

Graphics with Processing



2023-11 シェーディングとマッピング

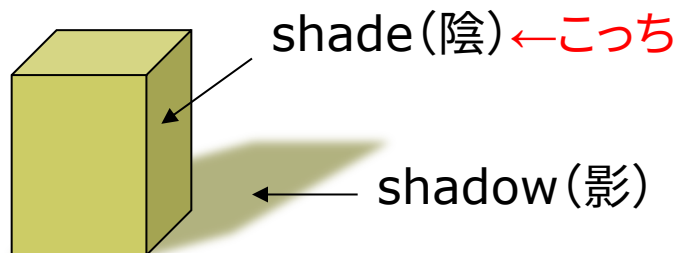
<http://vilab.org>

塩澤秀和

11.1* シェーディング

シェーディング (shading)

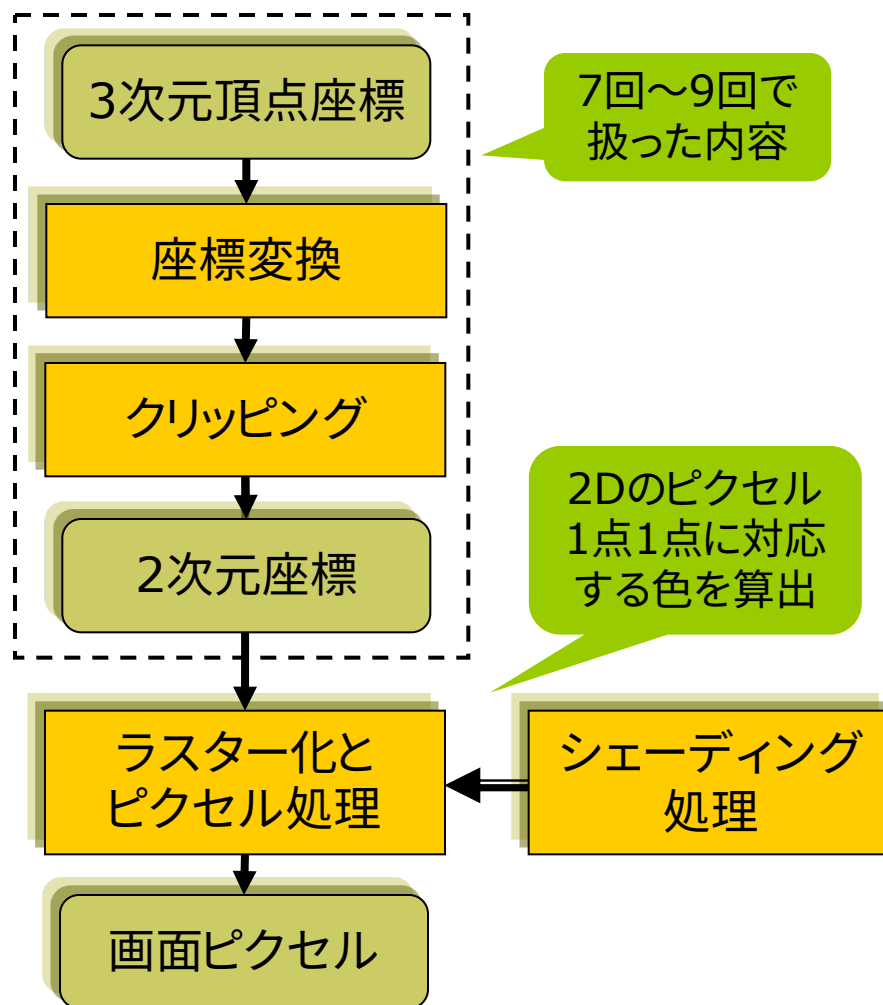
- シェーディングとは (p.141)
 - レンダリングの一過程
 - ラスター化したポリゴンの描画
 - 照明による面の明暗を計算して描画する ⇒ 立体感



□ 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

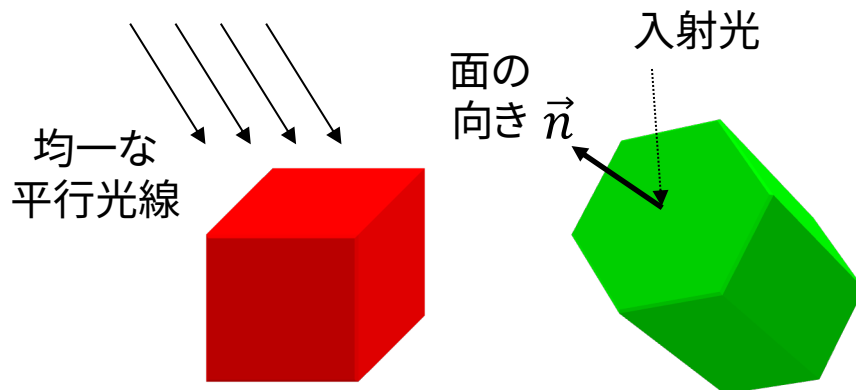
シェーディングは2D化後



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塗る
 - 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なら正しい
 - 高速だがリアリティには欠ける

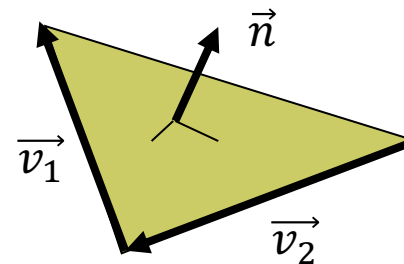


法線ベクトル (p.67)

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

$$\vec{N} = \vec{v}_1 \times \vec{v}_2$$

単位法線ベクトル

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

(大きさを1にする)

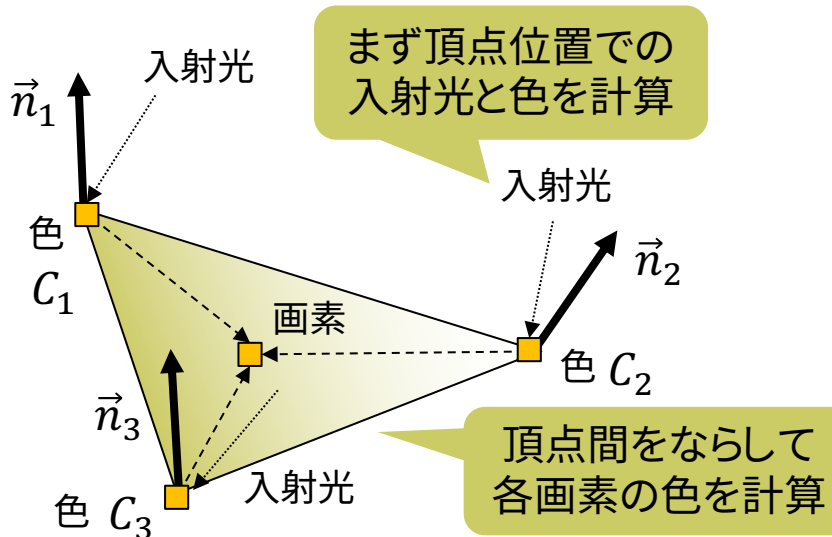
外積(クロス積)

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

11.3* スムーズシェーディング

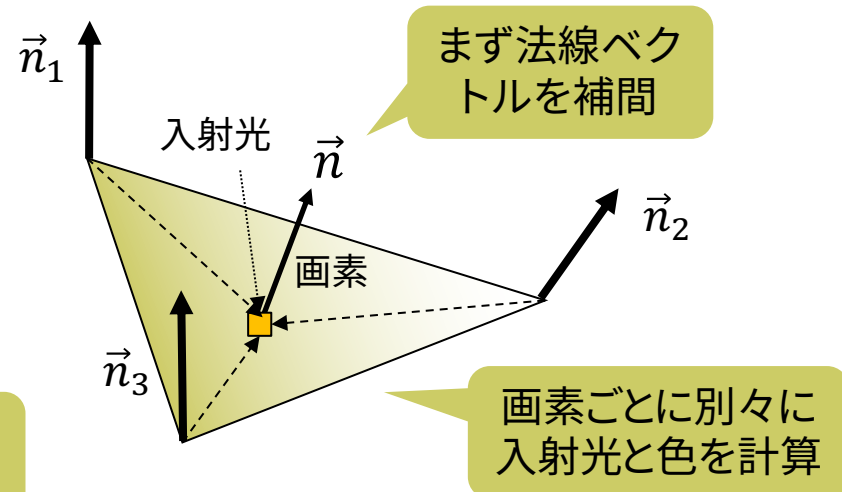
グロー (Gouraud) シェーディング

- 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

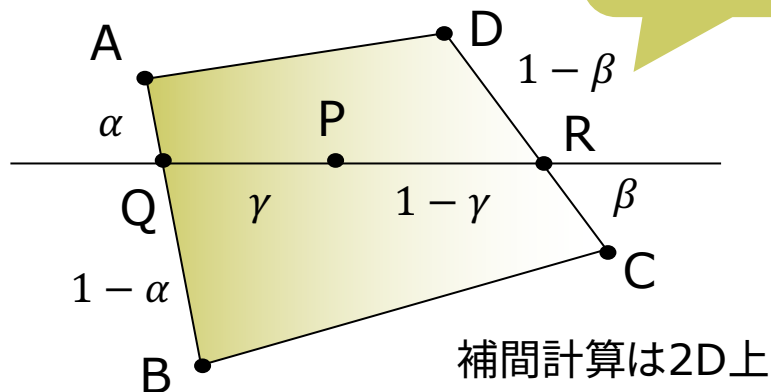


11.4 バイリニア補間

バイリニア補間 (p.156)

□ 2D上での線形補間

画面描画のために
ラスタライズされたポリゴン



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
から**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
成分から**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(後述)では、uv座標を**バイリニア補間**

11.5* ポリゴン曲面

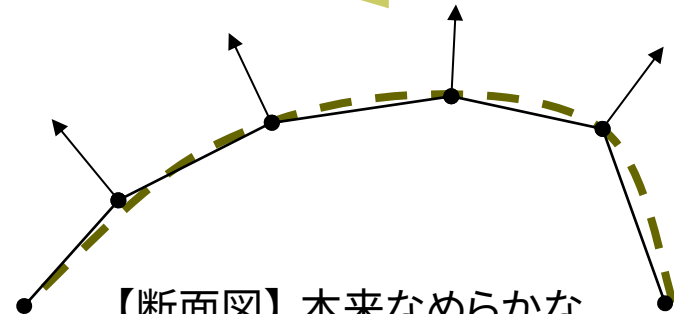
ポリゴン曲面 (p.94)

- 曲面をポリゴンの集合で近似
 - ポリゴンメッシュともいう
 - 三角形を使うことが多い(頂点が必ず同一平面上に乗るから)
 - 各頂点には、元の曲面から計算した法線ベクトルを持たせる
 - スムーズシェーディングで描画すると、ポリゴンの色が滑らかに連続し、曲面として見える

法線ベクトル設定関数

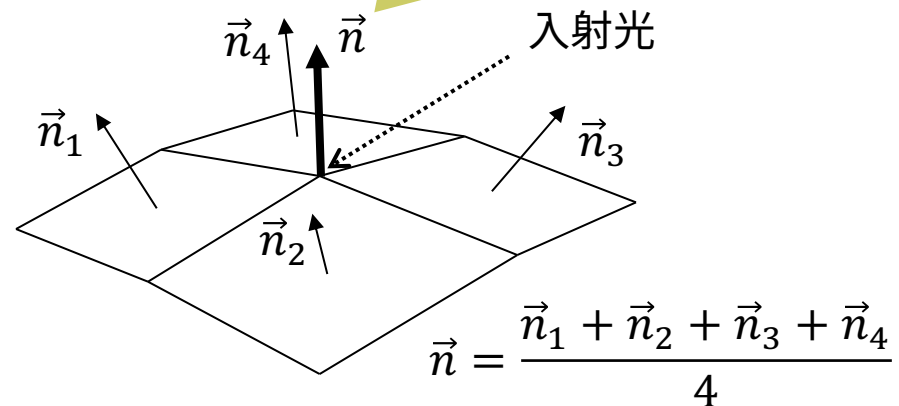
- `normal(nx, ny, nz)`
 - 頂点にシェーディングのための法線ベクトルを個別に設定する
 - 対応するvertexの直前で使う

ポリゴンモデルでは、各頂点に法線ベクトルを設定する



【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

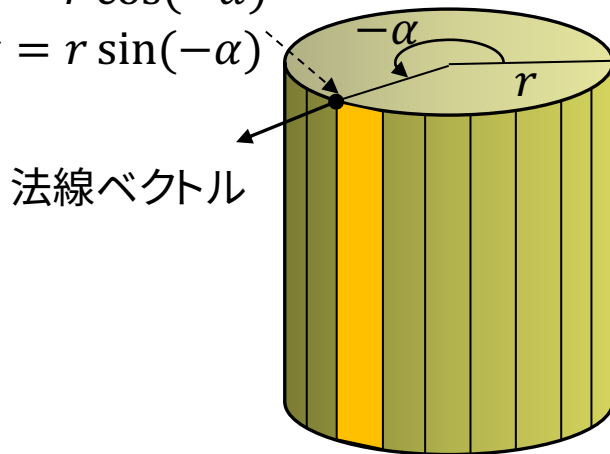


11.6 ポリゴン曲面の例

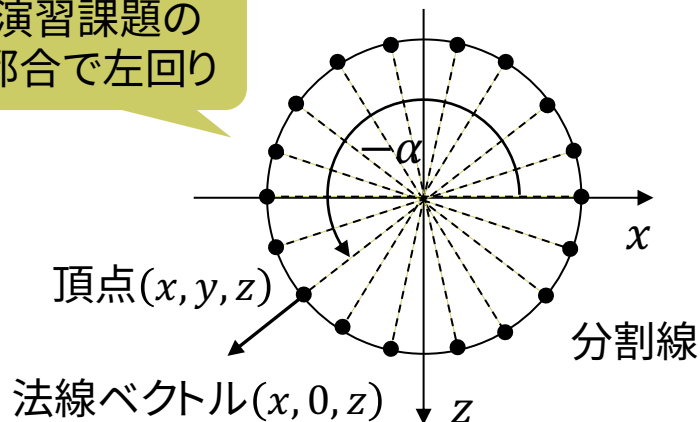
実は、QUAD_STRIPで描けば Processingが滑らかに見える 法線ベクトルを設定してくれる

円柱のモデリング

$$\begin{cases} x = r \cos(-\alpha) \\ z = r \sin(-\alpha) \end{cases}$$



演習課題の
都合で左回り



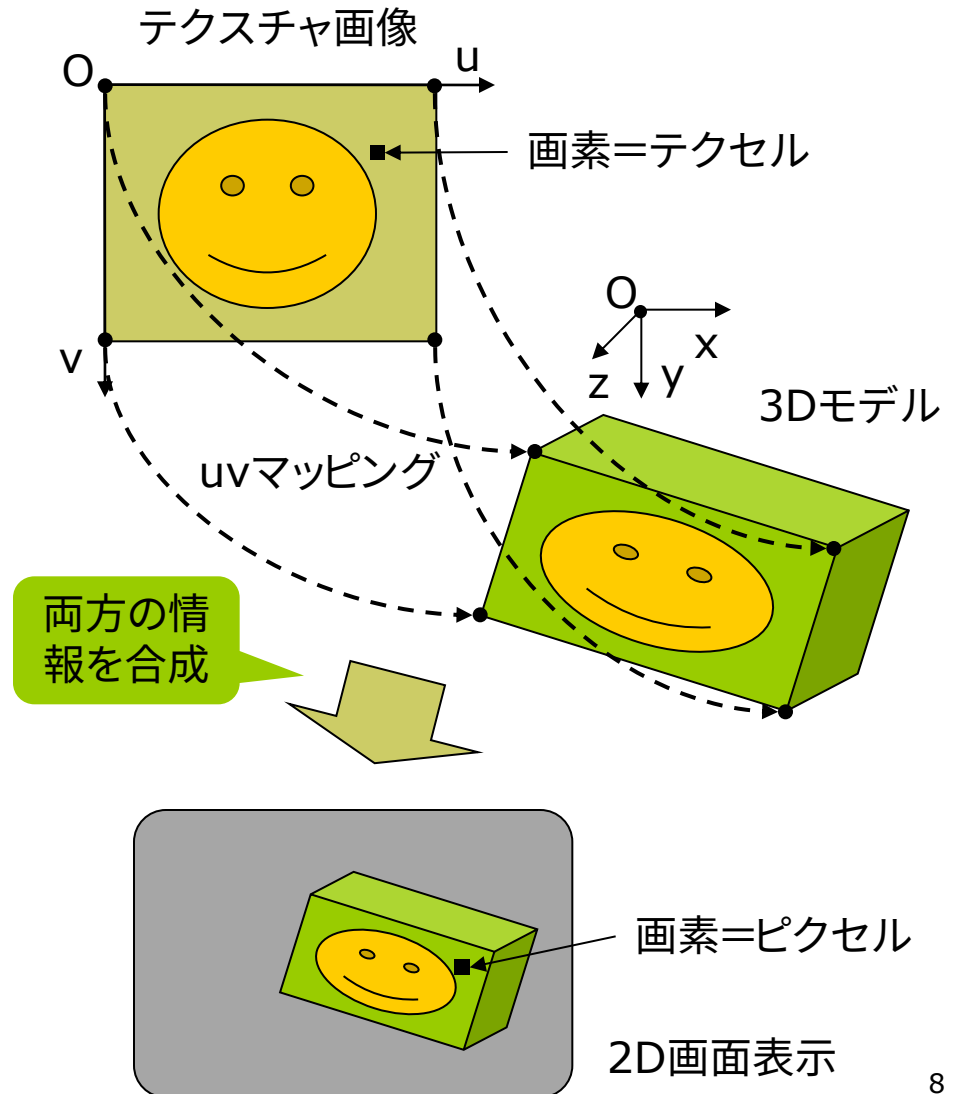
```
noStroke();
beginShape(QUADS);
int r = 200, N = 18; // 円周の分割数
for (int i = 0; i < N; i++) {
  float u1 = (float) i / N;
  float a1 = 2 * PI * u1;
  float x1 = r * cos(-a1);
  float z1 = r * sin(-a1);
  if (mousePressed) normal(x1, 0, z1);
  vertex(x1, -r, z1);
  vertex(x1, r, z1);

  float u2 = (float)(i + 1) / N;
  float a2 = 2 * PI * u2;
  float x2 = r * cos(-a2);
  float z2 = r * sin(-a2);
  if (mousePressed) normal(x2, 0, z2);
  vertex(x2, r, z2);
  vertex(x2, -r, 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
 - 質感を表すのに絶大な効果
 - 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 $(u, v) \rightarrow$ 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)

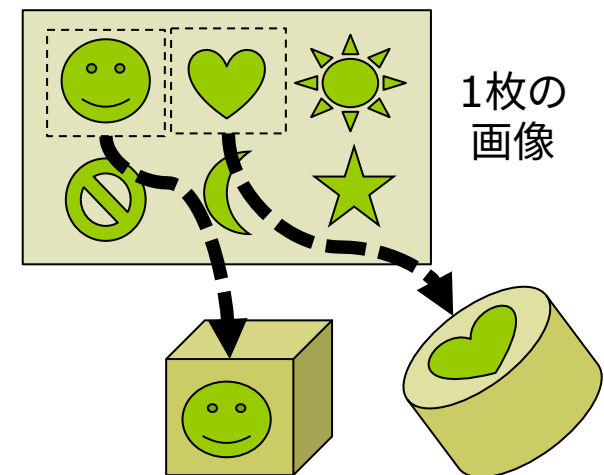
□ textureMode(モード)

- 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)



別々のポリゴンやオブジェクト

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;
```

画像はグローバル変数で定義する

```
void setup() {
  size(800, 600, P3D);
  frameRate(30);
  tex = loadImage("maskpop.jpg");
}
```

画像はsetupの中で一度だけ読み込む

```
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);
```

テクスチャを指定し、uv座標は0~1モード

```
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();
```

```
}
```

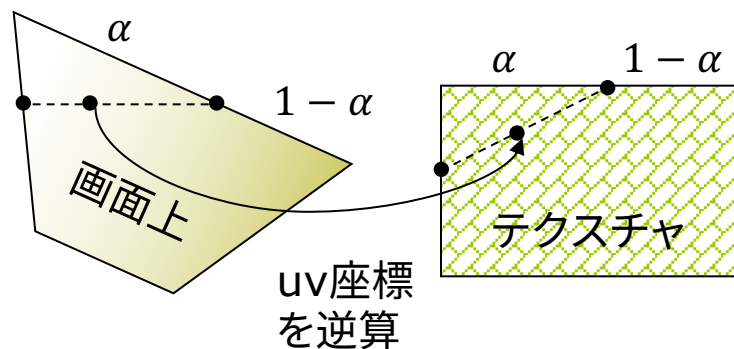
0~1で数値を色々変更してみよ

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



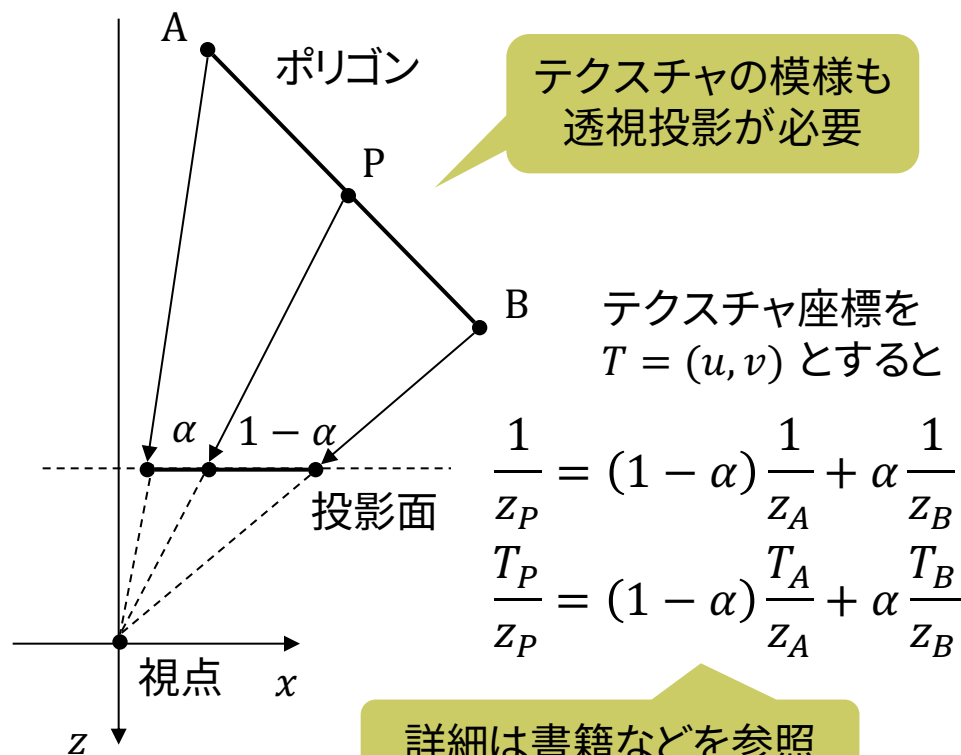
□ 段階2:uv座標→描画色

- 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



11.11* その他のマッピング技術

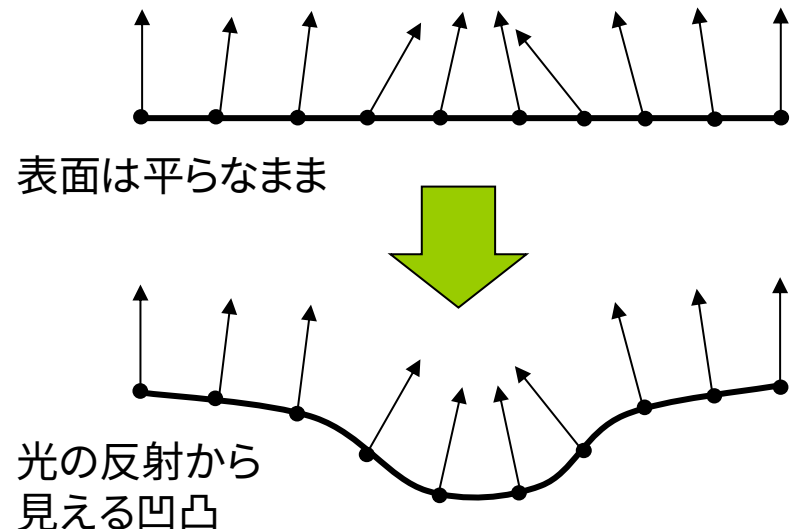
テクスチャマッピングの発展

- 投影マッピング (p.164)
 - プロジェクタで投影するように、面にテクスチャを貼り付ける
 - 影の表現に応用できる
- 環境マッピング (p.168)
 - 金属などへの周囲の景色の映り込みをテクスチャで表現する
 - シーンをレンダリングして環境マップ(一種の背景画像)を作り、それを変形して面に貼りつける
- 視差マッピング
 - 表面の凹凸による模様のをずれを2D画像の変形だけで表現する
 - 凸部分は奥に、凹部分は手前に見えるように、uv座標をずらしてテクスチャを貼る

色や模様以外のマッピング

- 法線(ノーマル)マッピング
 - 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
 - バンプマッピング (p.166) では、高低の分布(ハイトマップ)から法線を計算して貼りつける

法線ベクトルをマッピング



11.12 演習課題

課題

- 11.6の例を参考にして、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]

