Graphics with Processing

2023-13 高品質レンダリング

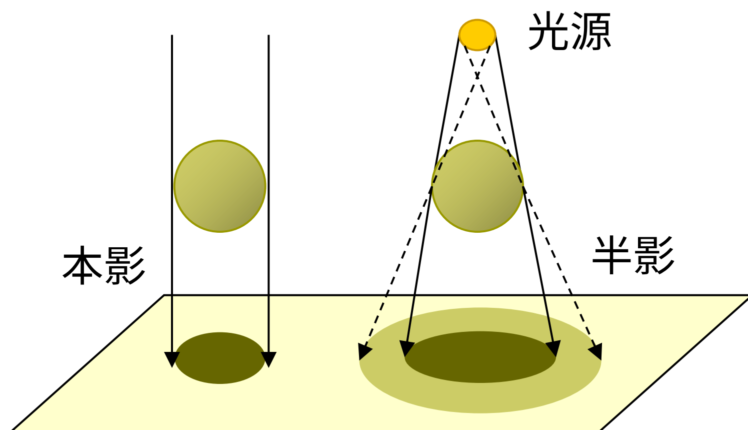
http://vilab.org

塩澤秀和

13.1* 影付け

影の種類(p.158)

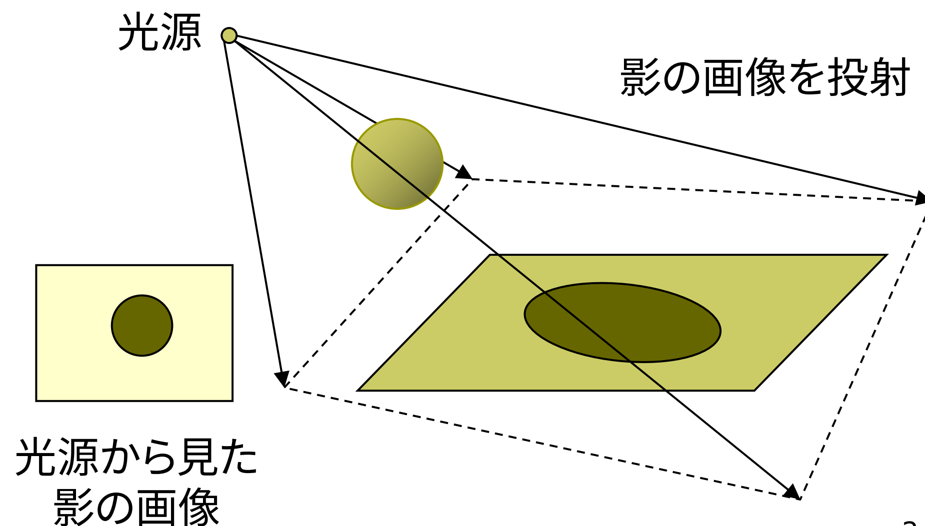
- 本影と半影
  - 点光源や平行光線ではくっきりした影(本影)だけができる
  - 光源に広がりがあると、半影を含むソフトシャドウができる



  - 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
  - リアルタイムな影の生成では、基本的に本影部分を扱う

主な影付け方式

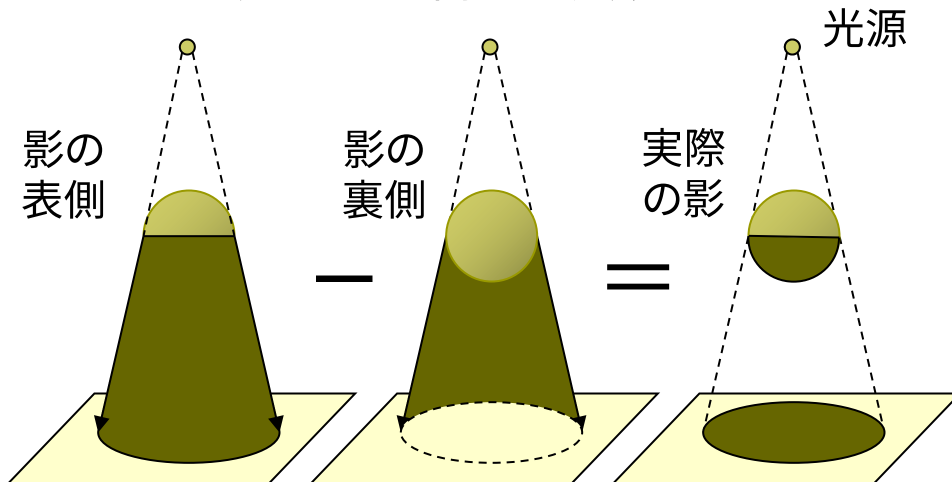
- 影の投影テクスチャマッピング
  - 視点を光源に置いて物体のシルエットを描画すると、光源から見たその物体の影が得られる
  - (視点は戻し、)影の画像を光源の位置から物体の下の床面などに投影テクスチャマッピングする

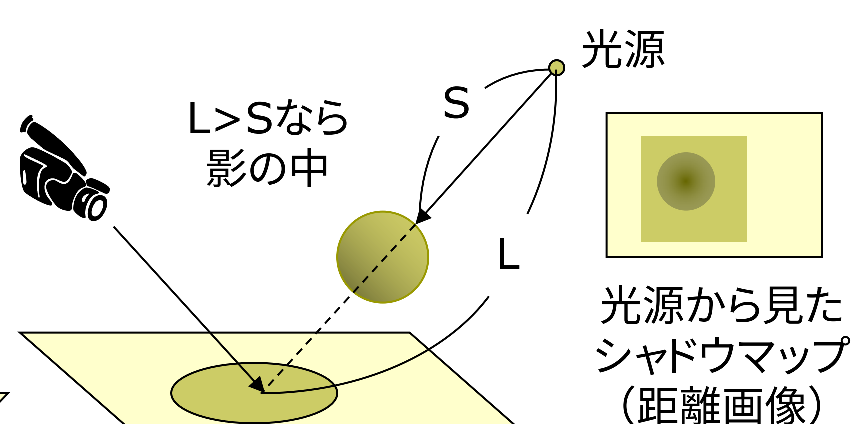


13.2* 影付け（続き）

- 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）を囲む“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、視点から見た影が描画できる
  - 「ステンシル（型抜き）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）
  - 視点を光源に置き、Zバッファだけを描画すると、光の到達距離Sの分布図（シャドウマップ）ができる
  - 視点を戻し、描画対象の3D座標から光源までの距離Lを計算し、シャドウマップ上の対応点の値Sと比較すると、描画対象まで光が届いているか判定できる





13.3 もっと単純な影の例

```
float x = 0, y = -200, z = -200;

void draw() {
  background(50, 50, 100);
  perspective();
  camera(-150, -500, 500,
            0, 0, 0, 0, 1, 0) ;
  directionalLight(200, 200, 200,
                   0, 1, 0);
  ambientLight(128, 128, 128);

  z++;

  fill(0, 150, 0);
  beginShape(QUADS);
  float w = 500;
  vertex(-w, 0, -w); vertex(-w, 0, w);
  vertex(w, 0, w); vertex(w, 0, -w);
  endShape();

  // 本体の表示
  pushMatrix();
    fill(255);
    drawObjects();
  popMatrix();

  // 縦方向に潰して真上からの影を作成
  pushMatrix();
    fill(0, 180);          // 黒色(半透明)
    translate(0, -1, 0); // 地面の少し上
    scale(1, 0.1, 1);     // y方向に潰す
    drawObjects();
  popMatrix();
}

void drawObjects() {
  translate(x, y, z);
  rotateX(PI/3); rotateY(PI/6);
  box(100);
}
```

13.4* 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
  - 60fps以上（最低限10fps程度）
- 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

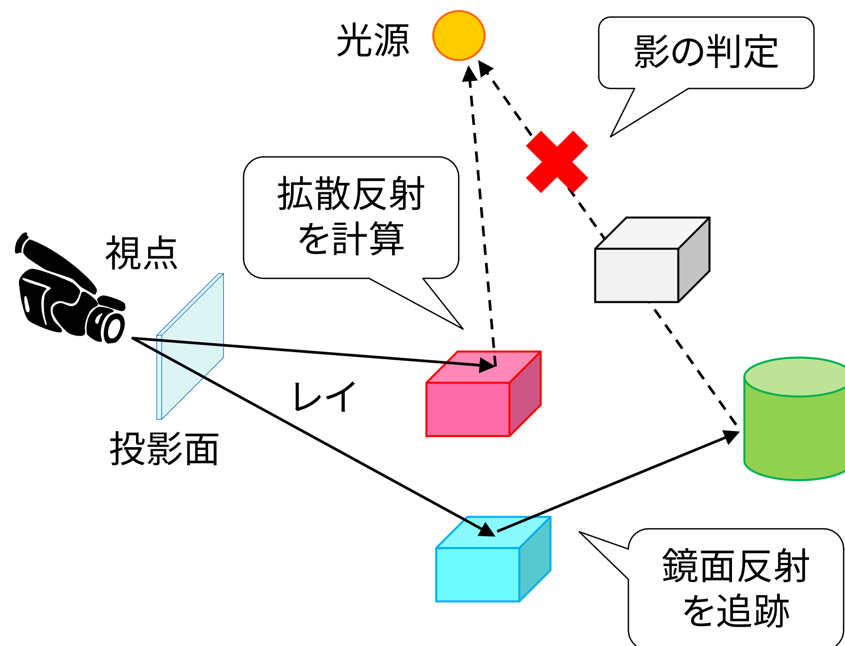
高品質レンダリングの例

- POV-Ray
  - hof.povray.org
- Blender+Cycles
  - www.cycles-renderer.org
- LuxCoreRender
  - luxcorerender.org/gallery/
- Sunflow
  - sunflow.sourceforge.net/index.php?pg=gall
- Unreal Engine
  - リアルタイムレイトレーシング
  - docs.unrealengine.com/ray-tracing-and-path-tracing-features-in-unreal-engine/

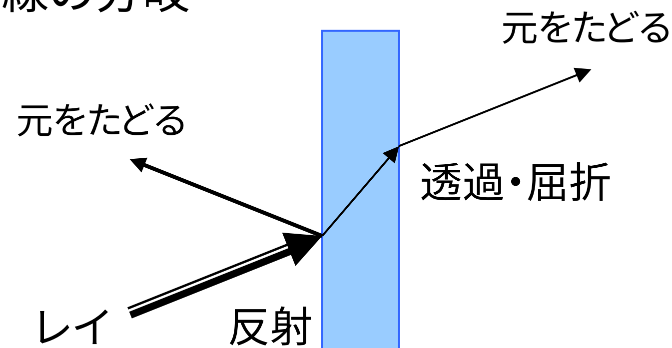
13.5* レイトレーシング(p.135)

レイトレーシング(光線追跡)法

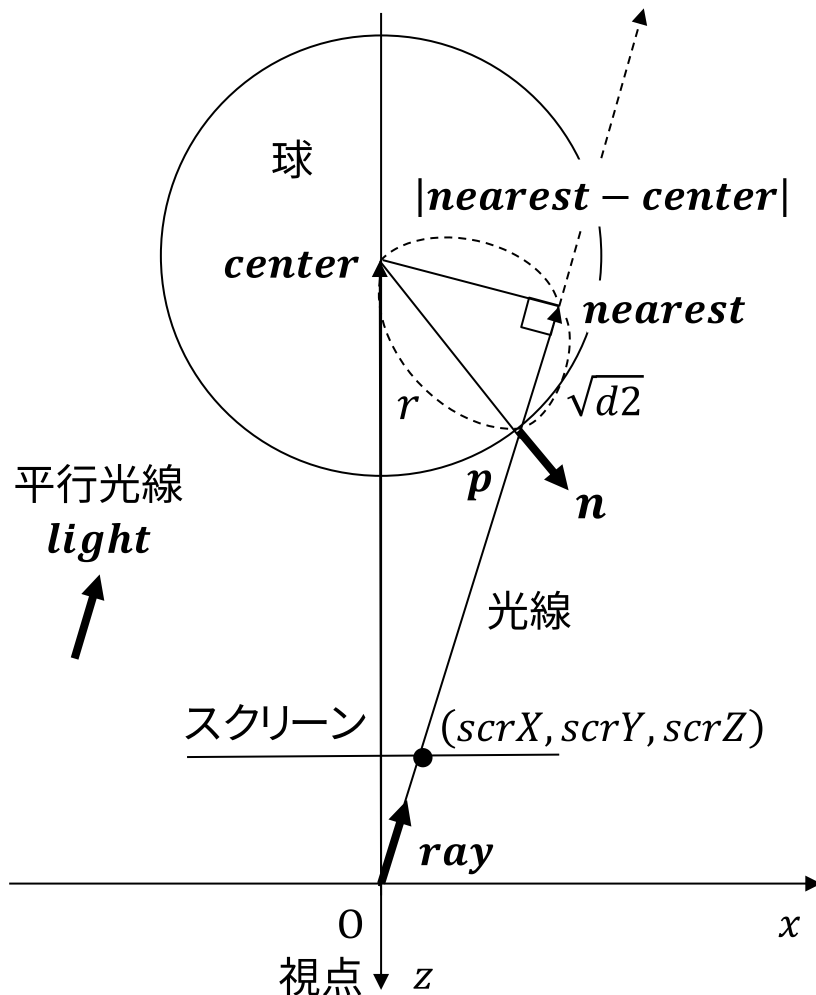
- 概要
  - 画面の各画素に届く光線(レイ)を視点から逆方向に追跡する
  - 視点から、各画素に対応するレイ(半直線)を"飛ばす"
  - レイが物体と交差(衝突)したら、材質と照明から画素の色を求める
  - 影を描画する場合、衝突点と光源の間に障害物があるか判定する
  - 鏡面反射、透過・屈折を扱う場合、レイを分岐して再帰的に追跡する
- 特徴
  - 隠面消去や影付けが容易
  - 透明、レンズ、映り込み等も再現
  - 映像作品(映画等)では一般的
  - まだリアルタイム処理には不向き



光線の分岐



13.6 レイを飛ばす処理の基本




```
// レイトレーシングの基本となるレイと球の
// 交差判定の例（レイキャスティング）
void setup() { size(600, 600);  noLoop(); }

// 視点座標系における球の中心と半径
PVector center = new PVector(0, 0, -10);
float r = 1.0;
// 照明（平行光線）の方向ベクトル
PVector light =
  new PVector(1, 1, -3).normalize();

void draw() {
  // 全ピクセルに対してレイを飛ばして描画
  loadPixels();
  for (int x = 0; x < width; x++)
    for (int y = 0; y < height; y++)
      pixels[y * width + x] = raycast(x, y);
  updatePixels();
}
```

13.7 レイを飛ばす処理の基本（続き）

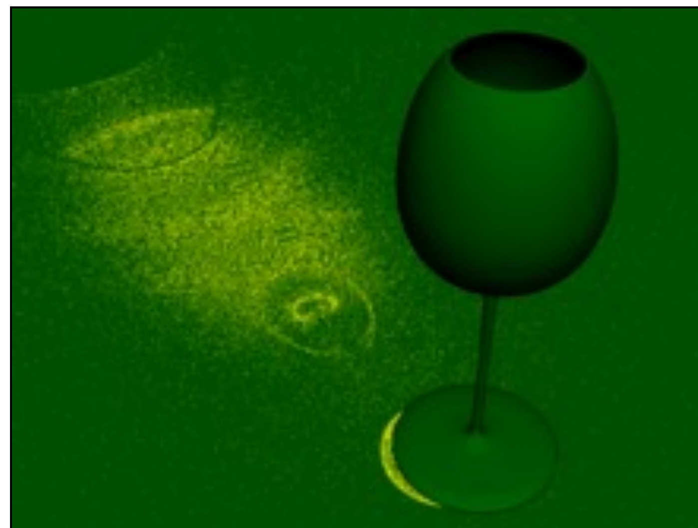
```
color raycast(int x, int y) {

  // 視点座標系で視点（原点）の前（z=-2）に
  // xy座標が-1～+1のスクリーンを想定
  float scrX = (x * 2.0 - width) / width;
  float scrY = (y * 2.0 - height) / height;
  float scrZ = -2.0;

  // 視点から仮想スクリーンのピクセルを
  // 貫いていくようなレイを飛ばす
  PVector ray =
    new PVector(scrX, scrY, scrZ);
  ray.normalize();

  // レイを飛ばした延長線上で球の中心に
  // 最も近づく点を求める
  PVector nearest =
    PVector.mult(ray, center.dot(ray));

  // 球の中心からその点までの距離を求める
  PVector l = PVector.sub(nearest, center);

  // その距離が球の半径よりも短ければ…
  float d2  = r * r - l.magSq();
  if (d2 > 0) {
    // レイは球に当たっているので、球面上の
    // 交点とそこでの法線ベクトルを求める
    PVector p = PVector.sub(nearest,
      PVector.mult(ray, sqrt(d2)));
    PVector n = PVector.sub(p, center);
    n.normalize();

    // ランバート反射による拡散反射の計算
    float f = -n.dot(light);  // cosなので内積
    if (f > 0) return color(f * 255);
  }
  return color(0);
}
```

13.8* 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使って、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よい
  - 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



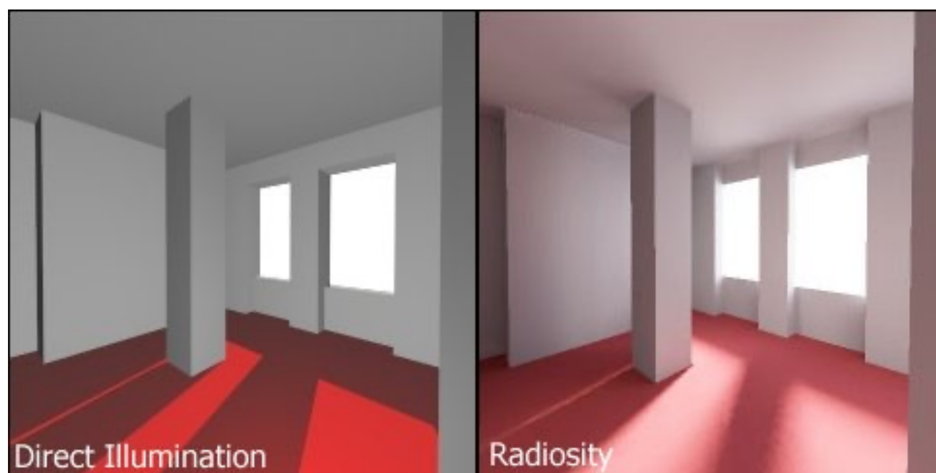
計算された光子の分布

13.9* 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



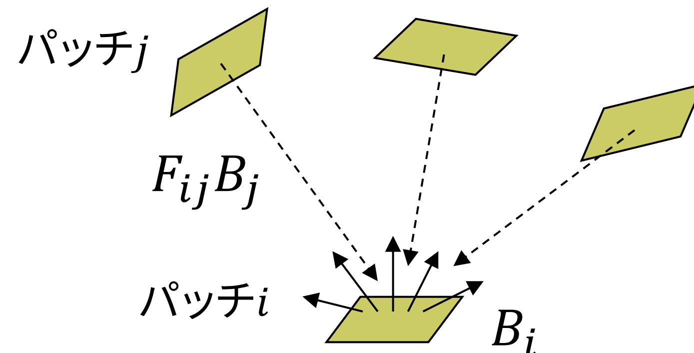
柔らかい影や壁の色の影響が表現されている

- 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
B_i パッチiの光の放射量(ラジオシティ)
E_i パッチiの発光量
ρ_i パッチiの反射率
F_{ij} フォームファクタ($F_{ij} = F_{ji}$)

- 巨大な「連立一次方程式」になる
  ⇒ コンピュータによる数値計算



13.10 その他のレンダリング関連技術

「ぼかす」方法

- 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
  - 遠くにあるほどかすんで、彩度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

- CGに画像を利用（p.171）
  - CGと画像処理技術を融合した多様な技術が開発されている
  - 実写画像による投影テクスチャマッピングや環境マッピング
  - イメージベーストライティング：360°画像に記録された照明の情報を利用してレンダリング
  - 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
  - 実写の中にCGを合成（AR）
  - CGの中に実写映像を合成
  - 自由視点画像：限られた台数で撮影したカメラ映像から、自由な視点から見た映像を合成する

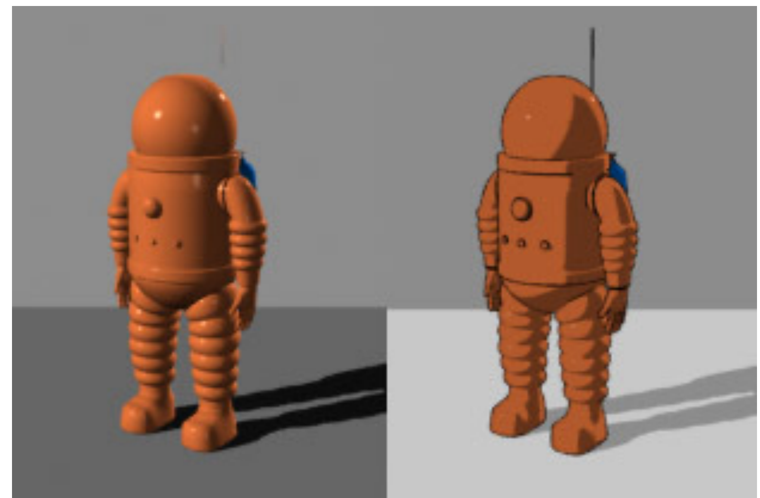
13.11* 非写実的レンダリング

ノンフォトリアリスティック（非写実的）レンダリング（NPR）（p.309）

- 概要
  - 現実の再現を目的としないCG
  - 例）油絵風、手描きタッチの再現、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、かなり完成された
  - 漫画・アニメーションへの応用
  - 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

docs.blender.org/manual/en/latest/render/freestyle/introduction.html

Wikipedia



Wikipedia

13.12 演習課題

自由課題

- レイトレーシング(自由提出)
  - 13.6のプログラムを改造し、複数の球を表示させる
  - または、任意のソフトウェアで、レイトレーシングの画像を作る

Processingでレイトレーシング

- joons-renderer
  - Sunflowを用いるライブラリ
  - github.com/joonhyublee/joons-renderer
  - 講義資料のWebサイトから joons102.zip を取得
  - 展開後、jonesrenderer フォルダをProcessingフォルダの中のlibrariesの中にコピー

```
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();
}
```