Graphics with Processing

2015-12 レンダリング技術

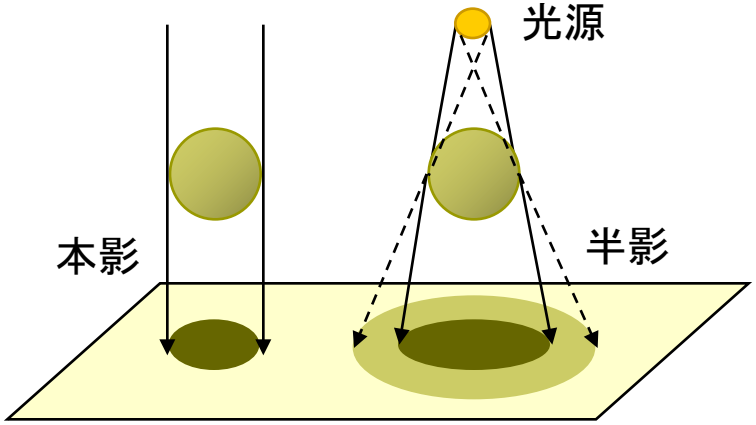
http://vilab.org

塩澤秀和

12.1 影付け

影の種類(p.158)

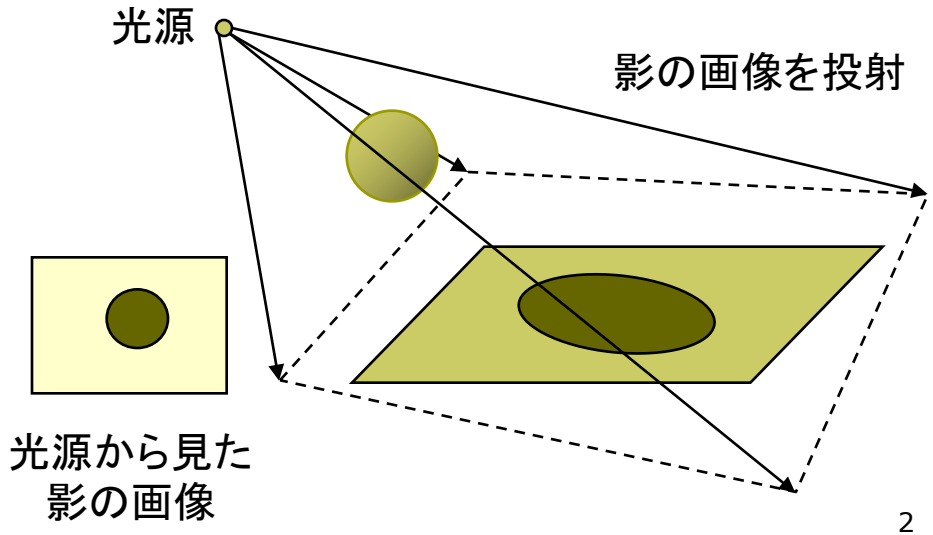
- 本影と半影
  - 点光源や平行光ではくっきりした影(本影)だけができる
  - 光源に広がりがあると，半影を含むソフトシャドウができる



  - 光源が複数ある場合，それぞれの光(影)を重ね合せればよい
  - リアルタイムな影生成では基本的に本影部分を扱う

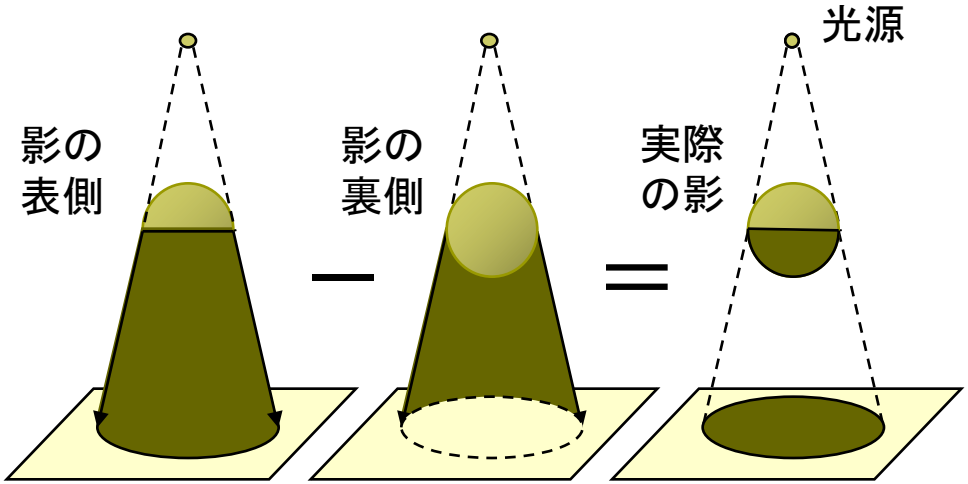
主な影付け方式

- 影の投影マッピング
  - いったん視点を光源に置き，物体のシルエットを描画すると，光源から見たその物体の影になる
  - 視点は戻して，影の画像を光源の位置から物体の下の地面などに投影テクスチャマッピングする

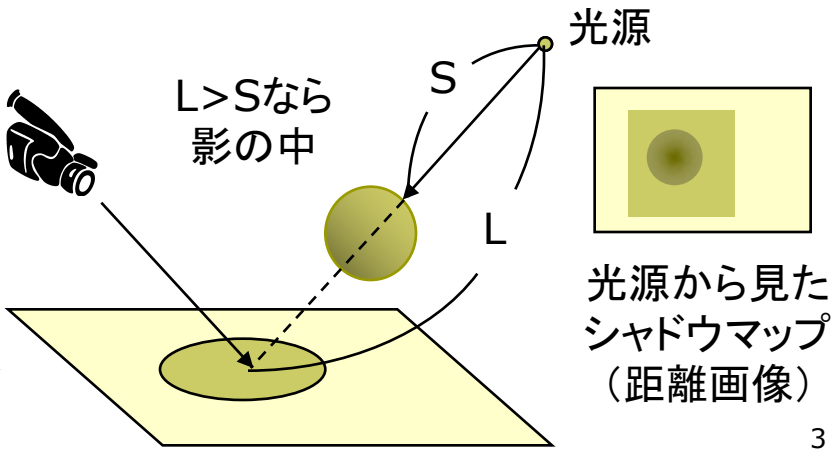


12.2 影付け（続き）

- シャドウボリューム法
  - 物体が光をさえぎってできる影の空間（シャドウボリューム）を囲う“影ポリゴン”を算出する
  - 視点から見て表を向いている影ポリゴンの像から，裏を向いている影ポリゴンの像を引くと，視点から見た影の形が分かる
  - 「ステンシルバッファ」を用いると，高速に実現できる



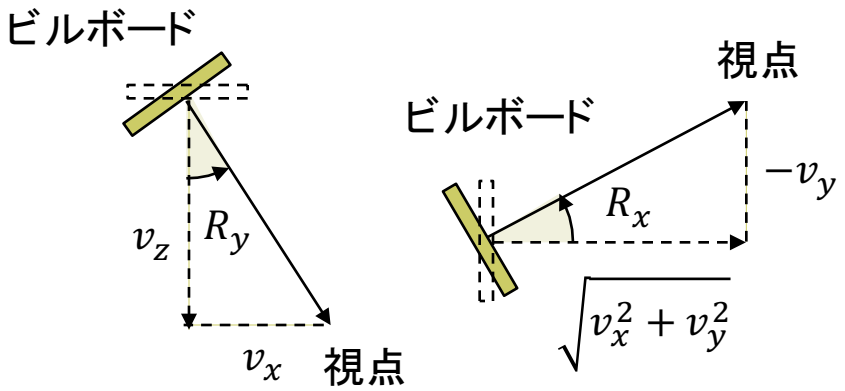
- シャドウマップ法（p.159）
  - （Zバッファを用いた2段階法）
  - 光源から見た場合のZバッファを構成すると，光の到達距離Sのマップ（シャドウマップ）ができる
  - 視点を戻し，レンダリングするオブジェクトから光源までの距離Lとシャドウマップ上の対応点の内容（S）を比較し，光がそこまで届いているか判定する



12.3 ビルボード

ビルボード

- Billboard＝立看板，掲示板
  - 細かいオブジェクトなどを，板に画像を貼ったもので表示する
  - 板はペラペラなので常に視点を向くように調整する
  - パーティクル（大量の粒子による表現）では特に有効



上から見た図　　横から見た図

```
// 常に視点を向く「立看板」にテクスチャを貼る
PImage tex;
PVector[] bbpos = new PVector[100];
PVector camPos = new PVector();

void setup() {
    size(400, 400, P3D);
    frameRate(30);
    tex = loadImage("tree.png");

    // ビルボード（立看板）の設置
    // （背景が透過するように奥から手前に設置）
    int n = bbpos.length;
    for (int i = 0; i < n; i++) {
        bbpos[i] = new PVector(random(
            -400, 400), 0, 800 * i / n - 400);
    }
}
```

12.4 ビルボード（続き）

```
void draw() {
  background(#80e0ff);
  // カメラを動かす
  float a = radians(frameCount);
  camPos.x = 200 * sin(a);
  camPos.y = -10 + 10 * cos(a);
  camPos.z =  250 + 250 * cos(a);
  camera(camPos.x, camPos.y,
      camPos.z, 0, 0, -400, 0, 1, 0);

  noStroke(); fill(#602020);
  box(800, 1, 800);  // 地面を描く
  textureMode(NORMAL);

  for (PVector pos : bbpos) {
    // 視点に正面を向ける回転角の計算
    PVector v = camPos.copy();
    v.sub(pos);
    float ry = atan2(v.x, v.z);
    //float rx =
    // atan2(-v.y, dist(0, 0, v.x, v.z));

    pushMatrix();
    translate(pos.x, pos.y, pos.z);
    rotateY(ry);    // 横の向きを正面に
    //rotateX(rx);  // 縦の向きを正面に

    beginShape(); texture(tex);
    vertex(-20, -60, 0, 0, 0);
    vertex( 20, -60, 0, 1, 0);
    vertex( 20,   0, 0, 1, 1);
    vertex(-20,   0, 0, 0, 1);
    endShape();
    popMatrix();
  }
}
```

12.5 物理ベースレンダリング

物理ベースレンダリング

- Physically based rendering
  - 最近，ゲームなどでも採用されているリアルなレンダリング
  - 従来よりも物理的に精密な光の計算（明確な定義はない？）

エネルギー（光量）保存則

- 反射光の合計＜入射光
  - 鏡面反射が強くなれば拡散反射光はその分減る → 配分を計算
  - 厳密には，BRDF（双方向反射分布関数）によって表される
- リニア（線形）色空間
  - 表示デバイス向けの階調補正を物理量に戻す（逆ガンマ補正）

主な使用技術

- 鏡面反射（specular）
  - 入射角が大きいほど反射率が高いことを考慮（フレネル係数）
  - クック・トランスのモデルなど
- マテリアル（材質）
  - 基本色（アルベド）＋メタリック＋ラフネス（スムースネス）で指定
  - それぞれテクスチャで指定可能
- 照明
  - 環境光も拡散反射と鏡面反射の成分に分配してそれぞれ計算
  - 光源画像による照明（IBL）
- 拡散反射（diffuse）
  - オーレン・ネイヤーのモデルが用いられることもある

12.6 高品質レンダリング

目的別レンダリング

- リアルタイムレンダリング
  - 3Dゲーム ← ユーザが操作
  - 理想は60fps，最低限10fps
- 高品質レンダリング
  - 静止画，映画 ← 事前に“撮影”
  - やわらかい陰影やガラスの表現

⇒ レイトレーシング法＋大域照明

大域照明モデル（p.183）
（Global Illumination: GI）

- 間接光まで含む照明計算
  - 単純な環境光モデルではなく，間接光をより精密に計算する
  - 特に室内の陰影がより自然
  - ラジオシティ，フォトンマッピング

フリーソフトによるレンダリングの例

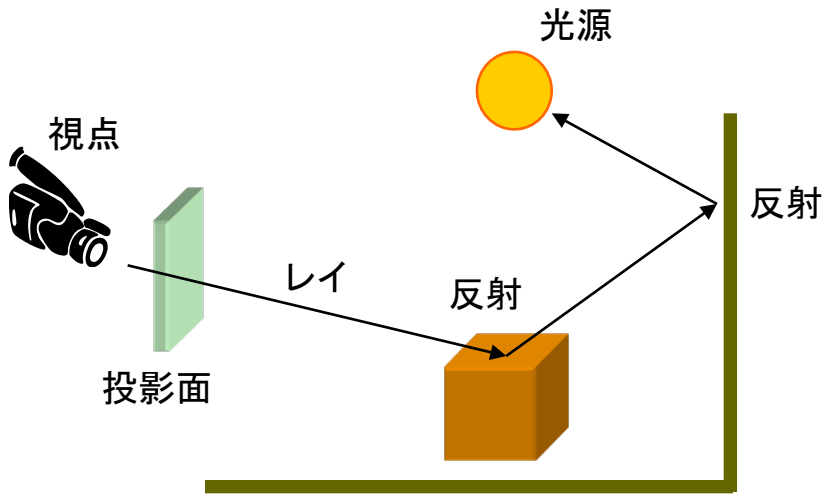
- POV-Ray
  http://www.povray.org
  → Hall of Fame
- Blender+Yafray
  http://www.blender.org
  → Feature & Gallery
  http://www.yafaray.org
  → Gallery
- Sunflow
  http://sunflow.sourceforge.net
  → Gallerly
- Art of Illusion
  http://www.artofillusion.org
  → Art Gallery

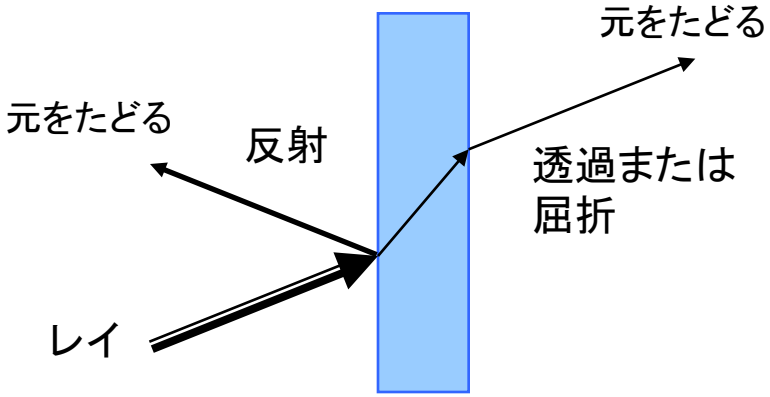
12.7 レイトレーシング (p.135)

レイトレーシング法

- 概要
  - Ray Tracing＝光線追跡
  - 各ピクセルに届く光の軌跡を，視点から光源にさかのぼるように追跡するレンダリング技術
- 高品質
  - 3DCGの初期からあるが，より正しく光学現象を再現するように研究され続けている
  - 原理的に隠面消去される
  - 透明，影，レンズも自然に表現
- 用途
  - リアルだが時間がかかるので，まだゲームなどには向かない
  - 映像作品（映画）製作で一般的

- 光線追跡の概念図



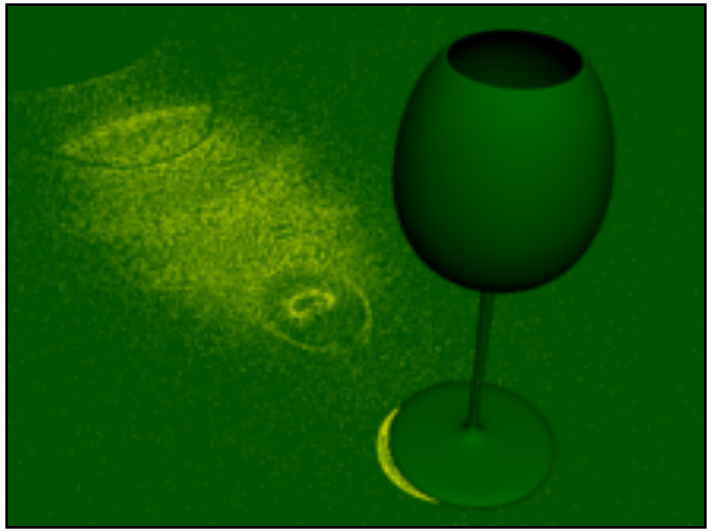


12.8 フォトンマッピング (p.187)

フォトン(Photon)マッピング

- 概要
  - 光源から出る大量の光子を考え，その軌跡をシミュレーションする
  - すると，シーン全体の光の分布(間接光)が概算できる
  - この間接光を環境光の代わりにして，レイトレーシングを行う
- 特徴
  - レンズなどの集光現象(コースティックス)が表現できる
  - 逆方向のレイトレーシングといえ，レイトレーシング法と相性が良い
  - 着想は簡単だが，アルゴリズムは複雑で膨大な時間がかかる

Wikipedia



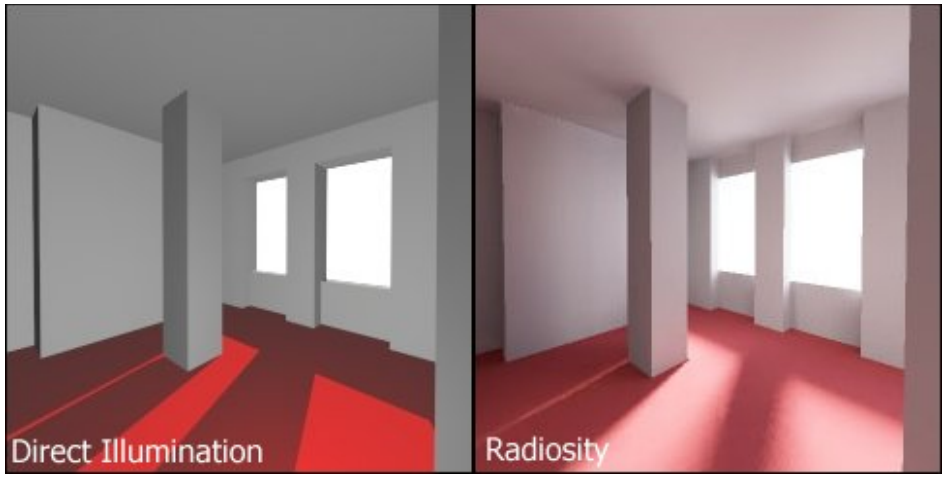
計算された光子の分布

12.9 ラジオシティ法（p.184）

ラジオシティ（Radiosity）法

- 概要
  - ポリゴンをパッチ（断片ポリゴン）に分割する
  - 2つのパッチの位置と向きの関係から，光の相互伝達率（フォームファクタ）を計算する
  - 全パッチ間での光エネルギーの放射発散の平衡状態を求める

Wikipedia



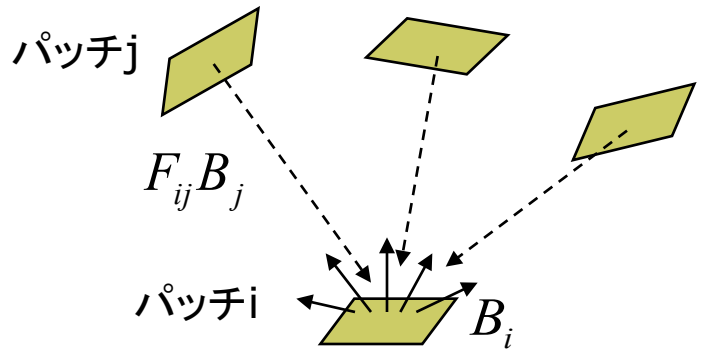
柔らかい影や壁の色の影響が表現されている

- ラジオシティ方程式（p.158）

$$B_i = E_i + \rho_i \sum_{j=1}^{n} F_{ij} B_j$$

n	シーン全体のパッチ数
Bi	パッチiの光の放射量（ラジオシティ）
Ei	パッチiの発光量
ρi	パッチiの反射率
Fij	フォームファクタ（Fij＝Fji）

- 本質的には「連立一次方程式」
  ⇒ ガウス・ザイデル法など



12.10 その他のレンダリング技術

ぼかし(ボケ)系

- アンチエイリアシング(p.255)
  - ドットのギザギザが目立たないように，輪郭を中間色でぼかす
- フォグ(霧)
  - 水蒸気やチリなどによる空気の「濁り」を再現する
  - 遠くにあるものがかすんでいき，色が落ちていく効果を与える
- 被写界深度(DOF)(p.301)
  - レンズの効果を再現し，ピントが合っていないところをぼかす
- モーションブラー
  - 速く動くものに見える残像(ボケ)をわざと表示する
  - 軌跡の画像を重ね合わせる

イメージベーストレンダリング

- 画像をCGに利用(p.171)
  - CGと画像処理技術との融合
  - テクスチャマッピングの応用(撮影地点から画像を投影など)
  - イメージベーストライティング：画像を光源として利用
  - 環境マッピング：周辺の景色の映り込みを表現
  - イメージベーストモデリング：写真から3Dモデルを自動生成
- 実写とCGの融合
  - 実写にCG映像を合成(AR)，または，CGに実写映像を合成
  - 自由視点画像：限られた台数で撮影したカメラ映像から，自由な視点からの映像を合成

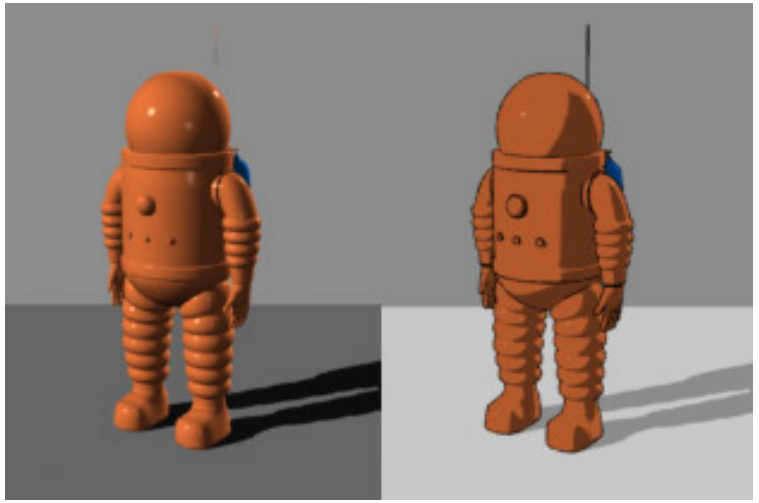
12.11 非写実的レンダリング（p.309）

ノンフォトリアリスティック（非写実的）レンダリング（NPR）

- 概要
  - 現実の再現を目的としないCG
  - 例）油絵風，手書きタッチの再現，製図風，2次元アニメ，芸術作品
- 背景
  - 写実的（フォトリアリスティック）なCG技術はかなり完成
  - 漫画・アニメーションでの利用
  - 芸術などへのCG利用の広がり
- Blender Freestyle
  - フリーの3DCGソフトウェアBlenderに付属のNPR機能

http://www.blender.org/manual/render/freestyle/introduction.html

Wikipedia



Wikipedia

12.12 演習課題

Processingでレイトレーシング

- joons-renderer
  - Processing から Sunflow を利用するライブラリ
  - github.com/joonhyublee/joons-renderer/wiki/
  - ダウンロード joons???.zip
  - ZIPを展開し，jonesrenderer フォルダをProcessingフォルダの中のlibrariesの中にコピー
  - 右のプログラムで動作確認

- 課題
  - サイトのサンプルを参考にして図形をレンダリングしてみよ
  - または，ビルボードを利用したプログラムを作成せよ

```
import joons.JoonsRenderer;
JoonsRenderer jr;

void setup() {
  size(800, 600, P3D);
  jr = new JoonsRenderer(this);
}

void draw() {
  jr.beginRecord();

  camera(0, 0, 120, 0, 0, -1, 0, 1, 0);
  perspective(PI/4, 4.0/3.0, 10, 1000);

  jr.background("cornell_box", 100, 100, 100);
  jr.background("gi_instant");

  jr.fill("diffuse", 255, 255, 255);
  translate(0,10,-10);
  rotateY(-PI/8); rotateX(-PI/8);
  box(20);

  jr.endRecord();
  jr.displayRendered(true);
}

void keyPressed() {
  if (key == 'r' || key == 'R') jr.render();
}
```

レンダリング結果を保存

色

図形描画

Rキーでレンダリング開始