

# INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 13: “Ground Truth”

# Welcome!



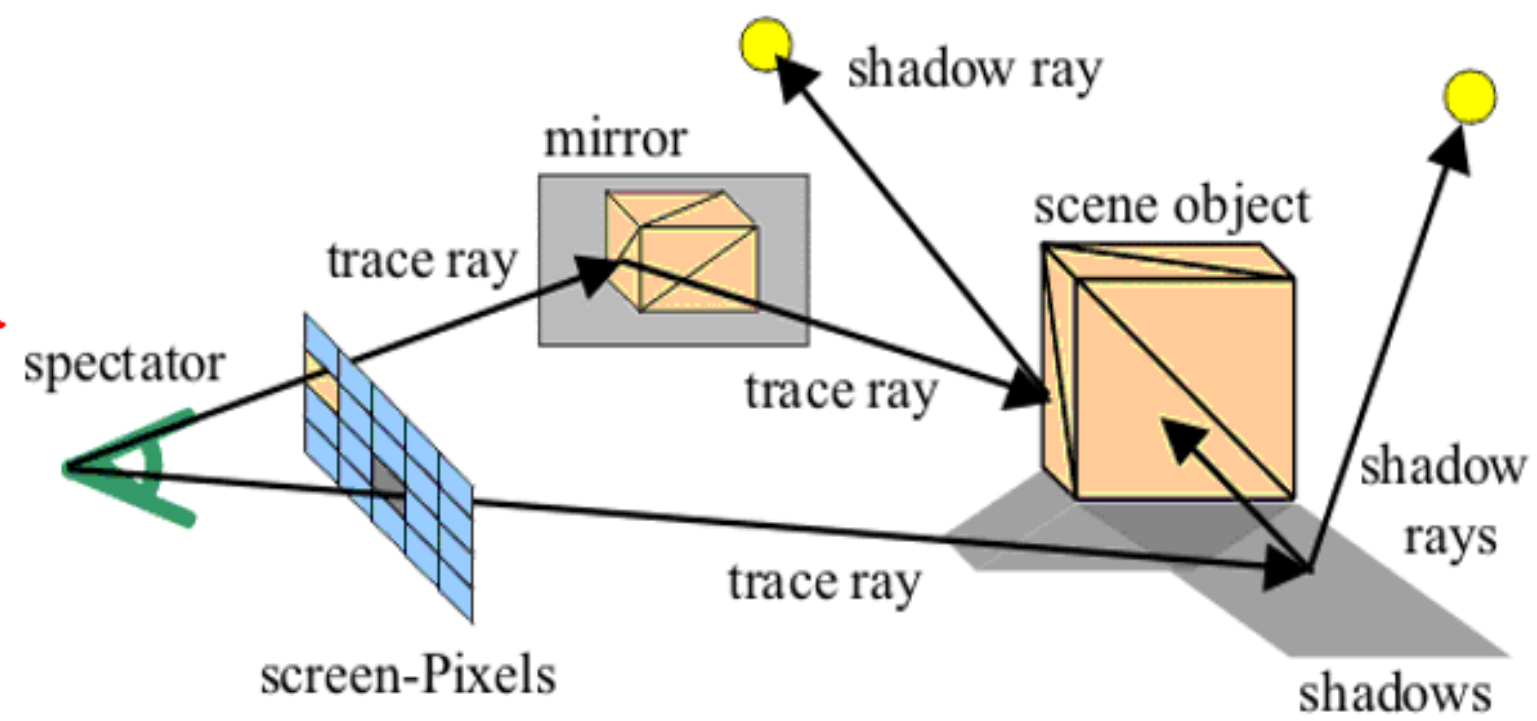
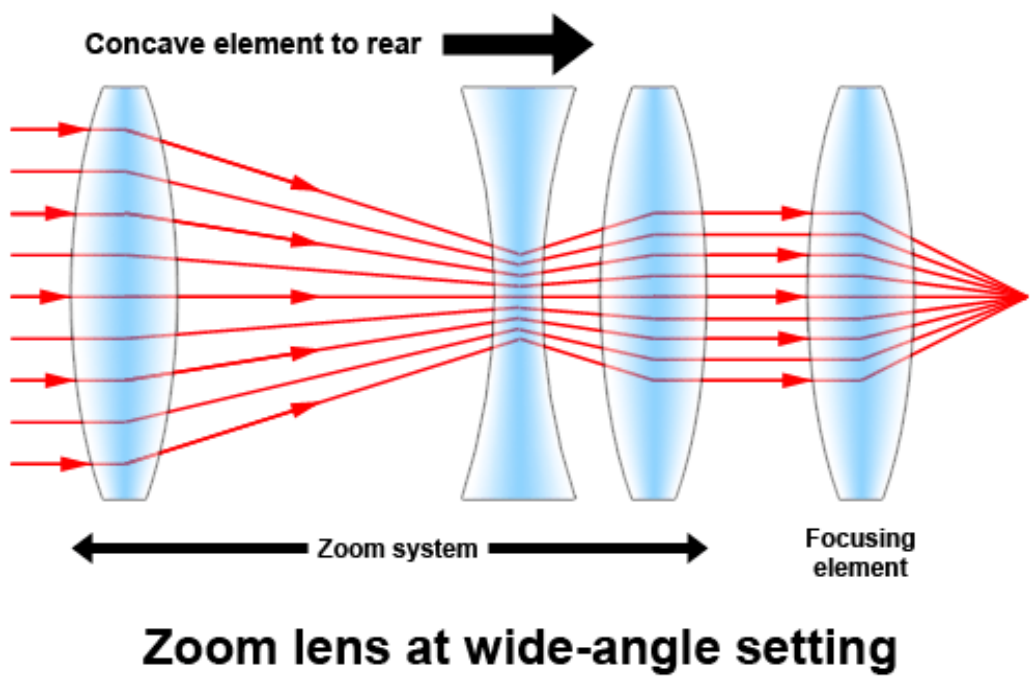
# Today's Agenda:

- Deterministic Rendering
- Monte Carlo
- Path Tracing



# Deterministic

## Whitted-style Ray Tracing



```
void walk = done properly, closely following the path of the ray (live);  
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, BR, &pdf );  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
mission = true;
```



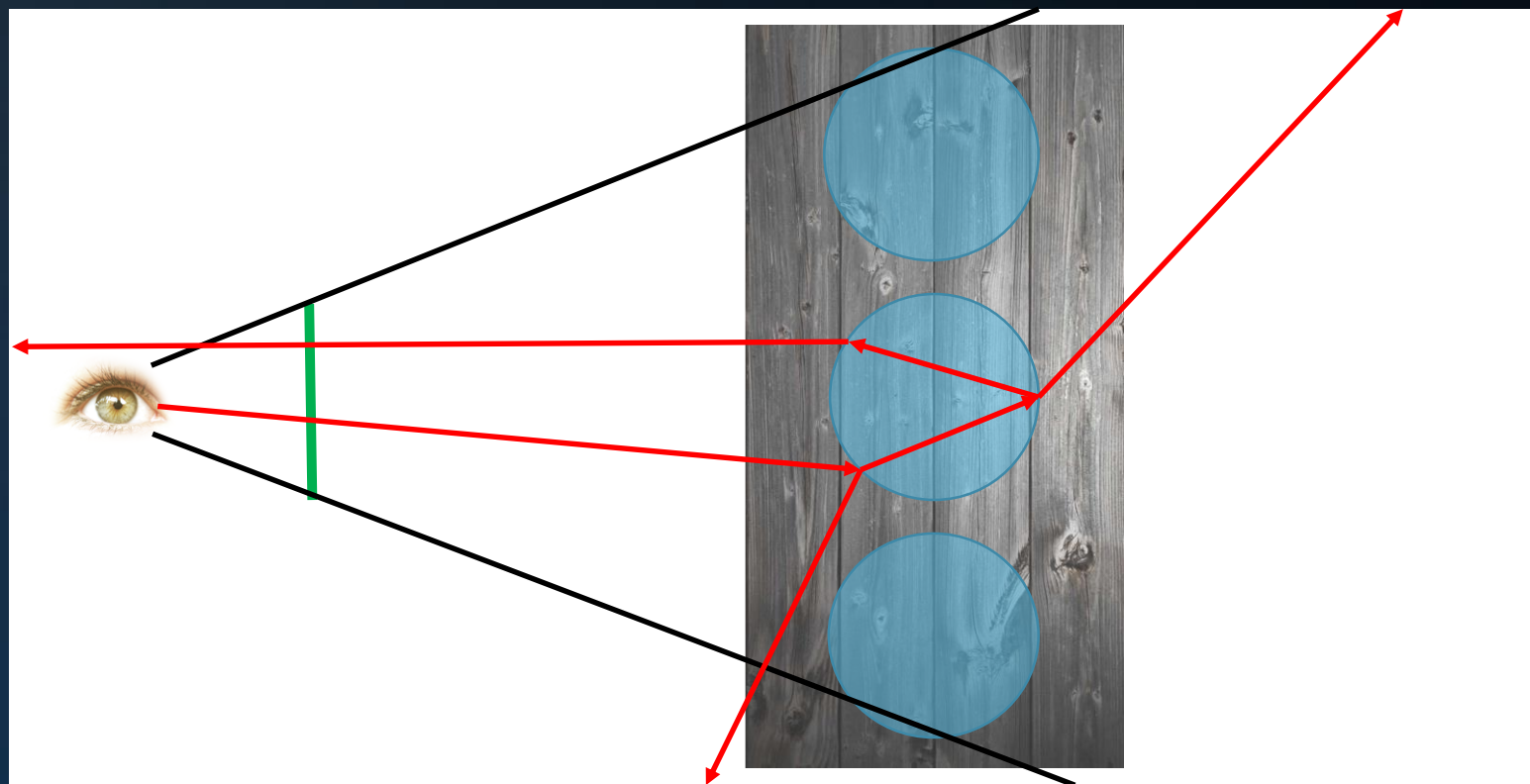
# Deterministic

## Whitted-style Ray Tracing

Color at pixel:

- sphere material color \*  
refracted ray
- + sphere material color \*  
reflected ray

This is a recursive process.



# Deterministic

## Whitted-style Ray Tracing

Color at pixel:

- sphere material color \* refracted ray
- + sphere material color \* reflected ray

*This is a recursive process.*

Fresnel equations

Snell's law

```
color Trace( O, D )
    I, N, mat = NearestIntersection( O, D )
    if (mat == DIFFUSE)
        return mat.color *
            DirectIllumination( I, N )
    if (mat == MIRROR)
        return mat.color *
            Trace( I, reflect( D, N ) )
    if (mat == GLASS)
        return mat.color *
            (X * Trace( I, reflect( D, N ) ) +
             (1-X) * Trace( I, refract( D, N ) ) )
```

angle of incidence = angle of reflection



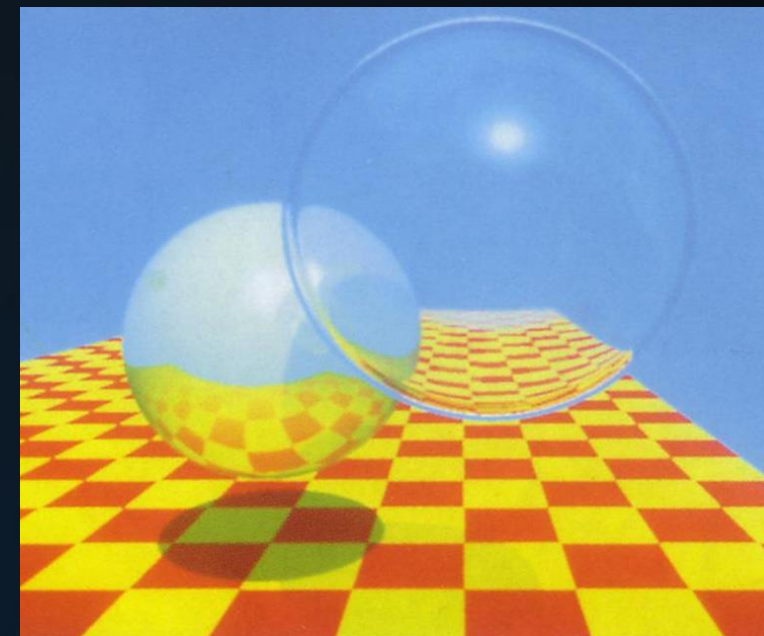
# Deterministic

## Whitted-style Ray Tracing

### Shading model:

- Based on classical ray optics

Whitted-style ray tracing is deterministic:  
it cannot simulate area lights, glossy  
reflections, and diffuse reflections.





# Deterministic

## Rasterization

```
color Shade( P, N )  
    if (mat == DIFFUSE)  
        return mat.color *  
            DirectIllumination( P, N )  
    if (mat == MIRROR)  
        return mat.color *  
            EnvironmentMap( ... )  
    if (mat == GLASS)  
        return mat.color *  
            HackyGlassEffect( ... )
```



Ray traced







# Today's Agenda:

- Deterministic Rendering
- Monte Carlo
- Path Tracing



# Monte-Carlo

## Distributed Ray Tracing\*

### Problem:

Ray tracing is currently limited to sharp shadows, sharp reflections, and sharp refraction.

### Goal:

- Augment Whitted-style ray tracing with glossy reflections and refractions, as well as soft shadows.



\*: “Distributed Ray Tracing”, Cook et al., 1984.









# Monte Carlo

```

    if (depth < MAXDEPTH)
    {
        Vec inside = L;
        Vec nt = nc;
        Vec nnt = nnt * nnt;
        Vec pos2t = 1.0f - nnt;
        Vec D, N;
        Vec a = nt - nc;
        Vec b = nt - nc;
        Vec Tr = 1 - (R0 + (1 - R0) * pos2t);
        Vec R = (D * nnt - N * pos2t);
        Vec E * diffuse;
        Vec refl;
        Vec refl + refr;
        Vec refl * E * diffuse;
        Vec refl;
        Vec MAXDEPTH);
        Vec survive = SurvivalProbability( diffuse );
        Vec estimation - doing it properly, closely following walk (survive);
        Vec radian = SampleLight( &rand, I, &L, &light );
        Vec e.x + radian.y + radian.z > 0) && (e.x + radian.y + radian.z > 0);
        Vec w = true;
        Vec brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        Vec factor = diffuse * INVPI;
        Vec weight = Mis2( directPdf, brdfPdf );
        Vec cosThetaOut = dot( N, L );
        Vec E * ((weight * cosThetaOut) / directPdf) * (radian);
        Vec random walk - done properly, closely following walk (survive);
        Vec survive);
        Vec brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec n;
        Vec n;
    }

```



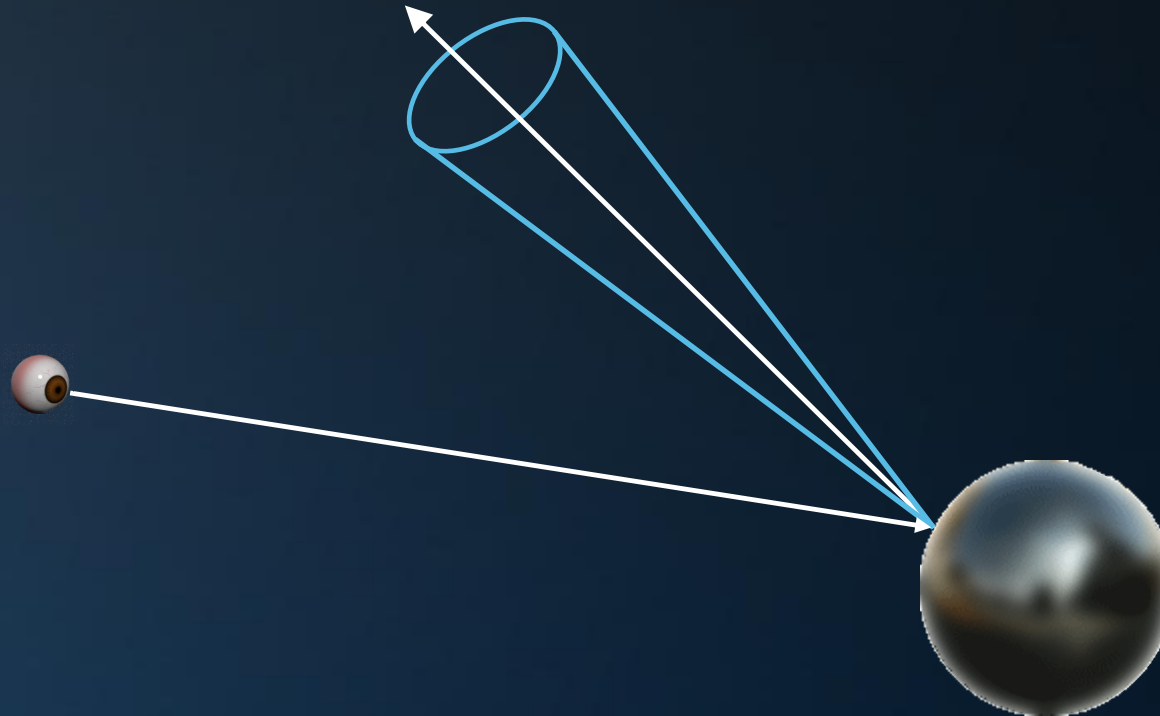


# Monte Carlo

```

    if (depth < MAXDEPTH)
    {
        Vec inside = L * N;
        Vec nt = nt / nc, nct = nc / nc;
        double r = 1.0f - nnt * nnt;
        double r2 = r * r;
        Vec D, N;
        Vec a = nt - nc, b = nt * nc;
        double Tr = 1 - (R0 + (1 - R0) * r2);
        Vec R = (D * nnt - N * (1 - Tr));
        Vec E * diffuse;
        bool = true;
        Vec refl + refr)) && (depth < MAXDEPTH)
        Vec D, N;
        Vec refl * E * diffuse;
        bool = true;
        Vec MAXDEPTH)
        Vec survive = SurvivalProbability( diffuse );
        Vec estimation - doing it properly, closely following
        if;
        Vec radiance = SampleLight( &rand, I, &L, &light);
        Vec e.x + radiance.y + radiance.z > 0) && (e.x + radiance.y + radiance.z > 0)
        Vec w = true;
        Vec brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        Vec factor = diffuse * INVPI;
        Vec weight = Mis2( directPdf, brdfPdf );
        Vec cosThetaOut = dot( N, L );
        Vec E * ((weight * cosThetaOut) / directPdf) * (radiance);
        Vec random walk - done properly, closely following
        Vec survive)
        Vec brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        Vec survive;
        Vec pdf;
        Vec n = E * brdf * (dot( N, R ) / pdf);
        Vec sion = true;

```



# Monte Carlo

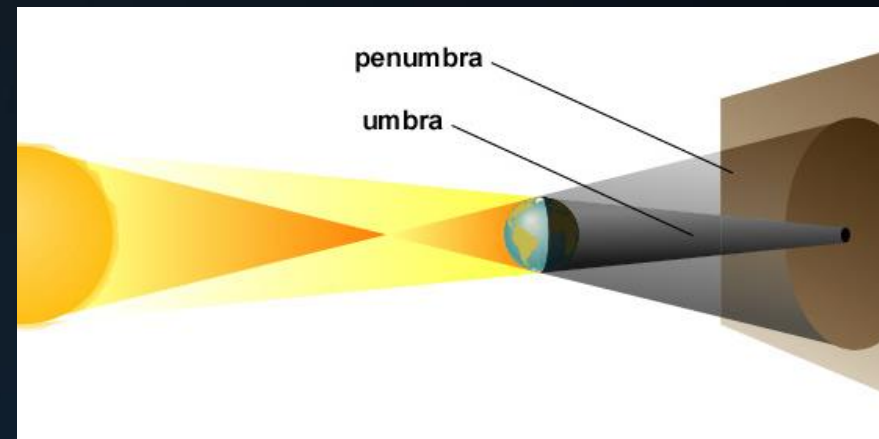
## Analytic Soft Shadows

### Anatomy of a shadow – regions

- Fully occluded area: *umbra*
- Partially occluded area: *penumbra*

*A soft shadow requires an area light source.*

In nature, all light sources are area lights (although some approximate point lights).



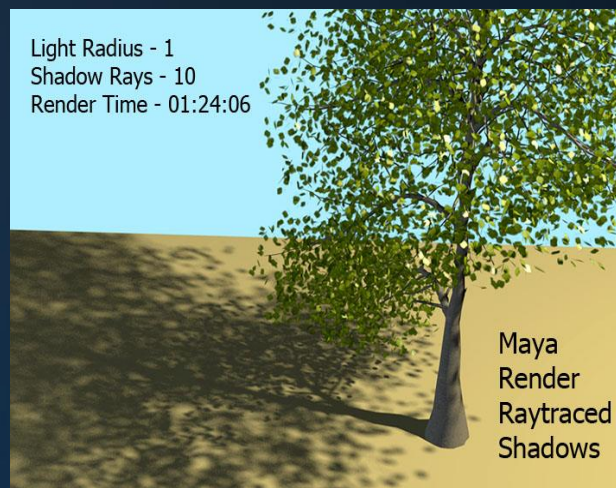
# Monte Carlo

## Analytic Soft Shadows

Surface points in the penumbra are lit by a part of the light source.

Rendering soft shadows requires that we determine the visible portion of the light source.

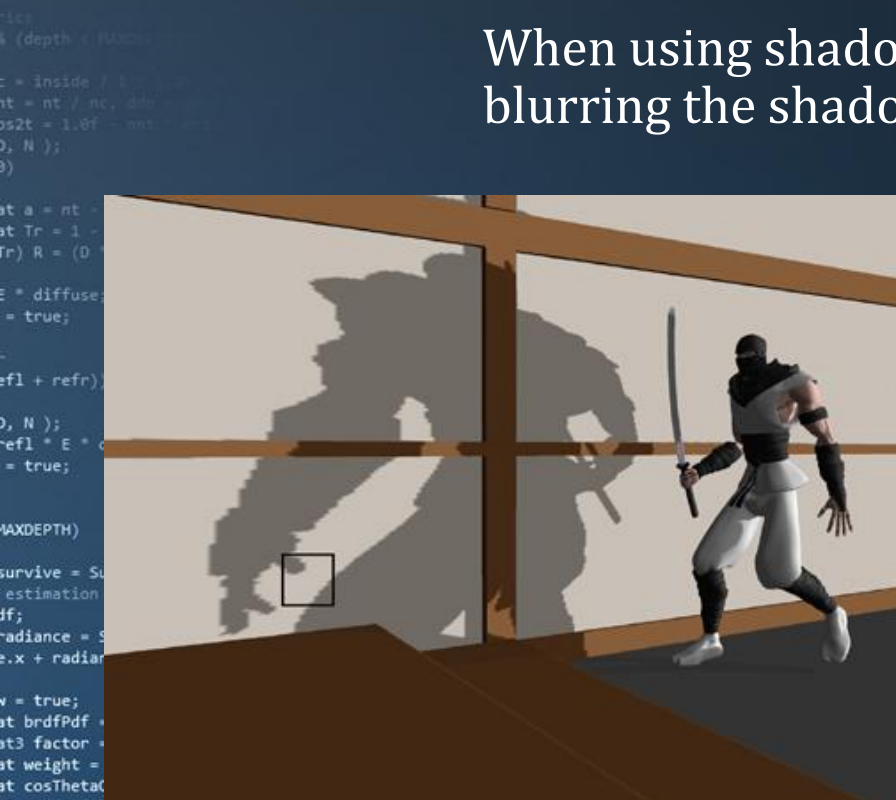
In most cases,  
this is a very  
hard problem.



# Monte Carlo

## Approximate Soft Shadows

When using shadow mapping, we can simulate soft shadows by blurring the shadow map.



In this example, filter kernel radius is adjusted based on the distance from the occluder.





# Monte Carlo

## Calculating Accurate Soft Shadows

*“Rendering soft shadows requires that we determine the visible portion of the light source.”*

In other words:

The amount of light cast on a surface point P by area light L is determined by the integral of the visibility between P and L over the surface of the light source:

$$I_{L \rightarrow P} = \int_{A_L} V(P, L)$$

## Monte-Carlo Integration

To solve this integral for the generic case, we will use Monte-Carlo integration.

Using Monte-Carlo, we replace the integral by the expected value of a stochastic experiment.





# Monte Carlo

## Stochastic shadows

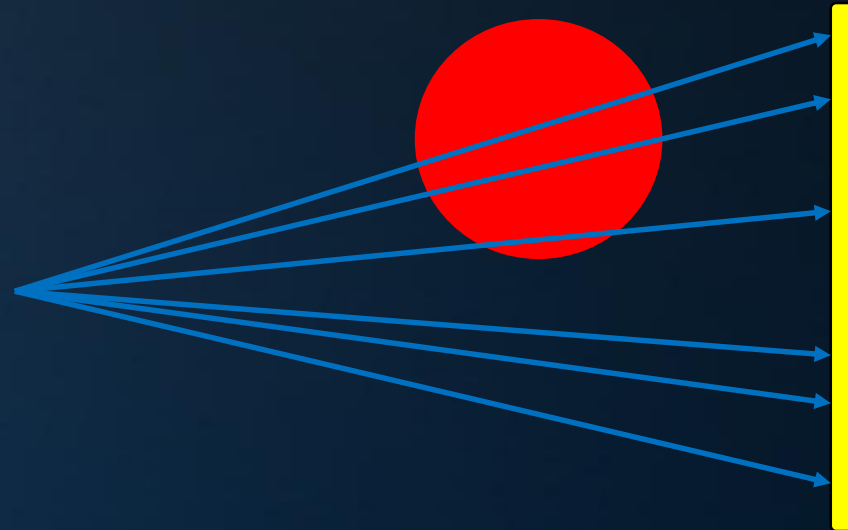
For soft shadows, we want to know the visible area of a light source, which can be 0..100%.

The light source could be (partially) obscured by any number of objects.

*We can approximate the visibility of the light source using a number of random rays.*

Using 6 rays:

$$V \approx \frac{1}{6} \sum_{i=1}^6 V_i$$



# Monte Carlo

## Stochastic shadows

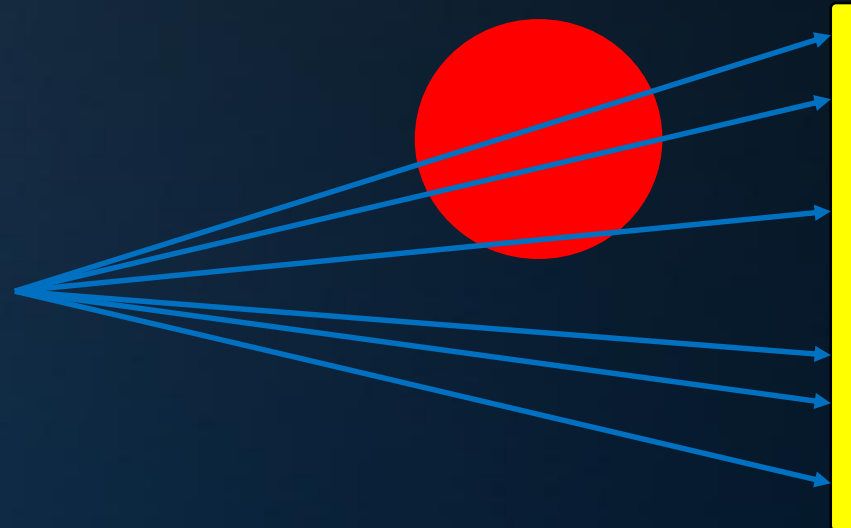
For soft shadows, we want to know the visible area of a light source, which can be 0..100%.

The light source could be (partially) obscured by any number of objects.

*We can approximate the visibility of the light source using a number of random rays.*

Using N rays:

$$V \approx \frac{1}{N} \sum_{i=1}^N V_i$$



# Monte Carlo

## Stochastic shadows

$$V \approx \frac{1}{N} \sum_{i=1}^N V_i$$

As  $N$  approaches infinity, the result becomes equal to the expected value, which is the integral we were looking for.

Before that, the result will exhibit *variance*.  
In the case of soft shadows, this shows up as noise.

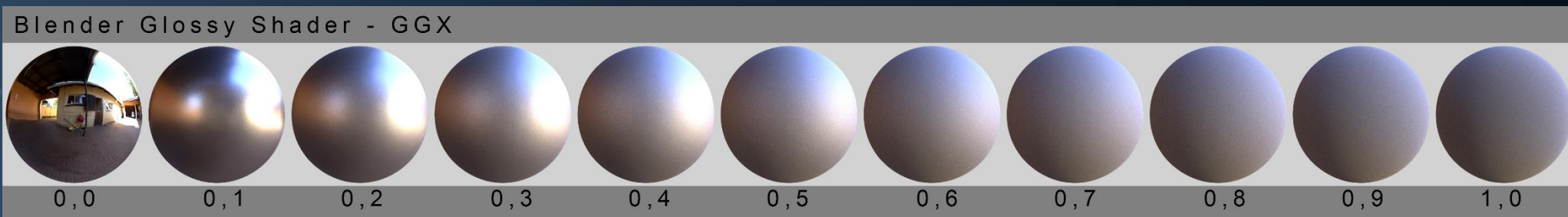


# Monte Carlo

## Approximate Diffuse Reflections

When rendering diffuse reflections, we face a similar problem:

A glossy surface reflects light arriving from a range of directions.



In rasterization, we can achieve this by blurring the environment map.





```

101
102 & (depth < MAXDEPTH)
103 {
104     nc = inside / 1.0f;
105     nt = nt / nc; ddn =
106     cos2t = 1.0f - nnt;
107     D, N);
108 }
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
```





```

100
    (depth < MAXDEPTH)
    {
        nc = inside / 1.0f + 1.0f;
        nt = nt / nc; ddo = dot(N, R);
        cos2t = 1.0f - nt * nt;
        D, N );
    }
}

    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (RB + (1 - RB) * a);
    Tr) R = (D * nnt - N * (dot(
    E * diffuse;
    = true;

    .
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;

    MAXDEPTH)

survive = SurvivalProbability( diffuse,
estimation - doing it properly, clearly
if;
radiance = SampleLight( &rand, I, &t, &align,
e.x + radiance.y + radiance.z) > 0) && (maxN <
w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mix2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following wall to
vive)

;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

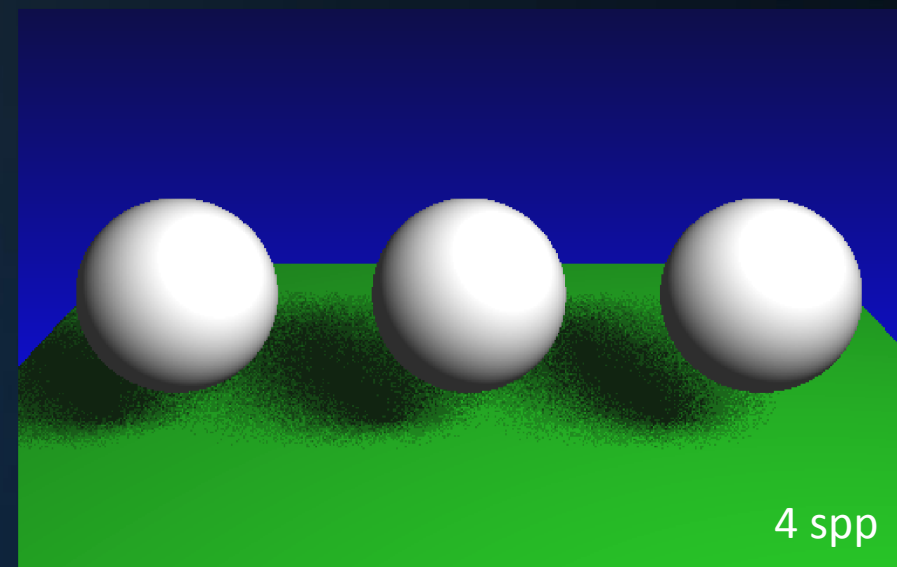
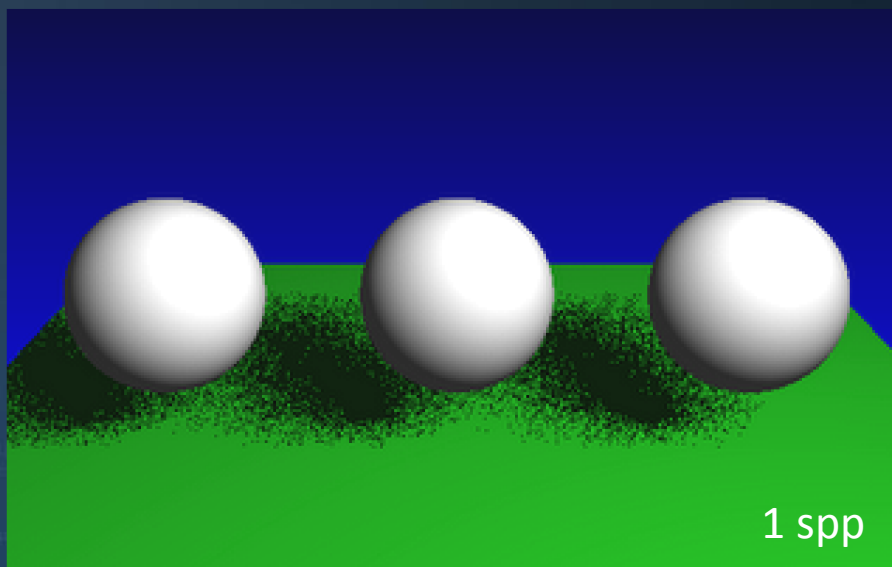
```



# Monte Carlo

## Variance

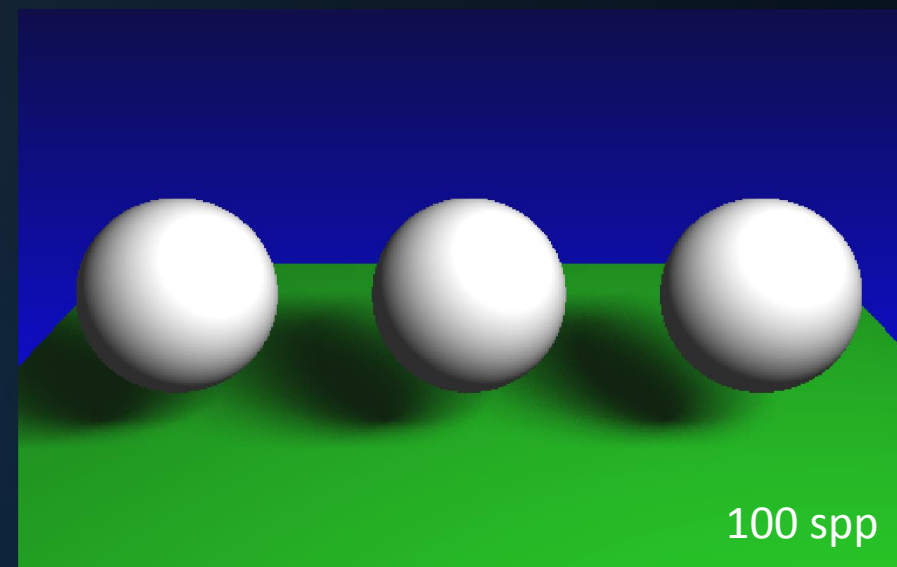
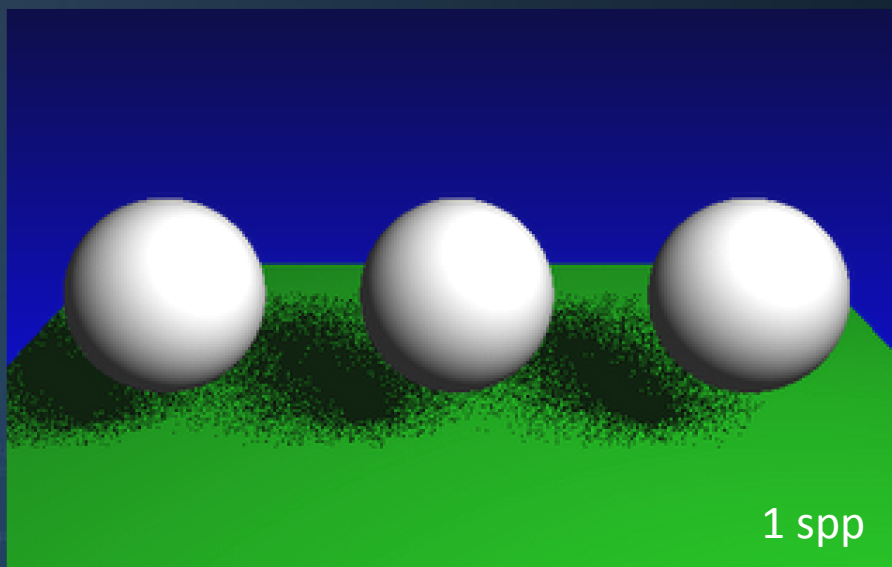
As long as we don't take an infinite amount of samples, the result of the stochastic process exhibits variance.



# Monte Carlo

## Variance

As long as we don't take an infinite amount of samples, the result of the stochastic process exhibits variance.



```
radiance = SampleLight( &rand, I, &l, &lightColor);  
e.x + radiance.y + radiance.z) > 0) && (comp <
```

```

    at brdfPdf = EvaluateDiffuse( L, N ) * Psurv);
    at3 factor = diffuse * INVPI;

```

```

    dot cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radius

```

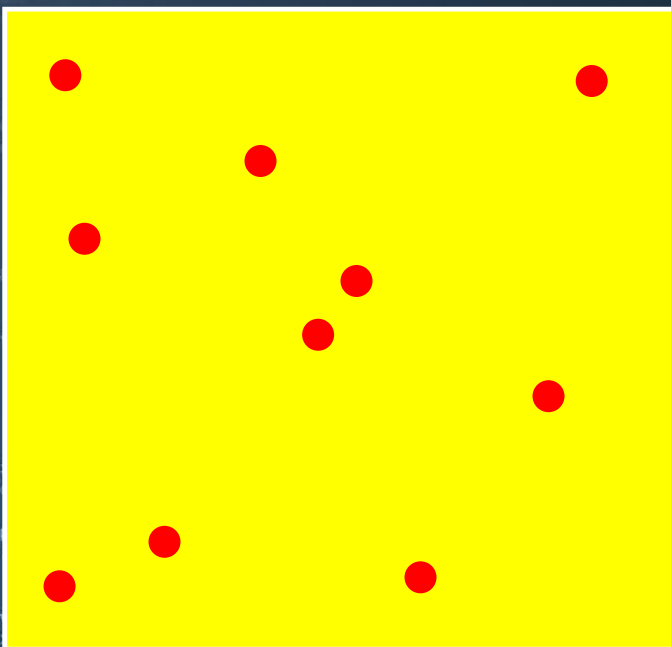
```
survive;
pdf;
```



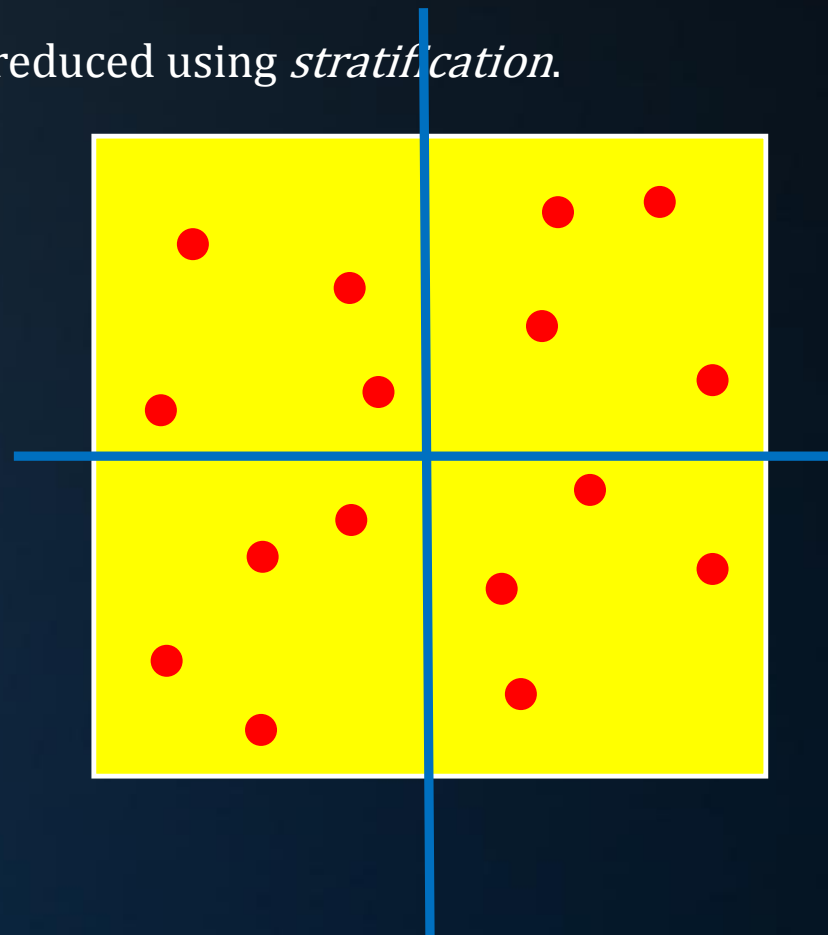
# Monte Carlo

## Variance reduction: stratification

The variance in random sampling can be reduced using *stratification*.



$N=16$

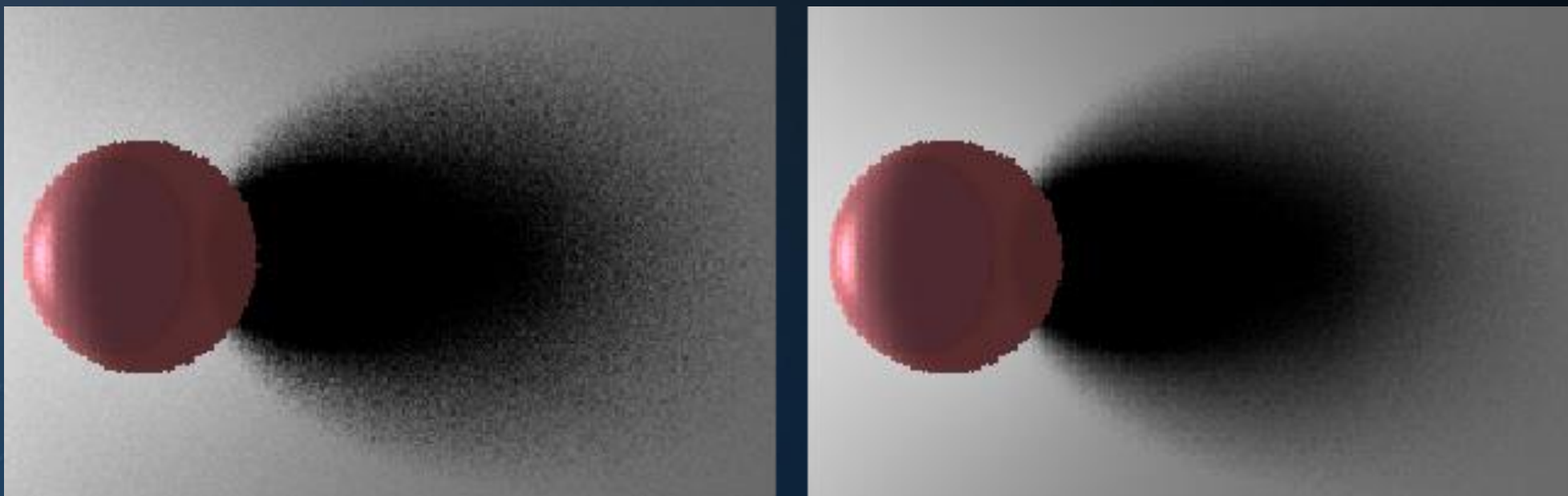




# Monte Carlo

## Variance reduction: stratification

The variance in random sampling can be reduced using *stratification*.



*Uniform vs stratified, 36 samples, 6x6 strata*



# Monte-Carlo

## Distributed Ray Tracing

Integrating over area of light sources: soft shadows

Integrating over reflection cone: glossy reflections

Integrating over pixel: anti-aliasing

Integrating over time: motion blur

Integrating over lens: depth of field

Integrating over wavelength: dispersion



# Monte Carlo

## Distributed Ray Tracing

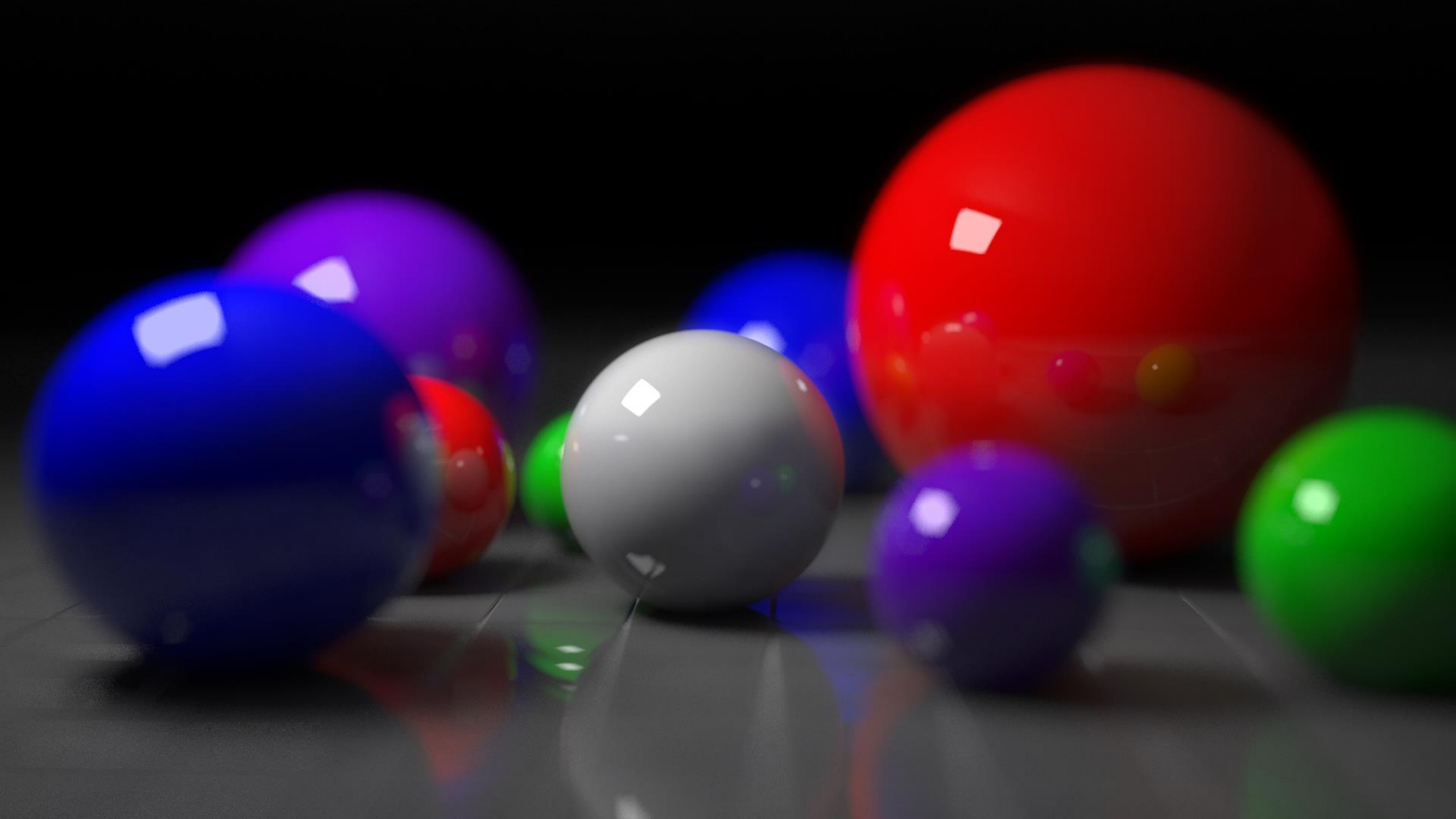
### Improved model:

- Still based on classical ray optics
- Combined with probability theory to solve integrals

Distributed ray tracing requires many rays to bring down variance to acceptable levels.







```

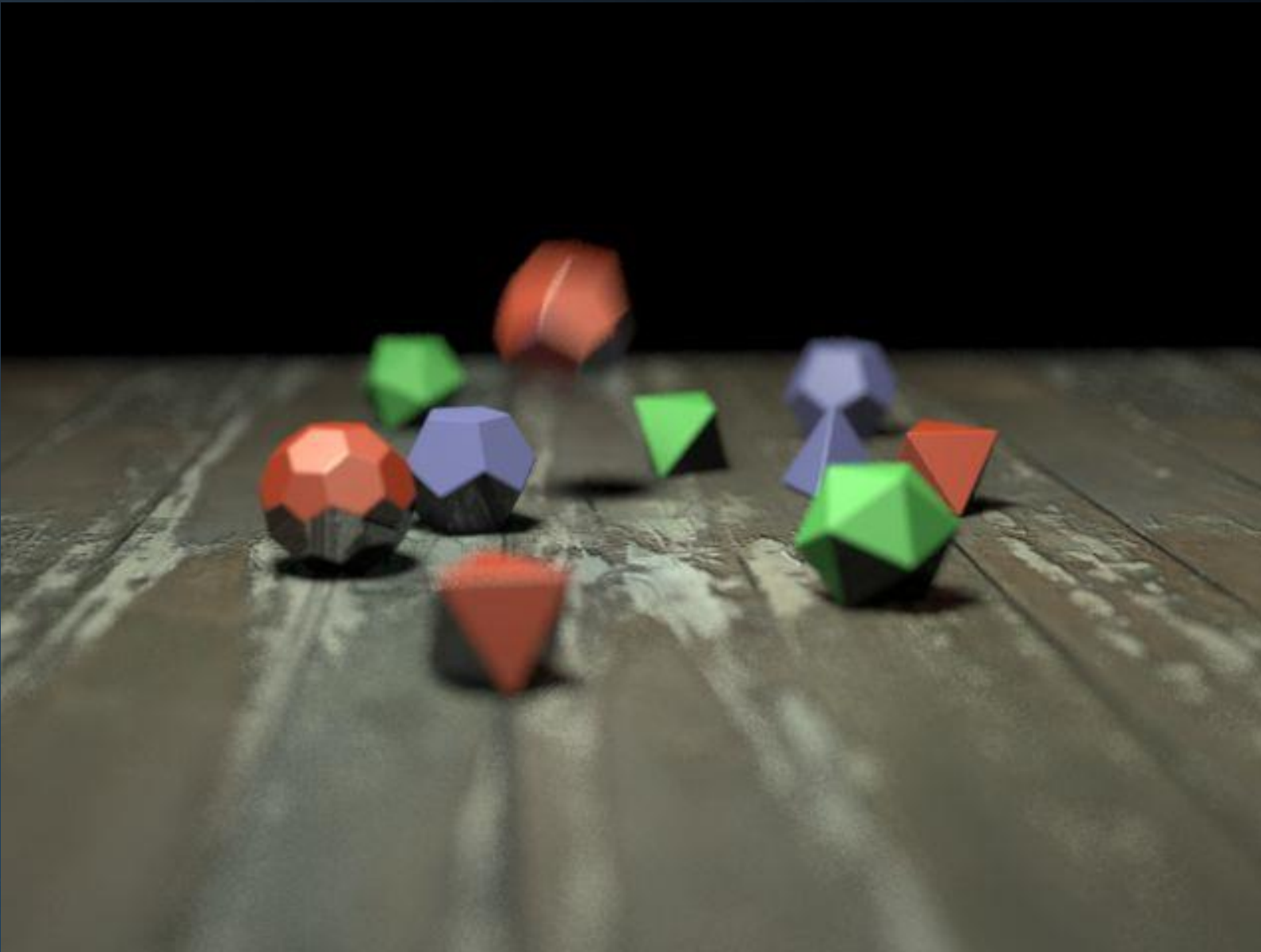
    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        t = inside / 1.5;
        nt = nt / nc; dde = dde / nc;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }

    // Inside the sphere
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (RB + (1 - RB) * t);
    Tr) R = (D * nnt - N * (dde
    // Inside the sphere
    E * diffuse;
    // Inside the sphere
    // Inside the sphere
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    // Inside the sphere
    // Inside the sphere
    MAXDEPTH)

    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &t, &light;
    e.x + radiance.y + radiance.z) > 0) && (depth <
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following walk
    survive)

    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```





```
rice  
& (depth  
t = insid  
nt = nt /  
os2t = 1.  
D, N );  
)  
at a = nt  
at Tr = 1  
Tr) R = (  
E * diffu  
= true;  
-  
efl + ref  
D, N );  
-refl * E  
= true;  
MAXDEPTH)  
survive =  
estimat  
if;  
-radiance  
e.x + rad  
v = true;  
at brdfPd  
at3 facto  
at weight  
at cosThe  
E * ((we  
random wal  
vive)
```

```
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf  
survive;  
pdf;  
n = E * brdf * (dot( N, R ) / pdf);  
sion = true;
```

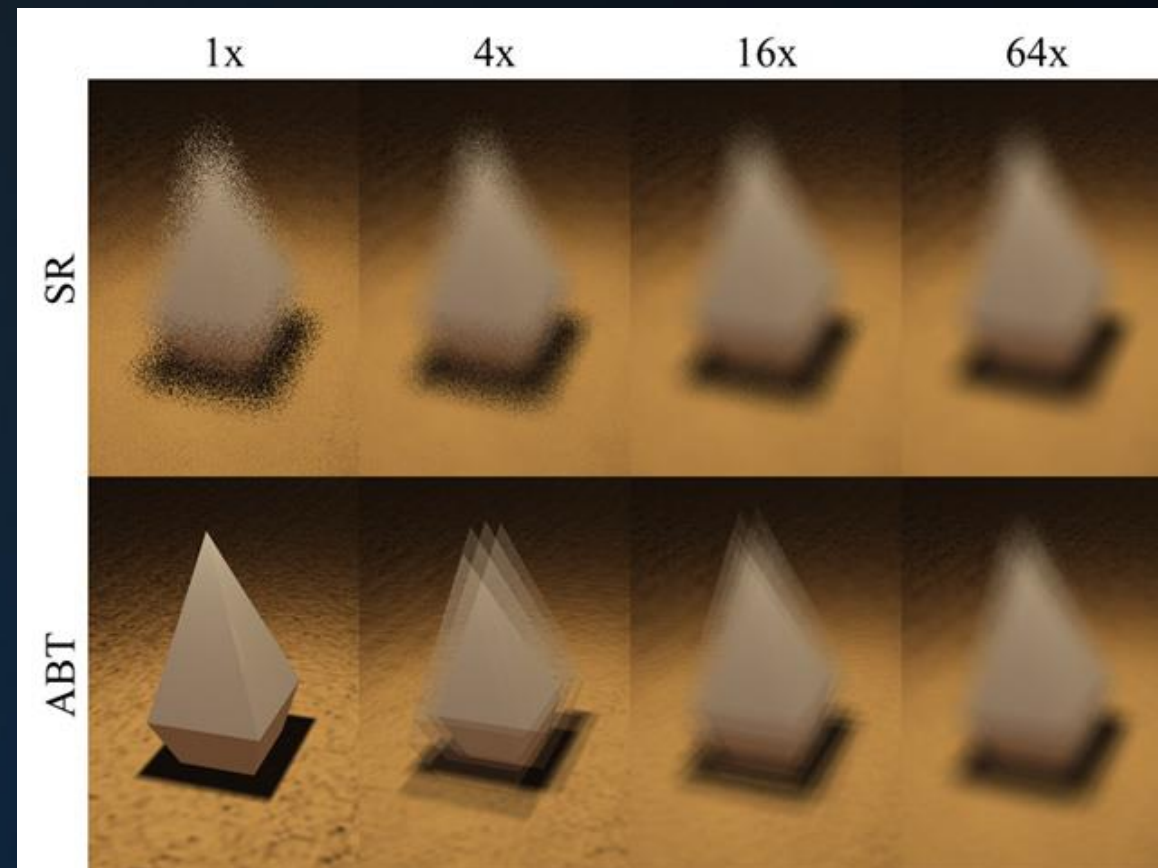


# Monte Carlo

## Monte Carlo in Rasterization

“Stochastic Depth of Field using Hardware Accelerated Rasterization”,

Robert Toth & Erik Lindler, 2008



# Monte Carlo in Rasterization

```

100
    (depth < MAXDEPTH)
    {
        nc = inside / L;
        nt = nt / nc; ddo = dot(N, D);
        cos2t = 1.0f - nnt * ddo;
        D, N );
    }
}

    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (RB + (1 - RB) * a);
    Tr) R = (D * nnt - N * ddo)
    {
        E * diffuse;
        = true;
    }
    {
        refl + refr)) && (depth < MAXDEPTH)
        {
            D, N );
            refl * E * diffuse;
            = true;
        }
    }
    MAXDEPTH)
    {
        survive = SurvivalProbability( diffuse );
        estimation - doing it properly, closely following
        diff;
        radiance = SampleLight( &rand, I, &L, &align,
        .x + radiance.y + radiance.z) > 0) && (not N
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mix2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (rad
        random walk - done properly, closely following
        survive)
    }
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }
}

```





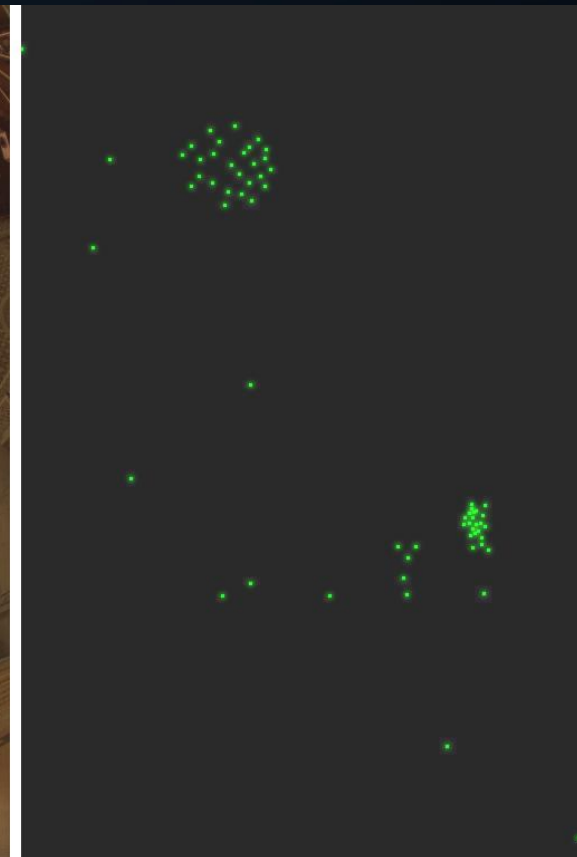
# Monte Carlo

## Monte Carlo in Rasterization

Light from an environment map,  
from:

“Wavelet Importance Sampling: Efficiently  
Evaluating Products of Complex  
Functions”,

Clarberg et al., 2005.



# Monte Carlo

## Cost of Distributed Ray Tracing

Distributed Ray Tracing is an expensive process:

- Per primary hit point, we need  $\sim 64$  shadow rays *per light*
- Per primary hit point on a glossy surface, we need  $\sim 64$  reflection rays,
  - ...and, for each reflection ray hit point, we need  $\sim 64$  shadow rays per light.

If we use 4x4 anti-aliasing per pixel, multiply the above by 16.

Now imagine a glossy surface reflects another glossy surface...





# Monte Carlo

```
...
    & (depth < MAXDEPTH)
{
    // Inside / Outside
    int nt = nt / nc, nde = nde / nc;
    float r2t = 1.0f - nnt * nnt;
    float D, N );
}

// ...
float a = nt - nc, b = nt - nc;
float Tr = 1 - (R0 + (1 - R0) * r2t);
float R = (D * nnt - N * (1 - R0));

// ...
E * diffuse;
= true;

// ...
refl + refr)) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;

    MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &t, &light );
e.x + radiance.y + radiance.z) > 0) && (e.x + radiance.y + radiance.z) > 0)
{
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance.x + radiance.y + radiance.z) > 0)
}

// ...
random walk - done properly, closely following walk
survive)

// ...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
}
```



# Today's Agenda:

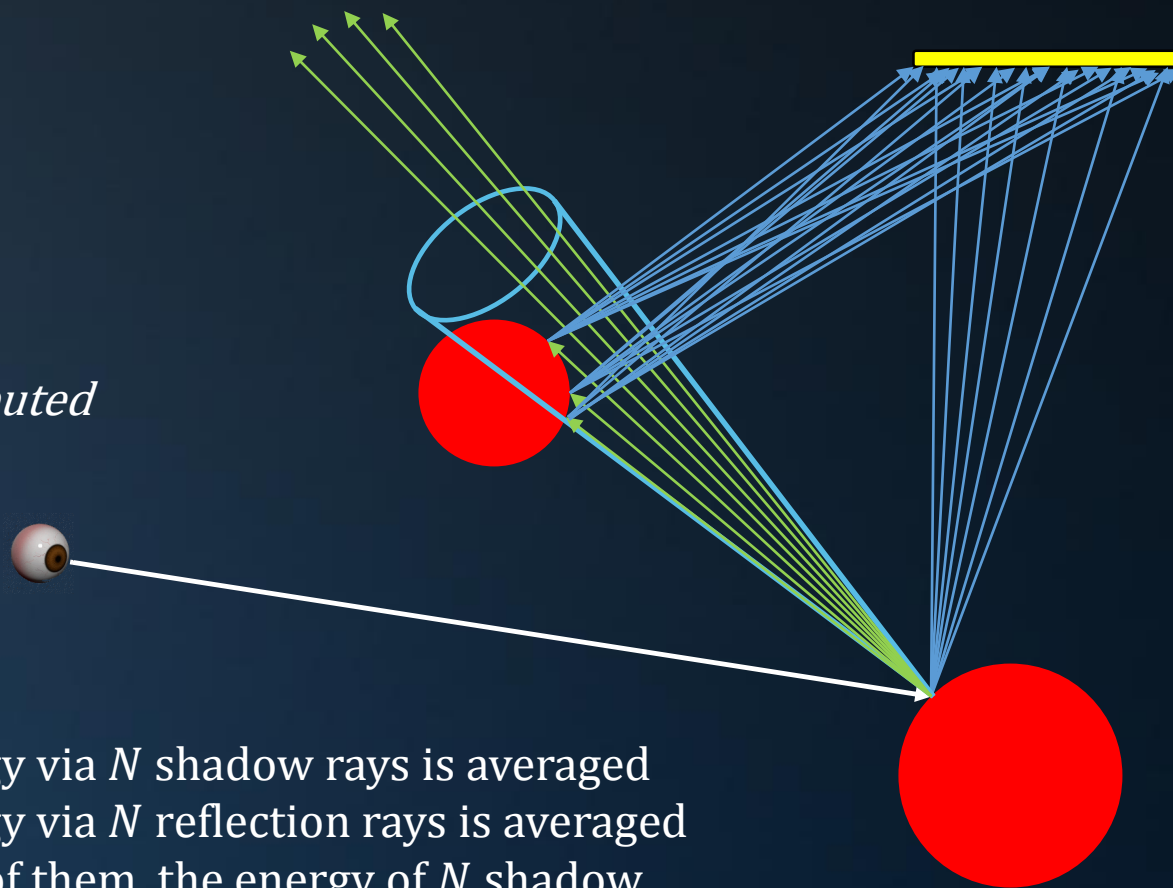
- Deterministic Rendering
- Monte Carlo
- Path Tracing



# Physically Based

## Ray Tree

*Using distributed ray tracing:*



- The energy via  $N$  shadow rays is averaged
- The energy via  $N$  reflection rays is averaged
- For each of them, the energy of  $N$  shadow rays is averaged.

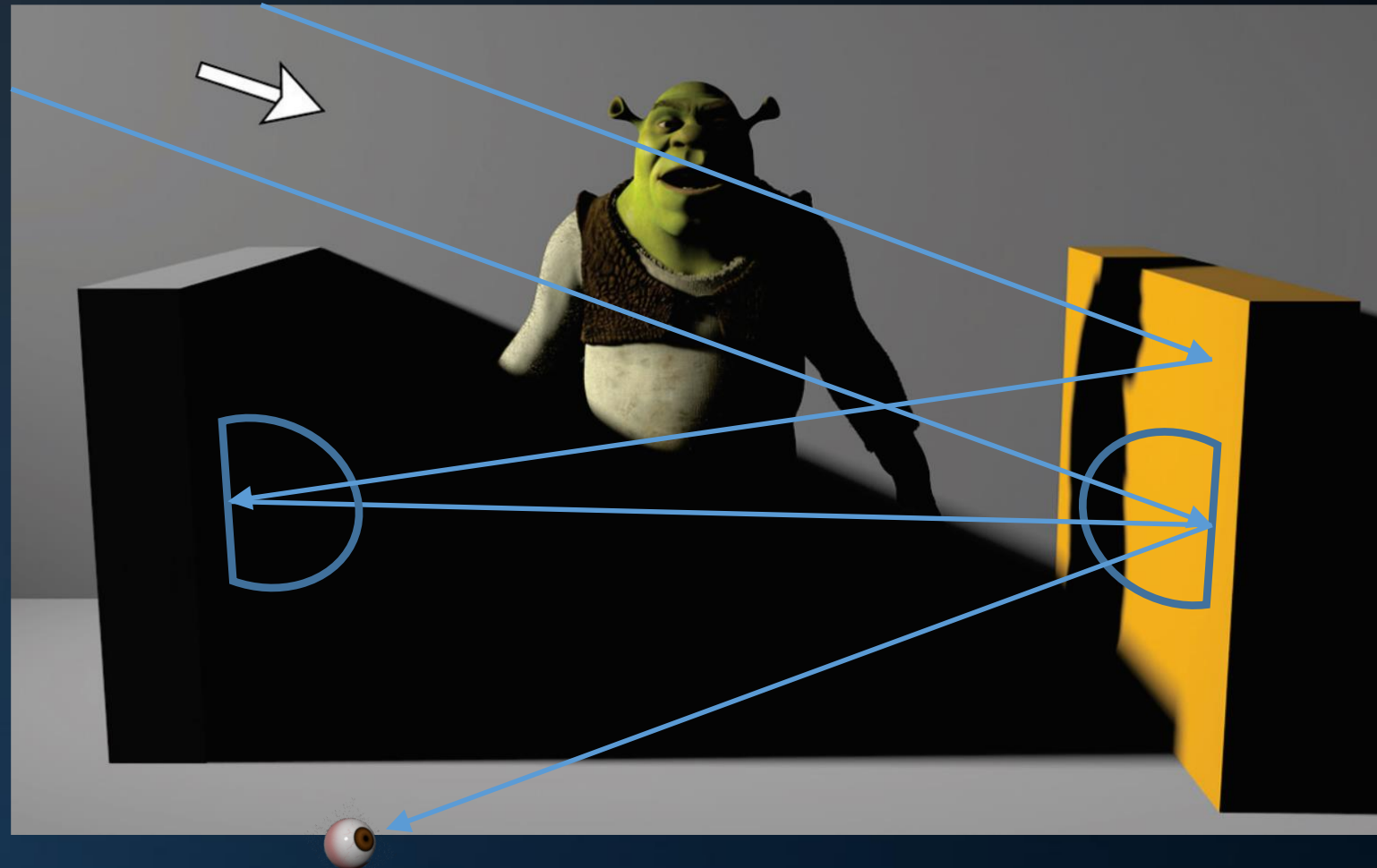
→ *The energy via each shadow ray is very low.*



# Physically Based

## Diffuse reflections

Apart from specular and glossy materials, diffuse materials also reflect light.



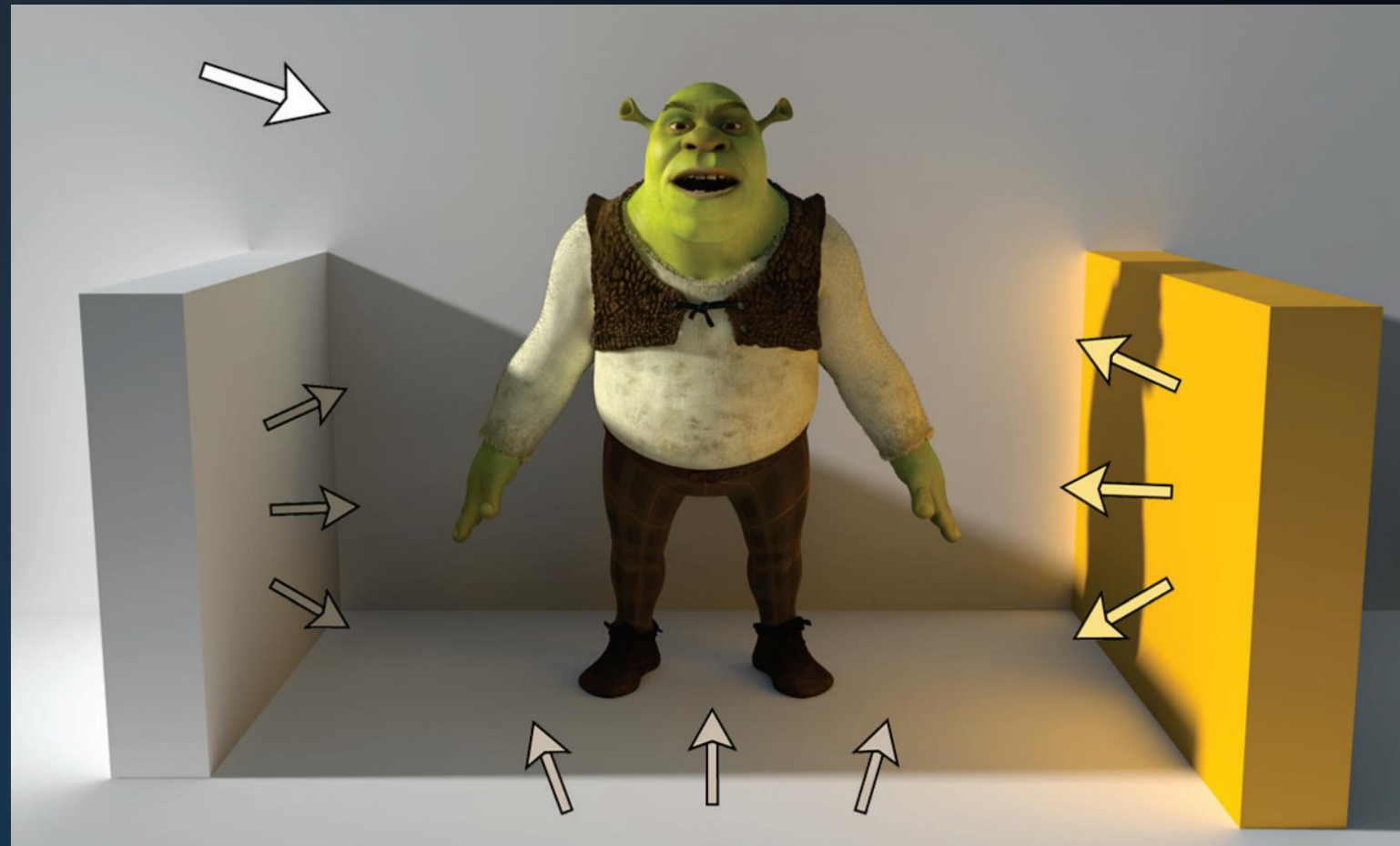


# Physically Based

## Diffuse reflections

Apart from specular and glossy materials, diffuse materials also reflect light.

This is why a shadow is seldom black.



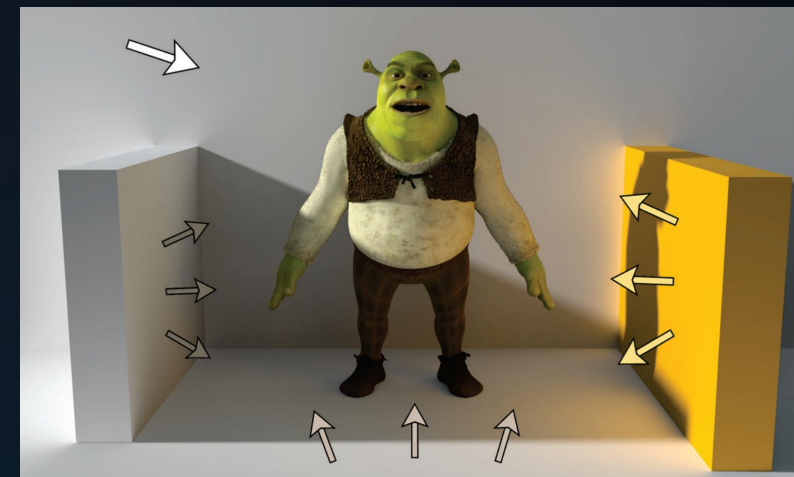


# Physically Based

## Physically based rendering

Calculating all light transport from the light sources to the camera, directly or via scene surfaces.

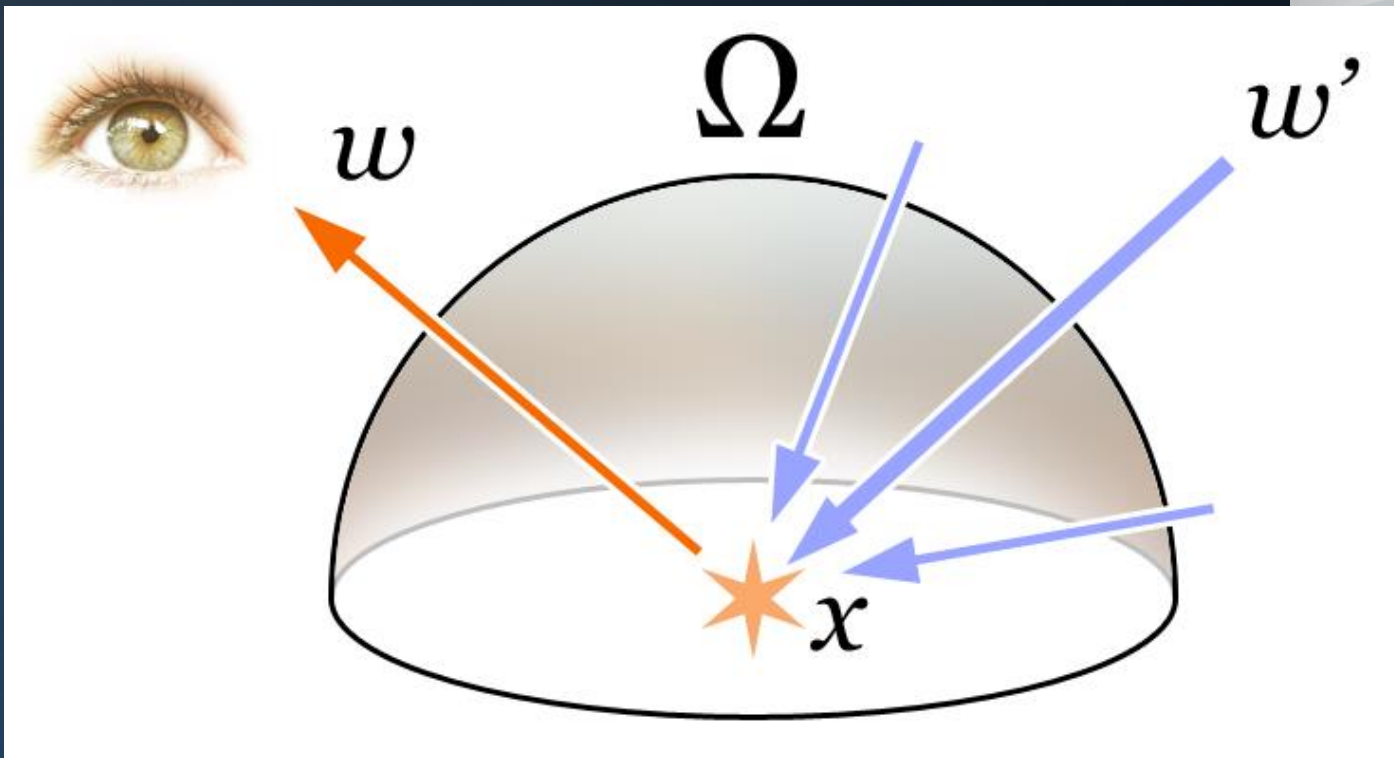
Nature solves this using a “random walk”: a large number of photons travelling through space from lights to sensors.



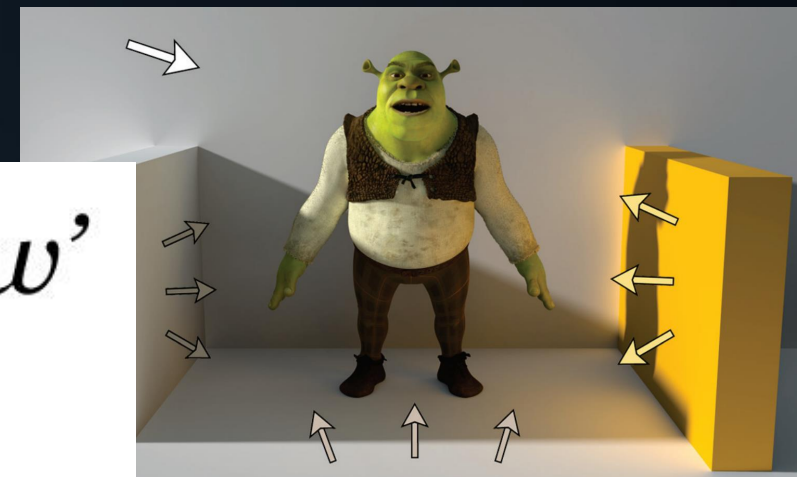
$$L_o(\mathbf{x}, \mathbf{w}) = L_e(\mathbf{x}, \mathbf{w}) + \int_{\Omega} f_r(\mathbf{x}, \mathbf{w}', \mathbf{w}) L_i(\mathbf{x}, \mathbf{w}') (-\mathbf{w}' \cdot \mathbf{n}) d\mathbf{w}'$$



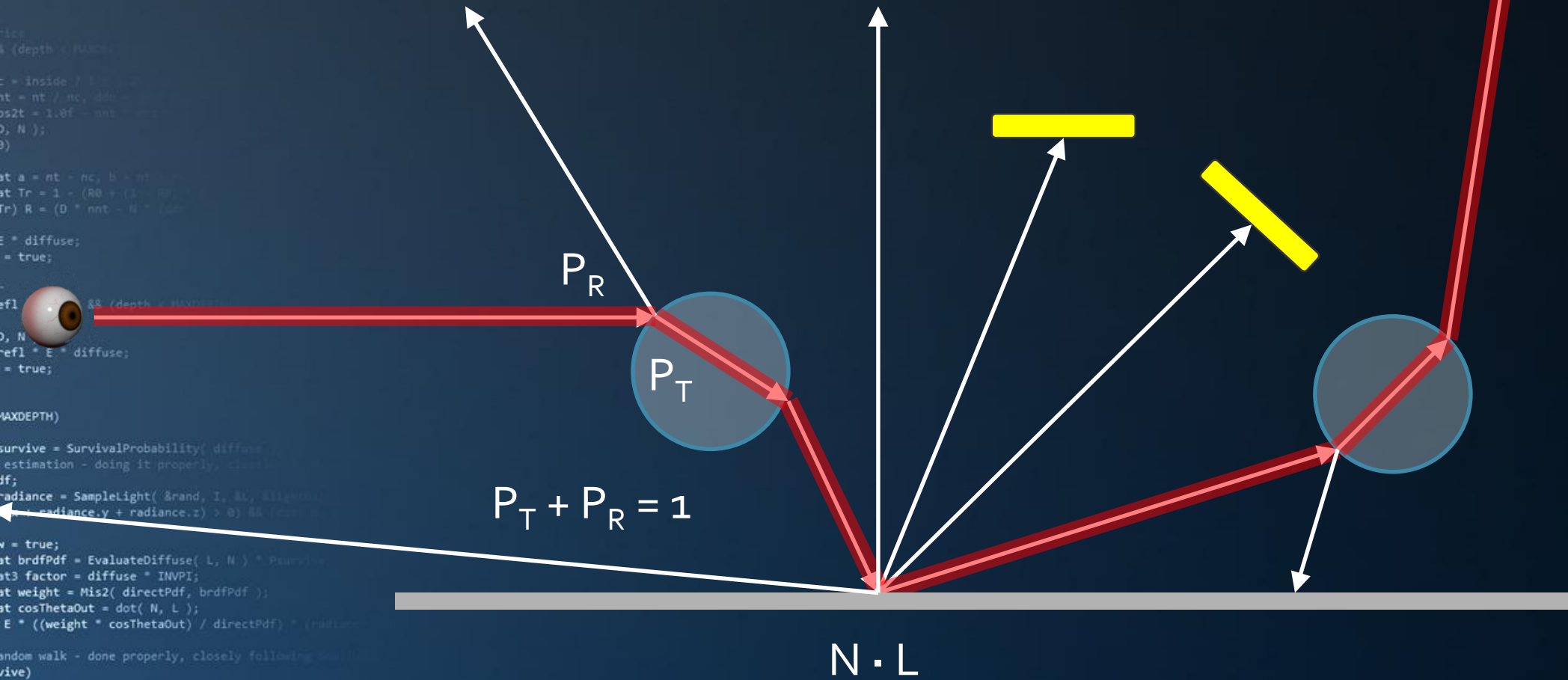
# Physically Based



$$L_o(\mathbf{x}, \mathbf{w}) = L_e(\mathbf{x}, \mathbf{w}) + \int_{\Omega} f_r(\mathbf{x}, \mathbf{w}', \mathbf{w}) L_i(\mathbf{x}, \mathbf{w}') (-\mathbf{w}' \cdot \mathbf{n}) d\mathbf{w}'$$



# Physically Based



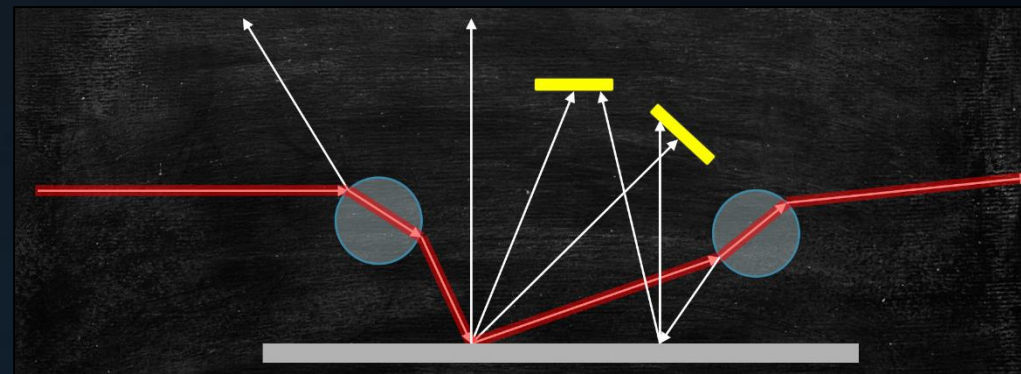


# Physically Based

## Path Tracing

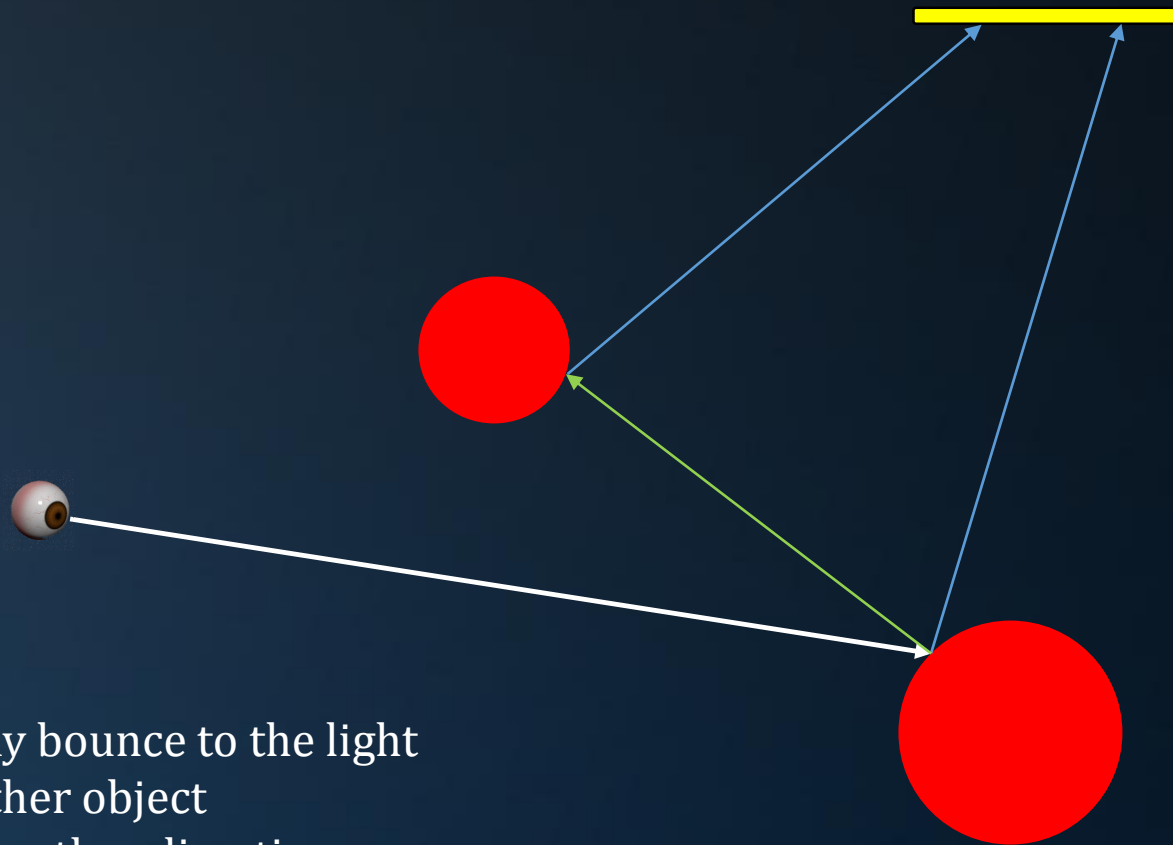
Color Trace( vec3 O, vec3 D )

```
{
    I,N,mat = Intersect( O, D );
    if (mat.IsLight()) return mat.emissive;
    vec3 R = RandomReflection( N );
    BRDF = mat.color;
    return BRDF * dot( N, R ) * Trace( I, R );
}
```



# Physically Based

## Ray Tree



- A path may bounce to the light
- Or to another object
- Or in some other direction





# Physically Based

## Path Tracing

Tracing ‘photons’ backwards, from the camera to the light source, by performing a random walk.

- Instead of splitting the path, we randomly evaluate one branch.
- By using many paths, we explore all possible branches.
- We have the same number of primary rays as we have ‘shadow rays’.

```

    // trace
    if (depth < MAXDEPTH)
    {
        // inside / outside
        int nt = nt / nc; add = 1;
        double r2 = 1.0f - nnt * nnt;
        double D, N );
    }

    // trace
    int a = nt - nc, b = nt - nc;
    double Tr = 1 - (R0 + (1 - R0) * r2);
    double R = (D * nnt - N * (add
    // trace
    E * diffuse;
    = true;
    // trace
    refl + refr)) && (depth < MAXDEPTH)
    {
        D, N );
        refl * E * diffuse;
        = true;
    }

    // trace
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // estimation - doing it properly, closely following walk
    if (survive)
    {
        radiance = SampleLight( &rand, I, &t, &light );
        double x = radiance.x + radiance.y + radiance.z > 0) && (rand.N < 1)
        // trace
        v = true;
        double brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        double factor = diffuse * INVPI;
        double weight = Mis2( directPdf, brdfPdf );
        double cosThetaOut = dot( N, L );
        double E = ((weight * cosThetaOut) / directPdf) * (radiance
    }

    // random walk - done properly, closely following walk
    // survive
    // trace
    double brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    // trace
    // survive

```



# Today's Agenda:

- Deterministic Rendering
- Monte Carlo
- Path Tracing



# INFOGR – Computer Graphics

J. Bikker - April-July 2016 - Lecture 13: “Ground Truth”

## END of “Ground Truth”

next lecture: “Grand Recap”

