

シンプルな4次元パストレーサーによるレンダリング

A simple path tracer for 4D scene rendering

<https://github.com/shizuo-kaji/smallpt4d>

九州大学
鍛治 静雄

- ▶ このワークショップ中に開設されたポータルサイト

<https://matheduvr.github.io>

- ▶ Slack (一応招待制)

matheduvr.slack.com

まず3次元の話

3次元シーンから2次元の絵を得る

レンダリング

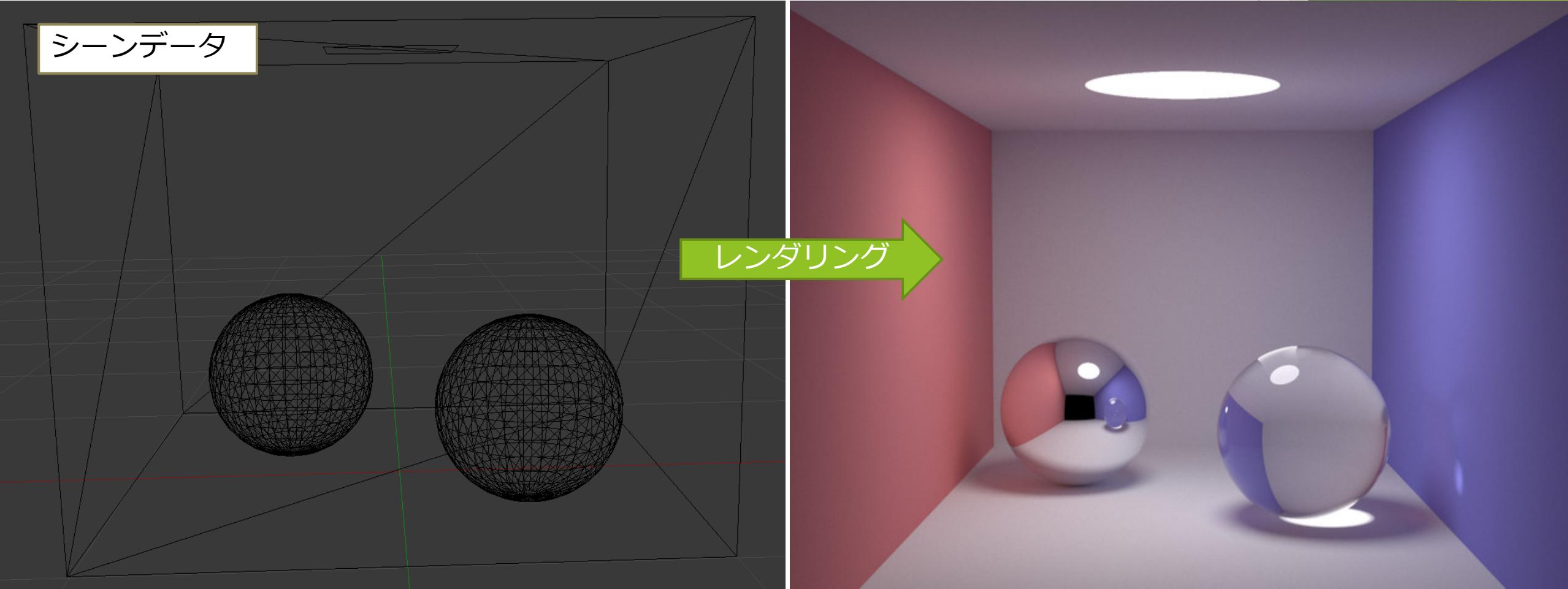
レイトレーシング

ラスタリ
ゼーション

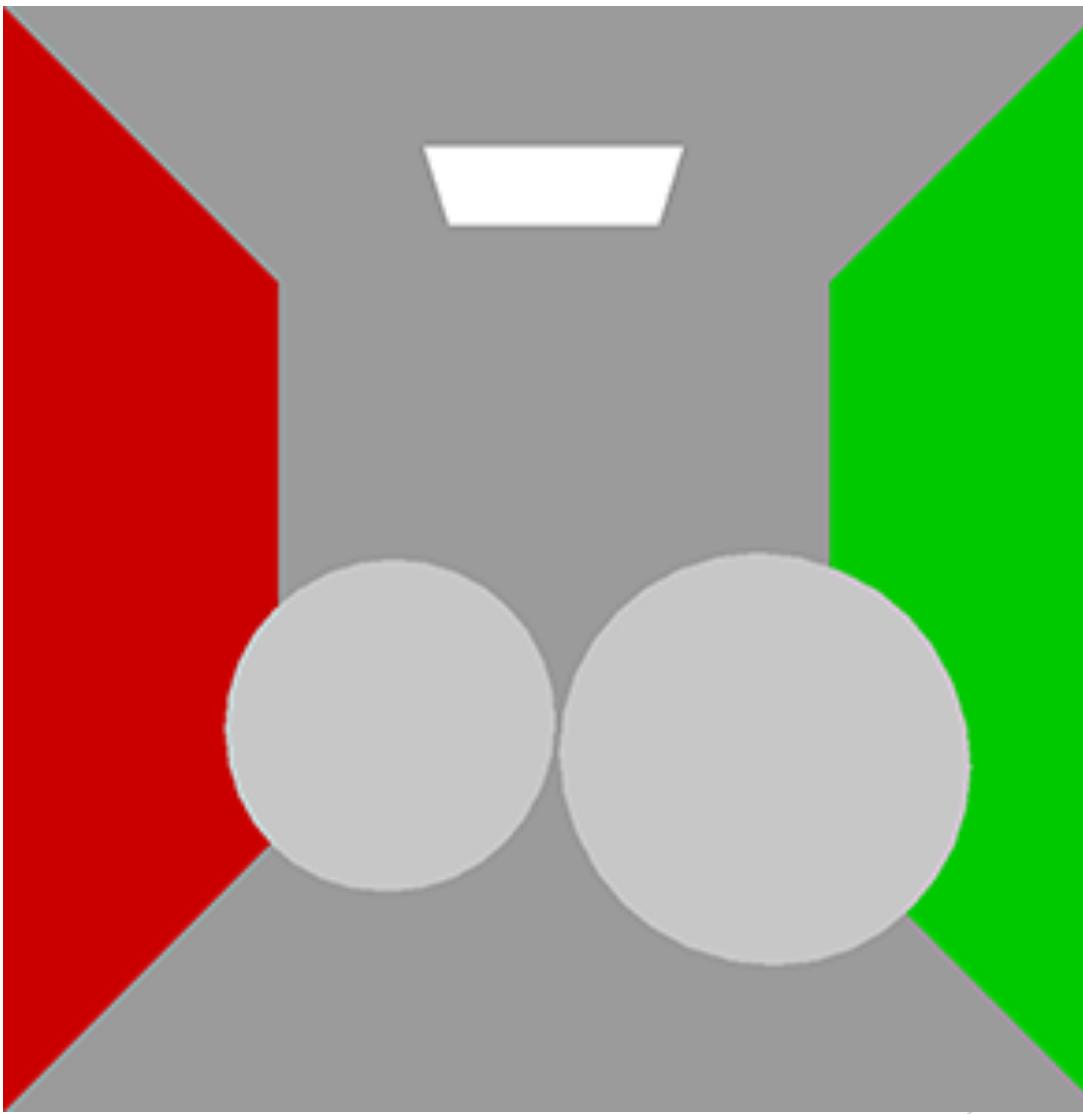
パストレー
シング

シーンデータ

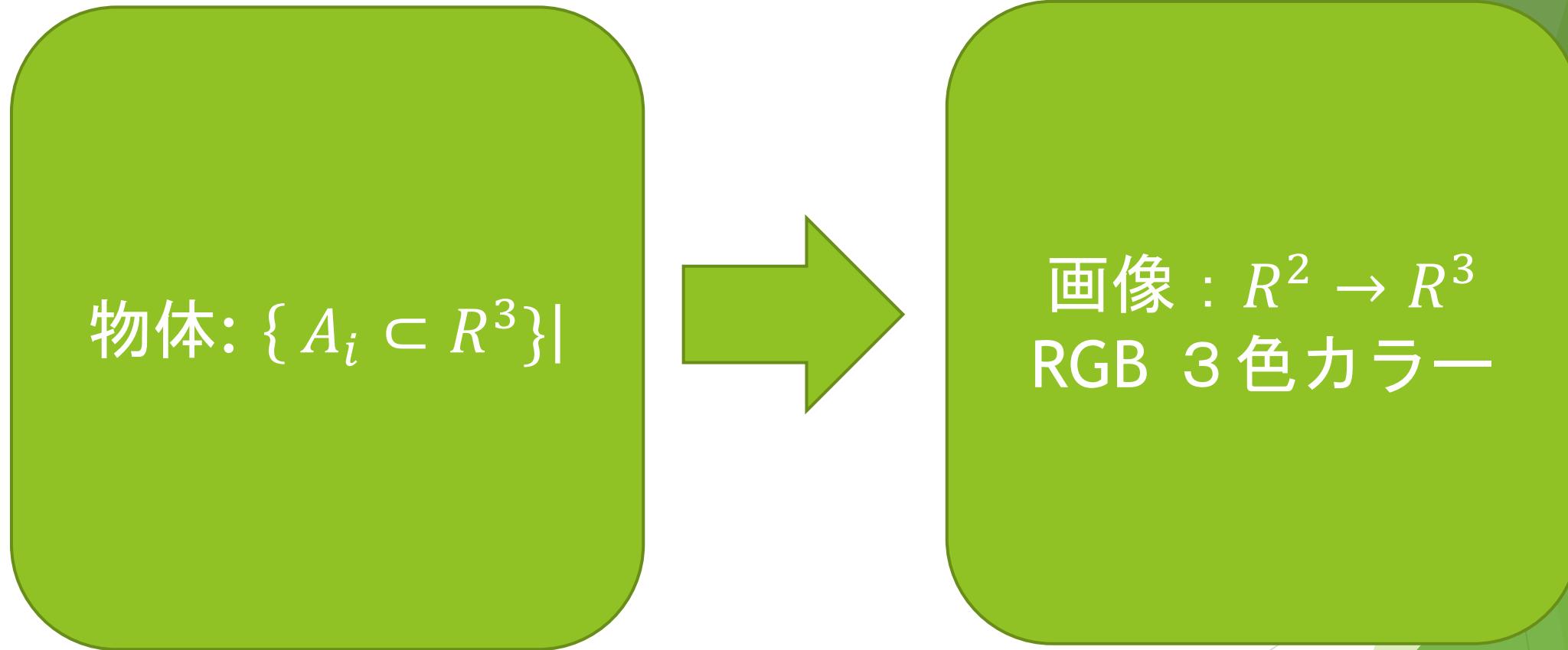
レンダリング



光学を考えないと

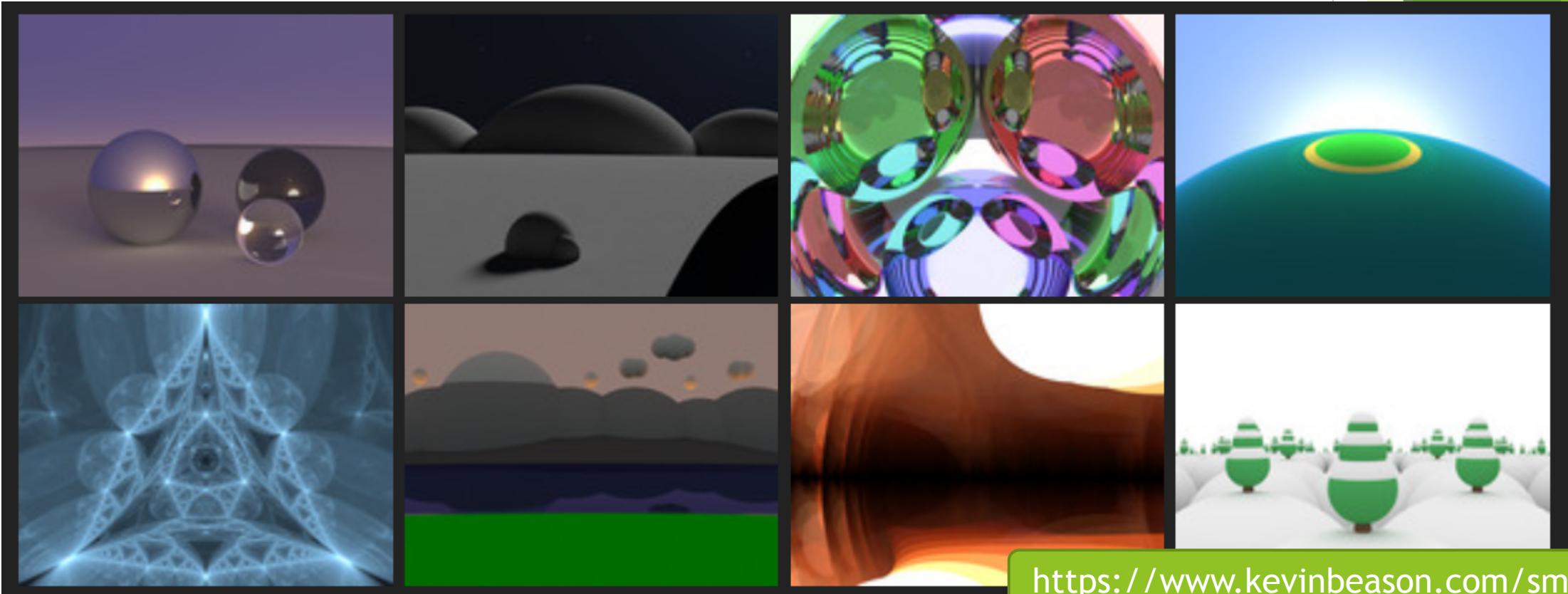


3次元シーンから2次元の絵を得る



smallpt

- ▶ written by Kevin Beason
- ▶ 99 line in C++
- ▶ Monte Carlo path tracing with Russian roulette path termination
- ▶ scenes must consist of spheres



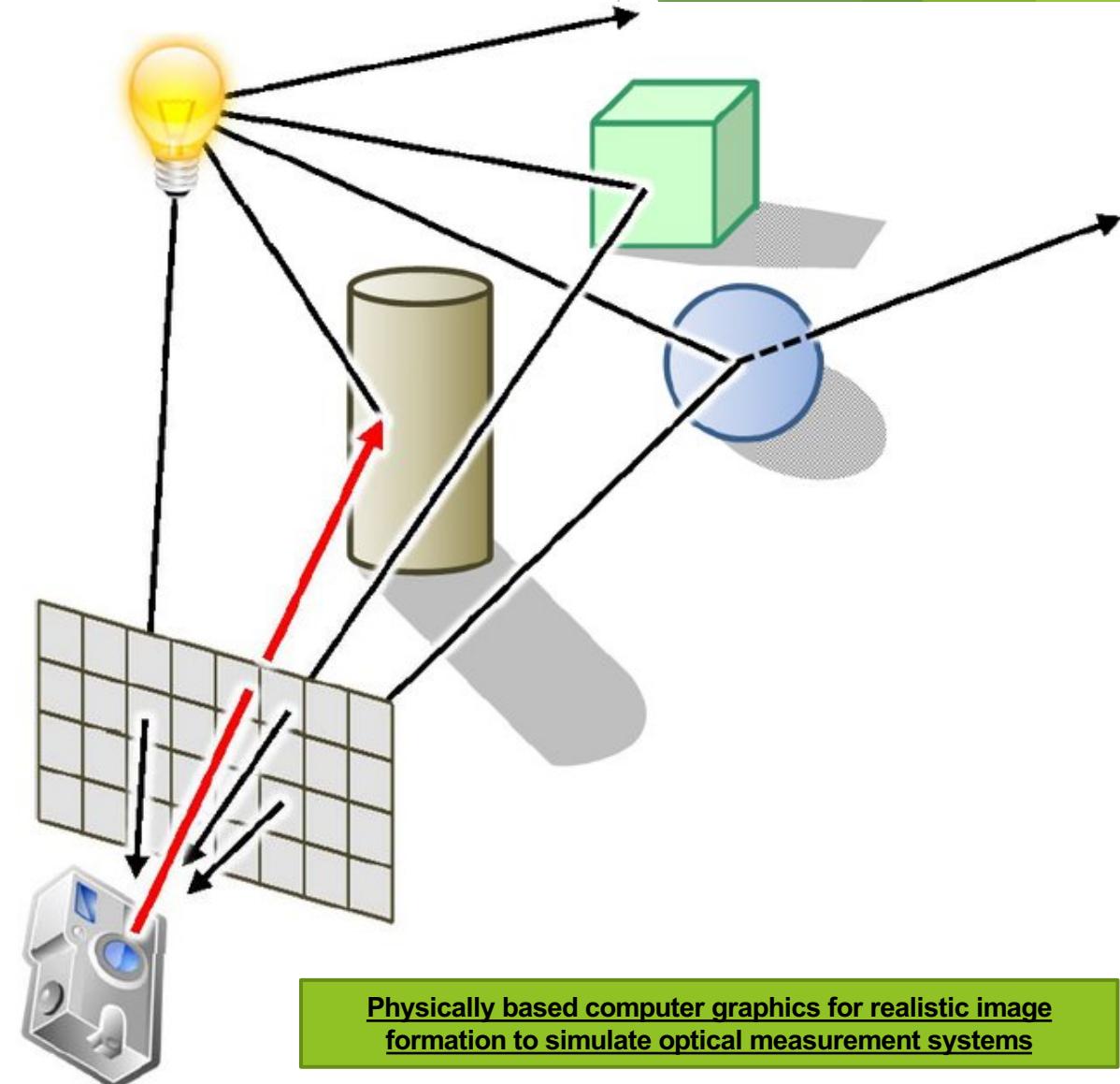
<https://www.kevinbeason.com/smallpt/>

smallpt4d: an extension for 4D scenes with binocular rendering



Optics for tracing light paths

- ▶ シーン=物体、ライト、カメラ
- ▶ 物体は、反射・屈折特性と色とを持つ
- ▶ ライトは色と強度を持つ
- ▶ カメラは向きを持つ

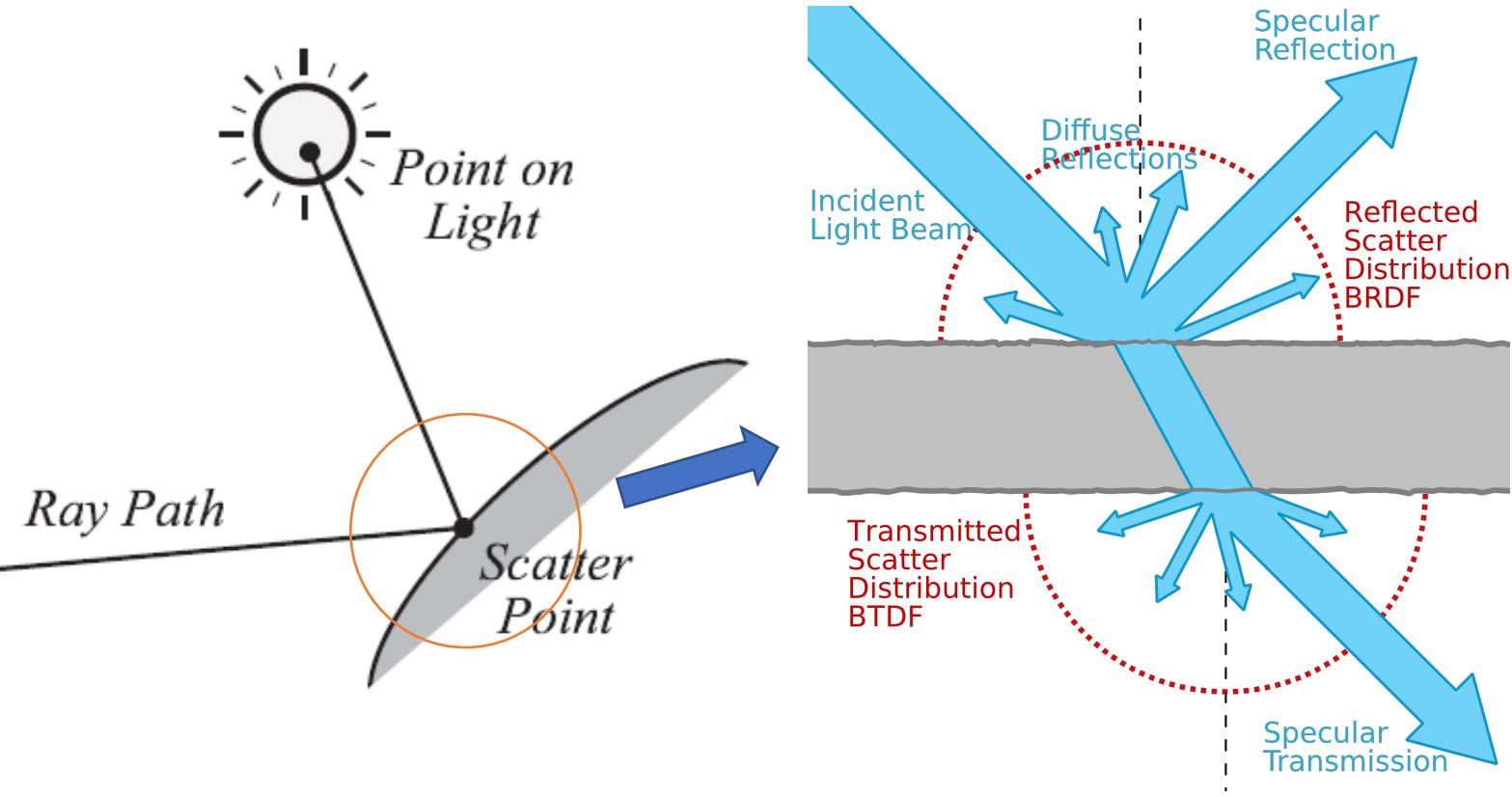
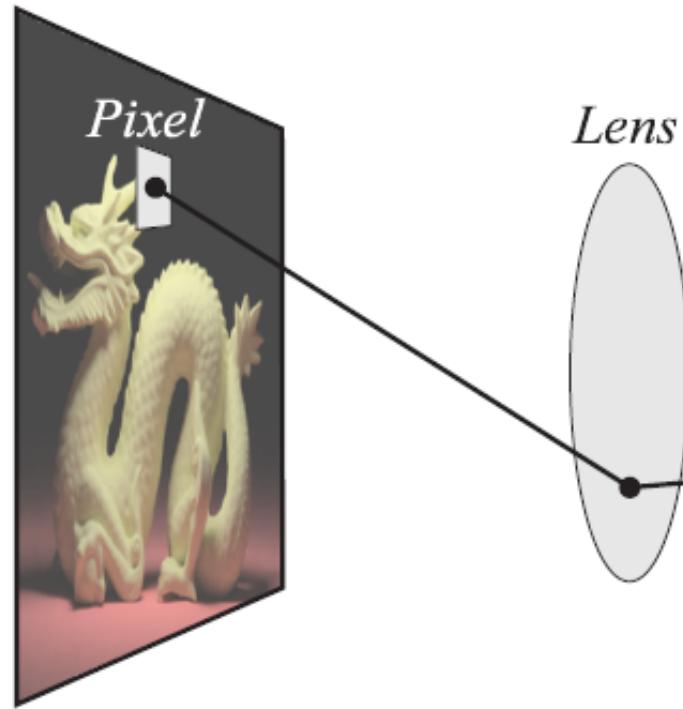


Physically based computer graphics for realistic image formation to simulate optical measurement systems

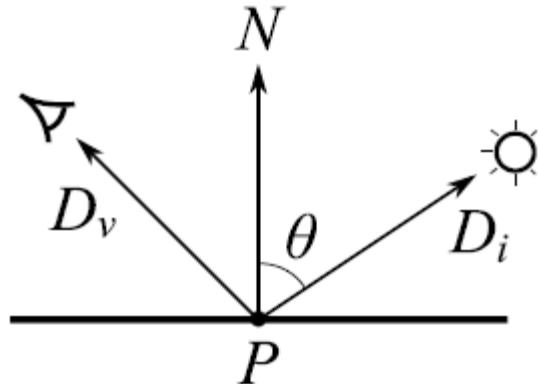
from the slides by Dr David Cline explaining smallpt
Available at <https://www.kevinbeason.com/smallpt/>

Light path

Time



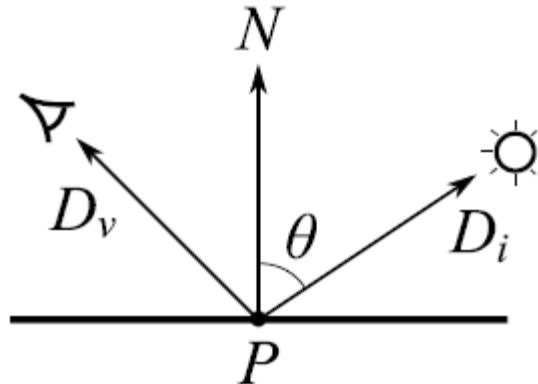
The Rendering Equation



$$L(P \rightarrow D_v) = L_e(P \rightarrow D_v) + \int_{\Omega} F_s(D_v, D_i) |\cos \theta| L(Y_i \rightarrow -D_i) dD_i$$

The radiance (intensity of light)
Coming from surface point P
In direction D_v . This is what we
Have to calculate.

The Rendering Equation

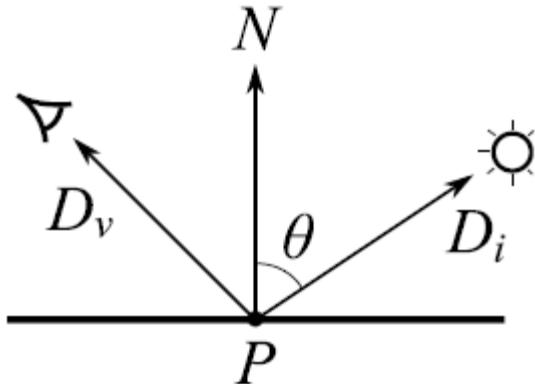


$$L(P \rightarrow D_v) = L_e(P \rightarrow D_v) + \int_{\Omega} F_s(D_v, D_i) |\cos \theta| L(Y_i \rightarrow -D_i) dD_i$$

|

The self-emitted radiance from P
In direction D_v (0 unless point P
Is a light source) This can be looked
Up as part of the scene description.

The Rendering Equation

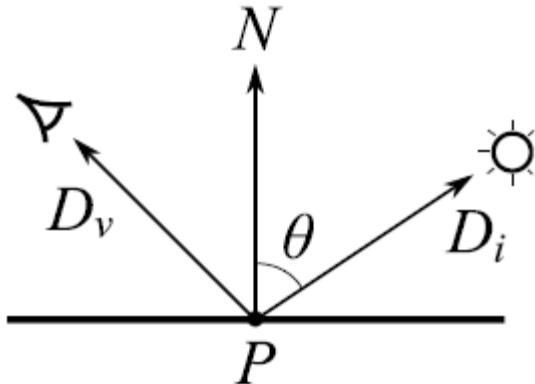


$$L(P \rightarrow D_v) = L_e(P \rightarrow D_v) + \int_{\Omega} F_s(D_v, D_i) |\cos \theta| L(Y_i \rightarrow -D_i) dD_i$$

|

The reflected light term. Here we must add Up (integrate) all of the light coming in to point P from all directions, modulated by the Chance that it scatters in direction D_v (based on the BRDF function, F_s)

Path Tracing Approximation



$$\hat{L}(P \rightarrow D_v) = L_e(P \rightarrow D_v) + \frac{F_s(D_v, D_i) |\cos \theta| \hat{L}(Y_i \rightarrow -D_i)}{p_{angle}^{tot}(D_i)}$$

|

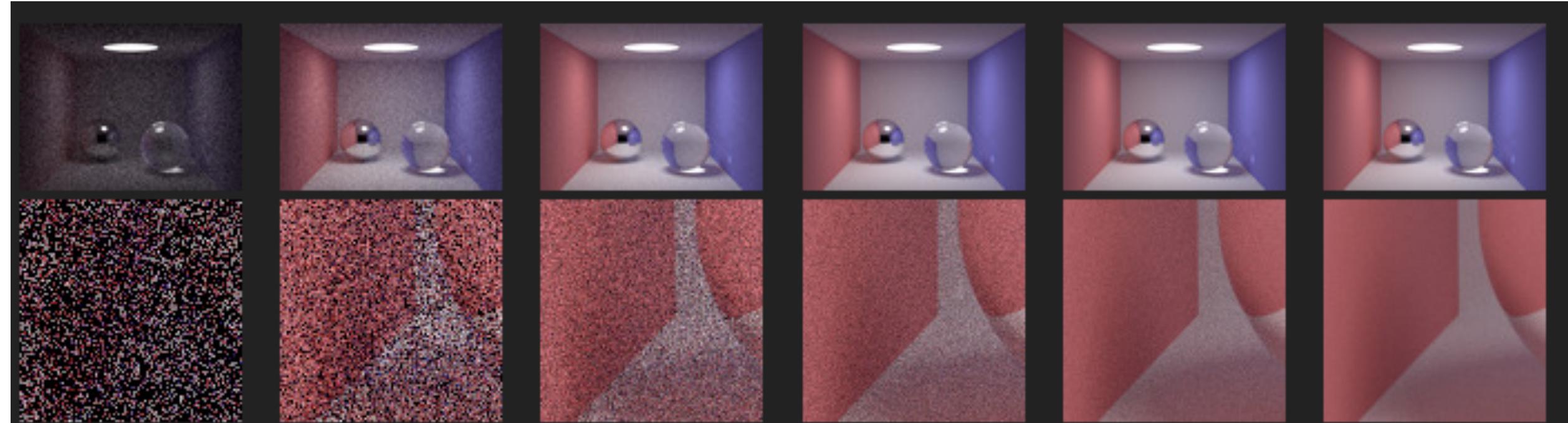
Replace the ray integral with a Monte Carlo
(random) Sample that has the same Expected
(average) Value. Then average a bunch of
samples for each pixel to create a smooth image.

Path Tracing Algorithm

Algorithm 3 Path Tracing Main Loop

```
1: for each pixel (i,j) do
2:   Vec3  $C = 0$ 
3:   for ( $k=0$ ;  $k < \text{samplesPerPixel}$ ;  $k++$ ) do
4:     Create random ray in pixel:
5:       Choose random point on lens  $P_{lens}$ 
6:       Choose random point on image plane  $P_{image}$ 
7:        $D = \text{normalize}(P_{image} - P_{lens})$ 
8:       Ray ray = Ray( $P_{lens}, D$ )
9:       castRay(ray, isect)
10:      if the ray hits something then
11:         $C += \text{radiance}(\text{ray}, \text{isect}, 0)$ 
12:      else
13:         $C += \text{backgroundColor}(D)$ 
14:      end if
15:    end for
16:    image(i,j) =  $C / \text{samplesPerPixel}$ 
17:  end for
```

Convergence



8 spp

13 sec

40 spp

63 sec

200 spp

5 min

1000 spp

25 min

5000 spp

124 min

25000 spp

10.3 hrs

Timings and resulting images for different numbers of samples per pixel (spp) on a 2.4 GHz Intel Core 2 Quad CPU using 4 threads.

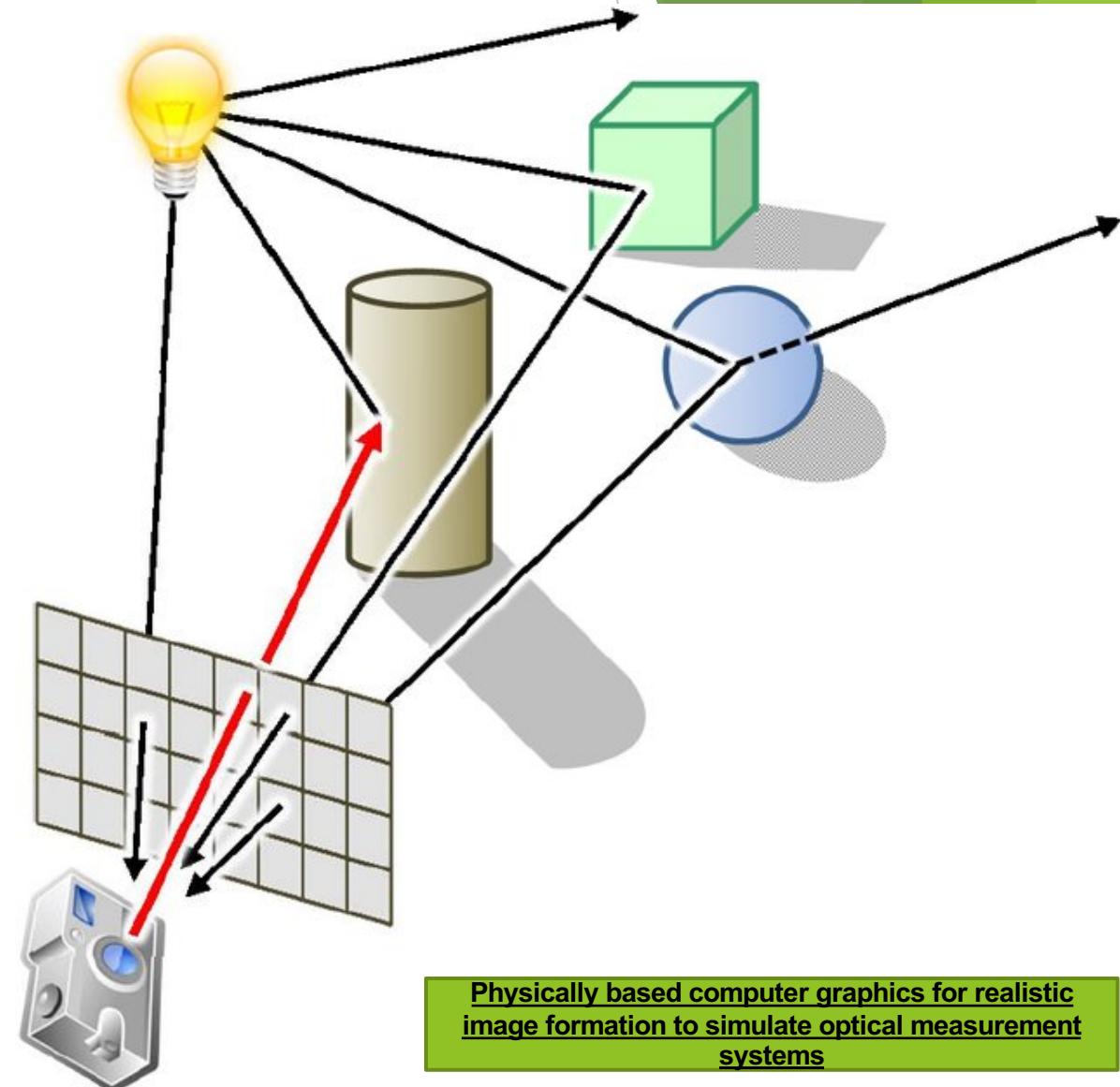
<https://www.kevinbeason.com/smallpt/>

Pixel, Ray ごとに完全に並列に計算できることに注意
=> GPU 処理に非常に向いている

4D Optics for tracing light paths

- ▶ 物体 : 余次元1
- ▶ ライト : 余次元1
- ▶ カメラ : 2次元スクリーン + 目1点
- ▶ 光線 : 1次元
- ▶ => 光線と物体交叉は点になる

他にも、1次元の目とカメラを結ぶ光面(2次元)と2次元物体で光学を考えるというような可能性もある。

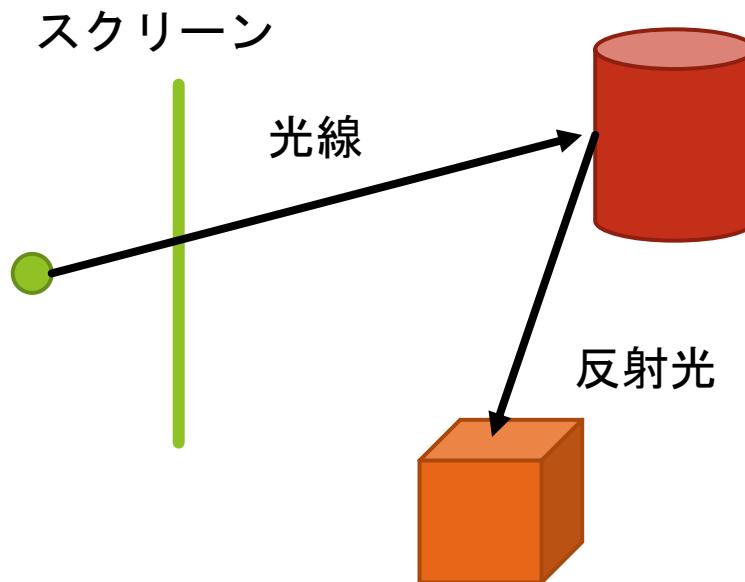


Physically based computer graphics for realistic
image formation to simulate optical measurement
systems

余次元 2 のスクリーン

- ▶ スクリーンを 余次元 2 に取るところだけ, straightforward ではない.
- ▶ このアナロジーを 3D のレンダリングで考えてみると

高次元のことを理解するには, “Flatland”式に
次元を下げてイメージすると良い



光線は最初, カメラ(線分 : スクリーン, 視点 ; 点)で
張られる 2 次元空間(cone)を進む.
物体に当たって反射・屈折・散乱するときには, そ
の 2 次元 cone から外れる.
=> 反射光が高次元の情報を capture

smallpt4d

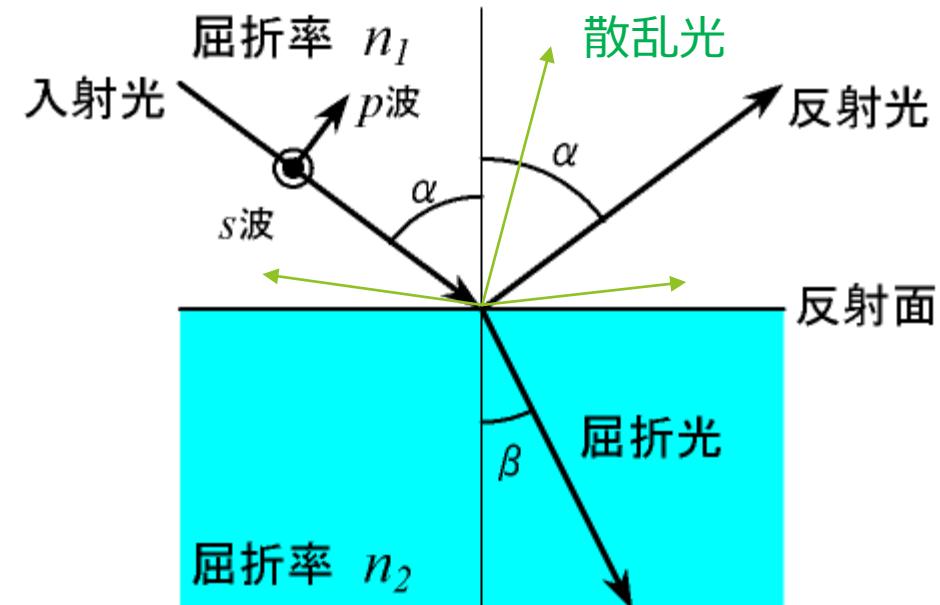
▶ <https://github.com/shizuo-kaji/smallpt4d>

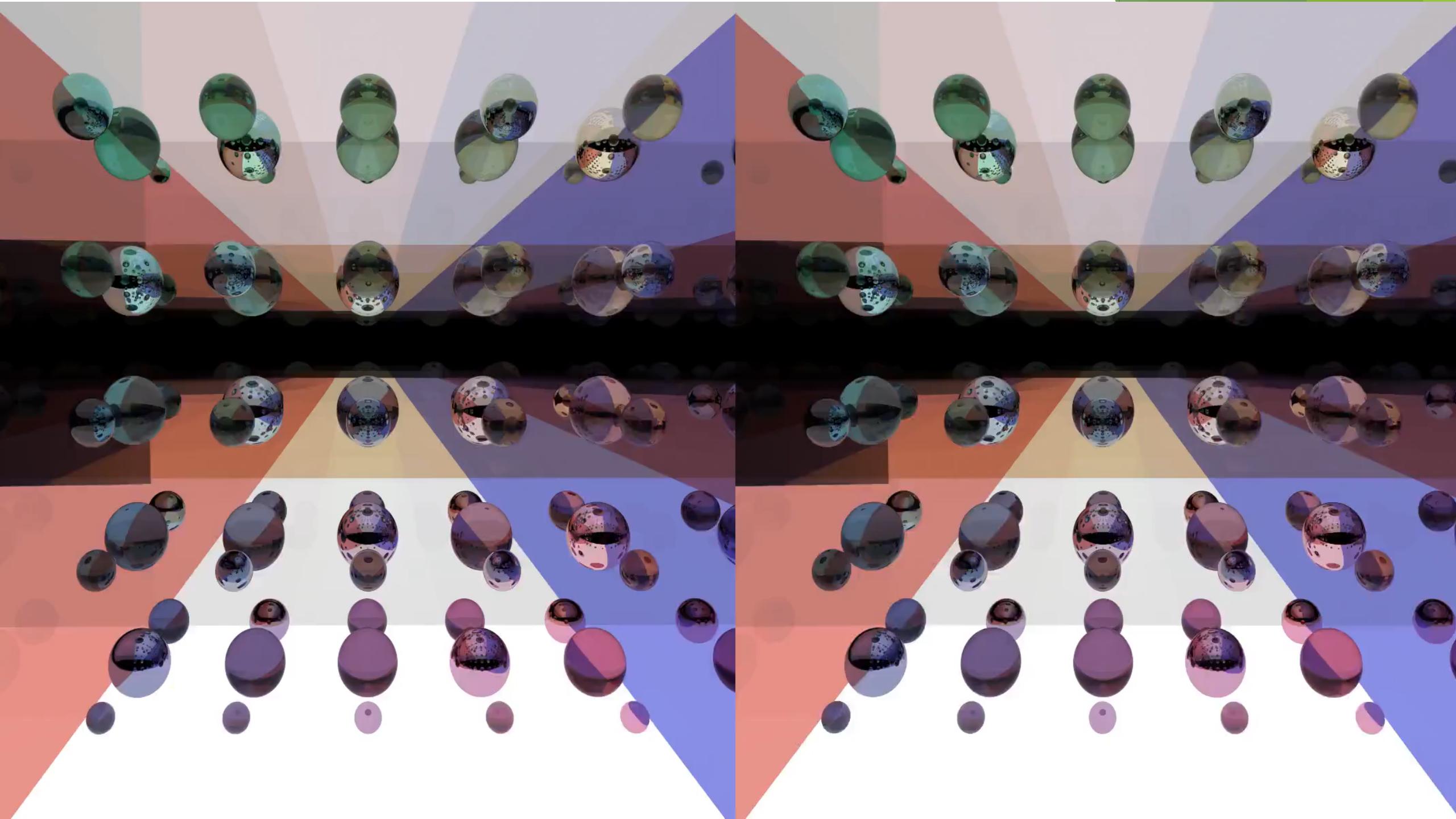
▶ 今の設定で、反射・屈折は2次元的。

つまり、入射光と法線のはる2次元部分空間で起きる

=> 3次元の光学を straightforward に拡張可能

▶ 散乱はランダムなので、これも straightforward





問題提起

問題設定

- ▶ 何を見たいのか. どんな対象の, いかなる性質?
- ▶ 何をもって「高次元が見えた」と言えるのか?

光学”代数”

- ▶ 反射や屈折は、アフィン部分空間の間の演算とみなせる



出力情報の次元を増やす方法

- ▶ 色 ("4D Draw" by Jeffrey Weeks <= "The Shape of Space" の著者, "Curved Space" の作者)
- ▶ 両眼：観測装置の window 分割
 - ▶ 両眼視差 (PolyVision)
 - ▶ 視野闘争
- ▶ 時間変化：時間的 window 分割
 - ▶ 運動視差 (PolyVision)
 - ▶ 断面のアニメーション “モース理論的”理解 ("4D Toys", "Miegakure" by Marc ten Bosch)
- ▶ 空間的 window 分割
 - ▶ 複数の射影を同時表示 (PolyVision)
- ▶ インタラクション・物理
 - ▶ 4次元の運動学 ("4D Toys")
 - ▶ 4次元の光学 (smallpt4d)
 - ▶ 4次元の遠近法(?)

基本方針 1

人間の持っている認識のためのリソースを何らかの形で分割して割り当てる

基本方針 2

関係による手がかりを構成する

どの方法がどの点で良いかを評価する
実験タスクや指標(ベンチマーク)の設計が重要

ベンチマークの例

タスク

- ▶ 同相か・isotope か判定する
- ▶ 高次のホモトピー群やホモロジ一群の生成元を認識する
- ▶ 鏡像対象かどうかを判定
- ▶ 合同かどうかを判定
- ▶ 合同な剛体を同じ向きに揃える
- ▶ 積み木をお手本通りに並べる
- ▶ 空間中の任意の一点を指す
- ▶ 迷路を解く
- ▶ 長さ・体積を比べる

受動的
(操作を要求しない)

能動的

評価指標

- ・正答率
- ・正答時間
- ・訓練による改善率

3次元の心理実験の拡張で、幾何的情報把握を見ている。
4次元特有のタスクは作れるか？

4次元のインタラクション: 岡体

- ▶ 四元数と4次元ユークリッド空間を同一視すると

$$\mathbb{R}^4 \simeq \mathbb{H}$$

四元数の掛け算(非可換)により、右掛け算と左掛け算で変換を定義できる

特に、絶対値1の四元数に制限して、次のパラメトリゼーションを得る

$$\mathbb{H}_1 = \{q \in \mathbb{H} \mid q\bar{q} = 1\}$$

$$\mathbb{H}_1 \times \mathbb{H}_1 \rightarrow SO(4)$$

$$(q_1, q_2) \mapsto (v \mapsto q_1 v \bar{q}_2)$$

これは二重被覆(2対1写像)となっている。
特に、自由度は6

平行移動も加えた合同変換(自由度10)はどうやって指定する?
(dual-dual quaternion?)

合同変換で良いのか？

- ▶ 着目したい性質は、本当に回転不变か？

$\mathbb{C}^2 \simeq \mathbb{R}^4$ では $U(2)$ or $SU(2)$ の方が自然？

さらに metric を変えると

(局所的には)

$\mathbb{R}^{3,1}$ ローレンツ空間 (=2次エルミート行列) $O(3, 1) \simeq SL(2; \mathbb{C})$

$\mathbb{R}^{2,2}$ (=2次実行列) $O(2, 2) \simeq SL(2) \times SL(2)$

インタラクションにどの変換を紐づけるかは
どんな対象を見たいかに依存する！

