

Graphics with Processing



2012-10 照明と材質のモデル

<http://vilab.org>

塩澤秀和

10.1 光源のモデル

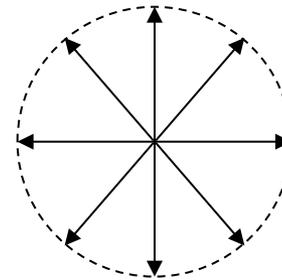
レンダリング (p.96)

- 座標変換後の画像生成
 - 3次元シーン → 2次元画像
 - 色, 陰影, 質感などの表現
 - 高品質 vs リアルタイム
 - 光と陰影 ⇒ シェーディング

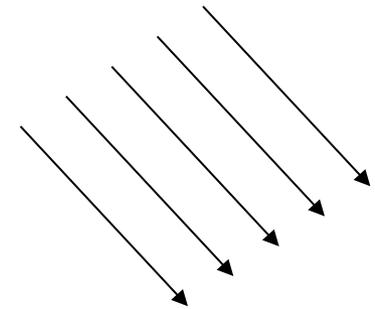
光源のモデル

- 照明の色
 - 太陽光・蛍光灯 ⇒ 白
 - 白熱電球 ⇒ 白っぽいオレンジ
- 照明の明るさ(照度) (p.118)
 - 照度(単位ルクス) = 単位面積あたりに当たる光の量
 - 点光源から距離 r 離れた場所での照度 ⇒ 逆2乗の法則

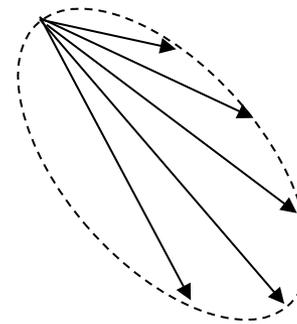
□ 光源の種類 (p.120)



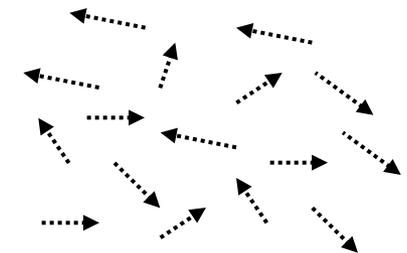
点光源
(電球など)



方向光
(太陽光など)



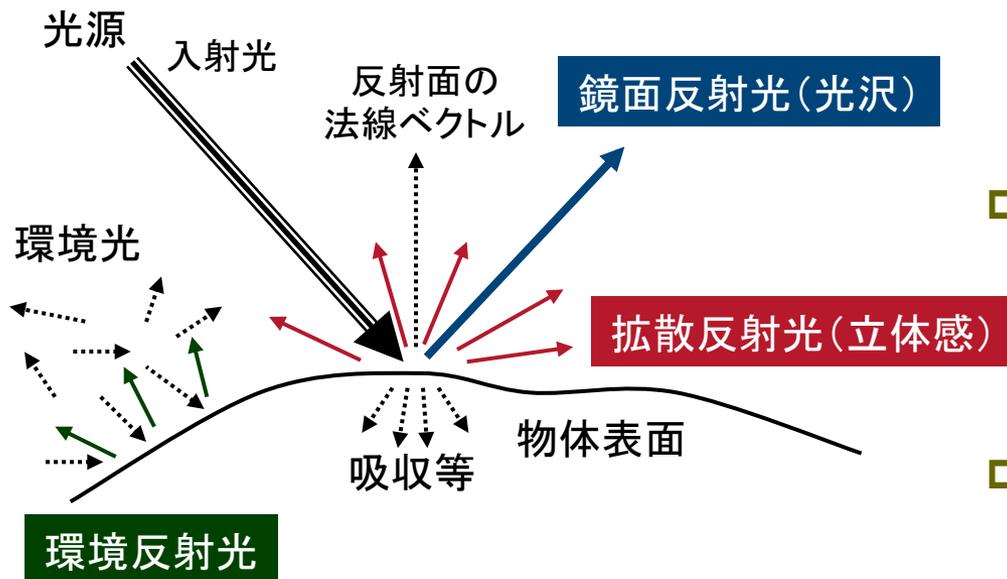
スポットライト



環境光(壁などに
何回も反射した
間接光のモデル化)

10.2 反射光のモデル

反射光のモデル(p.120)



反射光のR成分

= 入射光のR成分 × 物体の反射率のR成分

反射光のG成分

= 入射光のG成分 × 物体の反射率のG成分

反射光のB成分

= 入射光のB成分 × 物体の反射率のB成分

- 観測される色
 - 色 = 拡散反射光 + 鏡面反射光 + 環境反射光 + 放射光
 - ※ 色は照明にも影響される
- 拡散反射光
 - 光源からの光をザラザラの面が四方八方に拡散反射した光
 - 光の入射角に依存 ⇒ 立体感
- 鏡面反射光
 - 光源からの光をツルツルの面が鏡のように反射した光
 - 見る角度に依存 ⇒ “光沢”
- 環境反射光
 - 特定の光源ではなく、空間全体の間接光に対する反射光
 - シーン全体が一様に照らされる

10.3 材質属性のモデル

材質(マテリアル)属性

- “色”の設定
 - 反射・吸収される光の波長は、物体表面の材質によって違う
 - 反射率(K) = 白色光を当てたときの反射光の色(RGB成分)
- 拡散反射色
 - 拡散反射率(Kd)
 - 物体表層で何度も透過・屈折し、拡散して反射する光への着色
 - 通常の意味での物体の色
- 鏡面反射色
 - 鏡面反射率(Ks)
 - 物体表面の分子でほぼ完全に反射する光で、着色が少ない
 - 金属光沢, ハイライト, つや

- 環境反射色
 - 環境光の反射率(Ka)
 - 通常は拡散反射色と同じ色
- 放射光
 - 電球など発光している物体
 - 周囲に関係なく一定の色(K_e)

$$I = K_e \quad (\text{常に一定の色})$$

材質による特徴

- 紙・木など
 - 鏡面反射(光沢)がほとんどない
- プラスチックなど
 - 若干の鏡面反射によるつやがある
- 金属など
 - 強く白っぽい鏡面反射(Ks≠Kd)

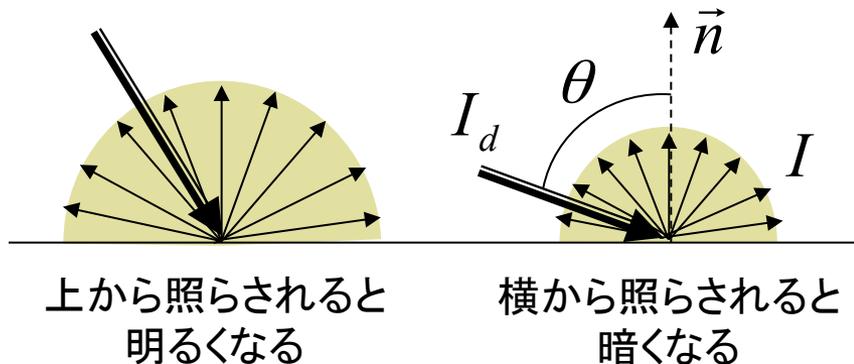
10.4 反射光の計算モデル

拡散反射光(p.123)

□ ランバートの余弦則

- 光がどの方向から入射しても、全方向に均等に拡散
- 入射角余弦の法則より、表面の明るさは入射角のcosに比例

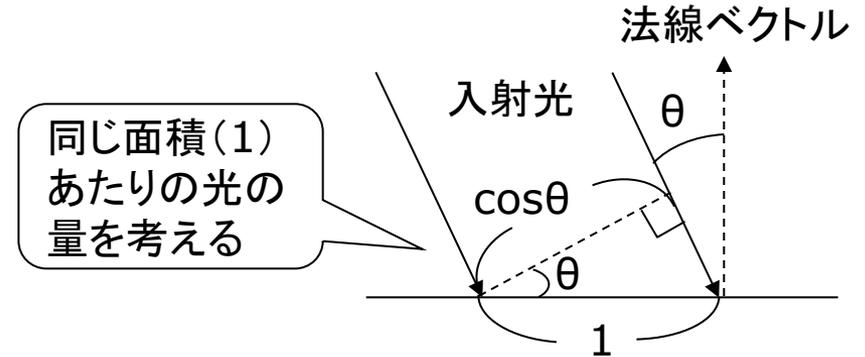
$$I = K_d I_d \cos \theta$$



I_d : 入射光の拡散反射成分 I : 反射光
 K_d : 物体表面の拡散反射率 θ : 入射角

□ 入射角余弦の法則

- 単位面積あたりに当たる入射光の量は入射角のcosに比例



環境反射光(p.122)

□ 環境光による拡散反射光

- 環境光は四方八方から均等に当たるので方向がない
- 常に同じ色に見える

$$I = K_a I_a \quad (K_a: \text{環境光の反射率})$$

10.5 照明と材質の関数

基本的な光源

- `pointLight(r, g, b, x, y, z)`
 - 点光源(例:電球)
 - `r, g, b`: 光の色(HSBモードの場合は, 色相, 彩度, 明度)
 - `x, y, z`: 光源の座標
- `directionalLight(r, g, b, nx, ny, nz)`
 - 方向光(例:太陽光, 天井照明)
 - `nx, ny, nz`: 光の方向ベクトル
- `ambientLight(r, g, b)`
 - 環境光(間接光のモデル化)
 - 全方向から均等にあたる光
- サンプル
 - 物体をおく前に, 光源をおくこと
 - Examples → 3D → Lights

標準の光源

- `lights()`
 - 下記の光源を設定
 - `ambientLight(128, 128, 128)`
 - `directionalLight(128, 128, 128, 0, 0, -1)`

基本的な材質特性

- `fill(色)`
 - 通常の色 = 拡散反射率 K_d
- `ambient(色)`
 - 環境反射率 K_a の設定
 - 無指定時にはfillと同じ色で計算
- `emissive(色)`
 - 放射光 K_e の設定(蛍光面)

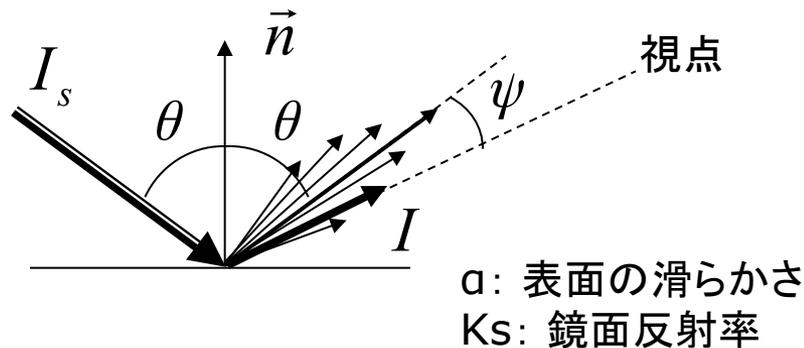
10.6 光沢の表現

鏡面反射光(p.126)

□ フォンの反射モデル

- 光がごく表層でほぼ完全に反射
⇒ 反射光が正反射方向に集中
- 近似的なモデルが一般的

$$I = K_s I_s \cos^\alpha \psi$$



□ より正確なモデル(p.156)

- ブリンの反射モデル
- クック・トランスの反射モデル

鏡面反射の材質特性

□ specular(色)

- 鏡面反射率 K_s

□ shininess(輝き)

- 鏡面反射光の集中度(α)
- 輝き: 10~50~500(金属)

光源のパラメータ

□ lightSpecular(r, g, b)

- 後に設置する光源に鏡面反射成分を追加
- 通常は光源と同じ色でよい

□ lightFallOff(c1, c2, c3)

- 光の減衰のしかたを変更する
- 距離dとして

$$\frac{1}{c_1 + c_2 d + c_3 d^2}$$

10.7 照明と材質の設定例

```
void setup() {
  size(400, 400, OPENGL);
}

void draw() {
  float a = radians(frameCount);
  background(0);
  perspective();
  camera(0, -100, 200, 0,0,0, 0,1,0);
  // 環境光
  ambientLight(50, 50, 50);
  // 回転する点光源を設置する
  // ボタンを押すと鏡面反射成分をつける
  float lx = 100 * cos(a);
  float ly = -100;
  float lz = 100 * sin(a);
  if (mousePressed)
    lightSpecular(128, 128, 128);
  pointLight(128, 128, 128, lx, ly, lz);
  stroke(128);
  line(lx, 0, lz, lx, ly, lz);

  noStroke();
  pushMatrix();
  rotateX(PI/2);
  fill(100); ellipse(0, 0, 200, 200);
  popMatrix();

  rotateY(a / 2);
  pushMatrix();
  translate(60, -20, 0);
  fill(250, 200, 10); // 拡散反射色
  specular(100, 100, 100); // 鏡面反射色
  shininess(20); // 輝きの集中度
  sphere(20);
  specular(0); // ゼロに戻す
  popMatrix();

  pushMatrix();
  translate(70, -20, 50);
  fill(40, 40, 230); // 拡散反射色
  box(20, 40, 20);
  popMatrix();
}
```

10.8 演習課題

課題

- スポットライト(下記)を使用したプログラムを作成しなさい
 - 床は右のようにタイルを敷き詰めるようにする(理由は次回)
 - 床の上に何か物を置くとよい
 - スポットライトの設置例
 - `spotLight(255, 0, 0, 50, -50, -50, -1, 1, 1, PI/2, 100)`
 - `spotLight(0, 255, 255, -50, -50, -50, 1, 0.6, 1, PI/2, 70)`

スポットライト関数

- `spotLight(r, g, b, x, y, z, nx, ny, nz, 角度, 集中度)`
 - 角度: 光の範囲($\sim \pi/2$ 程度)
 - 集中度: 1 \sim 100 \sim それ以上

床の描画例

```
noStroke();
for (x=-100; x<100; x+=10) {
  for (z=-100; z<100; z+=10) {
    beginShape(QUADS);
    vertex(x, 0, z);
    vertex(x, 0, z+10);
    vertex(x+10, 0, z+10);
    vertex(x+10, 0, z);
    endShape();
  }
}
```

