Graphics with Processing

2022-11 シェーディングとマッピング

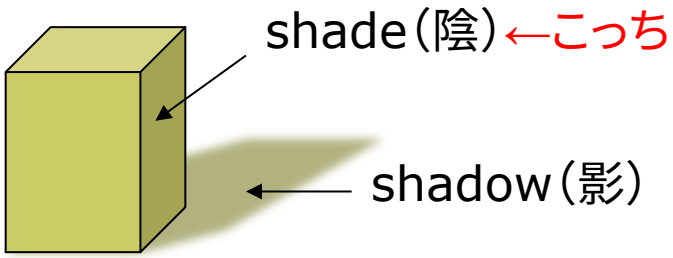
http://vilab.org

塩澤秀和

11.1* シェーディング

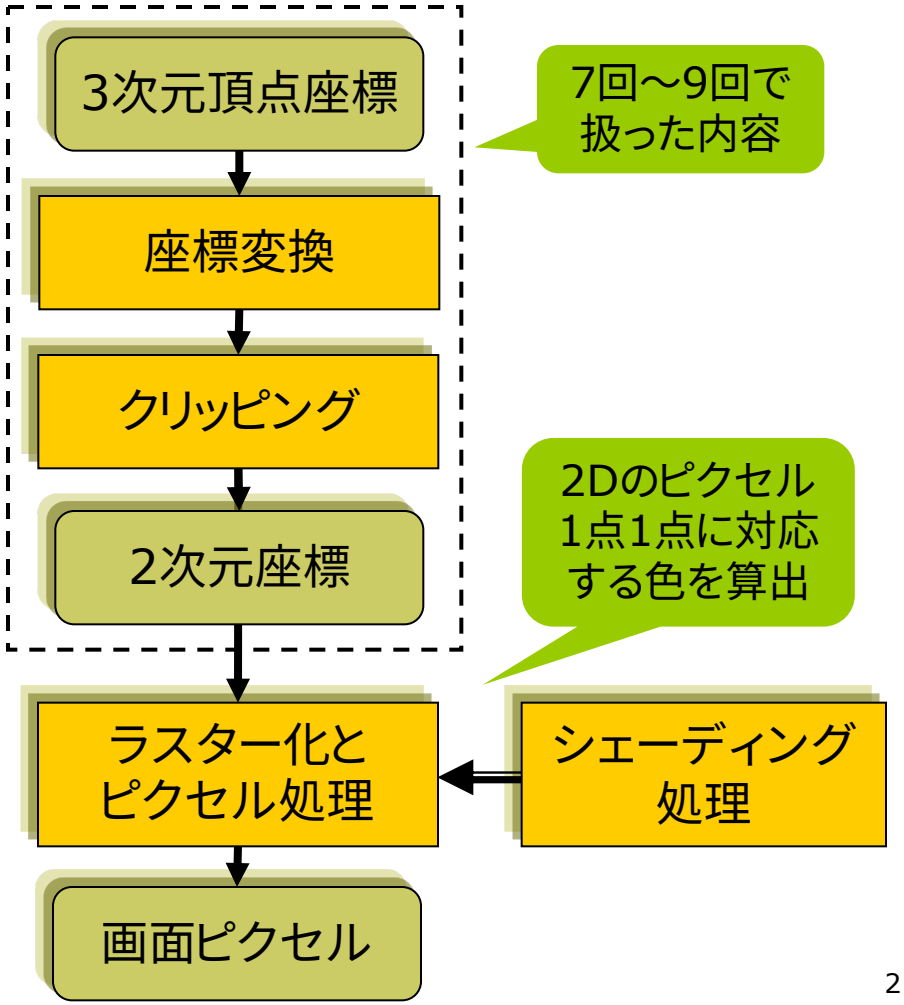
シェーディング(shading)

- シェーディングとは(p.141)
  - レンダリングの一過程
  - 物体の表面に照明による明暗をつけて描画する ⇒ 立体感
  - 2D化後にポリゴンを塗る処理



- 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
  - 皮膚,水面,マンガ風など特別なレンダリング技術が適用可能
  - 最終回で少し触れる予定

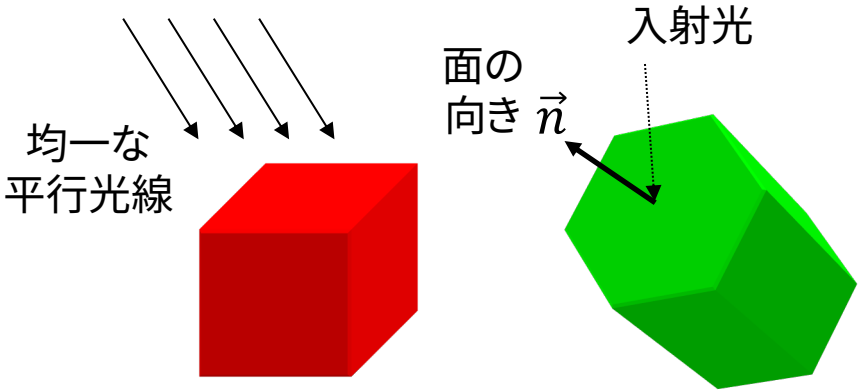
シェーディングは2D化後



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
  - まず，ポリゴンの面の向きを表す法線ベクトルを求める
  - 面上の1点で入射光と法線ベクトルから反射光（色）を計算する
  - 面全体をその色で塗り潰す
  - 均一な平行光線だけが平面に当たる状況は表現
  - 高速だがリアリティには欠ける



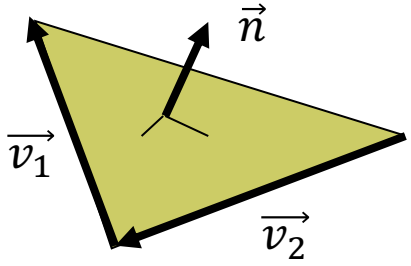
法線ベクトル（p.67）

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）
$\begin{pmatrix} y_1z_2 - z_1y_2 \\ z_1x_2 - x_1z_2 \\ x_1y_2 - y_1x_2 \end{pmatrix}$

単位法線ベクトル

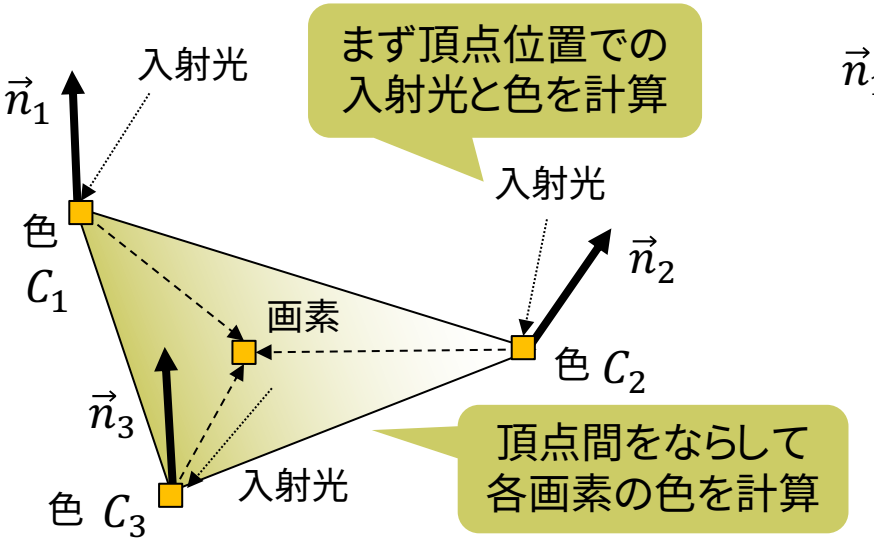
$$\vec{n} = \vec{N} / \left|\vec{N}\right|$$

（大きさを1にする）

11.3* スムーズシェーディング

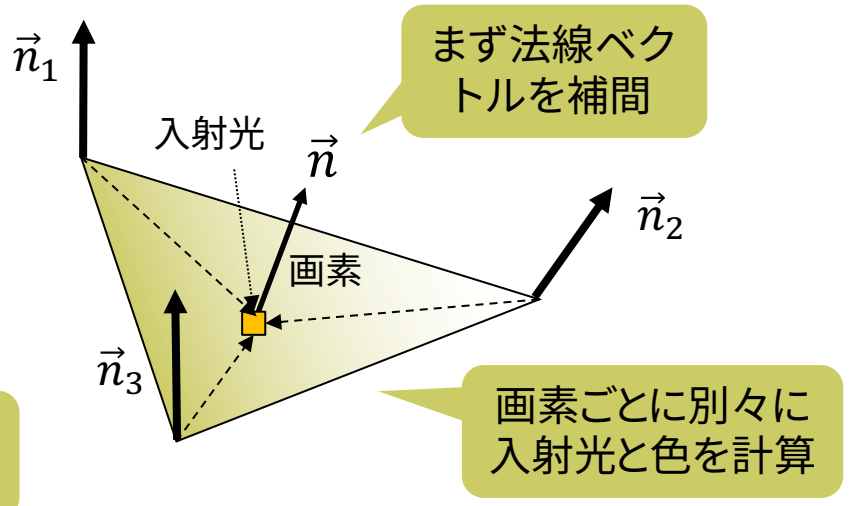
グロー(Gouraud)シェーディング

- 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
  - 高速だが光の計算点が少ない



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
  - 計算点が多く,光沢等がリアル



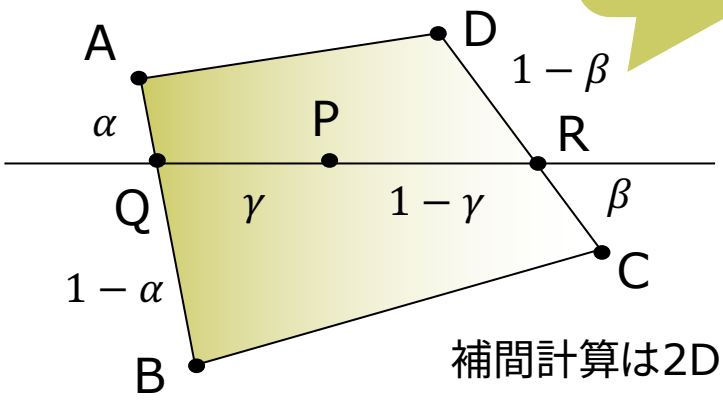
11.4 バイリニア補間

バイリニア補間(p.156)

- 2D上での線形補間

凸ポリゴンを横切る直線は必ず2辺で交わる

画面描画のためにラスター化されたポリゴン



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

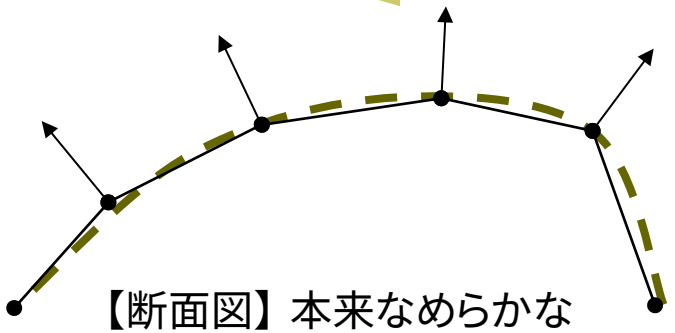
- グローシェーディング
  - 各頂点の色のR,G,B成分をそれぞれバイリニア補間
  - ↓
  - 各ピクセルの補間色を合成
- フォンシェーディング
  - 各頂点の法線ベクトルのx,y,z成分をそれぞれバイリニア補間
  - ↓
  - 各ピクセルの法線ベクトルを合成
- テクスチャマッピング(11.10)ではuv座標をバイリニア補間

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

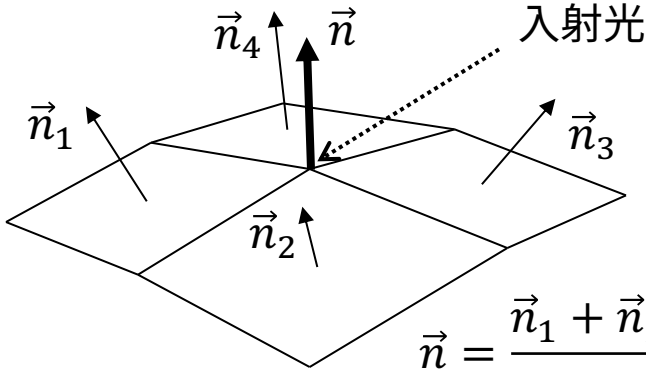
ポリゴン曲面（p.94）

- 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乗る）
  - 各頂点の属性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ズ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



【断面図】本来なめらかな曲面を多数のポリゴンで近似

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



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

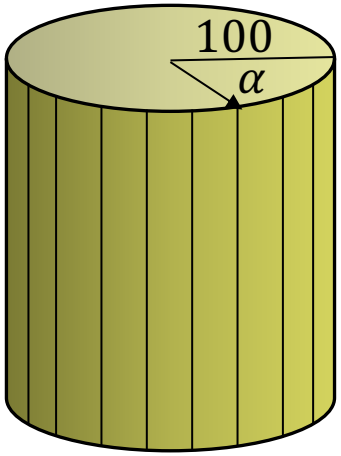
法線ベクトル設定関数

- 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個別に設定する
  - 対応するvertexの直前で使う

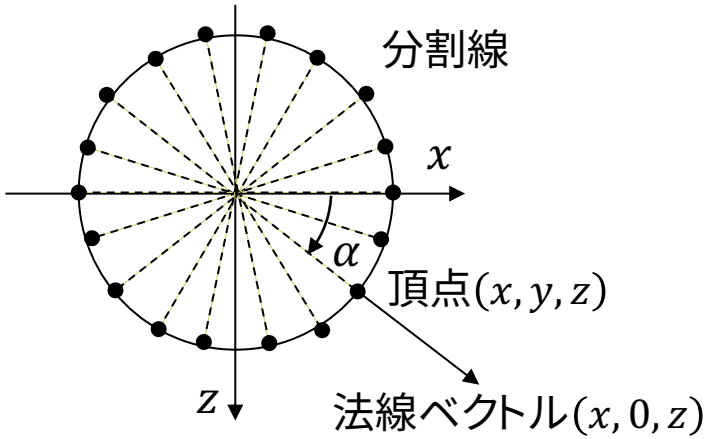
11.6 ポリゴン曲面の例

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



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



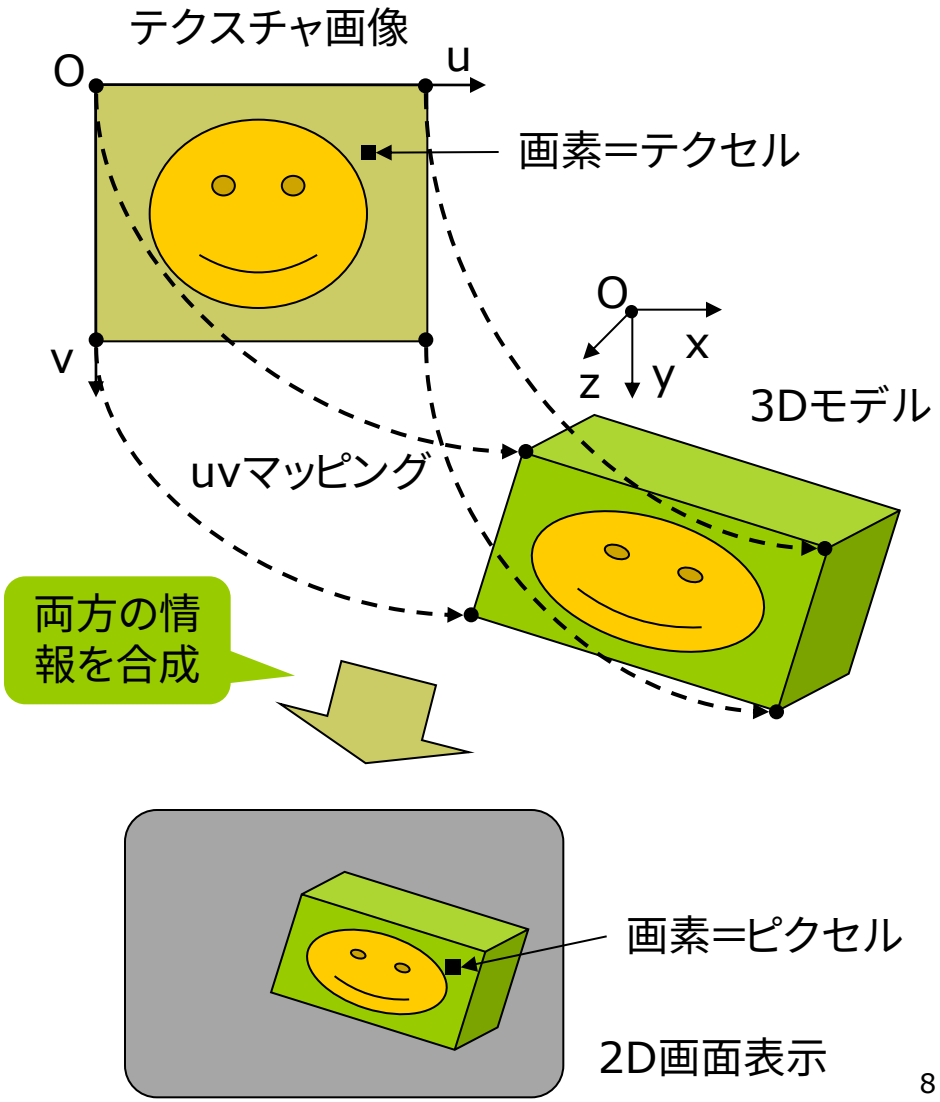

```
noStroke();
beginShape(QUADS);
int N = 18;  // 円周の分割数
for (int i = 0; i < N; i++) {
  float u1 = (float) i / N;
  float a1 = 2 * PI * u1;
  float x1 = 100 * cos(a1);
  float z1 = 100 * sin(a1);
  if (mousePressed) normal(x1, 0, z1);
  vertex(x1, -100, z1);
  vertex(x1,  100, z1);

  float u2 = (float)(i + 1) / N;
  float a2 = 2 * PI * u2;
  float x2 = 100 * cos(a2);
  float z2 = 100 * sin(a2);
  if (mousePressed) normal(x2, 0, z2);
  vertex(x2,  100, z2);
  vertex(x2, -100, z2);
}
endShape();
```

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

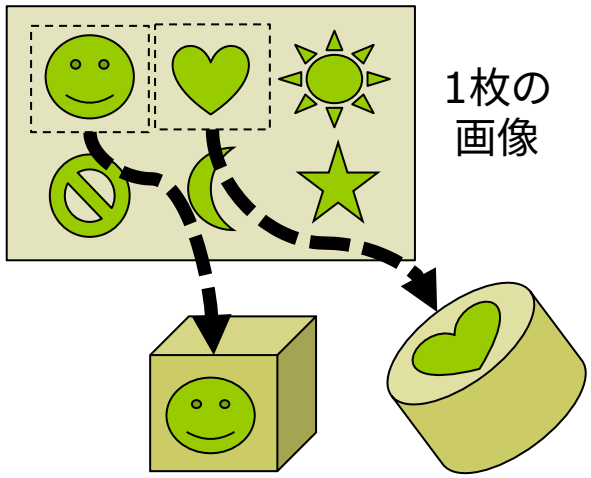
テクスチャマッピング関数

- 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]→[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ポリゴンに貼り付けることもできる

  ⇒ “テクスチャアトラス(地図)”



11.9 テクスチャマッピングの使用例

```
// 準備: 画像ファイル(maskpop.jpg)を
// あらかじめ講義ページからダウンロードし,
// 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(800, 600, P3D);
  frameRate(30);
  tex = loadImage("maskpop.jpg");
}

void draw() {
  background(0);
  perspective();
  directionalLight(255, 255, 255,
                   0, 1, -1);
  translate(width/2, height/2, 0);
  rotateY(-radians(frameCount));

  noStroke(); fill(255);
  beginShape(QUADS);
  texture(tex);
  textureMode(NORMAL);

  vertex( 100, -150,   0,  1.0, 0.5);
  vertex(-100, -150,   0,  0.0, 0.5);
  vertex(-100,  150, -50,  0.0, 0.0);
  vertex( 100,  150, -50,  1.0, 0.0);
  vertex(-100, -150,   0,   0.0, 0.5);
  vertex( 100, -150,   0,   1.0, 0.5);
  vertex( 100,  150,  50,  1.0, 1.0);
  vertex(-100,  150,  50,  0.0, 1.0);
  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

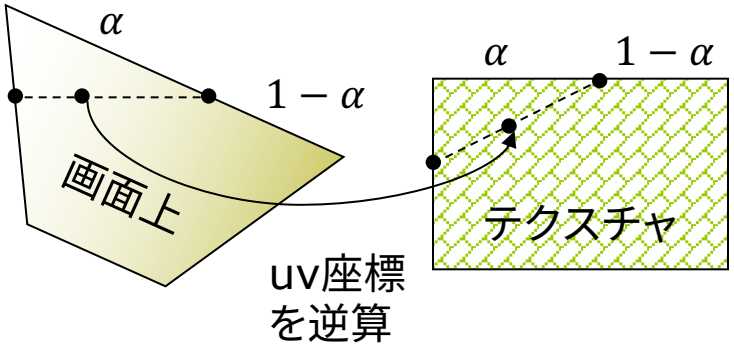
テクスチャを指定し，uv座標は0〜1モード

0〜1で数値を色々変更してみよ

11.10 テクスチャの描画処理

テクスチャの描画処理(p.148)

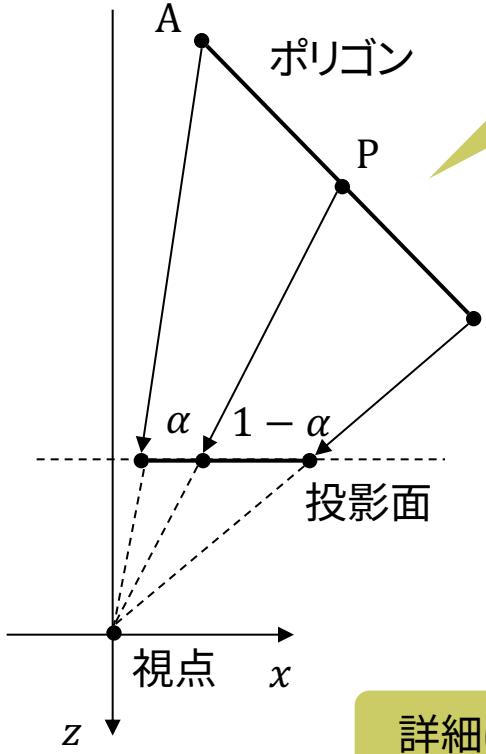
- 段階1:画面座標→uv座標
  - 画面上の各ピクセルの描画時に画像のuv座標を逆算する
  - 各頂点が持つuv座標の値からバイリニア補間で求める



- 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
  - 視点からの距離(z)に反比例するようにテクスチャを縮めて貼る



テクスチャの模様も透視投影が必要

テクスチャ座標を $T = (u, v)$ とすると

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

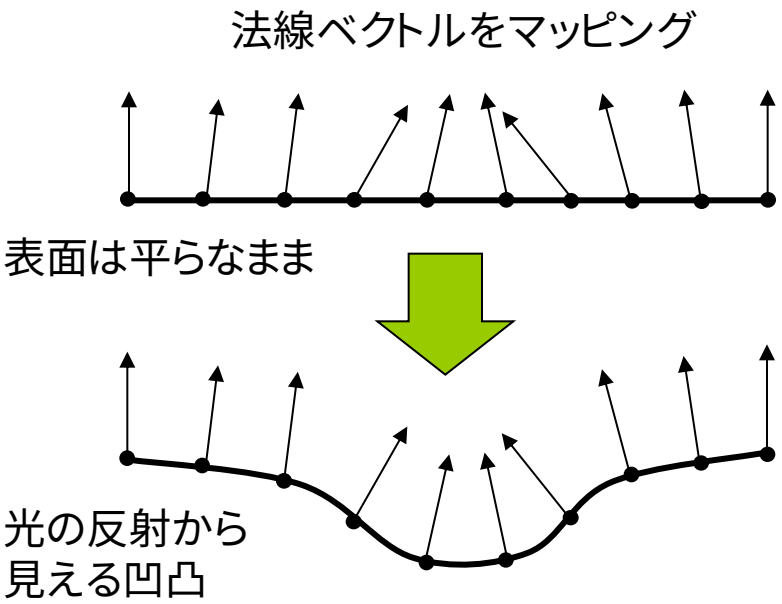
$$\frac{T_P}{z_P} = (1-\alpha)\frac{T_A}{z_A} + \alpha\frac{T_B}{z_B}$$

詳細は書籍などを参照

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の例をもとに，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
  - textureModeはNORMALとし，全てのvertexに画像内の位置を表すuv座標を追加する
  - 回転などによって全表面が確認できるようにすること
  - PNG形式などで透過色がある画像を貼るのも面白い
- 提出方法
  - 画像も含めたフォルダをZIPファイルにまとめて提出すること
  - [ツール]→[スケッチをアーカイブ]