Graphics with Processing

2019-11 シェーディングとマッピング

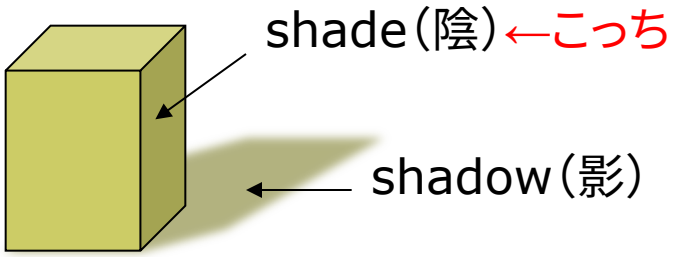
http://vilab.org

塩澤秀和

11.1* シェーディング

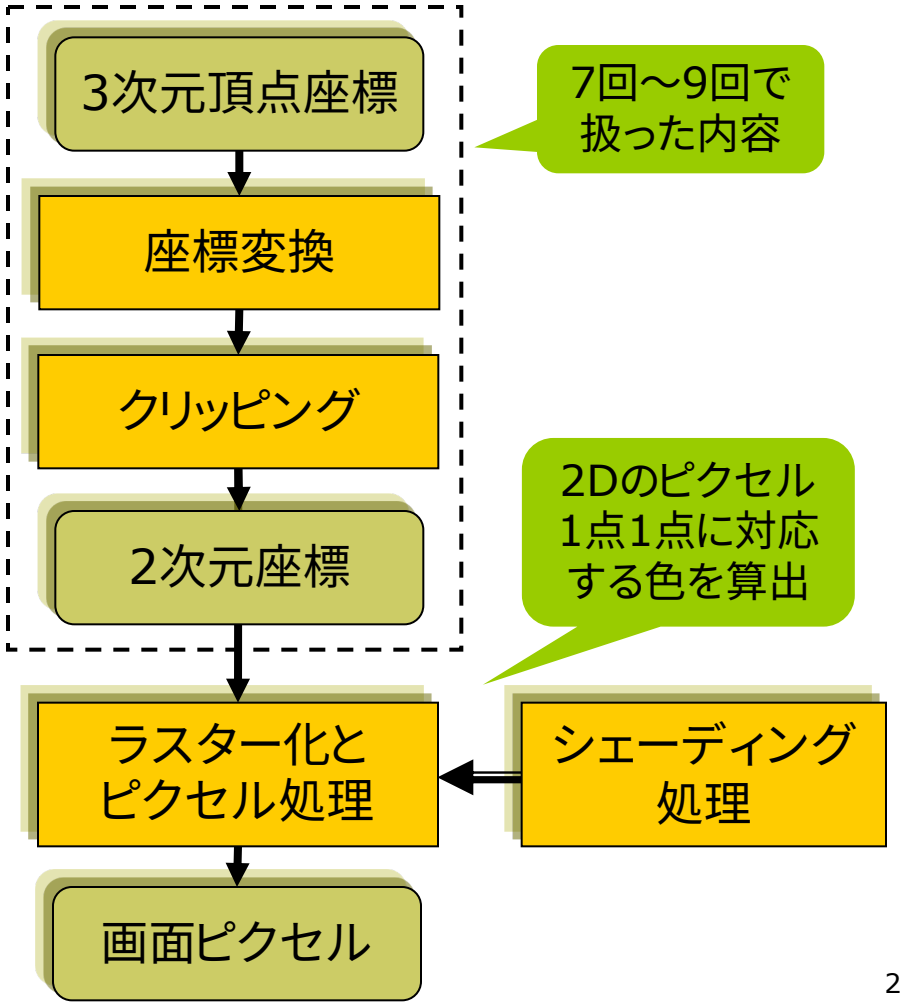
シェーディング(shading)

- シェーディングモデル(p.141)
  - 光の反射・材質のモデル(前回)
  - ポリゴンの明暗の計算モデル



- プログラマブルシェーダー
  - 最近のGPUはプログラム言語で内部処理を変更できる
  - 水面や人の肌など対象ごとに，シェーディングをプログラム
  - 最終回で少し触れる予定

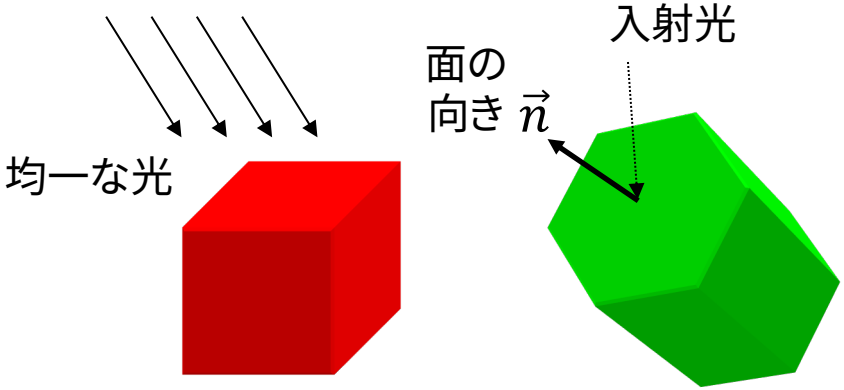
シェーディングは2D化後



11.2* フラットシェーディング

フラットシェーディング（p.155）

- 各ポリゴンを単一色で描画
  - ポリゴンの代表点（例：重心）を決め，法線ベクトルを求める
  - 代表点における入射光と法線ベクトルから反射光（色）を計算
  - 面全体をその色で塗り潰す
  - 均一な平行光が平面に当たる場合は，光学的に正しい
  - 高速だがリアリティには欠ける



法線ベクトル（p.67）

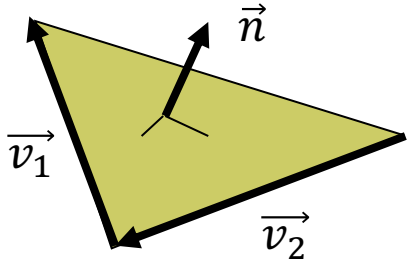
平面の方程式から求める方法

$$ax + by + cz + d = 0$$

$$\vec{N} = (a, b, c)$$

曲面の場合は？

$$\left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)$$



ポリゴンの辺（ベクトル）から求める方法

$$\vec{N} = \overrightarrow{v_1} \times \overrightarrow{v_2}$$

外積（クロス積）

$$\begin{pmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{pmatrix}$$

単位法線ベクトル

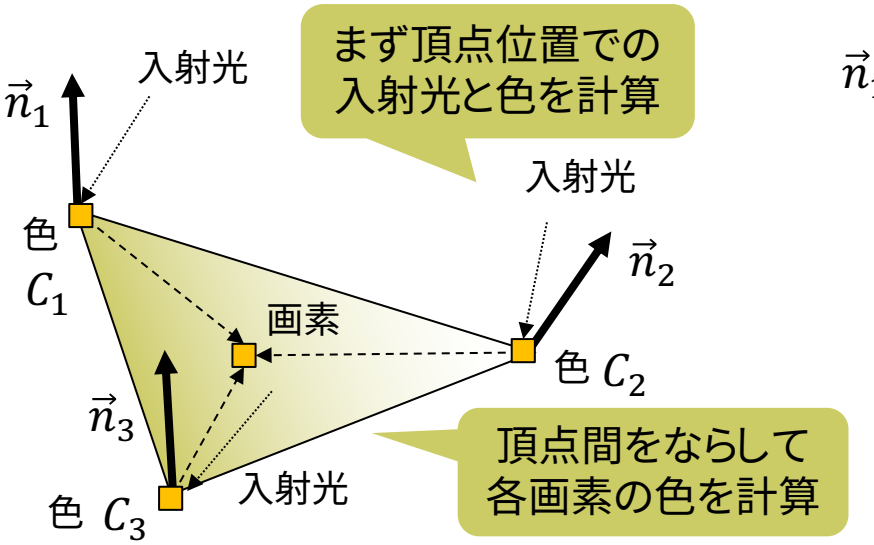
$$\vec{n} = \vec{N} / \left|\vec{N}\right|$$

（大きさを1にする）

11.3* スムースシェーディング

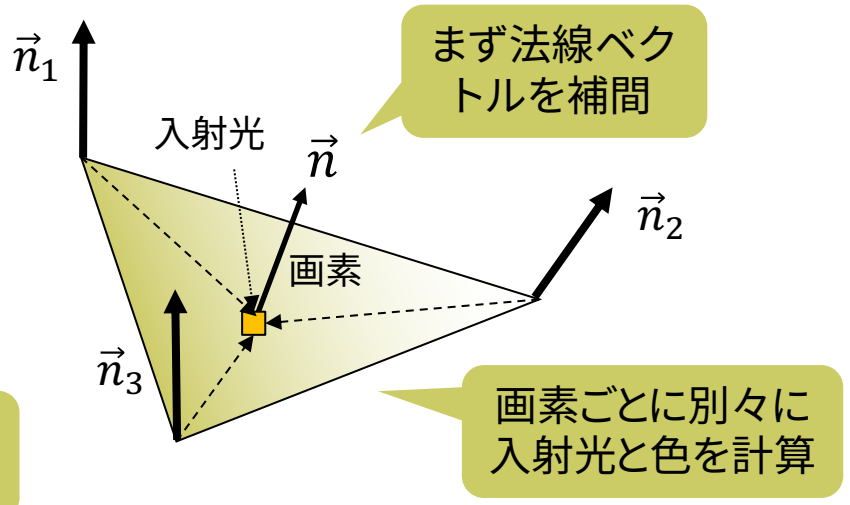
グロー(Gouraud)シェーディング

- 頂点間の色を補間(p.156)
  - 各頂点に法線ベクトルを設定
  - 各頂点で入射光と法線ベクトルから反射光(色)を計算する
  - その色を2D画面上で滑らかに補間して面全体を塗り潰す
  - 処理が高速



フォン(Phong)シェーディング

- 法線ベクトルを補間(p.157)
  - 各頂点に法線ベクトルを設定
  - 2D画面上の各画素に対応する法線ベクトルを補間して算出
  - 画素ごとに入射光と反射光(色)を計算し,領域を塗り潰す
  - 鏡面反射の光沢をリアルに表現



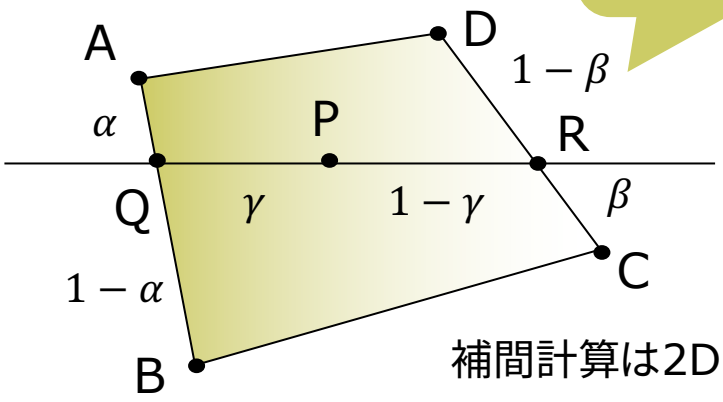
11.4 バイリニア補間

バイリニア補間(p.156)

- 2段階の線形補間

画面描画のために
ラスター化されたポリゴン

凸ポリゴンを横切る直線は必ず2辺で交わる



補間計算は2D上

1段階目

$$V_Q = (1-\alpha)V_A + \alpha V_B$$
$$V_R = (1-\beta)V_C + \beta V_D$$

2段階目

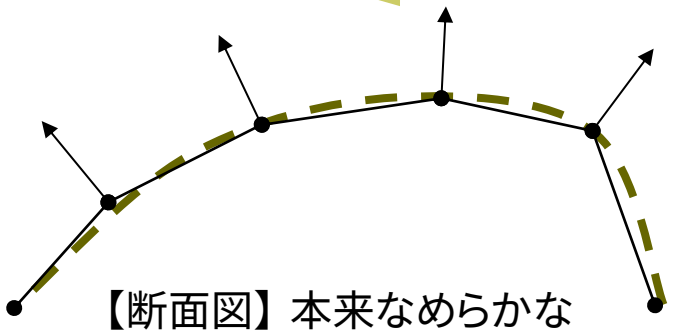
$$V_P = (1-\gamma)V_Q + \gamma V_R$$

各シェーディングでの補間処理

- グローシェーディング
  - 各頂点の色のR,G,B成分をそれぞれバイリニア補間
  - ↓
  - 各ピクセルの補間色を合成
- フォンシェーディング
  - 各頂点の法線ベクトルのx,y,z成分をそれぞれバイリニア補間
  - ↓
  - 各ピクセルの法線ベクトルを合成
- テクスチャマッピングにおける対応uv座標の計算にも使用

11.5* ポリゴン曲面

ポリゴンモデルでは，頂点に法線ベクトルを持たせる

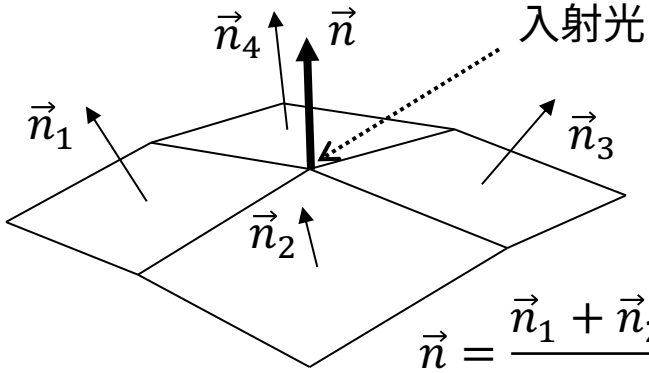


【断面図】本来なめらかな曲面を多数のポリゴンで近似

ポリゴン曲面（p.94）

- 曲面の近似
  - 曲面をポリゴンの集合で表す
  - 三角形を使うことが多い（頂点が必ず同一平面上にあるから）
  - 各頂点のデータとして元の曲面の法線ベクトルを持たせる
  - 各ポリゴンをスムースシェーディングで描画すると，色が滑らかに連続して曲面として見える

曲面の数式がない場合は周囲のポリゴンの法線ベクトルを平均化



$$\vec{n} = \frac{\vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4}{4}$$

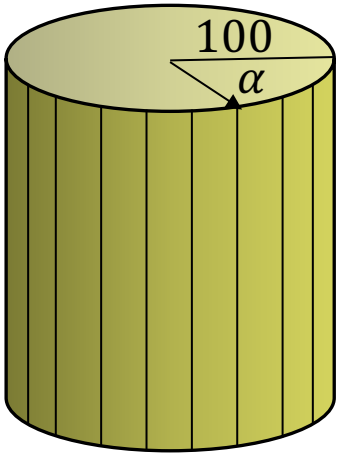
法線ベクトルの設定

- normal(nx, ny, nz)
  - 頂点にシェーディングのための法線ベクトルを明示的に設定
  - 対応するvertexの直前で使う

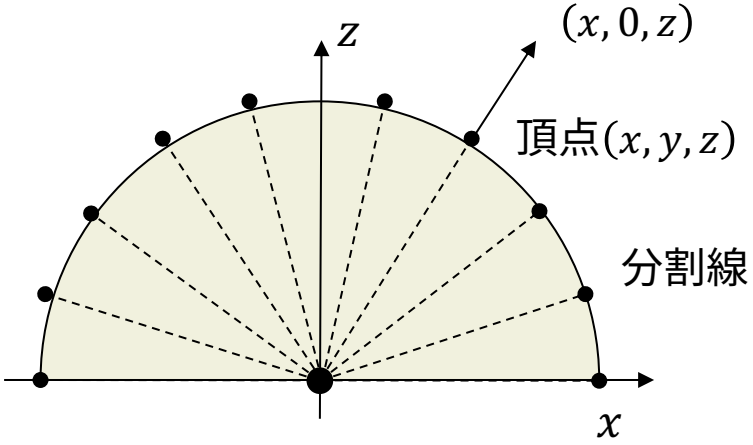
11.6 ポリゴン曲面の例

実は，QUAD_STRIPで描けば
Processingが滑らかに見える
法線ベクトルを設定してくれる

長方形を貼り合わせて
円柱をモデリング



$$x = 100\cos\alpha$$
$$z = 100\sin\alpha$$



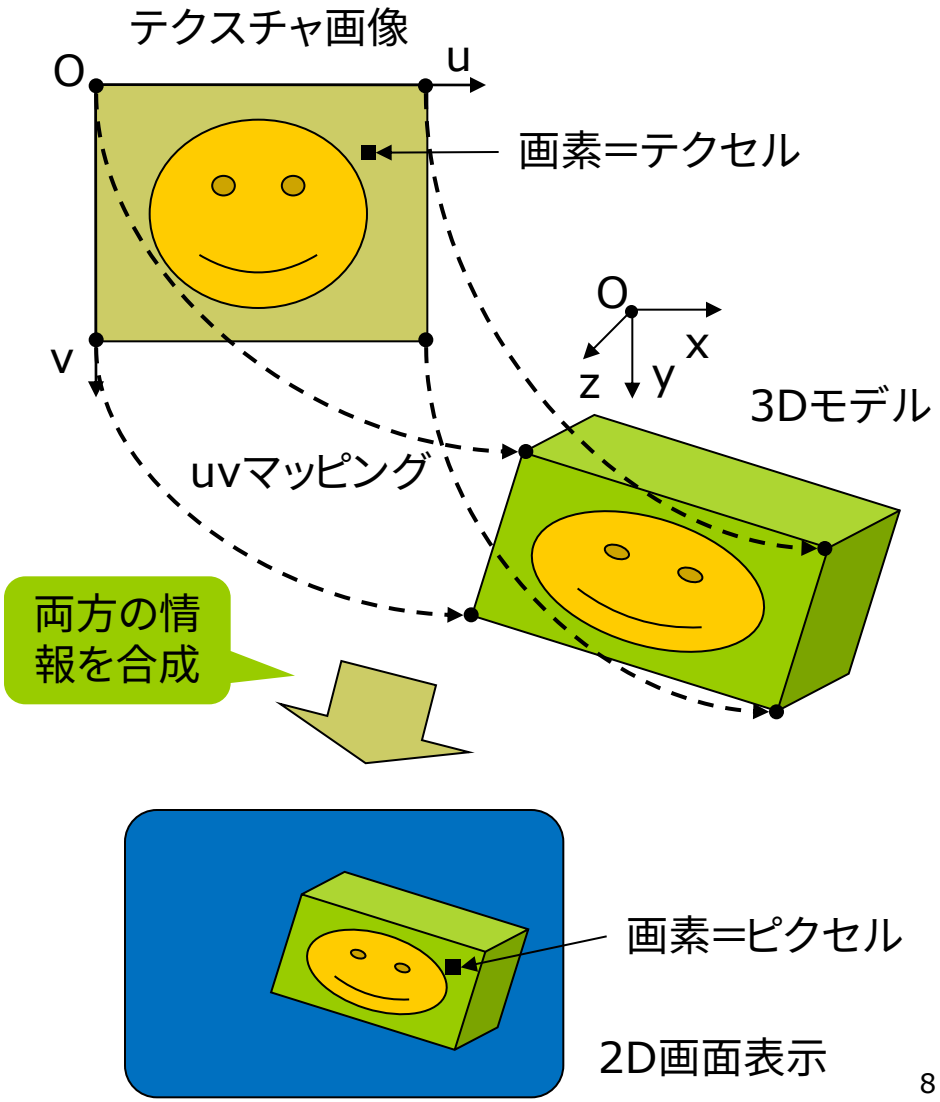

```
noStroke();
beginShape(QUADS);
float d = 1.0/18;  // 18分割
for (float f = 0.0; f < 1.0; f += d) {
    float a, x, z;
    a = TWO_PI * f;
    x = 100 * cos(a);
    z = 100 * sin(a);
    if (mousePressed) normal(x, 0, z);
    vertex(x, -100, z);
    vertex(x,  100, z);

    a = TWO_PI * (f + d);
    x = 100 * cos(a);
    z = 100 * sin(a);
    if (mousePressed) normal(x, 0, z);
    vertex(x,  100, z);
    vertex(x, -100, z);
}
endShape();
```

fは1周を1とした角度

11.7* テクスチャマッピング(p.162)

テクスチャマッピング

- texture=布目・模様
  - 立体表面に画像(=色分布)をシールのように貼りつける
  - 例) 球に世界地図を貼りつける
  - 質感を表すのに効果てきめん
  - テクスチャ画像の画素のことをテクセル(texel)という
- uv座標(テクスチャ座標)
  - テクスチャ画像内の2D座標
  - モデリング座標と区別するため,(u, v) (またはs,t)で表す
- uvマッピング
  - ポリゴンの各頂点に画像内の点(uv座標)を対応させる処理
  - 画像(u, v) → 空間(x, y, z)



11.8 テクスチャマッピング関数

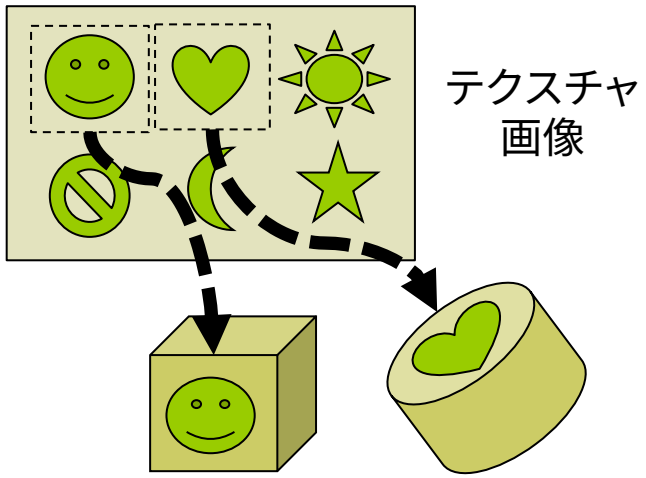
テクスチャマッピング関数

- texture(画像)
  - 画像: PImage型(第3回参照)
  - テクスチャ画像の設定
  - beginShape(), endShape()の中で指定する
- vertex(x, y, z, u, v)
  - 頂点(x, y, z)を追加し,そこにテクスチャ画像内の点(u, v)を対応づける
  - 2Dでの画像変形にも使える vertex(x, y, u, v)
- textureMode(モード)
  - uv座標の値の範囲
  - NORMAL: 0.0～1.0に正規化
  - IMAGE: 画像のピクセル座標

画像の右下隅が u=1.0, v=1.0

- テクスチャマッピングの活用
  - Topics (3D) → Textures
  - テクスチャとポリゴンの形状が違う場合,自動的に変形される
  - 画像の小領域を別々のオブジェクトに貼り付けることもできる

  ⇒ “テクスチャアトラス”



11.9 テクスチャマッピングの使用例

```
// 準備：画像ファイル（kouji50m.jpg）を
// あらかじめ講義ページからダウンロードして
// スケッチのdataフォルダに入れておく
// （メニューで Sketch → Add File...）

PImage tex;

void setup() {
  size(300, 300, P3D);
  tex = loadImage("kouji50m.jpg");
}

void draw() {
  background(0);
  translate(width/2, height/2, 0);
  rotateY(-radians(frameCount));
  noStroke();
  beginShape(QUADS);
  texture(tex);
  textureMode(NORMAL);
  vertex(-20,-50,  0,  0.0, 0.0);
  vertex( 20,-50,  0,  1.0, 0.0);
  vertex( 20, 50, 15,  1.0, 1.0);
  vertex(-20, 50, 15,  0.0, 1.0);
  endShape();

  fill(#ffffff, 128);
  stroke(#555555);
  beginShape(QUADS);
  vertex(-20,-50,   0);
  vertex( 20,-50,   0);
  vertex( 20, 50, -15);
  vertex(-20, 50, -15);
  endShape();
}
```

画像はグローバル変数で定義する

画像はsetupの中で一度だけ読み込む

長方形に画像texをテクスチャマッピング

uv座標は0～1モード

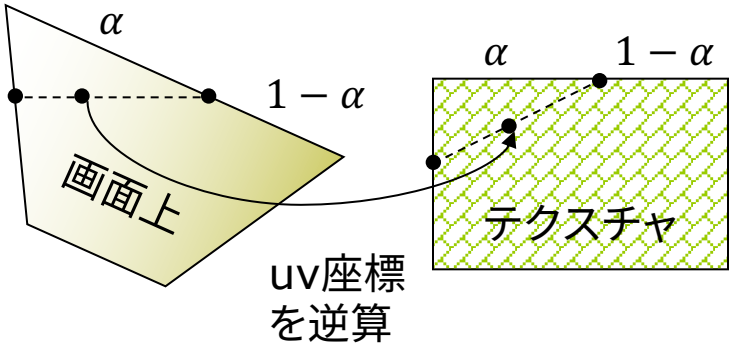
0以上1以下で数値を色々変更してみよ

半透明や透過テクスチャは最後に描画

11.10 テクスチャの描画処理

テクスチャの描画処理(p.148)

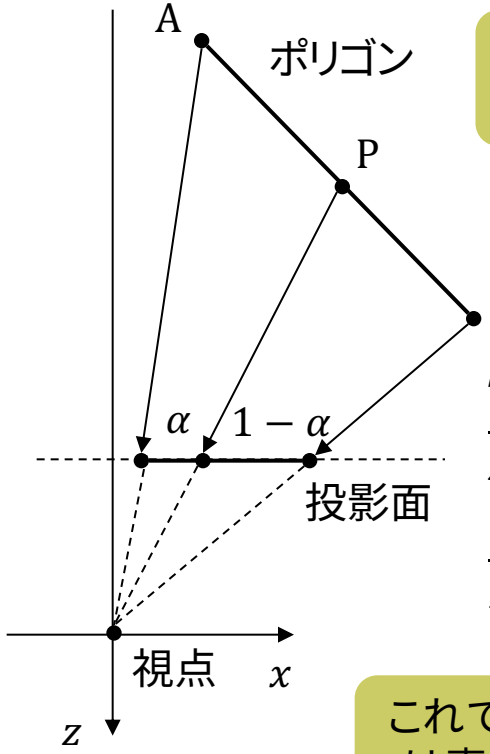
- 段階1:画面座標→uv座標
  - 画面上の各ピクセルの描画時に元のuv座標を逆算する
  - ポリゴンの頂点が持つuv座標値からバイリニア補間する



- 段階2:uv座標→描画色
  - 求めたuv座標の周辺テクセルから補間等で描画色を決める
  - 画像の拡大・変形技術と同じ

パースペクティブ補正

- 透視投影での補正
  - 遠くにいくほどテクスチャを縮めることで,“ゆがみ”を防ぐ



3D→2Dの投影で長さの比が狂う

テクスチャ座標を $T = (u, v)$ とすると

$$\frac{T_P}{z_P} = (1-\alpha)\frac{T_A}{z_A} + \alpha\frac{T_B}{z_B}$$

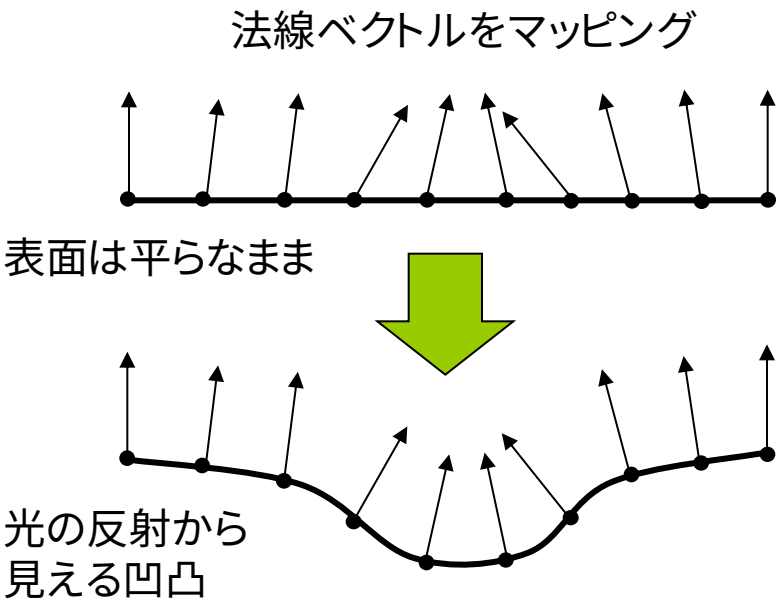
$$\frac{1}{z_P} = (1-\alpha)\frac{1}{z_A} + \alpha\frac{1}{z_B}$$

これで計算できる理由は専門書などを参照

11.11* その他のマッピング技術

バンプマッピング（p.166）

- 凹凸を表面にマッピング
  - 表面の細かい凹凸から計算した法線ベクトルの分布を貼り付け，凹凸があるような陰影をつける
  - 任意の法線分布を用意しておくものは「法線マッピング」という



その他のマッピング

- 視差マッピング
  - 凹凸のある表面を違う方向から見ると，でっぱりの高さによって位置がずれて見える（視差）
  - 視線と高さに応じて，uv座標をずらしてマッピングする
- 投影テクスチャマッピング（p.164）
  - プロジェクタで投影するように，テクスチャを貼り付ける
  - 影の表現にも応用できる
- 環境マッピング（p.168）
  - 金属などへの景色の映り込みをテクスチャで表現する
  - 視点から物体の表面に反射して見えるシーンをレンダリングし，その画像をテクスチャとして貼る

11.12 演習課題

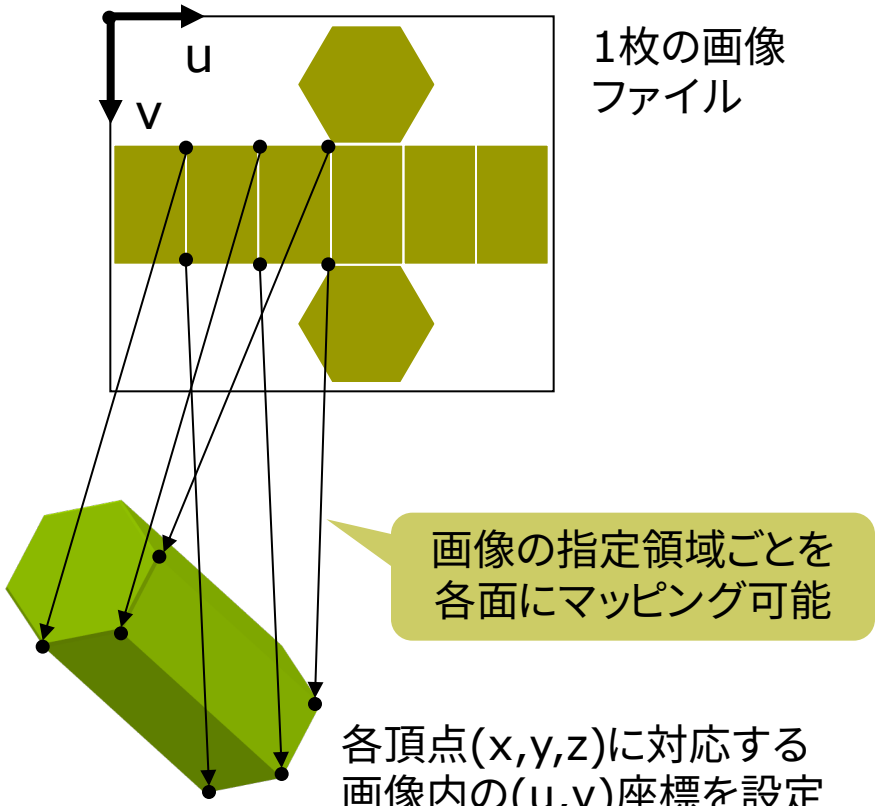
課題

- 11.6のサンプルをもとに円筒の表面に画像をぐるりと貼り付けるプログラムを作成しなさい
  - アニメーション等によって円筒の全表面が見えるようにすること
  - PNG形式などで透過色がある画像を貼ってみると面白い
  - 提出するファイル：ツール → スケッチをアーカイブで，画像もまとめたZIPファイルを作る

期末レポート予告

- 3DCGの作品制作
  - 3人までのチーム制
  - テーマ：「教室」，「島」，「飛ぶ」

展開図などの利用方法



ヒント：vertexのuvの値をうまく設定する

11.13 参考：オフスクリーンレンダリング

```
// いったん“隠し画面”に描いた図形を
// 画像としてポリゴンに貼り付ける例
PGraphics pg;  // 隠し画面用変数

void setup() {
  size(400, 300, P3D);
  // 隠し画面を開く
  // 3つの引数の意味はsize関数と同じ
  pg = createGraphics(100, 100,
         JAVA2D);
}

void draw() {
  // 隠し画面上での描画処理
  pg.beginDraw();  // 開始
  pg.background(255);
  pg.translate(50, 50);
  pg.fill(240, 180, 180);
  pg.rotate(radians(frameCount));
  pg.rect(-100, -3, 200, 6);
  pg.endDraw();    // 終了

  // 表示画面での処理
  background(255);
  lights();
  translate(width / 2, height / 2, 0);
  rotateX(radians(frameCount) / 8);
  noStroke();
  scale(90);

  beginShape(QUADS);
  texture(pg);  // 隠し画面を画像として使う
  textureMode(IMAGE);
  vertex(-1, 1, 1,   0,   0);
  vertex( 0, 1, 0,  50,   0);
  vertex( 0,-1, 0,  50, 100);
  vertex(-1,-1, 1,   0, 100);
  vertex( 0, 1, 0,  50,   0);
  vertex( 1, 1, 1, 100,   0);
  vertex( 1,-1, 1, 100, 100);
  vertex( 0,-1, 0,  50, 100);
  endShape();
}
```