

# INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

## Lecture 3: “Ray Tracing (2)”

# Welcome!



# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO



```

k (depth < MAXDEPTH)
{
    c = inside / L * L * L;
    nt = nt / nc; dde = 1.0f - nt;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    );
    at a = nt - nc, b = nt * c;
    at Tr = 1 - (RB + (1 - RB) * b);
    Tr) R = (D * nnt - N * dde)
    E * diffuse;
    = true;
    -
    refl + reflr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    -
    MAXDEPTH)
    survive = SurvivalProbability( diffuse,
    estimation - doing it properly, class
    ff;
    radiance = SampleLight( &rand, I, &L, &light
    .x + radiance.y + radiance.z) > 0) && (cos
    w = true;
    at brdfPdf = EvaluatedDiffuse( L, N ) * Psurv
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (
    random walk - done properly, closely following
    survive)

```



# Recap

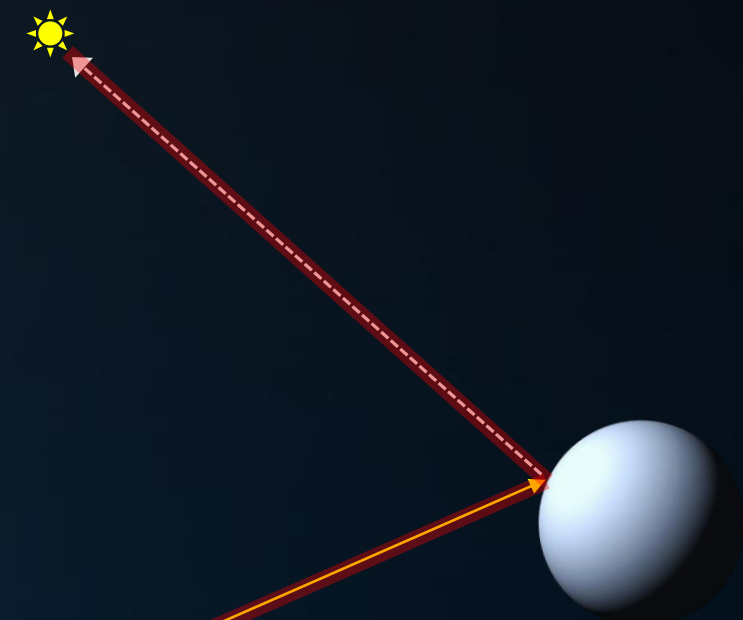
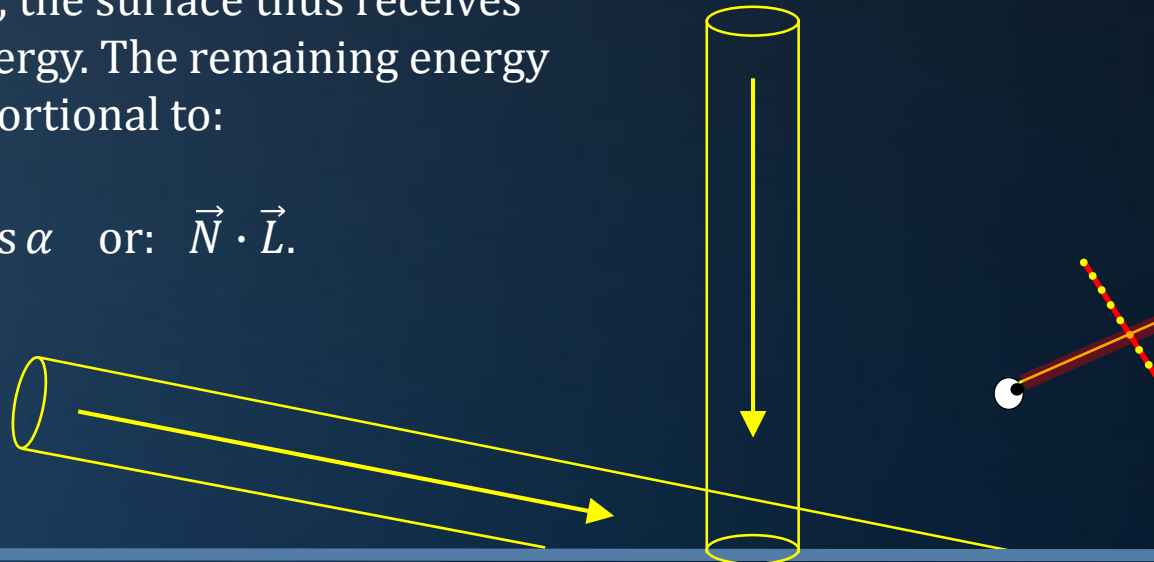
## Transport

Energy arriving at an angle:

A small bundle of light arriving at a surface affects a larger area than the cross-sectional area of the bundle.

Per  $m^2$ , the surface thus receives less energy. The remaining energy is proportional to:

$$\cos \alpha \quad \text{or: } \vec{N} \cdot \vec{L}.$$



# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO



# Normals

Fun fact:  $\begin{pmatrix} A \\ B \\ C \end{pmatrix}$  is the normal.

We Need a Normal

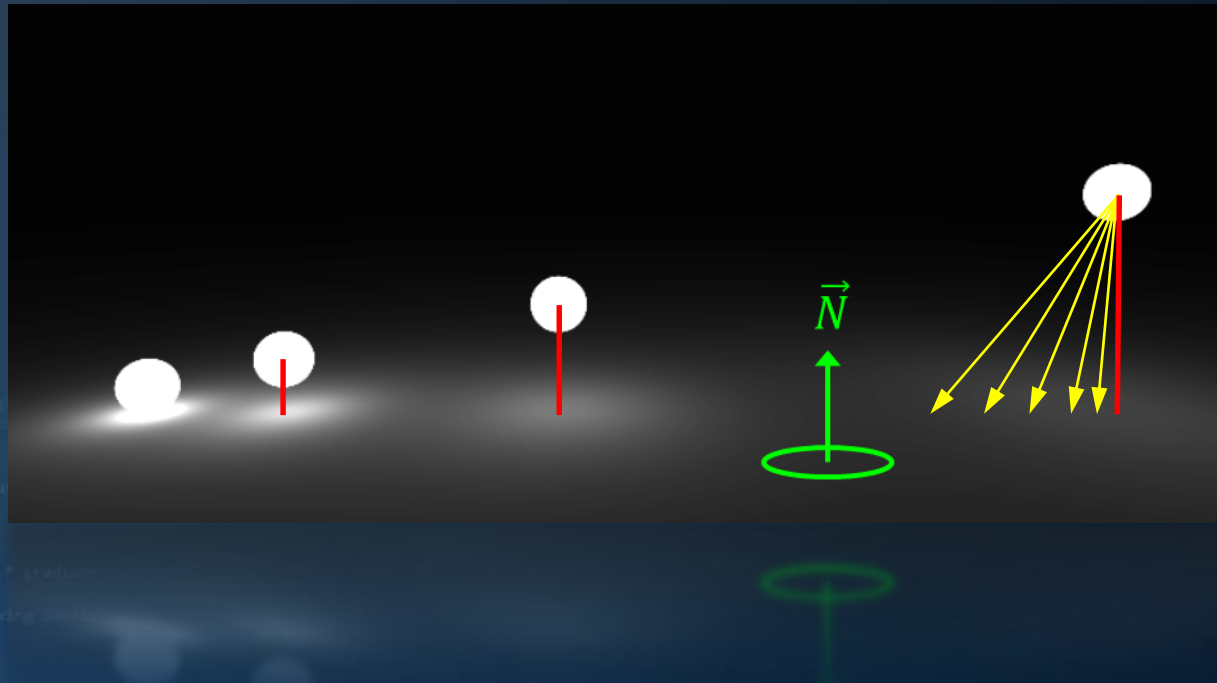
For a plane, we already have the normal.

$$Ax + By + Cz + D = 0 \quad \text{or} \quad (P \cdot \vec{N}) + D = 0$$

A plane is the set of points that are at distance 0 from the plane.

Distance increases when we move away from the plane.

We move away from the plane by moving in the direction of the normal.



Distance attenuation:  $1/r^2$

Angle of incidence:  $N \cdot L$



# Normals

## We Need a Normal

### Question:

How does light intensity relate to scene size?

i.e.: if I scale my scene by a factor 2, what should I do to my lights?

→ Distance attenuation requires scaling light intensity by  $2^2$

→ Scene scale does not affect  $N \cdot L$ .





# Normals

We Need a Normal

Question:

What happens when a light is near the horizon?

→ Angle approaches  $90^\circ$ ;  $\cos \alpha$  approaches 0  
(and so does  $\vec{N} \cdot \vec{L}$ )

→ Light is distributed over an infinitely large surface  
(so, per unit area it becomes 0)

Note: below the horizon,  $\cos \alpha$  becomes negative.

→ Clamp  $\vec{N} \cdot \vec{L}$  to zero.





# Normals

## We Need a Normal

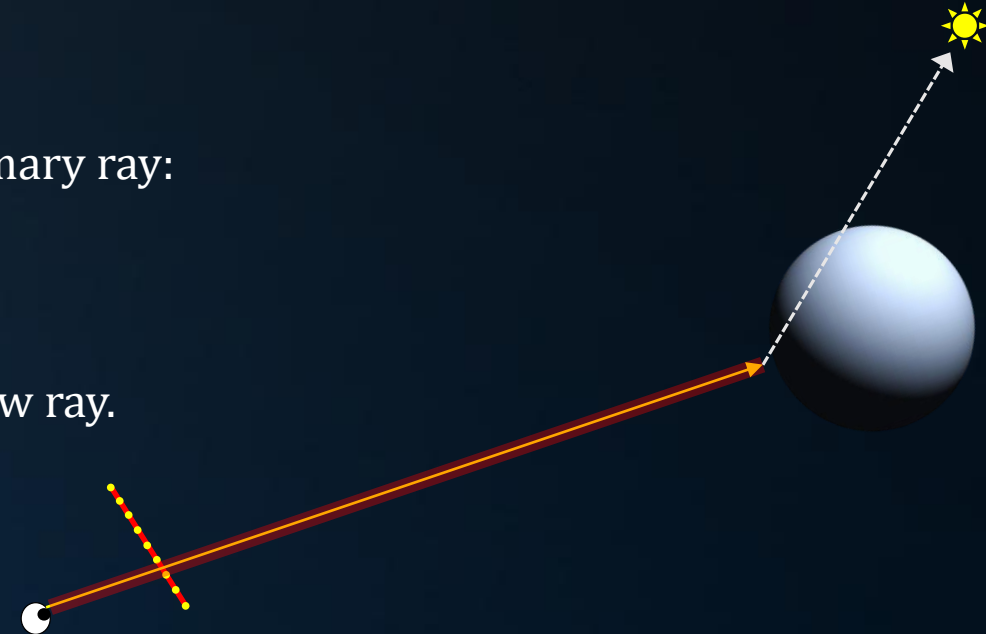
Normals are also used to *prevent* shadow rays.

Situation:

A light source is behind the surface we hit with the primary ray:

$$\vec{N} \cdot \vec{L} < 0$$

In this case, visibility is 0, and we do not cast the shadow ray.



# Normals

We Need a Normal

Normals for spheres:

The normal for a sphere at a point  $P$  on the sphere is parallel to the vector from the center of the sphere to  $P$ .

$$\vec{N}_P = \frac{P - C}{||P - C||}$$



# Normals

We Need a Normal

Normals for spheres:

When a sphere is hit from the inside, we need to *reverse* the normal.

$$\vec{N}_P = \frac{C - P}{||P - C||}$$

How to detect this situation when it is not trivial:

1. Calculate the normal in the usual manner ( $P - C$ );
2. If  $\vec{N}_P \cdot \vec{D}_{ray} < 0$  then  $\vec{N}_P = -\vec{N}_P$ .



# Normals

## Normal Interpolation

Simulating smooth surfaces using normal interpolation:

1. Generate *vertex normals*.

A vertex normal is calculated by averaging the normals of the triangles connected to the vertex and normalizing the result.

2. Interpolate the normals over the triangle.

In a ray tracer, use barycentric coordinates to do this. Normalize the interpolated normal.



# Normals

## Normal Interpolation

Using the interpolated normal:

- Use the interpolated normal in the  $\vec{N} \cdot \vec{L}$  calculation.
- Use the original face normal when checking if a light is visible.

```
...ics
& (depth < MAXDEPTH)
{
    t = inside / 1.5;
    nt = nt / nc; add = add * nt;
    pos2t = 1.0f - nnt;
    D, N );
}

at a = nt - nc; b = nt - nc;
at Tr = 1 - (R0 + (1 - R0) * t);
Tr) R = (D * nnt - N * (add

E * diffuse;
= true;

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

MAXDEPTH)
survive = SurvivalProbabil
estimation - doing it pro
if;
    radance = Sampl
    e.x + radance.y

v = true;
at brdfPdf = Eva
at3 factor = dif
at weight = Mis2
at cosThetaOut =
    E * ((weight *

andom walk - don
ive)

;
at3 brdf = Sampl
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```



# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO









# Assignment P2

Get on Slack!

SL

Ctrl+1

Ctrl+2

Ctrl+3

Ctrl+4

+

infogr2017

jbbikker\_boss

All Threads

CHANNELS (5)

# general

# partnerzoeken

# random

ta

DIRECT MESSAGES

slackbot

jbbikker\_boss (you)

bear

hsveer

hugo

jm

phaneron

sandervanheste

steven

ta\_blockcat

tijmen

#general

173

0

Company-wide announcements and work-based ma...

Yesterday

jeroen13m

10:24 PM

die heb ik in de init

j4stmart

10:24 PM

dan is het vs.txt en fs.txt

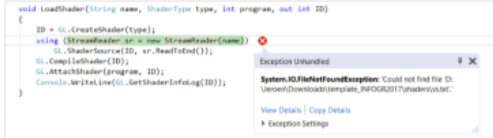
jeroen13m

10:24 PM

ja dat ja

jeroen13m

10:25 PM

uploaded this image: 

j4stmart

10:25 PM

je kan ook gewoon het hele pad van je bestand in je code kopiëren plakken

ruub

10:25 PM

leuk voor de nakijkers idd

jeroen13m

10:26 PM

het pad dat ie hier aangeeft klopt

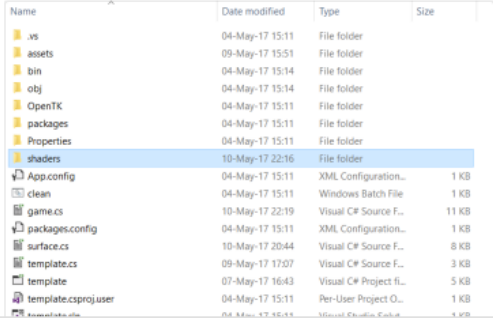
j4stmart

10:26 PM

ohja true

jeroen13m

10:26 PM

uploaded this image: 

+

Message #general

# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO



# Reflections

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        Vec r = inside / 1.5;
        Vec nt = nt / nc, nde = nde / nc;
        double cos2t = 1.0f - nnt * nnt;
        Vec tdir( tdir.x * r, tdir.y * r, tdir.z * r );
        Vec D, N );
    }

    // Russian roulette
    double a = nt - nc, b = nt + nc;
    double p1 = a * a * a * a;
    double p2 = 1 - (p1 + (1 - p1) * 0.5);
    double Tr = 1 - (D * nnt - N * nde);
    double R = (D * nnt - N * nde) * p1;

    // Russian roulette
    double E * diffuse;
    double refl;
    double refl = true;

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

    }

    MAXDEPTH)

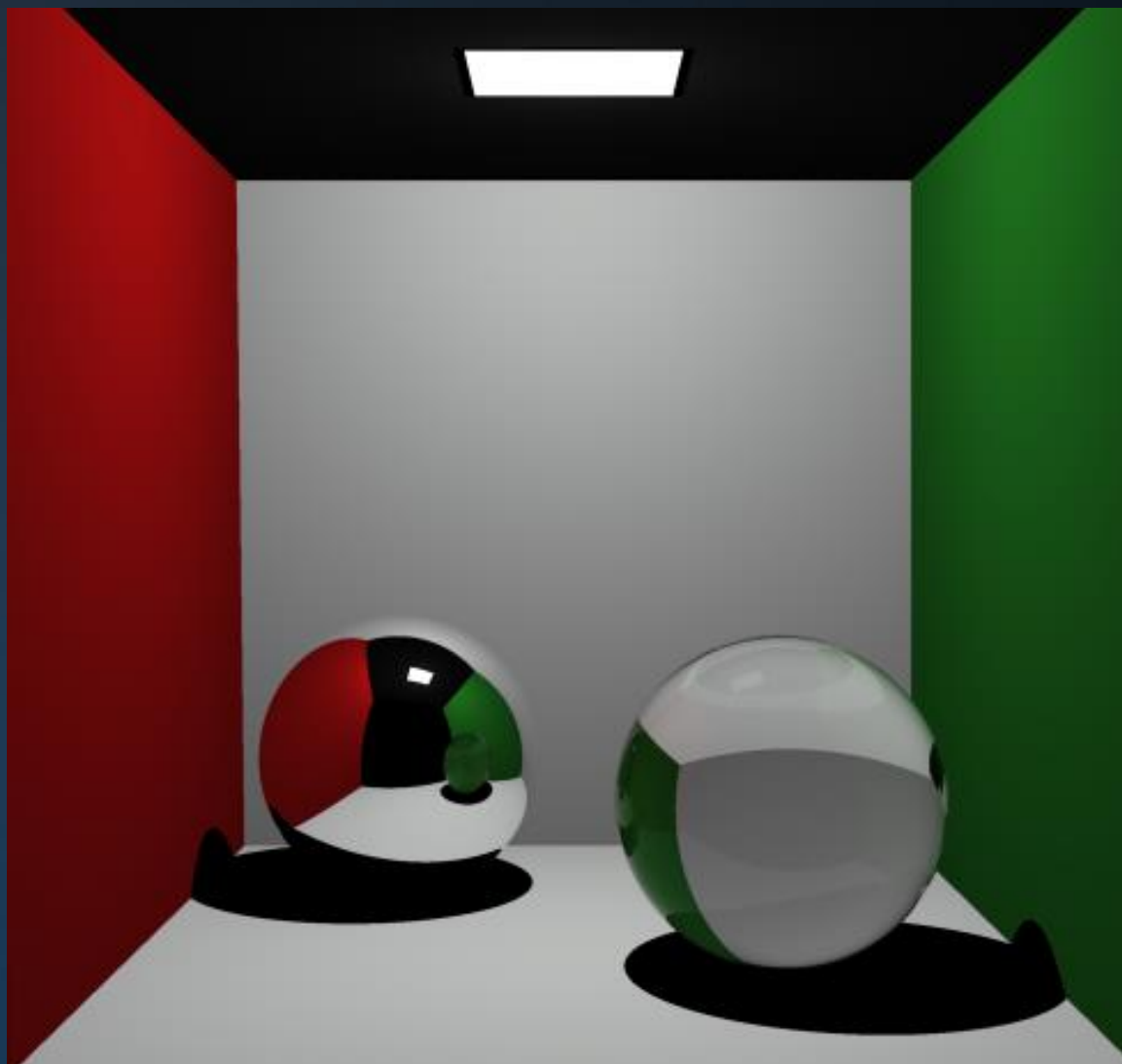
    survive = SurvivalProbability( diffuse );
    // Russian roulette - doing it properly, closely following survival
    if (survive)
    {
        Vec radiance = SampleLight( &rand, I, &L, &align );
        Vec e.x + radiance.y + radiance.z > 0) && (depth < MAXDEPTH)
    {
        double w = true;
        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);

    }

    // Russian roulette - done properly, closely following survival
    survive)

    ;
    double3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    double survive;
    double pdf;
    double n = E * brdf * (dot( N, R ) / pdf);
    double ion = true;

```



# Reflections

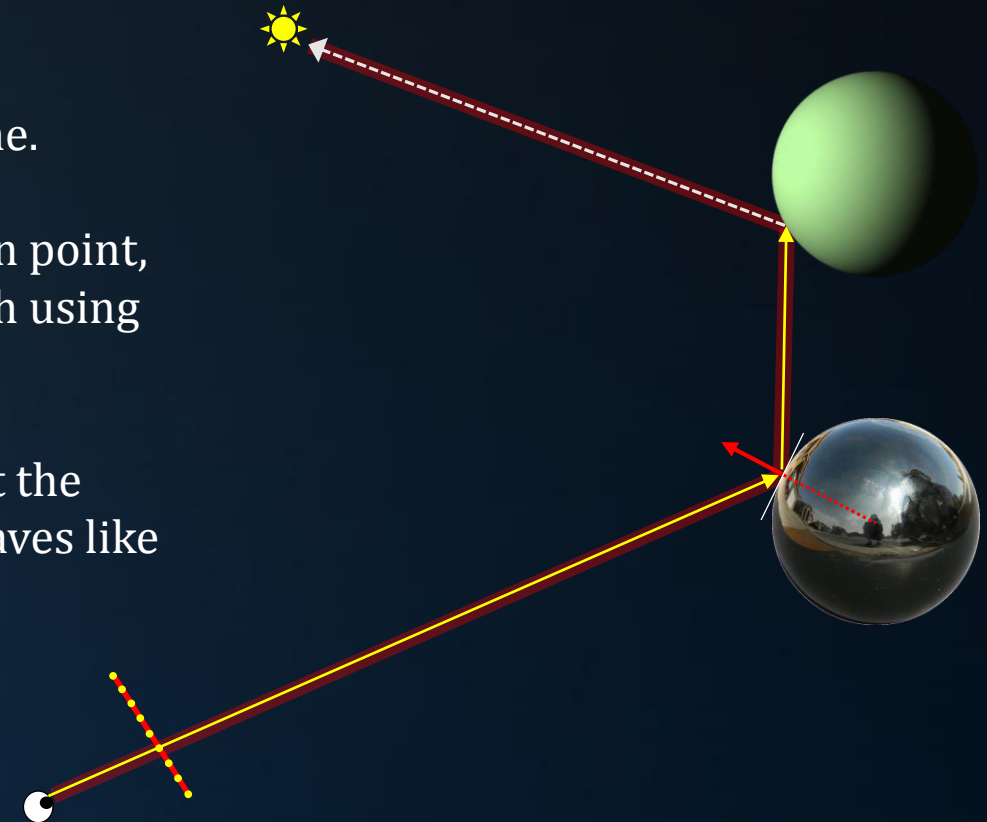
## Light Transport

We introduce a *pure specular* object in the scene.

Based on the normal at the primary intersection point, we calculate a new direction. We follow the path using a *secondary ray*.

At the primary intersection point, we ‘see’ what the secondary ray ‘sees’; i.e. the secondary ray behaves like a primary ray.

We still need a shadow ray at the new intersection point to establish light transport.





# Reflections

## Light Transport

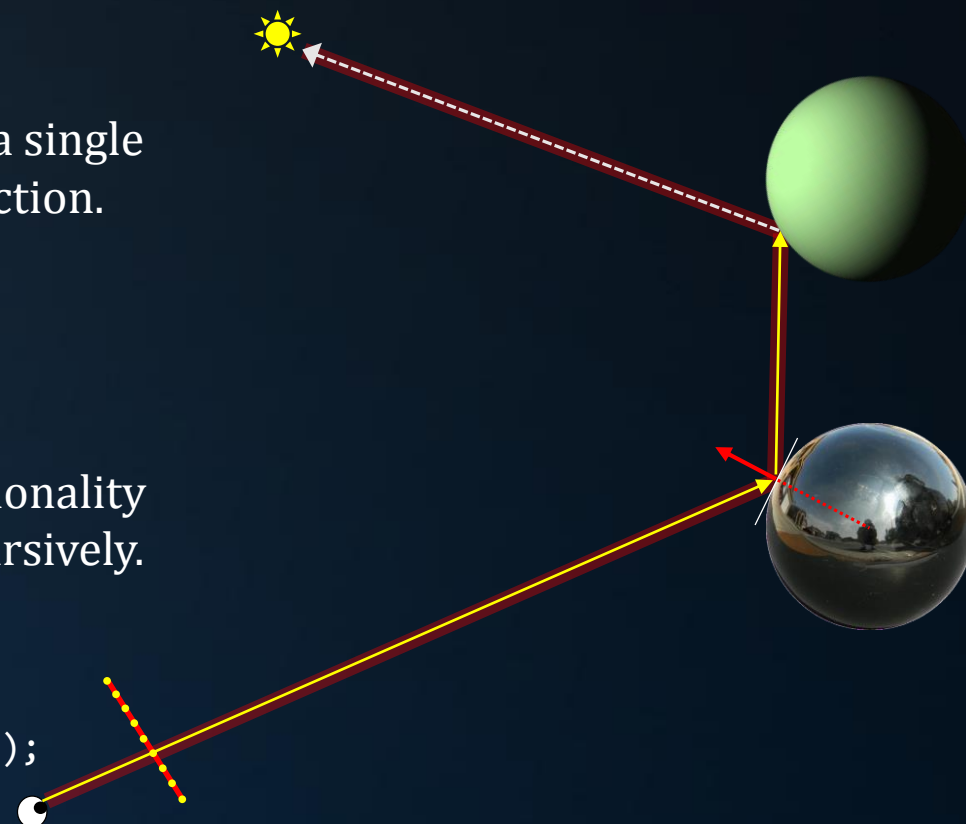
For a pure specular reflection, the energy from a single direction is reflected into a single outgoing direction.

- We do not apply  $\vec{N} \cdot \vec{L}$
- We do apply absorption

Since the reflection ray requires the same functionality as a primary ray, it helps to implement this recursively.

```
vec3 Trace( Ray ray )
```

```
{
    I, N, material = scene.GetIntersection( ray );
    if (material.isMirror)
        return material.color * Trace( ... );
    return DirectIllumination() * material.color;
}
```



```
refl + refr)) && (depth < MAXDEPTH)
```

MAXDEPTH)

```
survive = SurvivalProbability( diff
```

if;

```
if (e.x + radiance.y + radiance.z) > 0)
```

```
w = true;
```

```
at3 factor = diffuse * INVPI;
```

```
float cosThetaOut = dot( N, L );
```

(ive)

Survive;

```
radiance = E * brdf * (dot( N, R ) / pdf);
```





# Reflections



```

    if (depth > MAXDEPTH)
    {
        return Color(0, 0, 0);
    }
    int n = inside / 1000000;
    double r1 = nt / nc; double r2 = nt / nc;
    double cos2t = 1.0f - nnt;
    double D, N;
    D = 0; N = 0;
    for (int i = 0; i < N; i++)
    {
        double a = nt - nc; b = nt;
        double Tr = 1 - (R0 + (1 - R0) * i / N);
        double R = (D * nnt - N * i) / (N - i);
        double E * diffuse;
        double refl = true;
        double refl + refr)) && (depth < MAXDEPTH)
    {
        double D, N;
        double refl * E * diffuse;
        double refl = true;
    }
    double MAXDEPTH)
    {
        double survive = SurvivalProbability;
        double estimation - doing it properly;
        double rad = SampleLight(
            double x + radiance.y + radiance.z;
        }
        double w = true;
        double brdfPdf = EvaluateDiffuse(
        double t3 factor = diffuse * I;
        double weight = Mis2( direct
        double cosThetaOut = dot( N,
        double E * ((weight * cosThetaOut) / directPdf) * radiance;
    }
    double random walk - done properly, closely following the path
    double survive)
    {
        double t3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
        double survive;
        double pdf;
        double n = E * brdf * (dot( N, R ) / pdf);
        double estimation = true;
    }

```



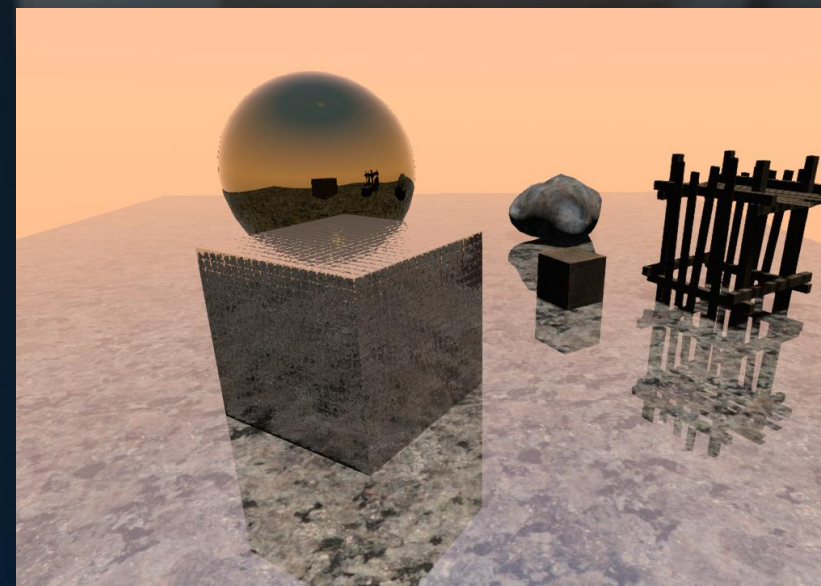
# Reflections

## Partially Reflective Surfaces

We can combine pure specular and diffuse properties.

Situation: our material is only 50% reflective.

In this case, we send out the reflected ray, and multiply its yield by 0.5. We also send out a shadow ray to get direct illumination, and multiply the received light by 0.5.





# Reflections

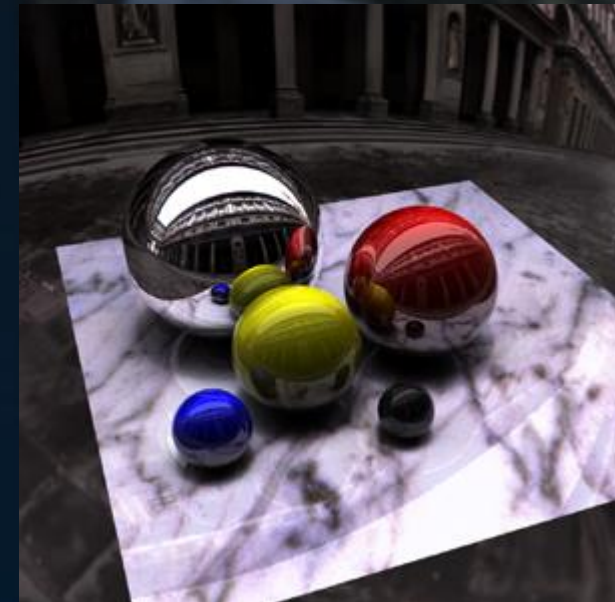
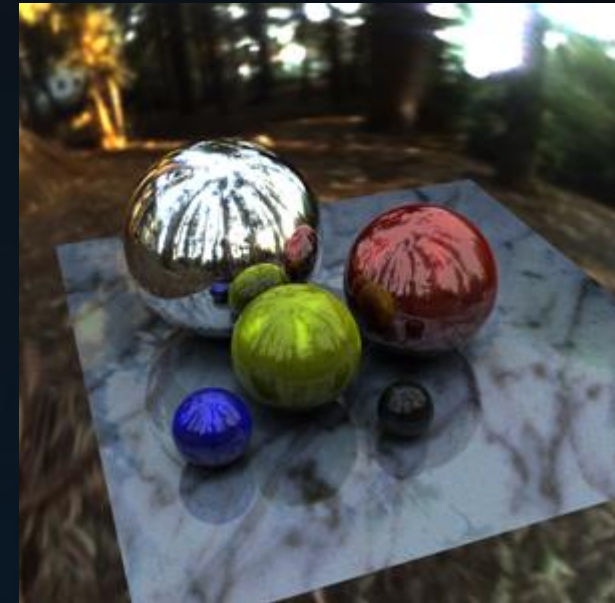
## Reflecting a HDR Sky

A dark object can be quite bright when reflecting something bright.

E.g., a bowling ball, pure specular, color = (0.01, 0.01, 0.01); reflecting a ‘sun’ stored in a HDR skydome, color = (100, 100, 100).

For a collection of HDR probes, visit Paul Debevec’s page:

<http://www.pauldebevec.com/Probes>



# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO



# Recursion

## Whitted-style Ray Tracing, Pseudocode

```
Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    return DirectIllumination( I, N ) * mat.diffuseColor;
}
```

```
Color DirectIllumination( vec3 I, vec3 N )
{
    vec3 L = lightPos - I;
    float dist = length( L );
    L *= (1.0f / dist);
    if (!IsVisible( I, L, dist )) return BLACK;
    float attenuation = 1 / (dist * dist);
    return lightColor * dot( N, L ) * attenuation;
}
```

### Todo:

- Implement IntersectScene
- Implement IsVisible



# Recursion

## Whitted-style Ray Tracing, Pseudocode

```

Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    if (mat.isMirror())
    {
        return Trace( I, reflect( D, N ) ) * mat.diffuseColor;
    }
    else
    {
        return DirectIllumination( I, N ) * mat.diffuseColor;
    }
}

```

Todo:

Handle partially reflective surfaces.



# Recursion

## Whitted-style Ray Tracing, Pseudocode

```

Color Trace( vec3 O, vec3 D )
{
    I, N, mat = IntersectScene( O, D );
    if (!I) return BLACK;
    if (mat.isMirror())
    {
        return Trace( I, reflect( D, N ) ) * mat.diffuseColor;
    }
    else if (mat.IsDielectric())
    {
        f = Fresnel( ... );
        return (f * Trace( I, reflect( D, N ) )
            + (1-f) * Trace( I, refract( D, N, ... ) ) ) * mat.DiffuseColor;
    }
    else
    {
        return DirectIllumination( I, N ) * mat.diffuseColor;
    }
}

```

### Todo:

- Implement reflect
- Implement refract
- Implement Fresnel
- Cap recursion





## Spheres: pure specular

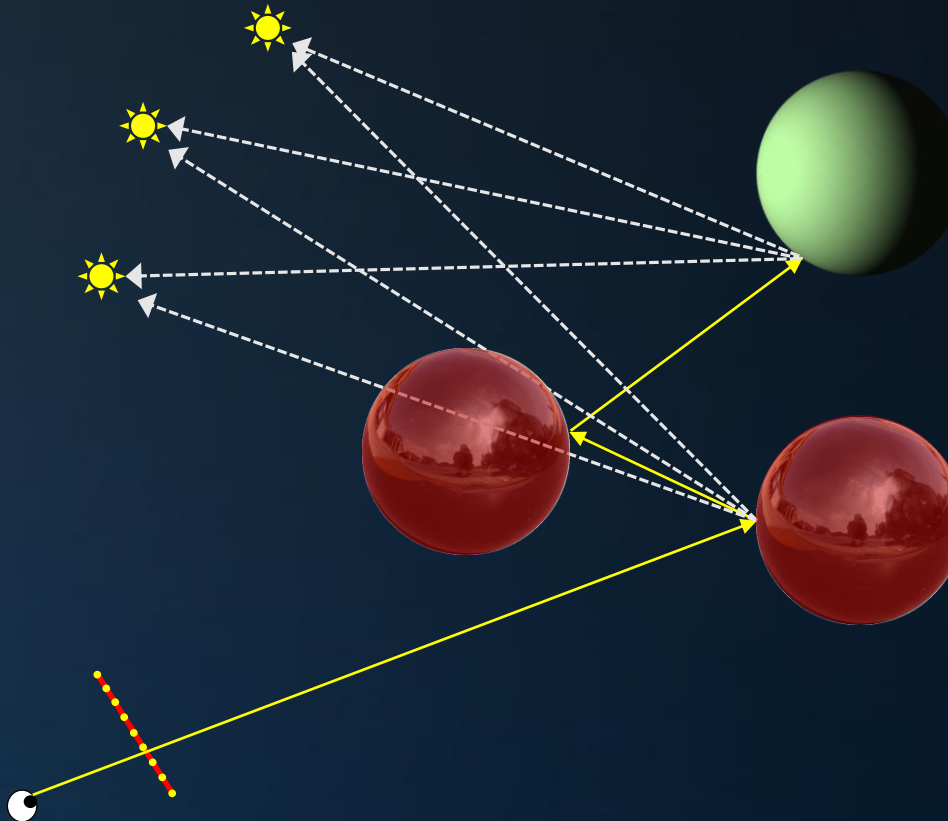


# Recursion

Spheres: 50% specular

```

    // Ray-sphere intersection
    float t = inside / 1.5;
    float nt = nt / nc;
    float nnt = nt * nt;
    float cos2t = 1.0f - nnt;
    float D, N;
    // ...
    float a = nt - nc, b = nt + nc;
    float Tr = 1 - (R0 + (1 - R0) * cos2t);
    float R = (D * nnt - N * cos2t);
    // ...
    E * diffuse;
    // ...
    refl + refr)) && (depth < MAXDEPTH)
    // ...
    D, N);
    refl * E * diffuse;
    // ...
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // ...
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    // ...
    e.x + radiance.y + radiance.z > 0) && (cosThetaOut > 0)
    // ...
    v = 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
    survive)
    // ...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    // ...
    survive;
    // ...
    pdf;
    // ...
    n = E * brdf * (dot( N, R ) / pdf);
    // ...
    sion = true;
  
```



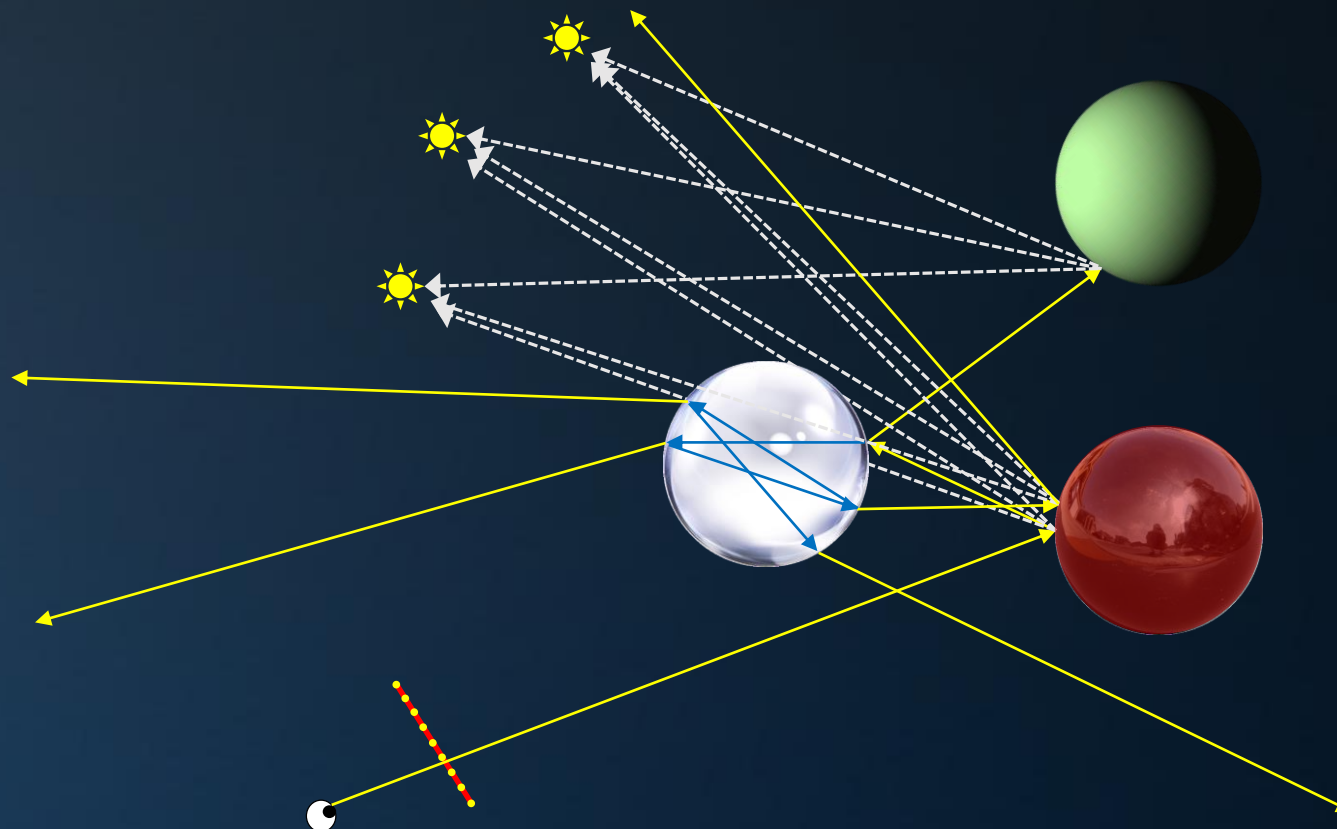
# Recursion

Spheres: one 50% specular, one glass sphere

```

ices
    & (depth < MAXDEPTH)
    {
        t = inside / 1.5;
        nt = nt / nc; nnt = nnt / nc;
        cos2t = 1.0f - nnt;
        D, N );
    }
    {
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) * t);
        Tr) R = (D * nnt - N * (cos2t > 0.5 ? 1 : -1));
    }
    {
        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) && (depth < MAXDEPTH)
    {
        v = 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
        survive)
    }
    {
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    }

```





# Recursion

```
...ics
& (depth < MAXD);
...
t = inside / 1.5;
nt = nt / nc;
os2t = 1.0f - nnt;
D, N );
...
at a = nt - nc, b = nt;
at Tr = 1 - (RB + (1 - RB) * t);
Tr) R = (D * nnt - N * (os2t
...
E * diffuse;
= true;
...
efl + refr)) && (depth < MAXDEPTH);
D, N );
refl * E * diffuse;
= true;
```



```
MAXDEPTH
survive
estimat
if;
-radiance
e.x + re
w = true
at brdff
at3 fact
at weigh
at cosTh
E * ((
andom wa
ive)
```

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



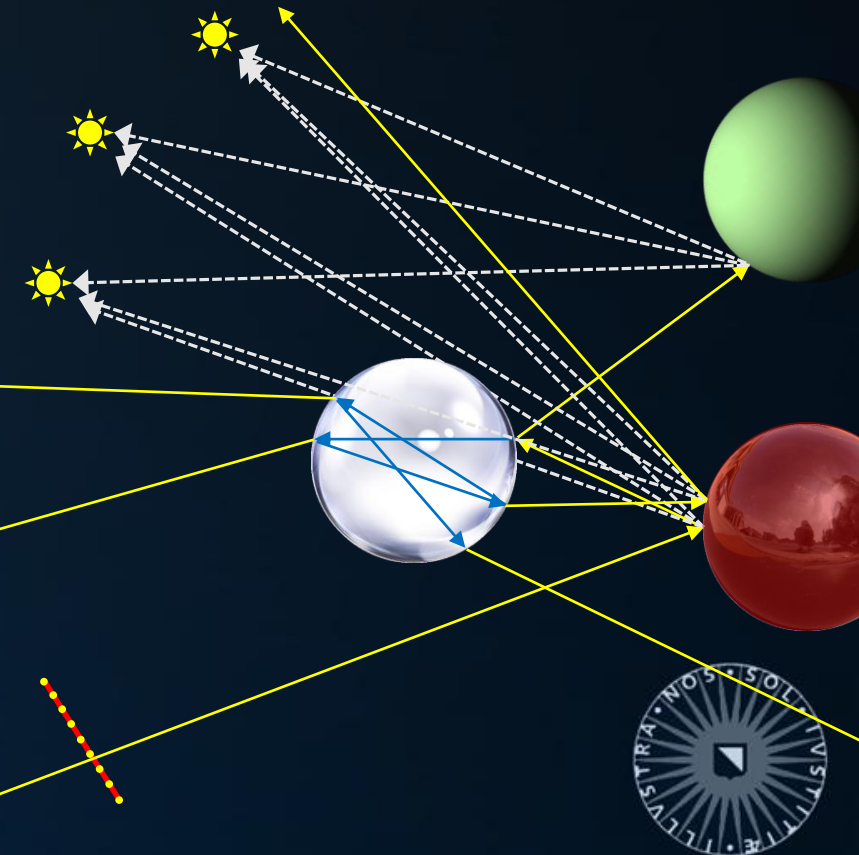
# Recursion

## Ray Tree

Recursion, multiple light sampling and path splitting in a Whitted-style ray tracer leads to a structure that we refer to as the *ray tree*.

All energy is ultimately transported by a single primary ray.

Since the energy does not increase deeper in the tree (on the contrary), the average amount of energy transported by rays decreases with depth.





# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO



# Shading

## Diffuse Material

A diffuse material scatters incoming light in all directions.

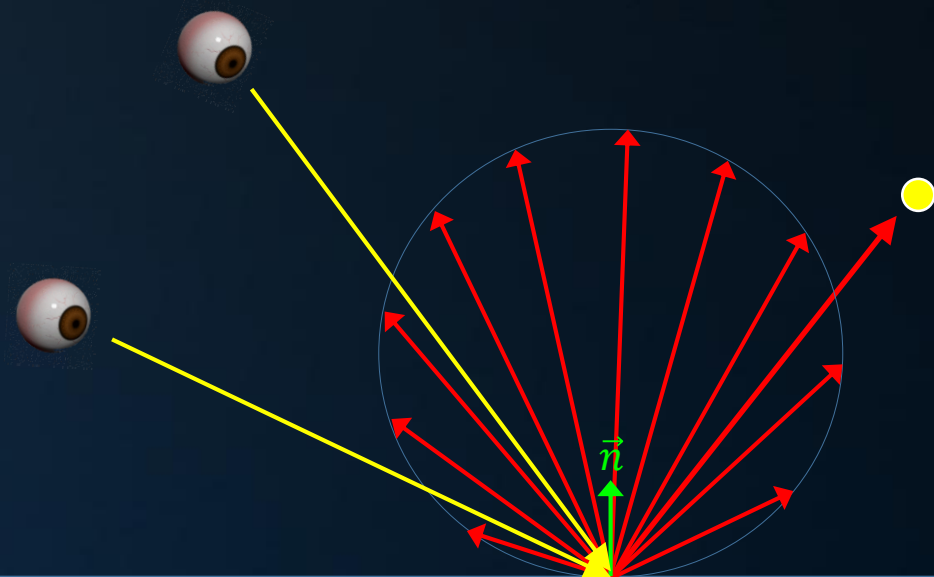
A diffuse material appears the same regardless of eye position.

Incoming:  $E_{light} * \frac{1}{dist^2} * \vec{N} \cdot \vec{L}$

Absorption:  $C_{material}$

Reflection:  $(\vec{V} \cdot \vec{N})$  | *terms cancel out.*

Eye sees:  $\frac{1}{(\vec{V} \cdot \vec{N})}$

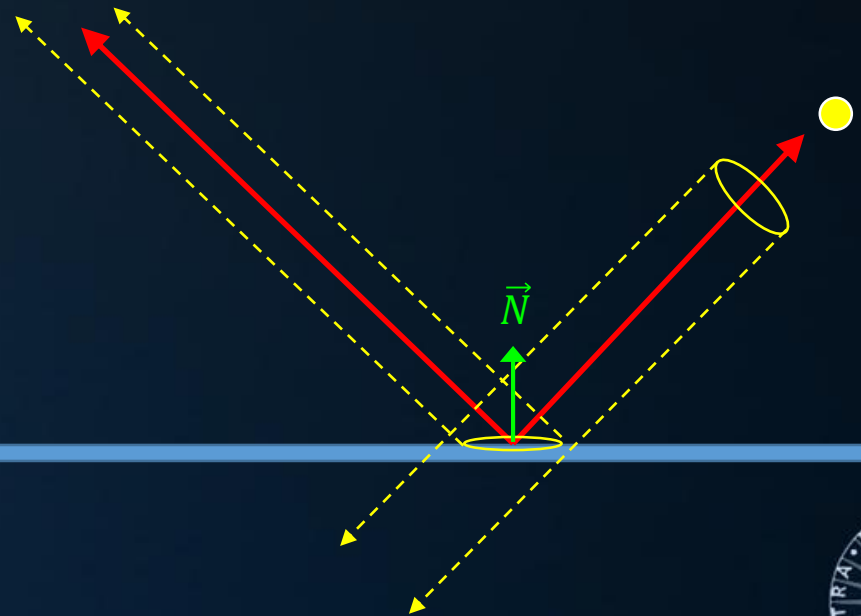




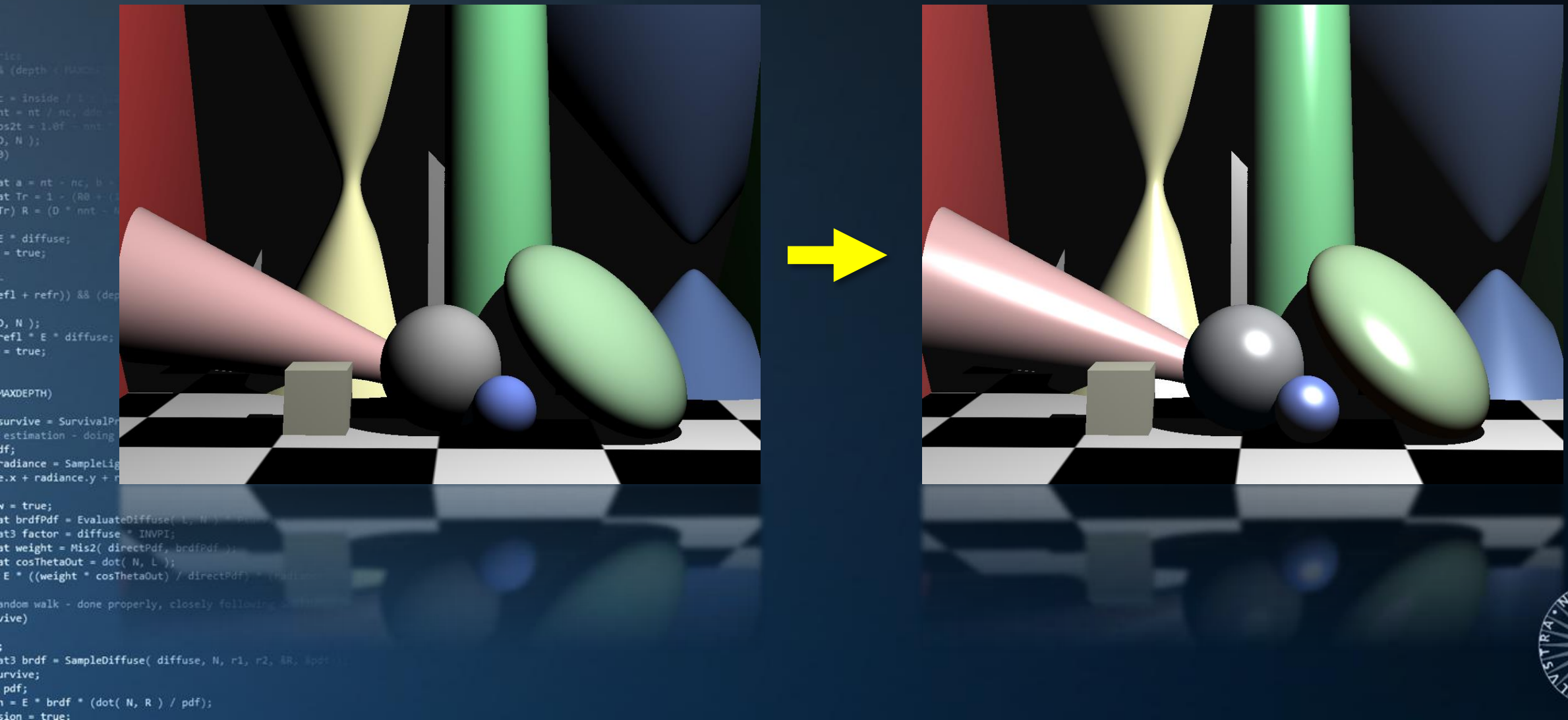
## Specular Material

```

100
101 (depth < MAXDEPTH)
102 {
103     nc = inside / 1.0f * 0.5f;
104     nt = nt / nc; ddx = dot(N, N);
105     cos2t = 1.0f - nt * nt; r = sqrt(cos2t);
106     D, N );
107     0);
108
109     at a = nt - nc, b = nt * nc;
110     at Tr = 1 - (RB + (1 - RB) * r);
111     Tr) R = (D * nt - N * ddx) *
112
113     E * diffuse;
114     = true;
115
116     -
117     refl + refr)) && (depth < MAXDEPTH)
118
119     D, N );
120     -refl * E * diffuse;
121     = true;
122
123     -
124     MAXDEPTH)
125
126     survive = SurvivalProbability( diffuse,
127     estimation - doing it properly, clearly
128     diff;
129     radiance = SampleLight( &rand, I, &I, &I);
130     .x + radiance.y + radiance.z) > 0) && (m
131
132     w = true;
133     at brdfPdf = EvaluateDiffuse( L, N ) * Pa
134     at3 factor = diffuse * INVPI;
135     at weight = Mis2( directPdf, brdf
136     at cosThetaOut = dot( N, L );
137     E * ((weight * cosThetaOut) / directPdf)
138
139     random walk - done properly, closely follow
140     (survive)
141
142     ;
143     at3 brdf = SampleDiffuse( diffuse, N, r1,
144     survive;
145     pdf;
146     n = E * brdf * (dot( N, R ) / pdf);
147     sion = true;
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
1
```



# Shading



# Shading

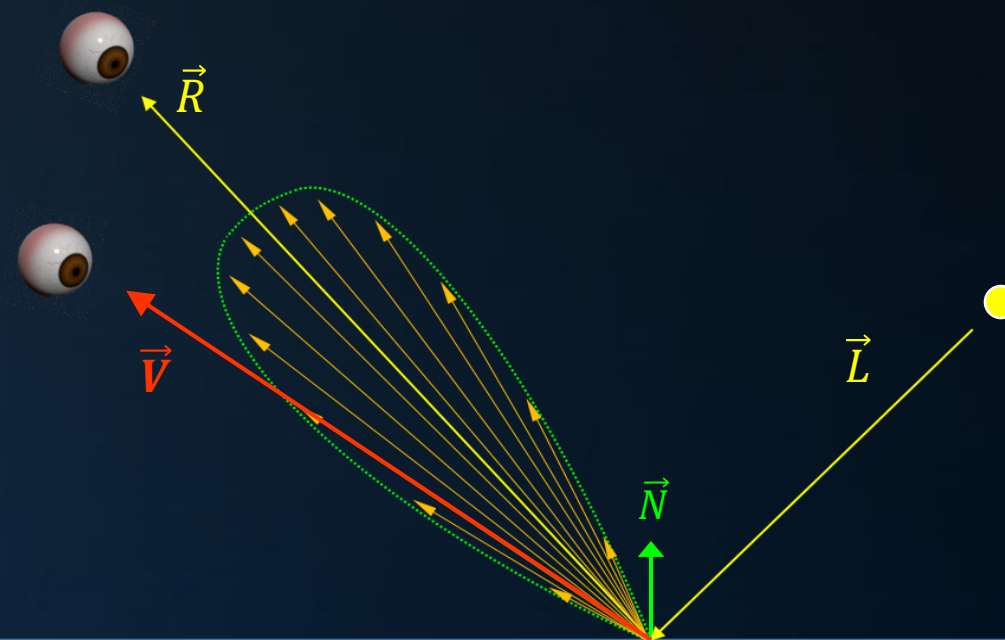
## Glossy Material

A glossy material reflects *most* light along the reflected vector.

$$\vec{R} = \vec{L} - 2(\vec{L} \cdot \vec{N})\vec{N}$$

For other directions, the amount of energy is:

$(\vec{V} \cdot \vec{R})^\alpha$ , where exponent  $\alpha$  determines the specularity of the surface.



# Shading

## Phong Shading

Complex materials can be obtained by blending diffuse and glossy.

$$I = C_{material} * \left( (1 - f_{spec}) (\vec{N} \cdot \vec{L}) + f_{spec} (\vec{V} \cdot \vec{R})^\alpha \right)$$

where

$\alpha$  is the specularity of the glossy reflection;

$f_{spec}$  is the glossy part of the reflection;

$1 - f_{spec}$  is the diffuse part of the reflection.

Note that the glossy reflection *only* reflects light sources.



# Today's Agenda:

- Recap
- Normals
- Assignment P2
- Reflections
- Recursion
- Shading models
- TODO





# TODO

## Limitations of Whitted-style

```

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

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

    E * diffuse;
    = true;

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

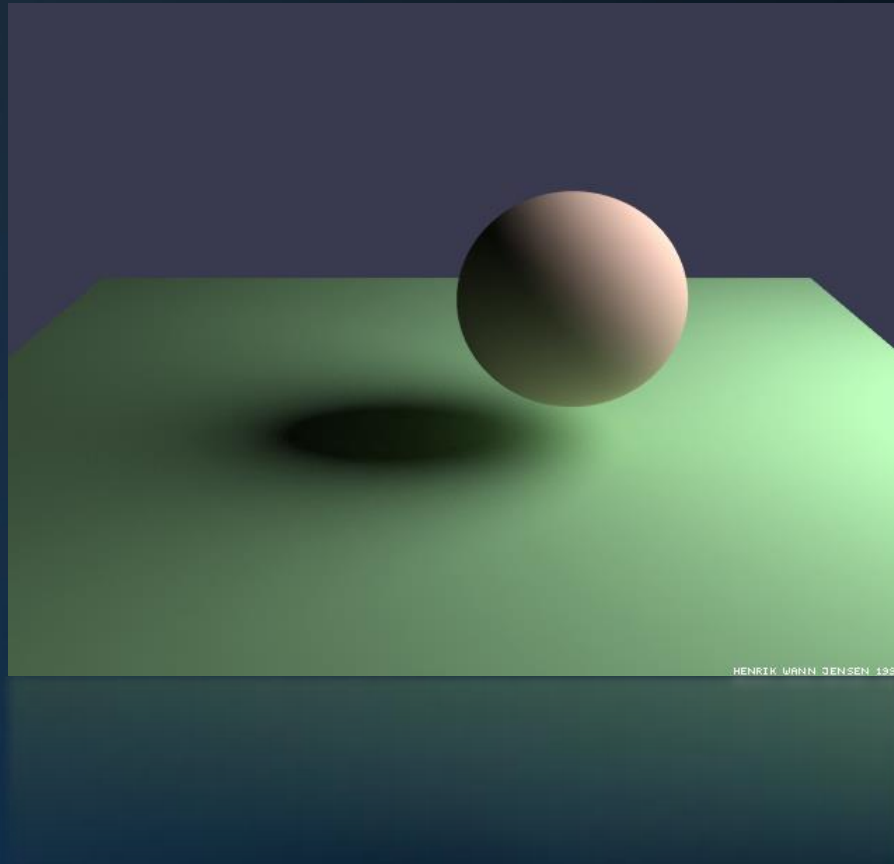
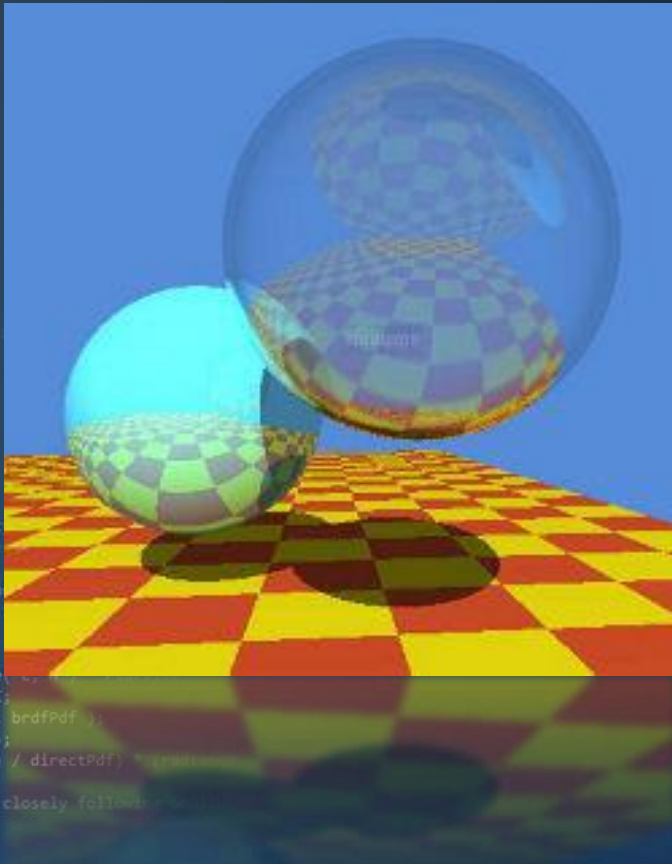
    MAXDEPTH)

    survive = SurvivalProbability(
    estimation - doing it properly
    if;
    radiance = SampleLight( &rand,
    e.x + radiance.y + radiance.z);

    v = true;
    at brdfPdf = EvaluateDiffuse( E, N, r1, r2, &rand, &pdf);
    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 a random walk
    survive)

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



# TODO

## Limitations of Whitted-style

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        nt = inside / 1.5;
        nc = nt / (1 - nt);
        ndc2 = 1.0f - nt * nt;
        D = N;
        N = D * N;
        D = D * D;
        R = (D * nt - N * (1 - nt));
        E * diffuse;
        = true;
    }
    if (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, &L, &light );
    e.x + radiance.y + radiance.z) > 0) && (rand < 1)
    v = 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
    rive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



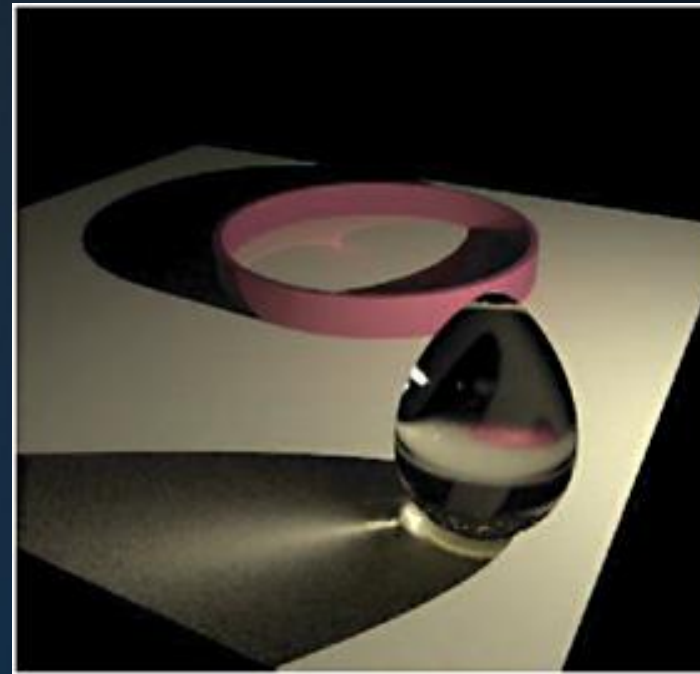
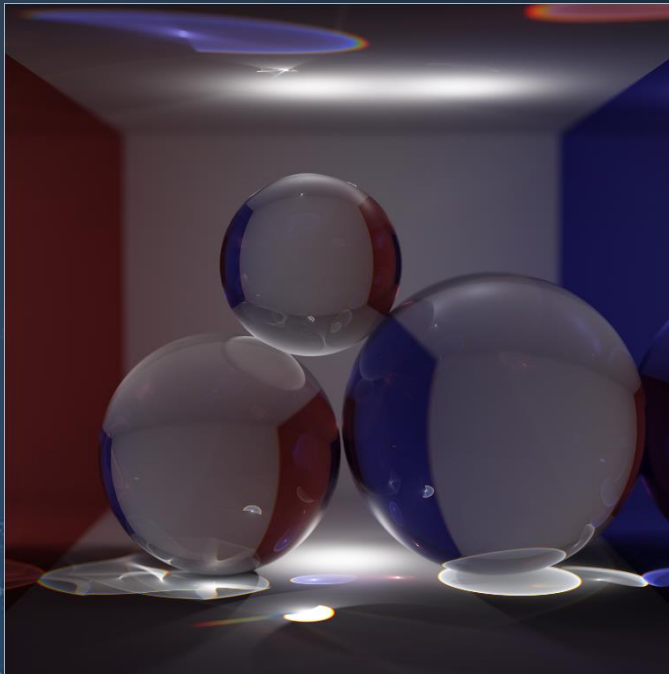
# TODO

## Limitations of Whitted-style

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        nt = inside / 1.5;
        nc = nt / (1 - nt);
        n1t = nt / (1 - nt);
        n2t = 1.0f - n1t;
        D = N;
        N = N * nc;
        R = (D * n1t + N * n2t);
        E = diffuse;
        refl = true;
        refl + refr) && (depth < MAXDEPTH)
        D, N);
        refl * E * diffuse;
        = true;
        MAXDEPTH)
        survive = SurvivalProbability(
        estimation - doing it properly
        if;
        radiance = SampleLight( &rand
        e.x + radiance.y + radiance.z
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N, r1, r2, &R, &pdf );
        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 walls (survive)
        survive)
        ;
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;

```



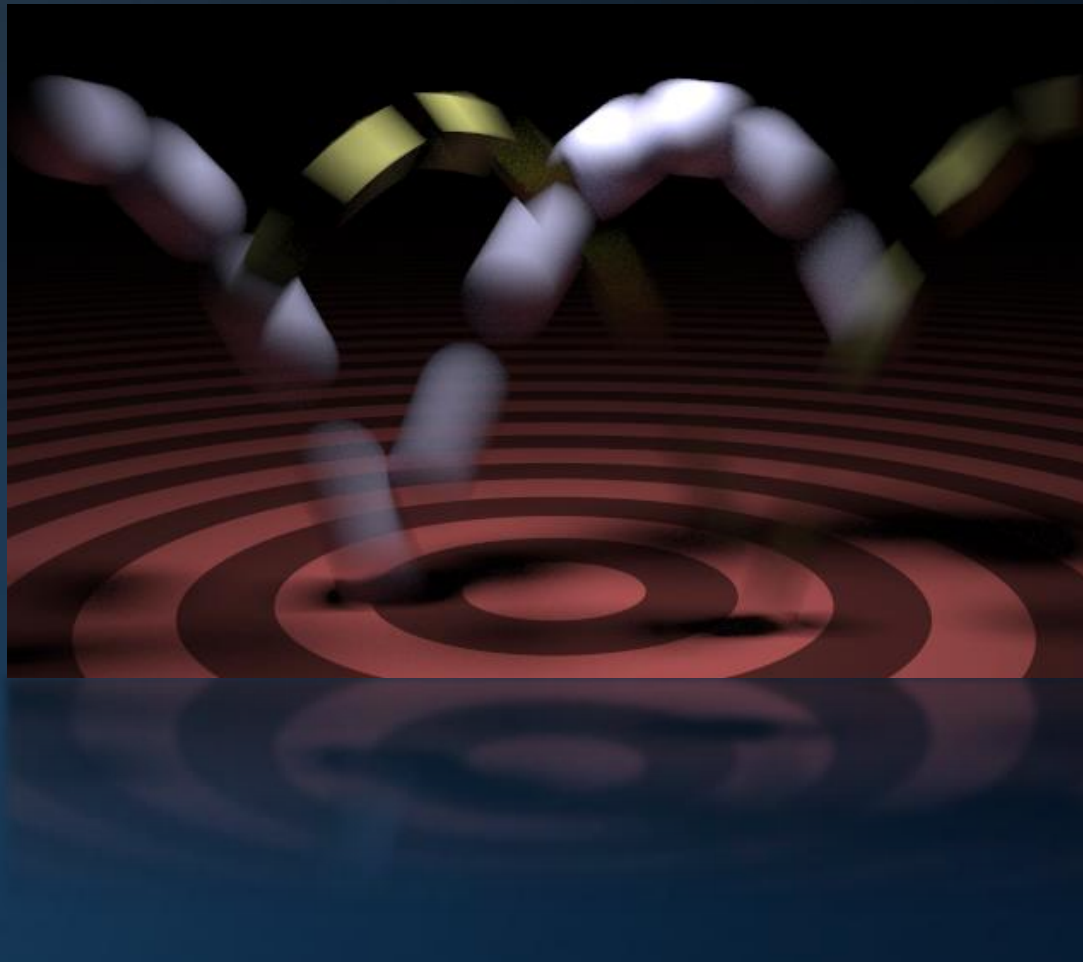
# TODO

## Limitations of Whitted-style

```

    if (depth < MAXDEPTH)
    {
        // Inside the sphere
        // Ray-sphere intersection
        double nc = inside / 1.5;
        double nt = nt / nc, nde = nde / nc;
        double os2t = 1.0f - nnt * nnt;
        double o, N;
        if (os2t < 0)
            return 0;
        double a = nt - nc, b = nt + nc;
        double Tr = 1 - (RB + (1 - RB) * os2t);
        double R = (D * nnt - N * (nde - Tr));
        double E = diffuse;
        double refl = true;
        // Refractive index
        double refl + refr) && (depth < MAXDEPTH)
        {
            double D, N;
            double refl * E * diffuse;
            double refl = true;
        }
    }
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    // estimation - doing it properly, closely following
    if (
        radiance = SampleLight( &rand, I, &L, &align );
        e.x + radiance.y + radiance.z > 0) && (depth <
    w = 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
    survive)
    ;
    double brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    ion = true;

```

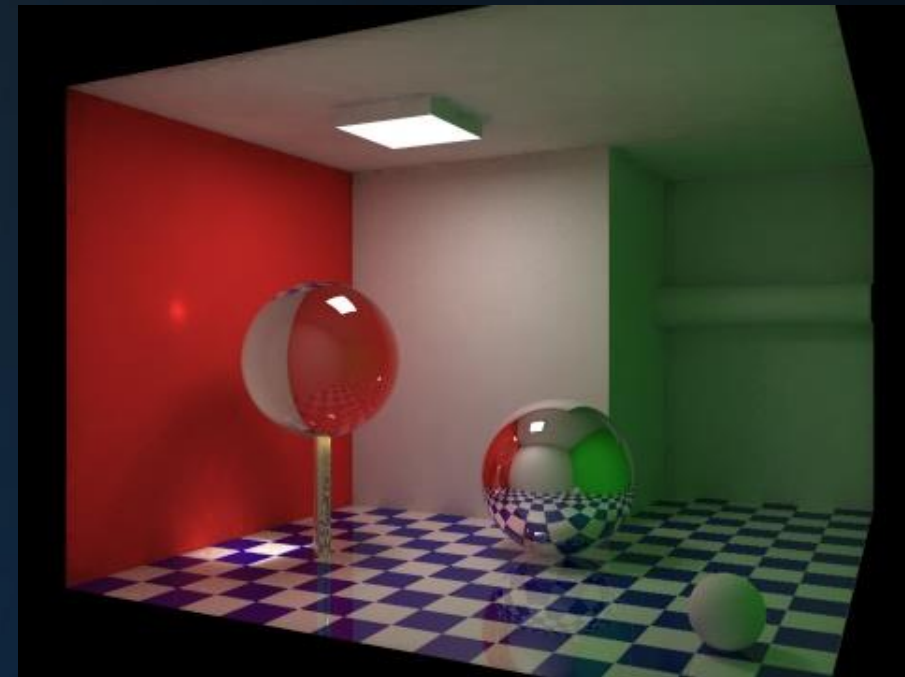
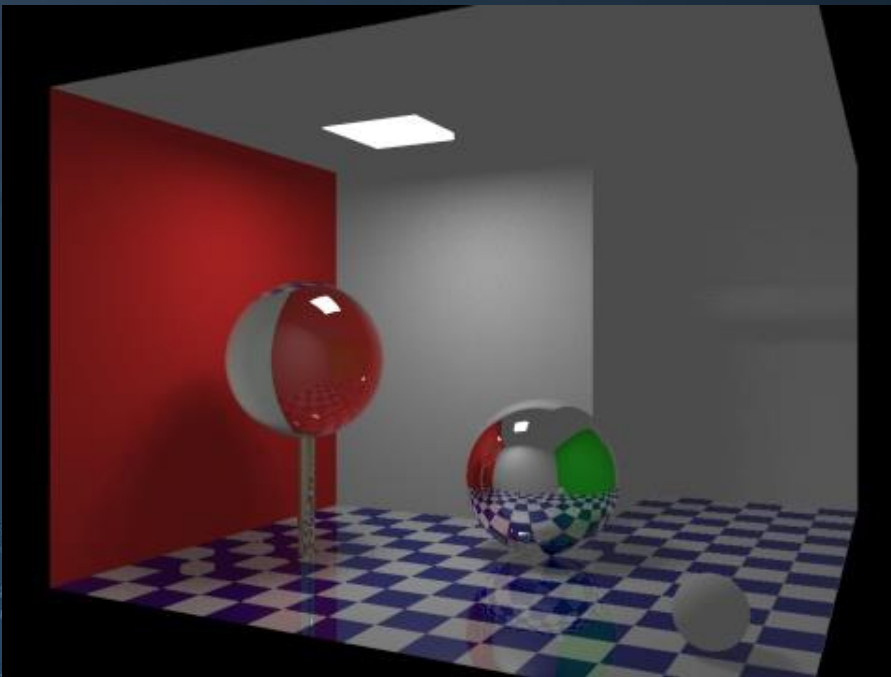




# TODO

## Limitations of Whitted-style

```
...ics  
    & (depth < MAXDEPTH)  
    {  
        t = inside / 1.5f; // 1.5f is a magic number  
        nt = nt / nc; nnt = nnt / nc;  
        cos2t = 1.0f - nnt; // nnt is the probability of a  
        D, N );  
        R = (D * nnt + N * cos2t);  
        E * diffuse;  
        = true;  
        refl + refr)) && (depth < MAXDEPTH)  
        D, N );  
        refl * E * diffuse;  
        = true;  
        survive = SurvivalProbability(0.5f);  
        estimation - doing it properly  
        ff;  
        radiance = SampleLight(0.5f);  
        e.x + radiance.y + radiance.z;  
        v = true;  
        at brdfPdf = EvaluateDiffuse( diffuse, N, r1, r2, &R, &pdf );  
        at3 factor = diffuse * INVPI;  
        at weight = Mis2( directPdf, brdfPdf );  
        at cosThetaOut = dot( N, L );  
        E * ((weight * cosThetaOut) / directPdf);  
        random walk - done properly, closely following the path  
        rive)  
        ;  
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );  
        survive;  
        pdf;  
        n = E * brdf * (dot( N, R ) / pdf);  
        sion = true;
```





# INFOGR – Computer Graphics

Jacco Bikker & Debabrata Panja - April-July 2017

END OF lecture 3: “Ray Tracing (2)”

*Next lecture: “Accelerate”*

