GLSL-BASED RAY TRACING

PROGRAMAÇÃO 3D

MEIC/IST

Motivation

- Ray Tracing: “embarrassingly parallel” problem
- Independent pixel processing
- Suitable for GPU
- Take advantage of the last programming stage of the GPU rasterization pipeline: pixel (fragment) shader that just outputs a colour
- Fragment program is evaluated at every pixel.
- Assignment 2: to program a Progressive Ray Tracer shader with GLSL by using a Shadertoy-based tool

Tools - Shadertoy.com

- https://www.shadertoy.com/ - website that lets you write small programs (shaders) in a c-like language called GLSL which are ran for every pixel in an image, on your GPU (not CPU).
- Shadertoy uses WebGL but manages everything except the pixel shader(s) for you.
- Limited input: pixel coordinate, frame number, time, and mouse position

Tools: Desktop platform

- Visual Code + Shadertoy plugin:
  - https://code.visualstudio.com/download
  - https://marketplace.visualstudio.com/items?itemName=stevensona.shader-toy – read it carefully
- VS Code: Use the "Show GLSL Preview" command
- Minor differences from Shadertoy.com. Shader entry point:
  - Shadertoy.com: void mainImage(out vec4 fragColor, in vec2 fragCoord)
  - VS Code: void main(); and you use gl_FragCoord and gl_FragColor
- The Course will provide a GLSL template, in VS Code, to the Assignment

GLSL vs C++

- No object-oriented
- No recursion
- already knows vectors and matrices; data types and maths: vec2, mat4, length(), cross(), dot(), etc.
- already knows 3D graphics operations reflect(), refract(), faceforward()
- provides many goodies like clamp, mix (linear interpolation), smoothstep (Hermite weighting+clamp)
- Warning:local memory and number of registers is an ultra-critical resource on GPU. In particular, arrays eat a lot of resources. Use them when they are *really* the only solution (or in small harmless cases).

Assignment 2 – GLSL Progressive Path Tracer

- This Course : programming the last stage of the GPU rasterization pipeline: pixel (fragment) shader that just outputs a colour
- Fragment program is evaluated at every pixel.
- State-of-the-art: RTX technology – Hardware Ray Tracing Pipeline which will be discussed later
- GLSL template in VS Code available

Casual Shadertoy Path Tracing

- Suggested Posts in this series:
  - Basic Camera, Diffuse, Emissive: https://blog.demofox.org/2020/05/25/casual-shadertoy-path-tracing-1-basic-camera-diffuse-emissive/)
  - Image Improvement and Glossy Reflections: https://blog.demofox.org/2020/06/06/casual-shadertoy-path-tracing-2-image-improvement-and-glossy-reflections/
  - Fresnel, Rough Refraction & Absorption, Orbit Camera: https://blog.demofox.org/2020/06/14/casual-shadertoy-path-tracing-3-fresnel-rough-refraction-absorption-orbit-camera/

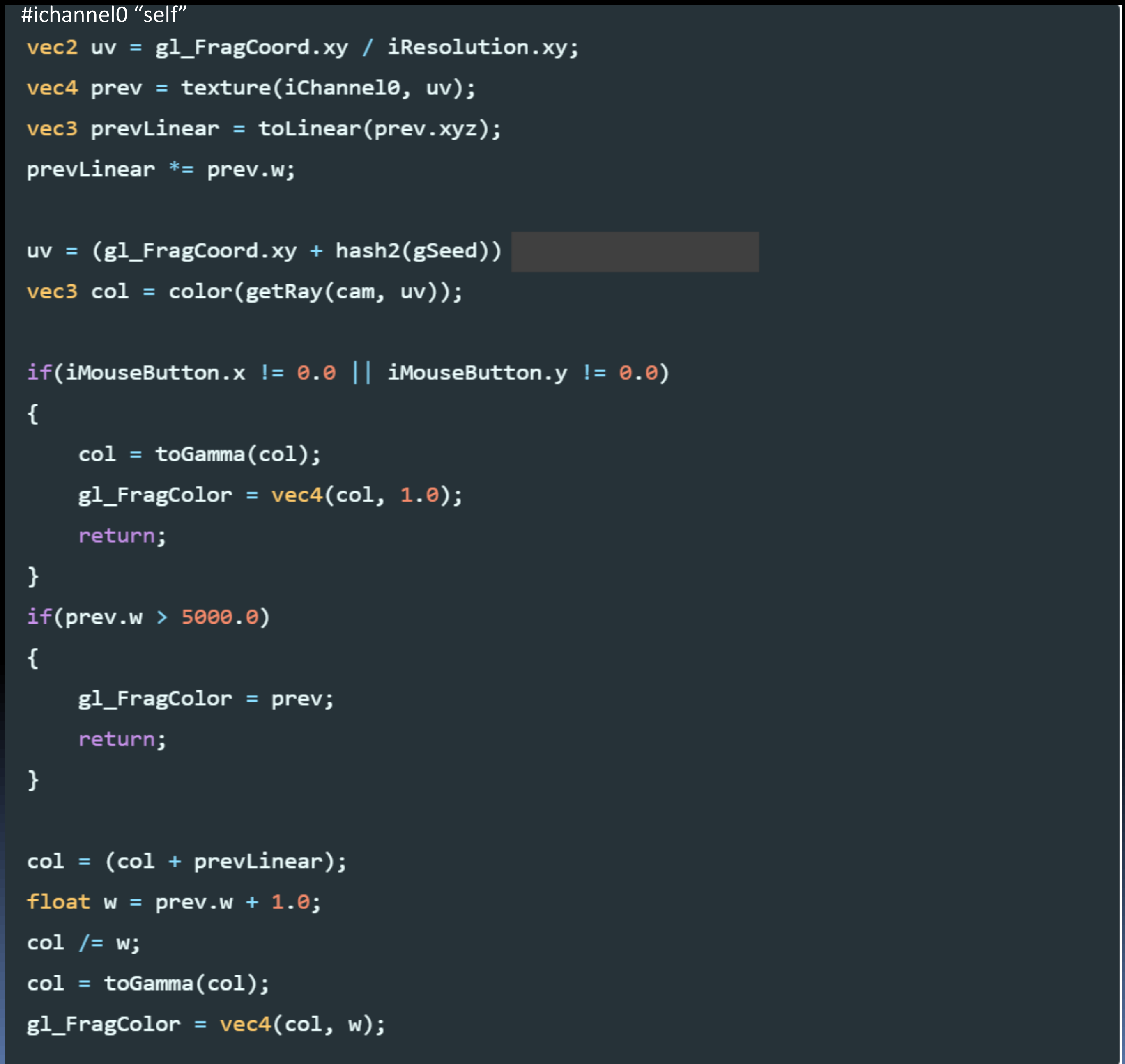
Utilities for Random Scattering

- Several implementations of GPU-based pseudo-random numbers
- common.h file: hash functions from Nimitz (https://www.shadertoy.com/view/Xt3cDn)
- Other useful functions: random_in_unit_disk and random_in_unit_sphere

Progressive Ray Tracing

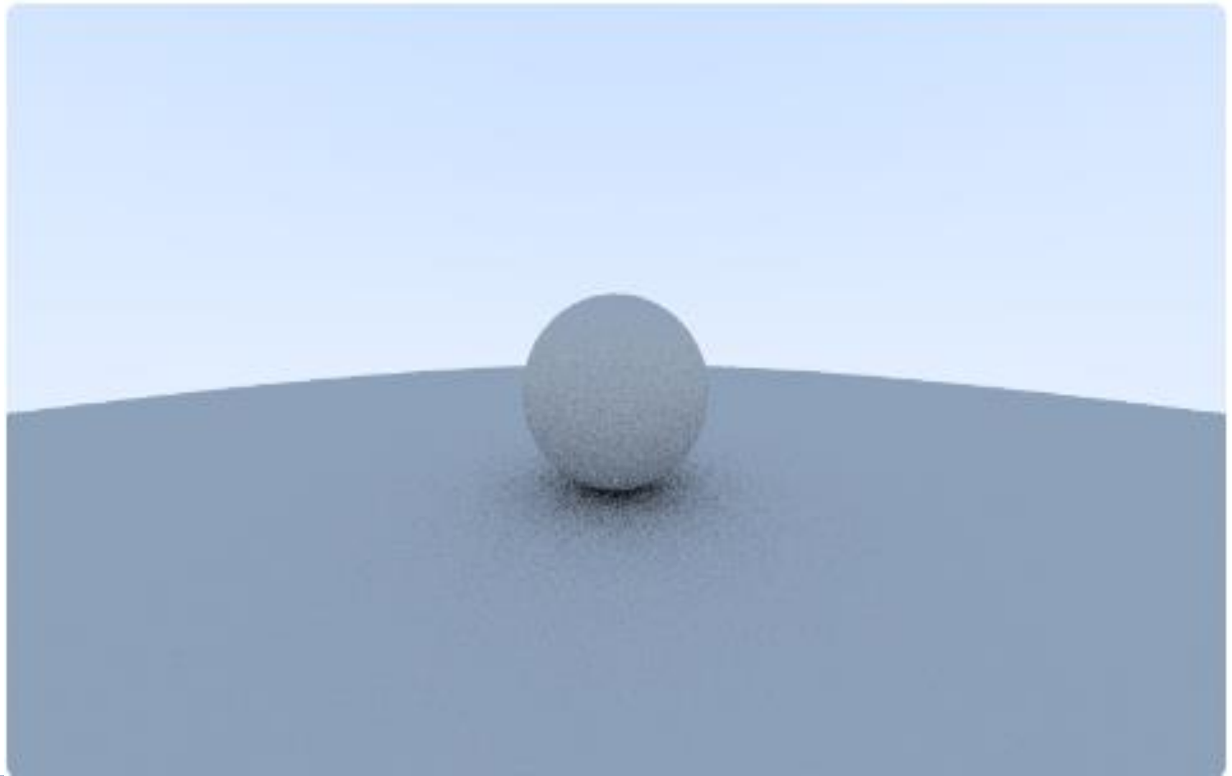
- Before: in each frame, multiple samples per pixel were taken and averaged
- Now: previous results are fed into current frame
  - Calculate the color within the current pixel offset by a random function and have its result accumulated to a running average of all previous frames

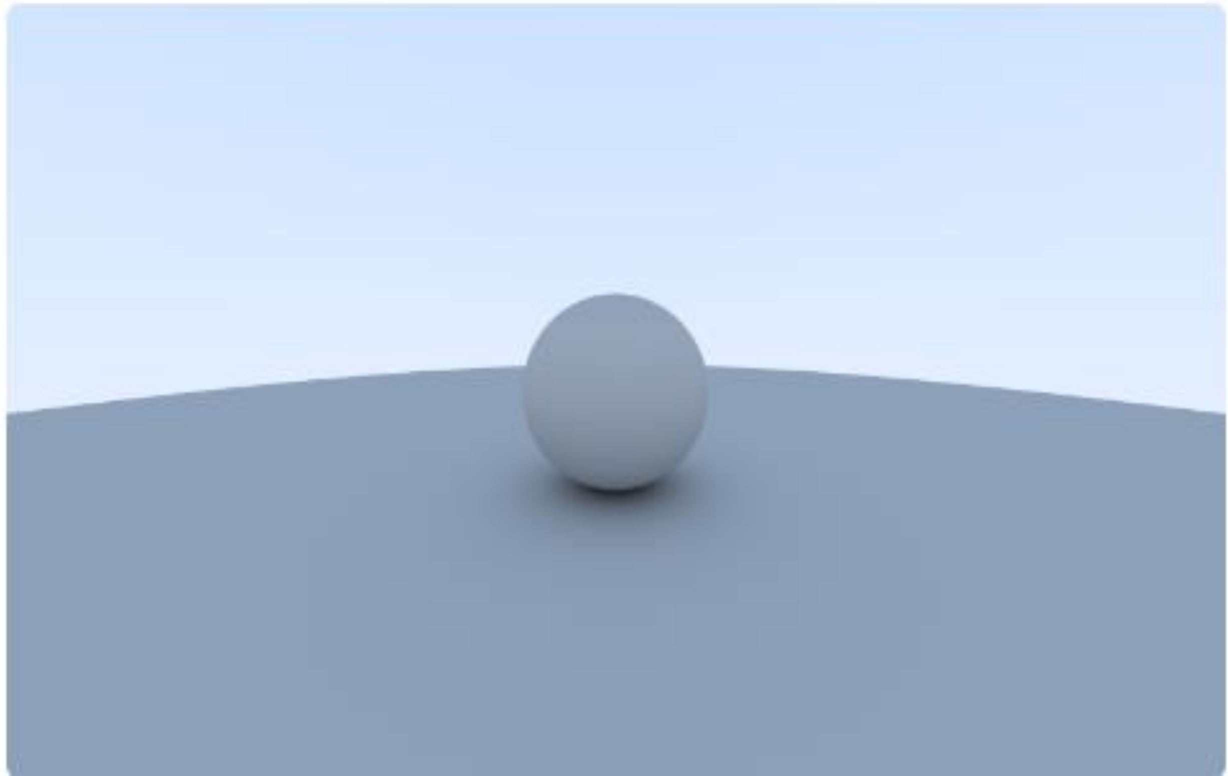
Progressive Ray Tracing



```
#ichannel0 “self”
vec2 uv = gl_FragCoord.xy / iResolution.xy;
vec4 prev = texture(iChannel0, uv);
vec3 prevLinear = toLinear(prev.xyz);
prevLinear *= prev.w;

uv = (gl_FragCoord.xy + hash2(gSeed)) [redacted]
vec3 col = color(getRay(cam, uv));

if(iMouseButton.x != 0.0 || iMouseButton.y != 0.0)
{
    col = toGamma(col);
    gl_FragColor = vec4(col, 1.0);
    return;
}
if(prev.w > 5000.0)
{
    gl_FragColor = prev;
    return;
}

col = (col + prevLinear);
float w = prev.w + 1.0;
col /= w;
col = toGamma(col);
gl_FragColor = vec4(col, w);
```

Progressive Ray Tracing





Looping instead of recursion

```
vec3 rayColor(Ray r)
    vec3 col = vec3(0.0);
    vec3 throughput = vec3(1.0f, 1.0f, 1.0f);
    for(int i = 0; i < MAX_BOUNCES; ++i)
    {
        if(hit_world(r, 0.001, 10000.0, rec))
        {
            //calculate direct lighting
             col += local_color * throughput;

            //calculate secondary rays and their throughputs
            Ray scatterRay;
            vec3 atten;
            if(scatter(r, rec, atten, scatterRay))
            {
                r = scatterRay;
                throughput *= atten;
            }
            else //it never happens. The material always scatters the
incoming ray
            {
                break;
            }
        }
        else //background color varying with y direction of the ray
        {
            float t = 0.8 * (r.d.y + 1.0);
            col += throughput * mix(vec3(1.0), vec3(0.5, 0.7, 1.0), t);
            break;
        }
    }
    return col;
```

Scattering

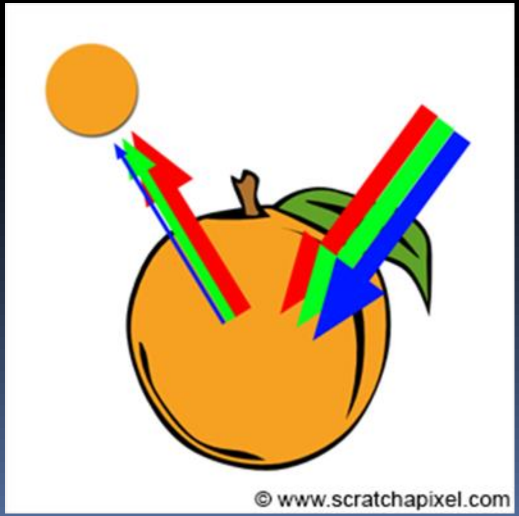
- bool scatter(Ray rIn, HitRecord rec, out vec3 atten, out Ray rScattered)
- Check the material type: diffuse, metal, dieletric
- Types of bouncing off:
  - Now we will consider diffuse (Lambertian) scattering: color bleeding
  - Metal: surface specular reflections modulated by a roughness parameter
  - Dieletric: reflection and refraction weighted by Schlick approximation of Fresnel equation. Use probabilistic maths to decide if refract or reflect:

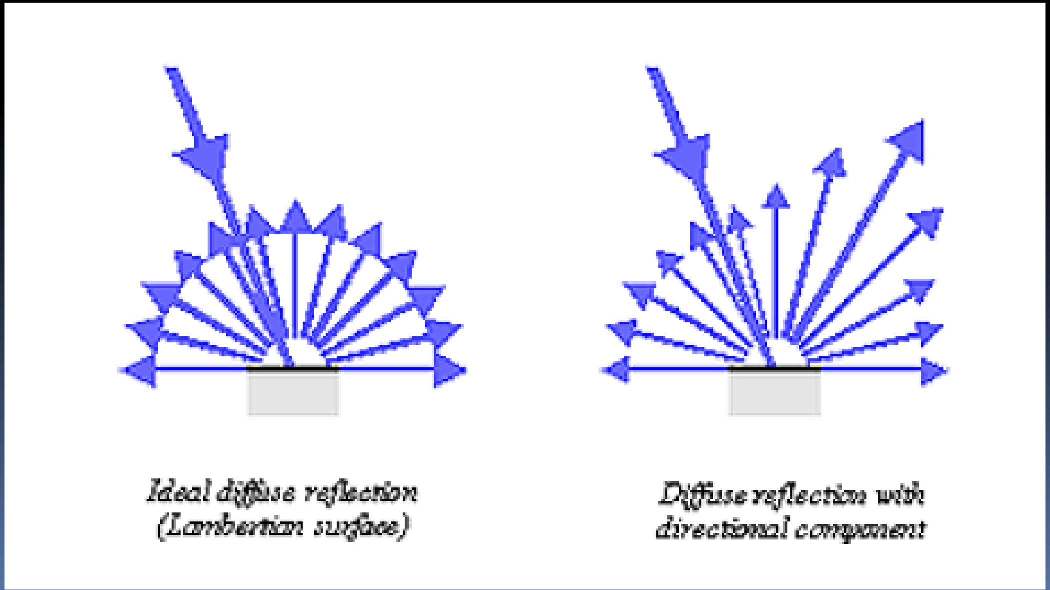
```
reflectProb = schlick(cosine, rec.material.refIdx);  //or = 1 if total reflection
if hash1(seed) < reflectProb
    create reflected ray
else
    create refracted ray
```

  - Dieletric: by using the above random ray generation, the attenuation will be just the refraction albedo, ex. (1.0, 1.0, 1.0) for clear transparent materials
  - Dielectric: use Beer's law for coloured transparent -> if ray inside then: $e^{-refractionColor * distance}$

Diffuse inter-reflections

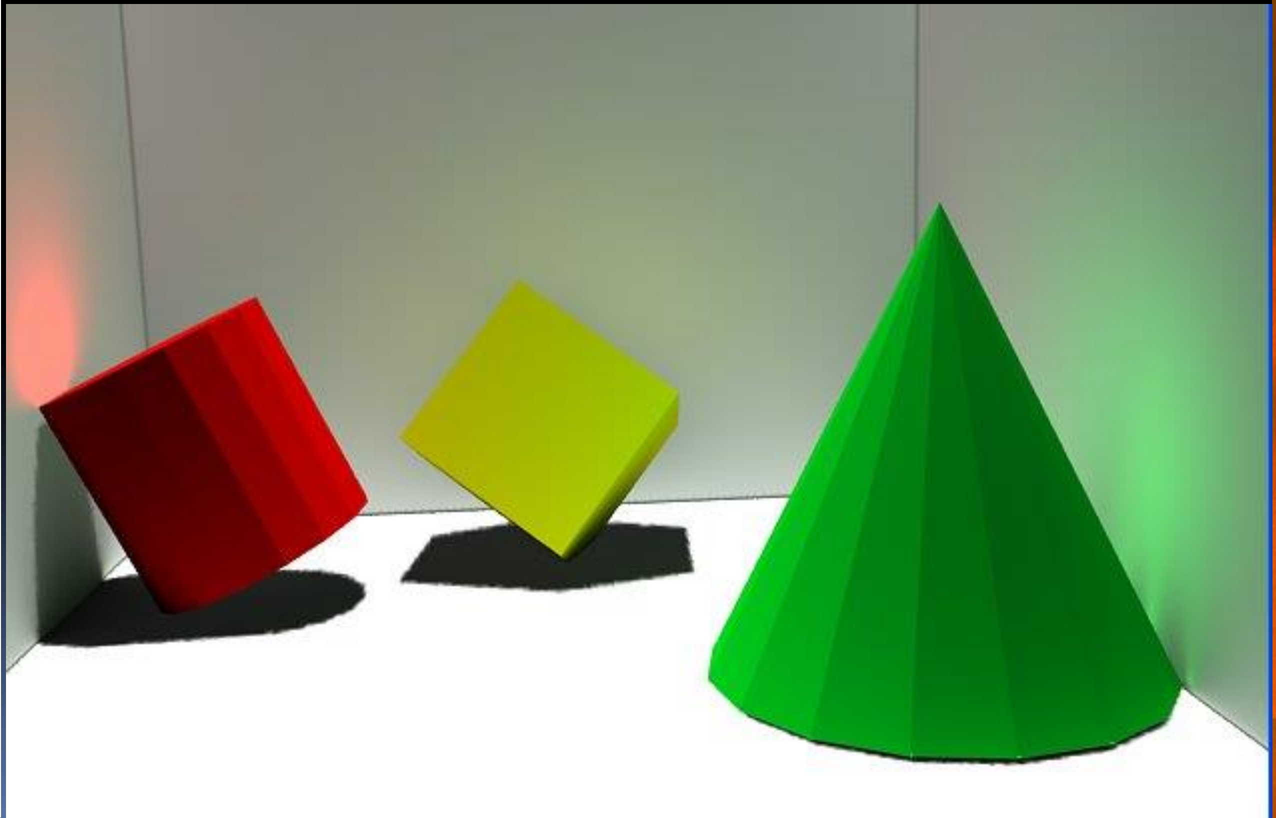
- How do we see the color: the surface absorbs photons of certain range of wavelengths and scatters the others (subsurface scattering around the hit point)
- A diffuse surface reflects light in all directions, ie. It has not a specific direction behaviour like the specular one
  - A special case of diffuse reflection is Lambertian: the reflection is uniform over the whole hemisphere regardless the irradiance.





Diffuse inter-reflections

- Color bleeding: objects take on color of their surroundings and modulate that with their own intrinsic diffuse color (aka albedo)

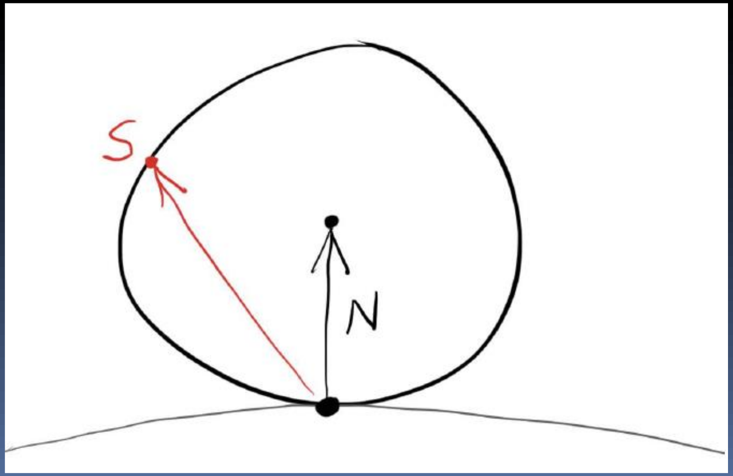


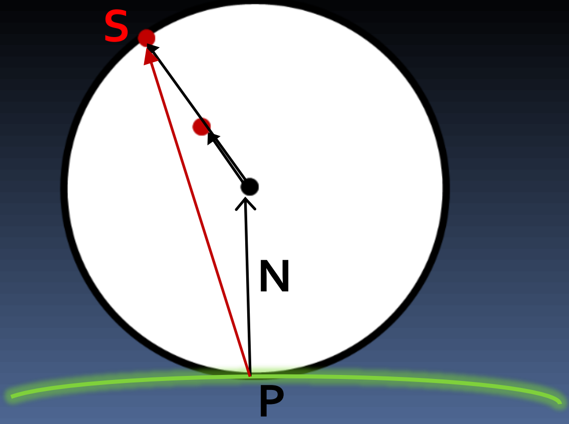
LightSprint

Random diffuse scattered ray direction

- Lambertian distribution means a cosine distribution: higher probability for a ray scattering close to the normal at the hit point; aka cosine importance sampling
- Consider a unit radius sphere tangent to the hit point P and calculate point S on the surface of it

S = P + N + normalize(random_in_unit_sphere)





Visual effects

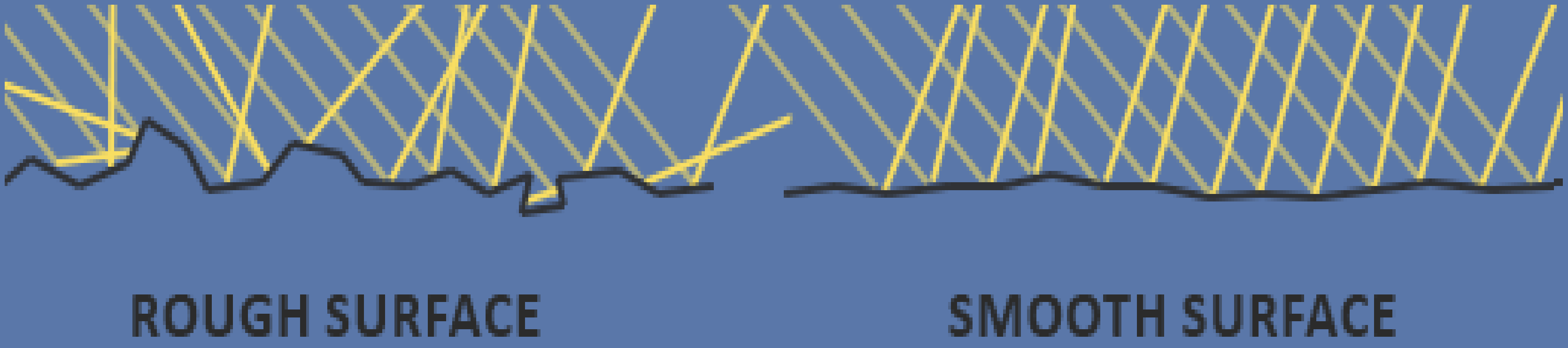
- Gamma correction for accurate color intensity
  - Gamma: 2.2
  - Use linear color for ray tracing computation: pow(c, vec3(2.2))
  - Store gamma corrected: pow(c, vec3(1.0 / 2.2))
- Motion blur with moving spheres
- Depth of field
- Modelling a hollow glass sphere: create a dieletric sphere with a negative radius

Scene representation

- Building the scene on each invocation of a hit world function
- Simple but no flexible.
- Constraint by Shadertoy model: it encourages the use of implicit form to model a scene
- 3D: use of implicit distance fields
- Primitives define simple shapes in an analytical closed form. Spheres, rounded boxes, helixes, etc are good examples
- Domain operators allow to scale, bend, twist and repeat shapes. Range operators allow to combine and displace and deform shapes.
- https://iquilezles.untergrund.net/www/articles/distfunctions/distfunctions.htm: distance functions for basic primitives, plus the formulas for combining them together for building more complex shapes, as well as some distortion functions that you can use to shape your objects.

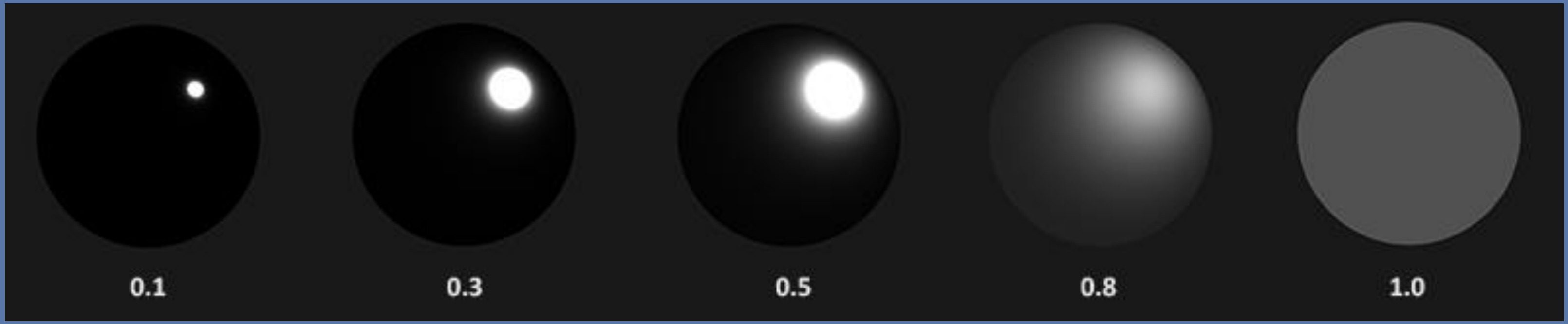
Physically-based Shading

- https://learnopengl.com/PBR/Theory
- originally explored by Disney and adopted for real-time display by Epic Games
- Microfacets based



$$h = \frac{l + v}{\|l + v\|}$$

- https://www.shadertoy.com/view/4sSfzK


0.1
0.3
0.5
0.8
1.0