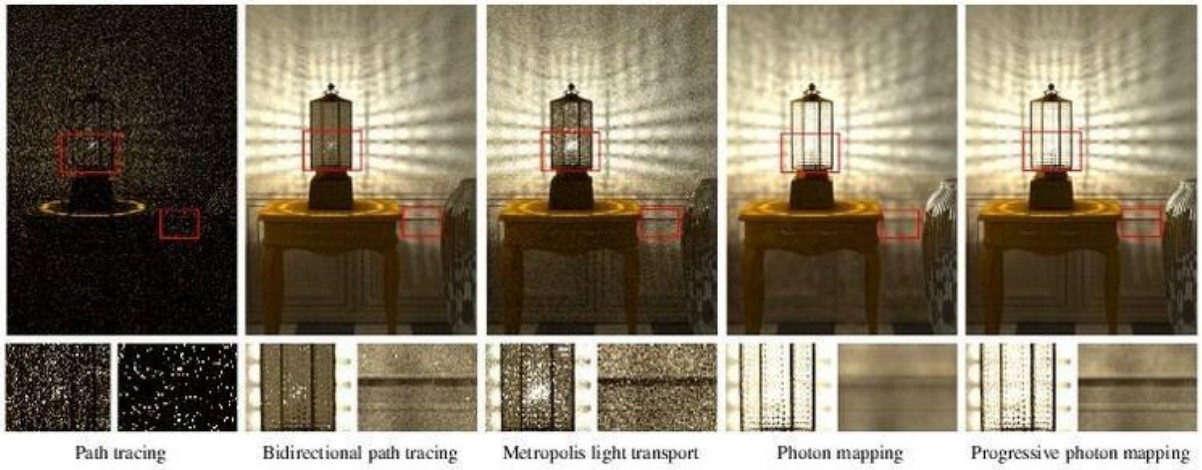
Compute flux for pixel sampled

- Iterate through each vertex on the eye path connecting the vértices on the light path.
- For each connection apply the corresponing contributions.
- Apply another weight based on the angle between the shadow ray and the normal of the light vertex.

```
//try all the different combination between
for (unsigned i = 0; i < eye_path.size(); i++)
{
    //get eye vtx
    glm::vec3 temp_color = glm::vec3(0.0f);
    PathVtx eye_vtx = eye_path[i];

    for (unsigned j = 0; j < light_path.size(); j++)
    {
        //get light vtx
        PathVtx light_vtx = light_path[j];

        //check shadow ray intersection
        intersection_stats stats;
        ray m_ray = { eye_vtx.pos, glm::normalize(light_vtx.pos - eye_vtx.pos) };
        model* obj = CheckRay(m_ray, stats);
        glm::vec3 check_pos = m_ray.at(stats.t);
        if (!obj) continue;

        //check for dielectric objects
        glm::vec3 extra_color = glm::vec3(1.0f);
        if (!obj->mbLight && obj->mMaterial->mType == MaterialType::dielectric){ ... }

        //weight value for the light intensity
        float angle = glm::clamp(glm::dot(eye_vtx.normal, m_ray.dir), 0.0f, 1.0f);

        //check if the shadow ray test was positive
        if ((obj->mbLight) || glm::distance(light_vtx.pos, check_pos) <= EPSILON)
            temp_color += (eye_vtx.color * light_vtx.color * extra_color) * angle * light_vtx.weight * eye_vtx.weight;
    }
    //update total color
    color += temp_color;
}

//average the resulting contribution
color /= static_cast<float>(light_path.size() * eye_path.size()) + valid_paths;
return color;
```

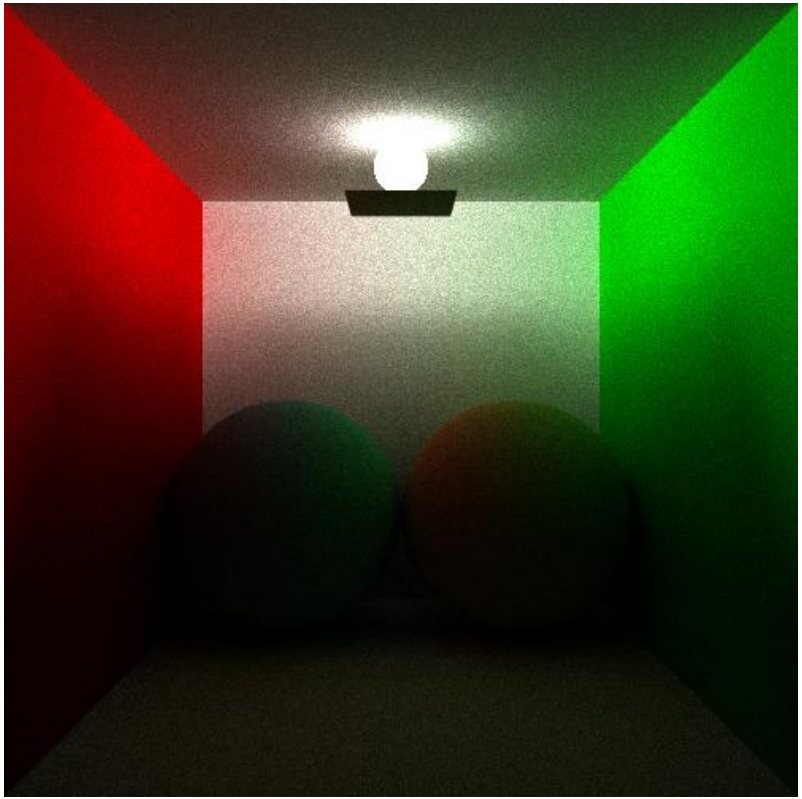
Conclusions

- There are different aproaches that also solve the same problem, such as **Photon Mapping** and **Metropolis Light Transport**.
- It works very good for small lights and lights that are difficult to reach.
- It isn't as good as photon mapping for caustics.
- It is difficult to handle shadow rays with dielectric objects in the middle.
- It doesn’t work for infinite area lights or environment maps.



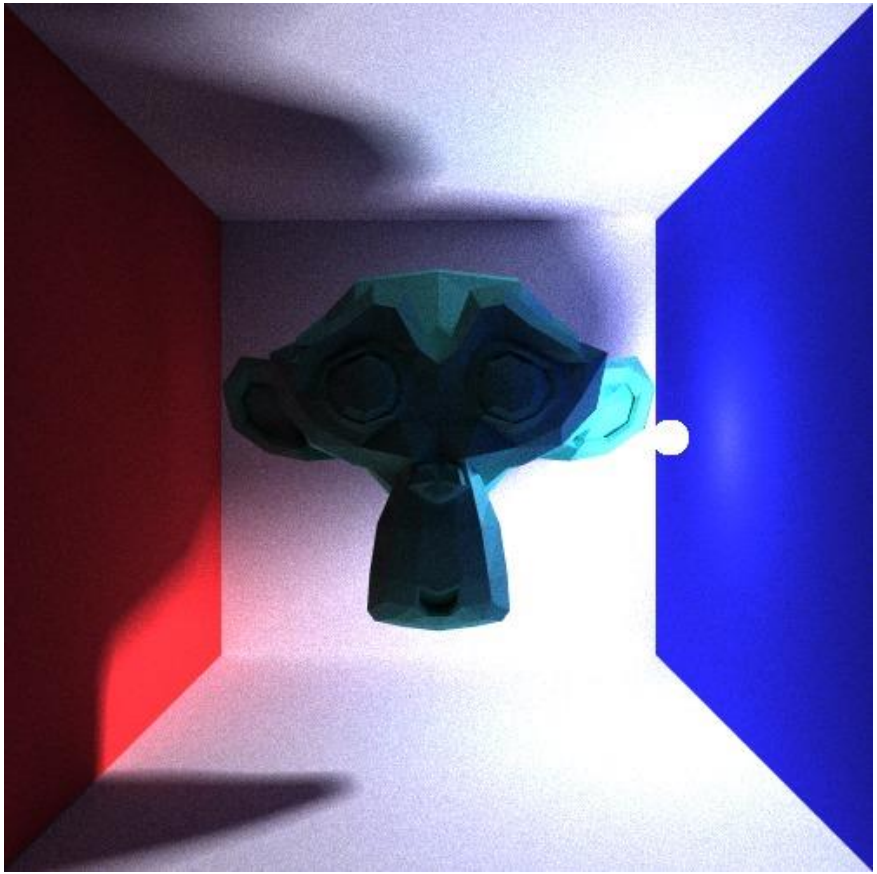
Results

Bidirectional Path Tracing (200 samples)

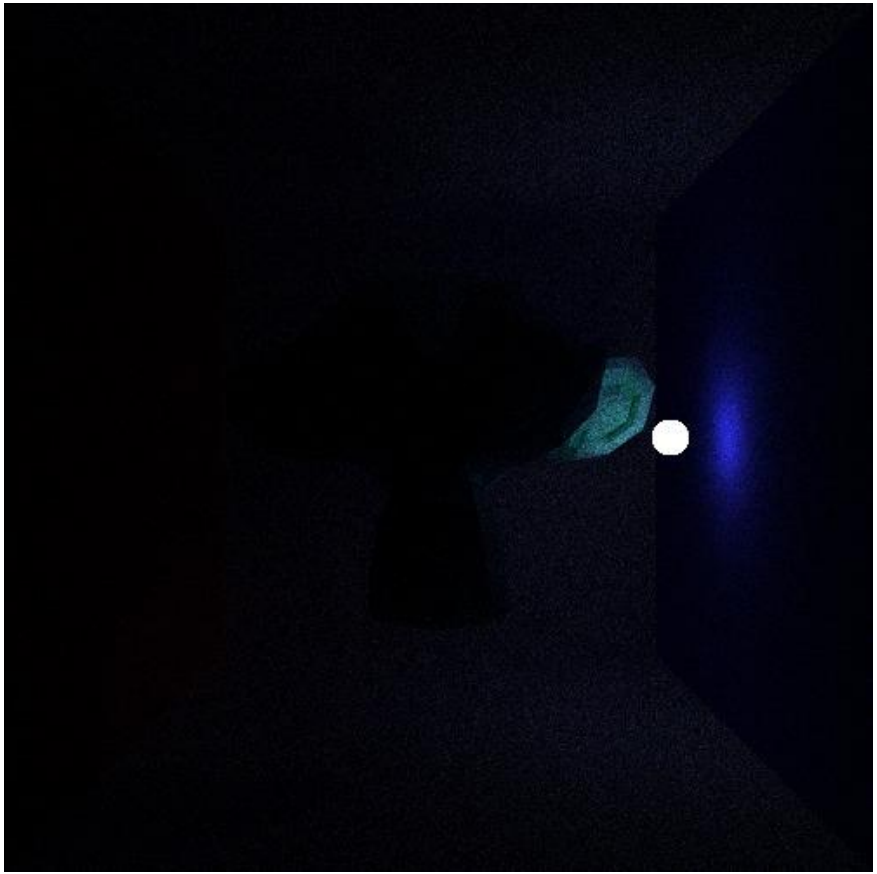


Traditional Path Tracing (200 samples)

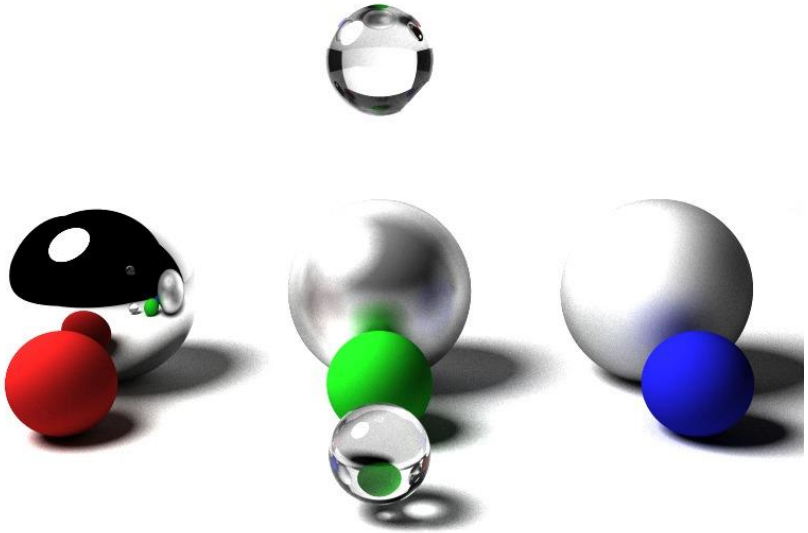
Bidirectional Path Tracing (500 samples)



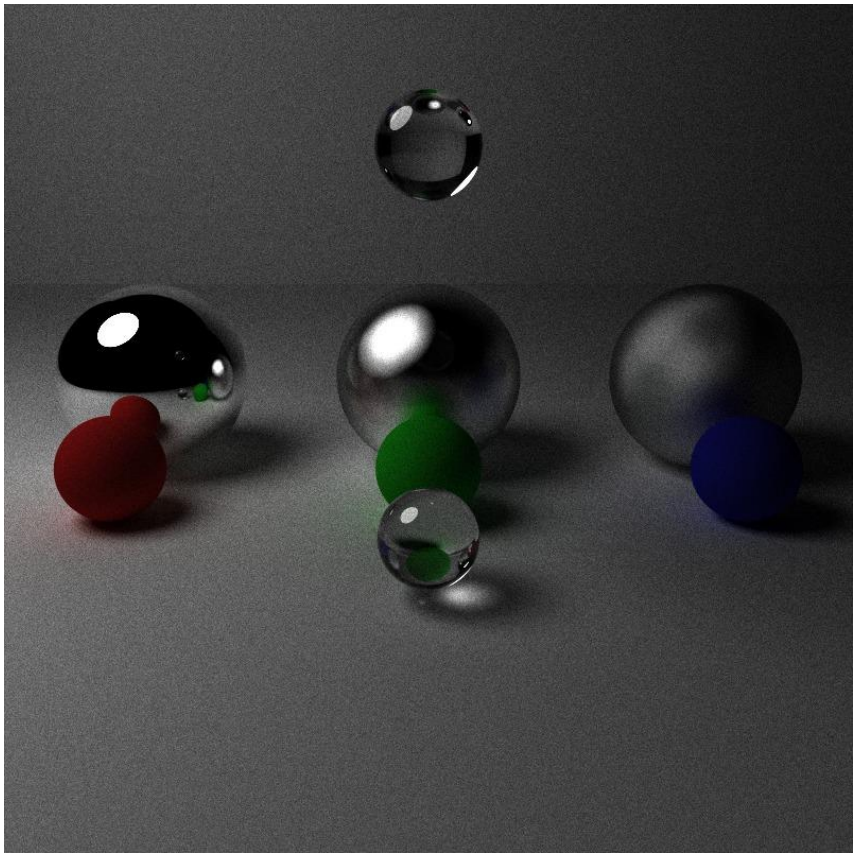
Traditional Path Tracing (500 samples)



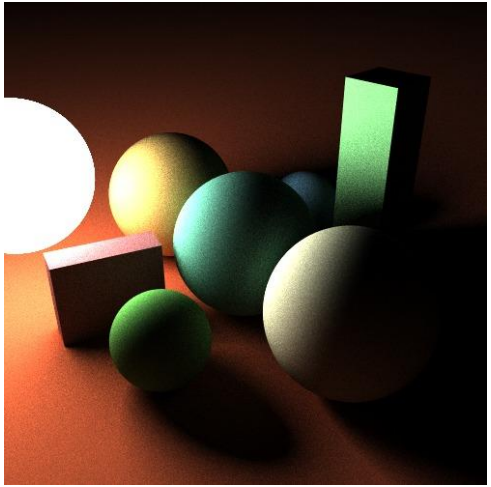
Bidirectional Path Tracing (1000 samples)

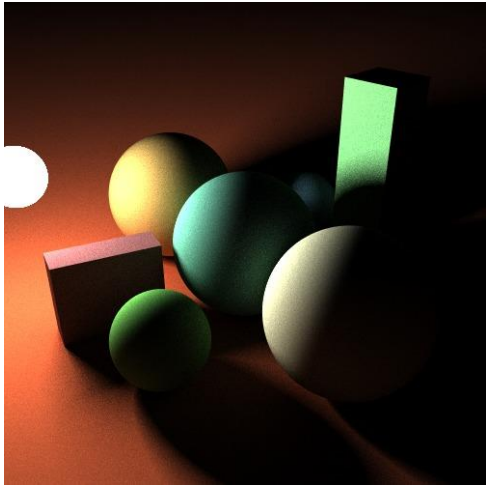


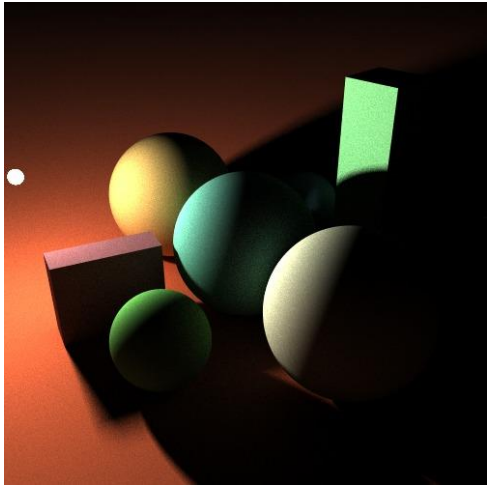
Traditional Path Tracing (1000 samples)



Bidirectional
Path Tracing
(200 samples)







Traditional
Path Tracing
(200 samples)