Graphics with Processing

2016-12 レンダリング技術

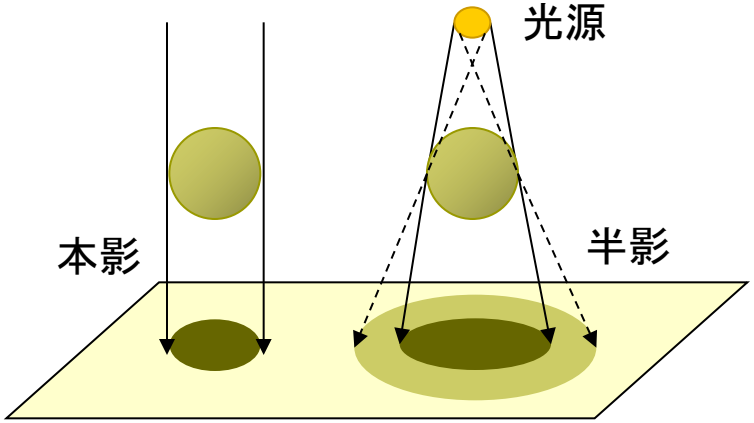
http://vilab.org

塩澤秀和

12.1 影付け

影の種類(p.158)

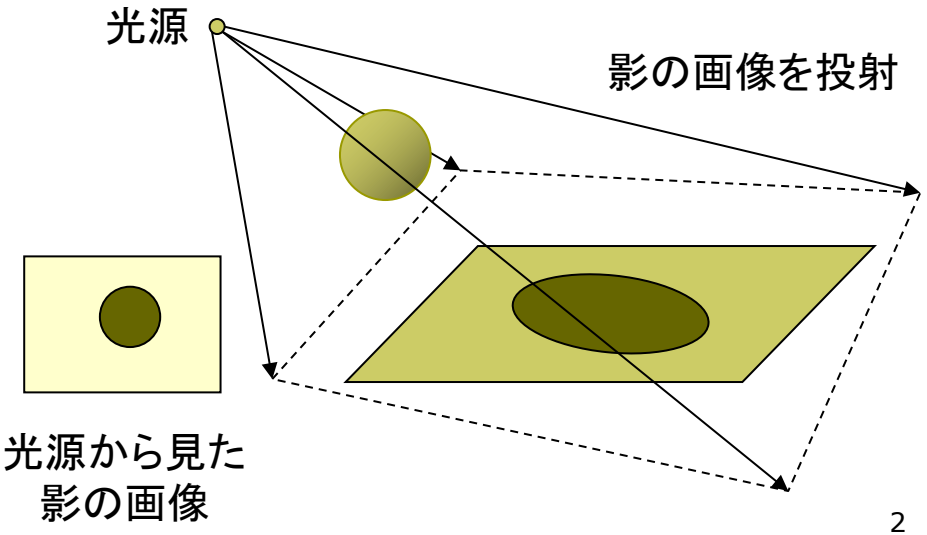
- 本影と半影
  - 点光源や平行光ではくっきりした影(本影)だけができる
  - 光源に広がりがあると，半影を含むソフトシャドウができる



  - 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

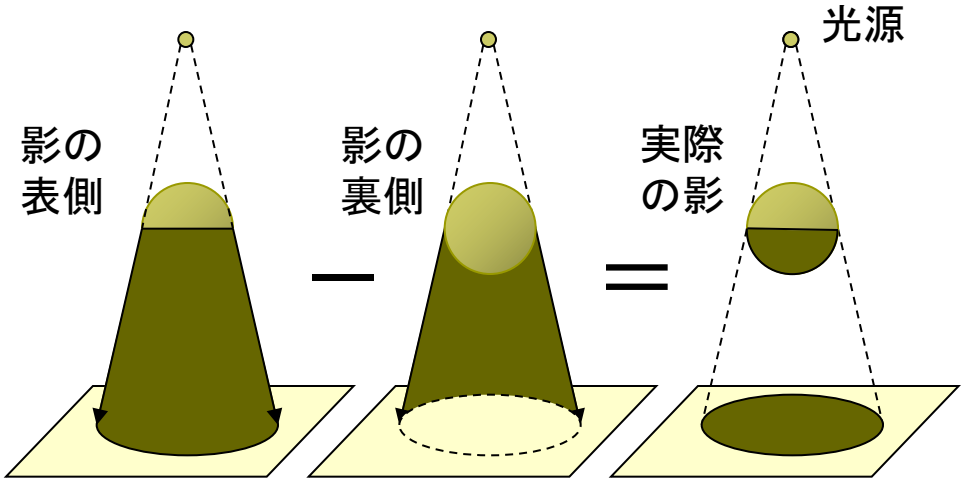
主な影付け方式

- 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

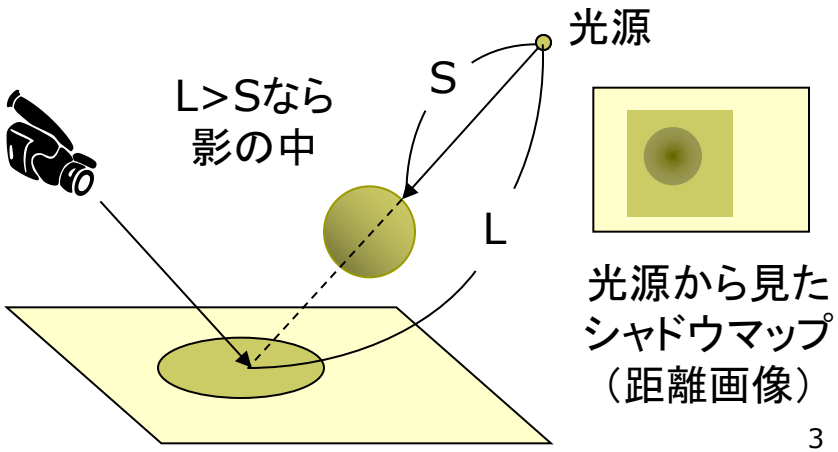


12.2 影付け（続き）

- 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



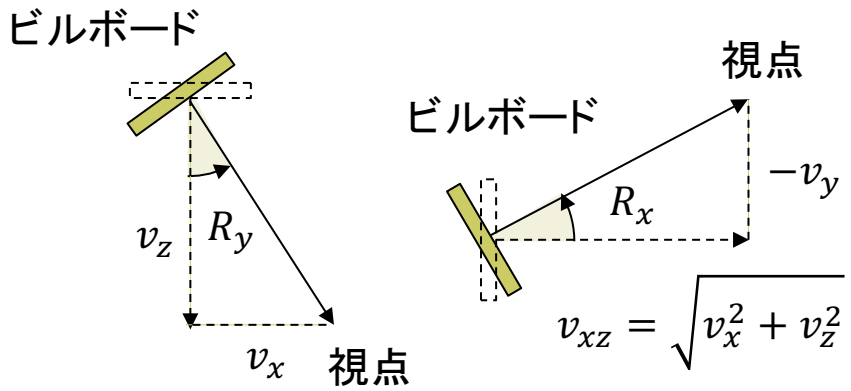
- シャドウマップ法（p.159）
  - （Zバッファを用いた2段階法）
  - 光源から見た場合のZバッファを構成すると，光の到達距離Sのマップ（シャドウマップ）ができる
  - 視点を戻し，レンダリングするオブジェクトから光源までの距離Lとシャドウマップ上の対応点の内容（S）を比較し，光がそこまで届いているか判定する



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
  - 遠方の雲やパーティクル（粒子による雪や火花の表現）で有効



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);

  // ビルボードの描画
  for (PVector pos : bbpos) {
    pushMatrix();
    translate(pos.x, pos.y, pos.z);
    // 視点へ向かうベクトルを求める
    PVector v = camPos.copy();
    v.sub(pos);
    // 横にRy回転し，正面を視点に向ける
    rotateY(atan2(v.x, v.z));
    // 縦にRx回転し，正面を視点に向ける
    //float vxz = dist(0, 0, v.x, v.z);
    //rotateX(atan2(-v.y, vxz));

    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 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

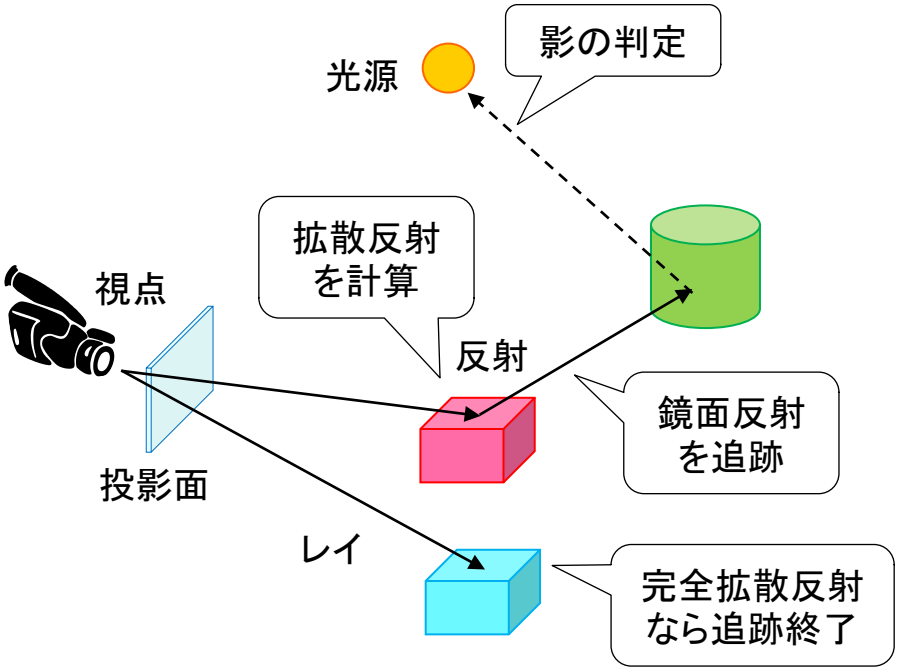
フリーソフトによるレンダリングの例

- 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（開発終了？）
- Art of Illusion
  http://www.artofillusion.org
  → Art Gallery

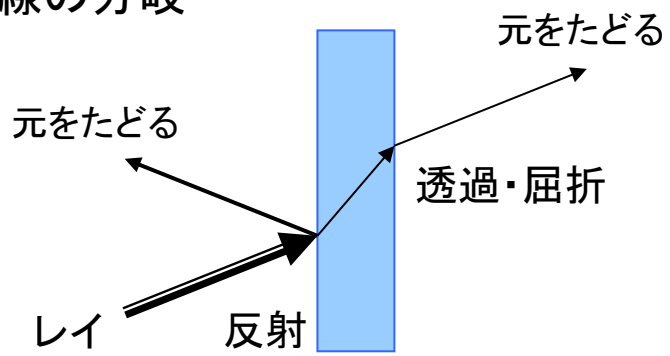
12.6 レイトレーシング (p.135)

レイトレーシング（光線追跡）法

- 概要
  - 画面の各ピクセルに届く光線（レイ）を視点から逆方向に追跡
  - 視点から各ピクセルに対応するレイ（半直線）を“飛ばす”
  - レイが物体と交差（衝突）したら，照明等から描画色を計算
  - 影を描画する場合，衝突点から光源にレイを飛ばして判定
  - 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
- 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
  - リアルタイム処理にはまだ不向き



光線の分岐



12.7 レイを飛ばす処理の基本

```
// レイトレーシングの基本（レイキャスティング）
// によるレイと球の交差判定の例
import static processing.core.PVector.*;

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（方向光）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，レイを飛ばして画面描画
  loadPixels();
  for (int x = 0; x < width; x++)
    for (int y = 0; y < height; y++)
      pixels[y * width + x] = raycast(x, y);
  updatePixels();
}
```

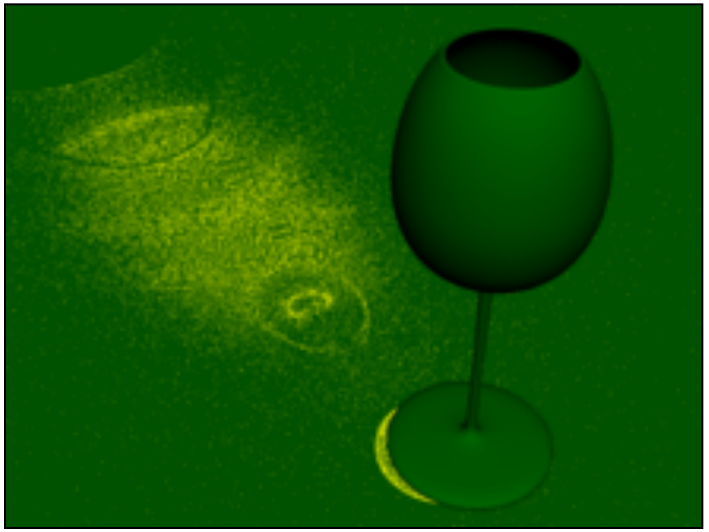
```
color raycast(int x, int y) {
  // 視点座標系で視点（原点）の前にスクリーンを想定
  float scrX = (x * 2.0 - width) / width;
  float scrY = (y * 2.0 - height) / height;
  float scrZ = -2.0;
  // 視点から仮想スクリーンの点の方向にレイを飛ばす
  PVector ray = new PVector(scrX, scrY, scrZ);
  ray.normalize();
  // レイの延長線上で球の中心に最も近づく点を求める
  PVector nearest = mult(ray, center.dot(ray));
  // その点が球の内側なら交差あり（効率優先の計算式）
  float d  = r * r - sub(nearest, center).magSq();
  if (d > 0) {
    // 球面上の交点とそこでの法線ベクトルを求める
    PVector p = sub(nearest, mult(ray, sqrt(d)));
    PVector n = sub(p, center).normalize();
    // ランバート反射によるシェーディング計算
    float f = - n.dot(light);
    if (f > 0) return color(f * 255);
  }
  return color(0);
}
```

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



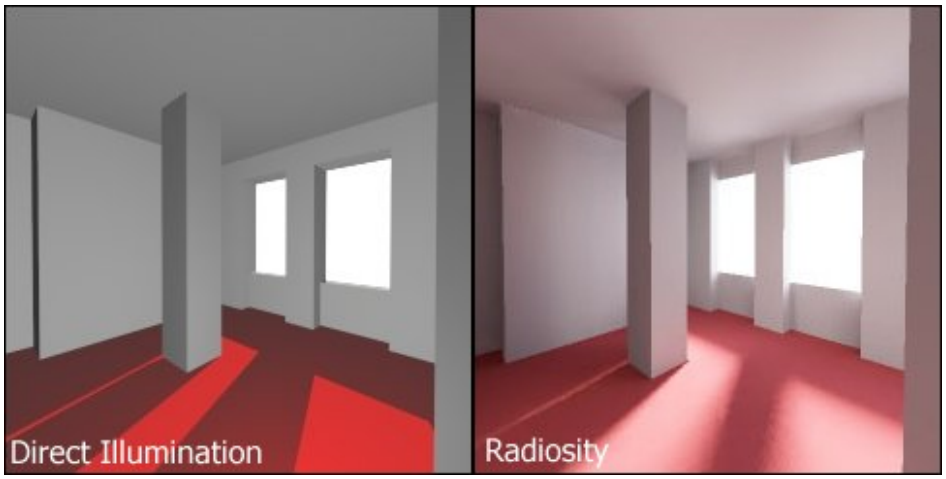
計算された光子の分布

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



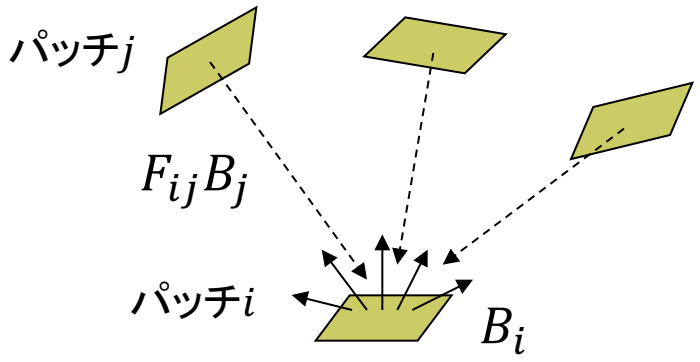
柔らかい影や壁の色の影響が表現されている

- 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
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（$F_{ij} = F_{ji}$）

- 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(IBL):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

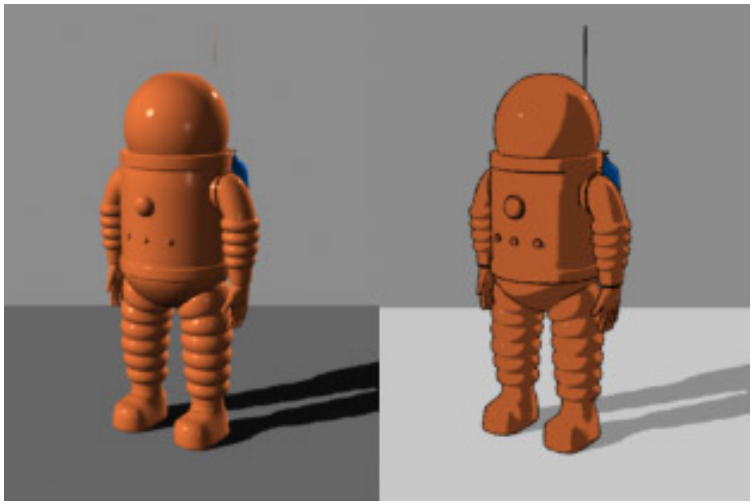
12.11 非写実的レンダリング（p.309）

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
  - Sunflowを利用するライブラリ
  - github.com/joonhyublee/joons-renderer/
  - コンパイル済み→ vilab.org/cg2016/joons102.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

- 自由課題
  - レイトレーシングで適当な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
  - 今回の課題は提出しなくてよい

```
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