ゲームグラフィックス特論

第9回 テクスチャマッピング (2)

テクスチャによるマスク

拡散反射係数以外の要素を制御する

材質マッピング (Material Mapping)

- 照明方程式

$$I_{diff} = \max(\mathbf{N} \cdot \mathbf{L}, 0)\, K_{diff} \otimes L_{diff}$$
$$I_{spec} = \max(\mathbf{N} \cdot \mathbf{H}, 0)^{K_{shi}}\, K_{spec} \otimes L_{spec}$$
$$I_{amb} = K_{amb} \otimes L_{amb}$$
$$\downarrow$$
$$I_{tot} = I_{amb} + I_{diff} + I_{spec}$$

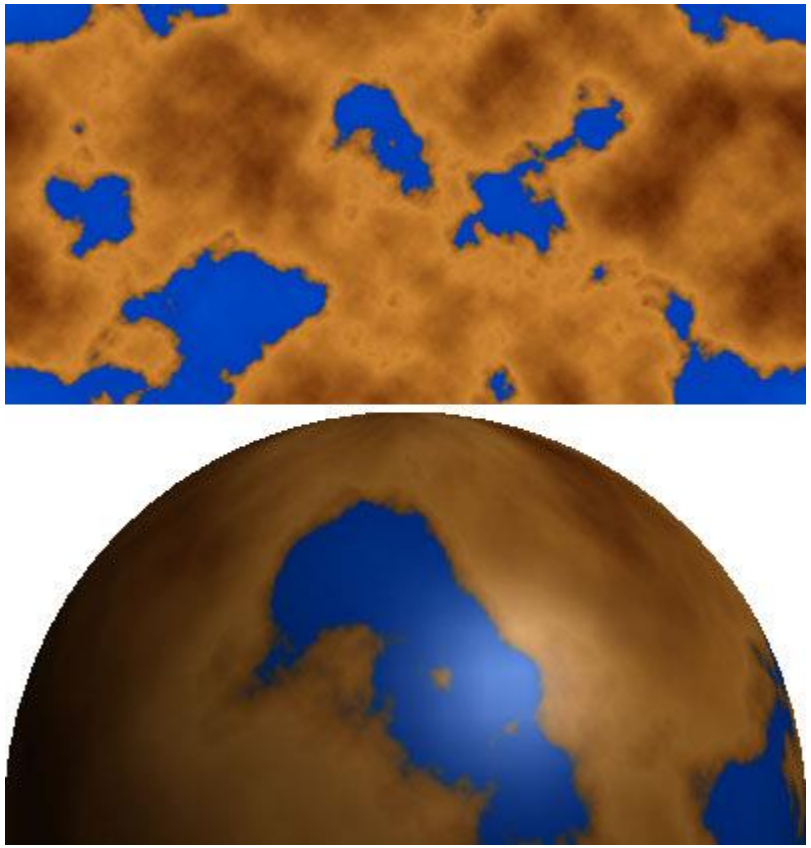
- K_{diff} と K_{amb} をテクスチャで制御
  - 通常のテクスチャマッピング
    - diffuse color map
- K_{spec} をテクスチャで制御
  - ハイライトマッピング
    - specular color map
    - 一般的にグレースケール画像を用いる
    - 金属ならカラー
- K_{shi} をテクスチャで制御
  - 輝きマッピング (gross map)

Specular Color Map の適用

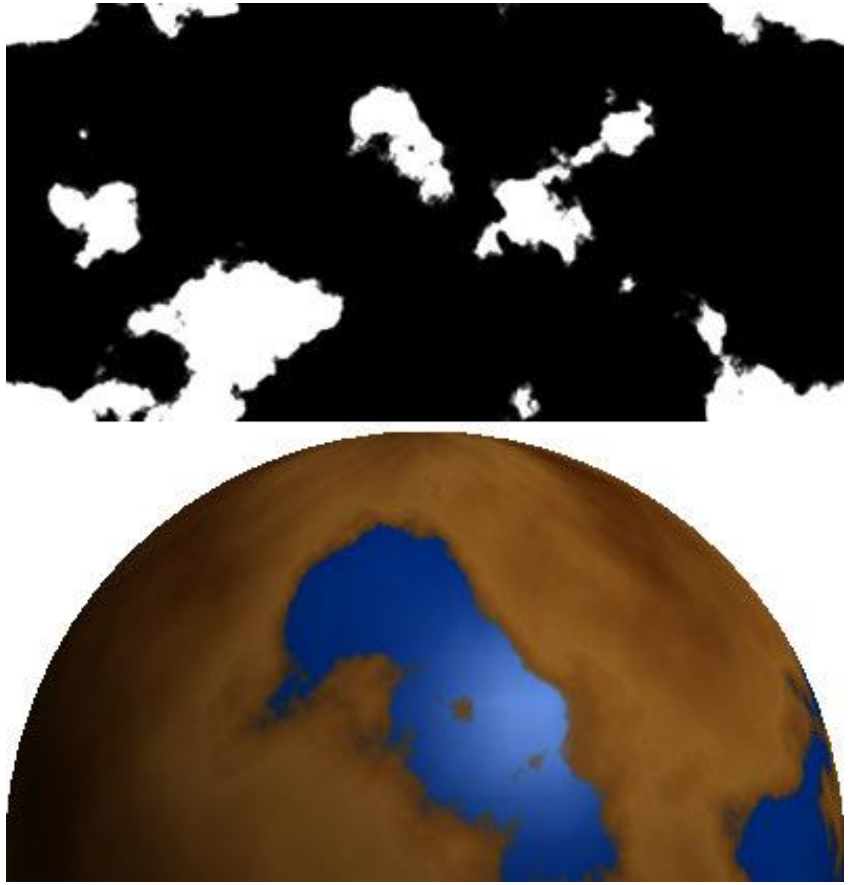
```
#version 410

in vec4 dc;                 // 拡散反射光強度
in vec4 sc;                 // 鏡面反射光強度
in vec2 tc;                 // テクスチャ座標

uniform sampler2D dmap;     // diffuse color map
uniform sampler2D smap;     // specular color map

out vec4 fc;                // カラーバッファへの出力

void main(void)
{
  fc = texture(dmap, tc) * dc + texture(smap, tc) * sc;
}
```

Specular Color Map の適用結果

Specular Color Map なし



Specular Color Map あり



不透明度マッピング (Alpha Mapping)

- テクスチャ画像のアルファ値の利用
  - Decal（デカール）
  - Cutout（切り抜き）
- 実装が容易
  - レンダリングパイプラインに対するわずかな拡張で実現可能
  - デプスバッファとの比較の前に実行する
- アルファ合成やテクスチャアニメーションとの組み合わせ
  - 炎の揺らぎ，植物の成長，爆発，大気の効果

ここではシェーダを使って
実装してみる

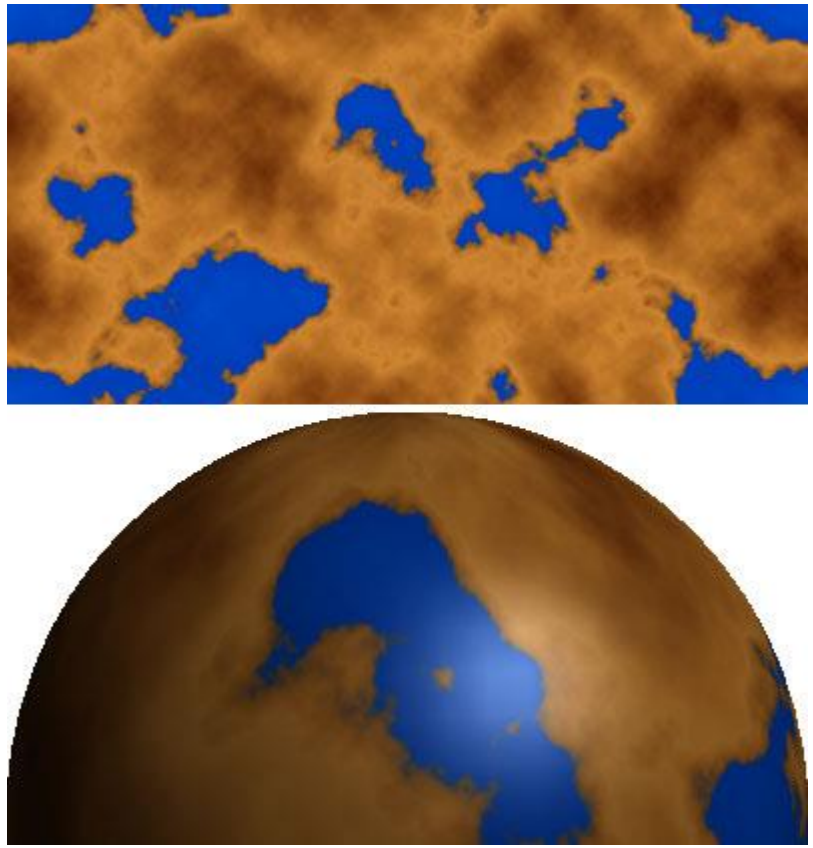
アルファ値を使って Decal

```
#version 150 core

in vec4 dc;                 // 拡散反射光強度（下地のポリゴンの色）
in vec2 tc;                 // テクスチャ座標

uniform sampler2D dmap;     // diffuse color map

out vec4 fc;                // カラーバッファへの出力

void main(void)
{
  vec4 color = texture(dmap, tc);
  fc = mix(vec4(color.rgb, 1.0), dc, color.a);
}
```

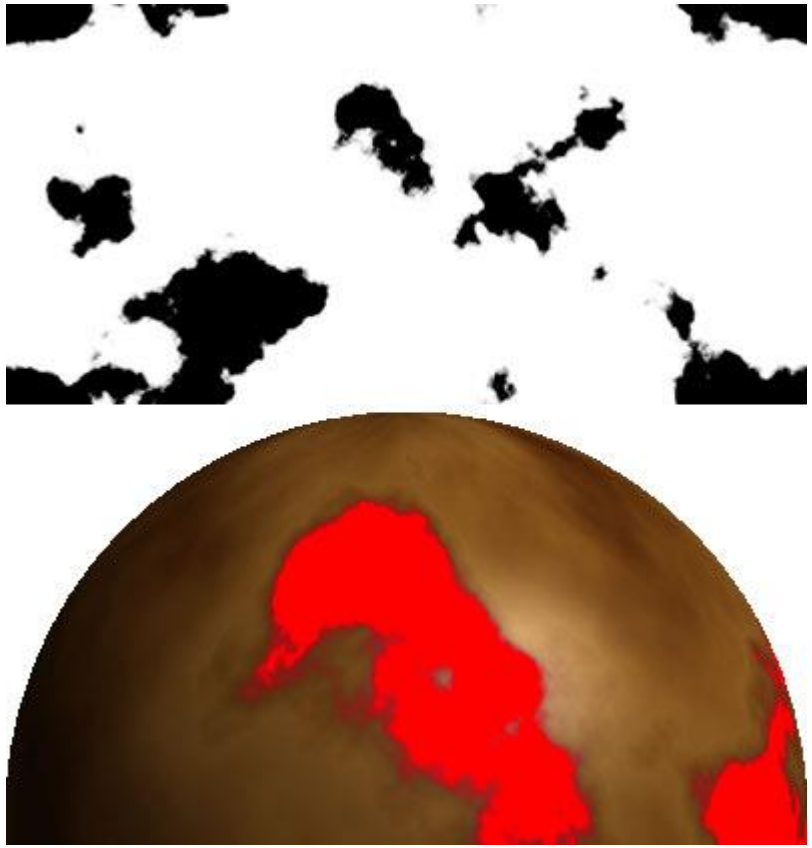
テクスチャの色と下地のポリゴンの色をアルファ値を使ってブレンドする

Decal

Decal なし



Decal あり



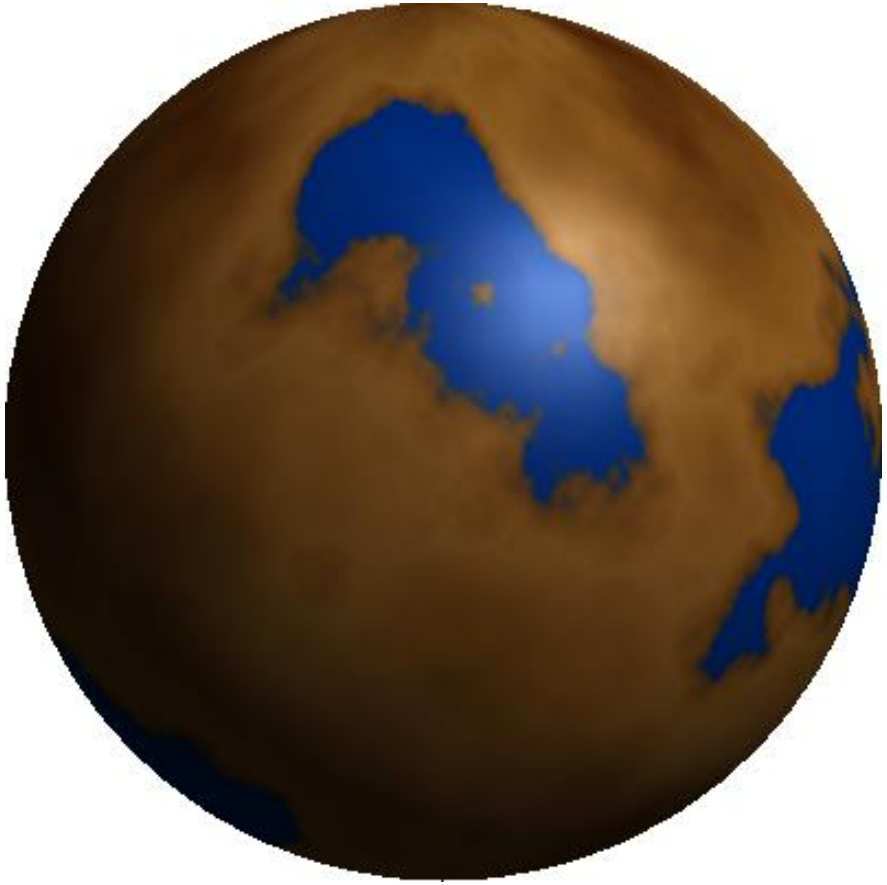
アルファ値を使って cutout

```
#version 150 core

const float threshold = 0.5;     // しきい値
in vec4 dc;                      // 拡散反射光強度（ポリゴンの色）
in vec2 tc;                      // テクスチャ座標

uniform sampler2D dmap;          // diffuse color map

out vec4 fc;                     // カラーバッファへの出力

void main(void)
{
  vec4 color = texture(dmap, tc);
  if (color.a < threshold) discard;
  fc = vec4(color.rgb, 1.0) *  dc;
}
```

discard はフラグメントシェーダの処理を打ち切り，そのフラグメントを捨てる（描画しない）

Cutout

Cutout なし



Cutout あり

Alpha Mapping の応用

画素単位の陰影付け

フラグメントシェーダで陰影を計算する

画素単位の陰影付け

- 陰影付けを**フラグメントシェーダ**で行う
  - Phong のスムーズシェーディング
    - 法線を補間するので画素ごとに陰影計算を行う必要がある
  - バーテックスシェーダは位置と法線ベクトルを out 変数に格納する
- 陰影付け方程式のさまざまな係数のテクスチャによる制御
  - 頂点における計算値の線形補間では対応しきれないものがある
    - 輝きマッピングは輝き係数 m が指数なので $(\mathbf{N} \cdot \mathbf{H})^m$ を画素ごとに計算する
- バンプマッピング（後述）ではテクスチャで法線を変化する
  - 画素ごとに変化後の法線を用いて陰影を計算する

バーテックスシェーダの変更

```
#version 410
…
out vec3 l;                    // フラグメントシェーダに送る光線ベクトル
out vec3 n;                    // フラグメントシェーダに送る頂点法線ベクトル
out vec3 h;                    // フラグメントシェーダに送る中間ベクトル
out vec2 tc;                   // フラグメントシェーダに送るテクスチャ座標
…
void main(void)
{
  …
  vec4 p = mw * pv;                          // 視点座標系の頂点位置
  vec3 v = normalize(p.xyz);                 // 視線ベクトル
  l = normalize(vec3(4.0, 5.0, 6.0));        // 光線ベクトル
  n = normalize((mg * nv).xyz);              // 頂点の法線ベクトル
  h = normalize(l + v);                      // 頂点の法線ベクトル
  tc = tv;                                   // テクスチャ座標
  gl_Position = mc * pv;
}
```

これは光線ベクトル方向を定数で与えている

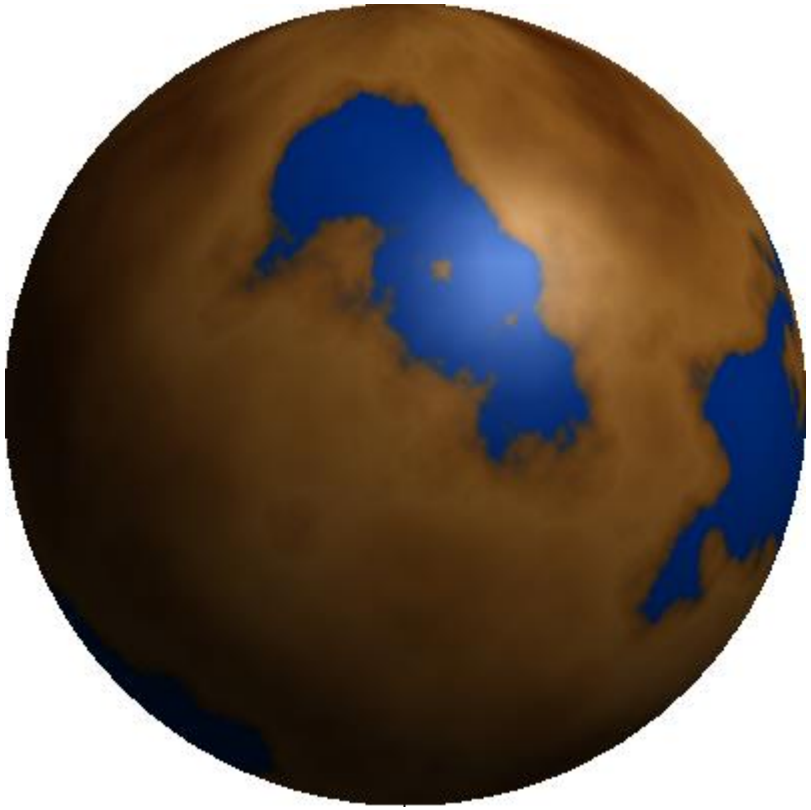
フラグメントシェーダの変更

```
#version 410
…
in vec3 l;                  // 補間された光線ベクトル
in vec3 n;                  // 補間された法線ベクトル
in vec3 h;                  // 補間された中間ベクトル
in vec2 tc;                 // 補間されたテクスチャ座標
uniform sampler2D dmap;     // diffuse color map
uniform sampler2D smap;     // specular color map
…
void main(void)
{
    vec3 nn = normalize(n); // 法線ベクトルの正規化
    vec4 iamp = kamb * lamb;
    vec4 idiff = max(dot(nn, l), 0) * kdiff * ldiff;
    vec4 ispec = pow(max(dot(nn, h), 0), kshi) * kspec * lspec;
    fc = texture(dmap, tc) * (iamb + idiff)
      + texture(smap, tc) * ispec;
}
```

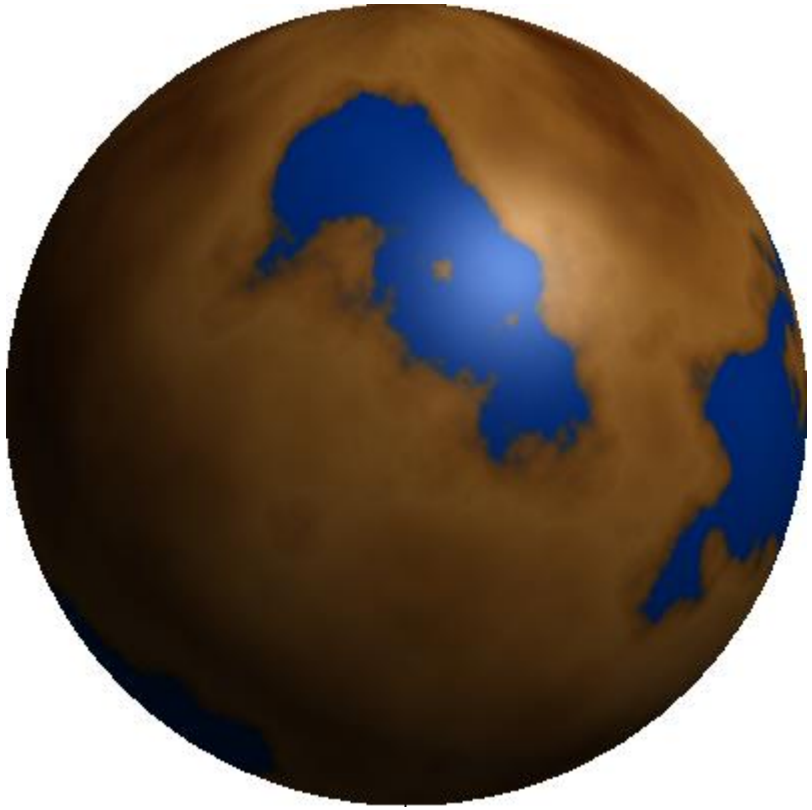
l, h も normalize() を使って正規化するべき

頂点単位と画素単位の陰影付け

頂点単位の陰影付け (Gouraud)



画素単位の陰影付け (Phong)

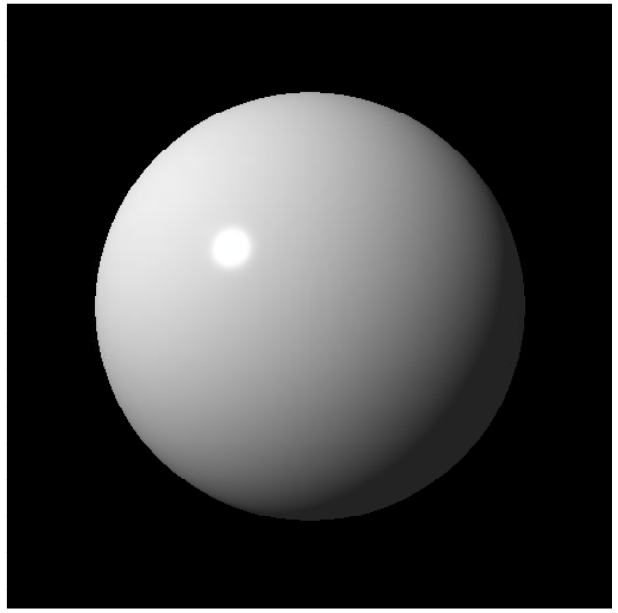


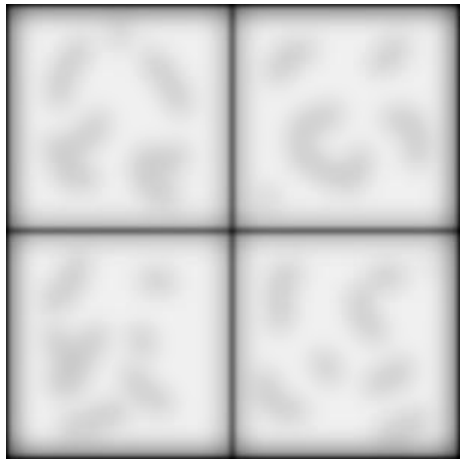
バンプマッピング

形のディティールをテクスチャで制御する

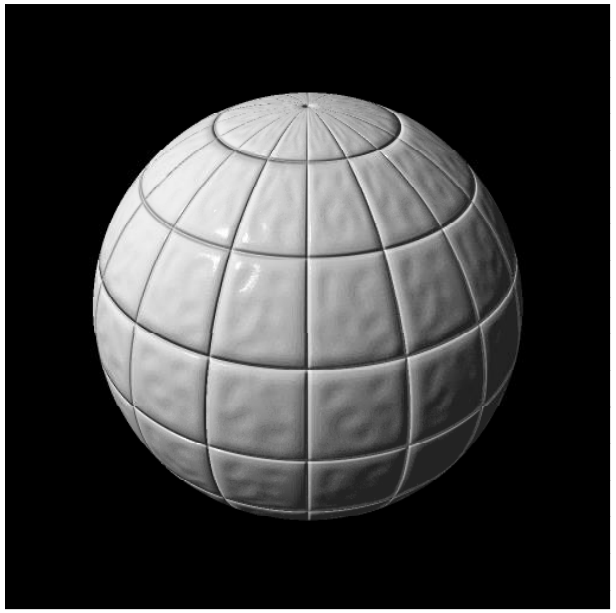
バンプマッピング (Bump Mapping)

- 面の法線ベクトルをテクスチャで制御する
  - 法線ベクトルの変化に伴って陰影が変わるために，物体表面に凹凸が付いたように見える
  - 実際の形状は変化しない



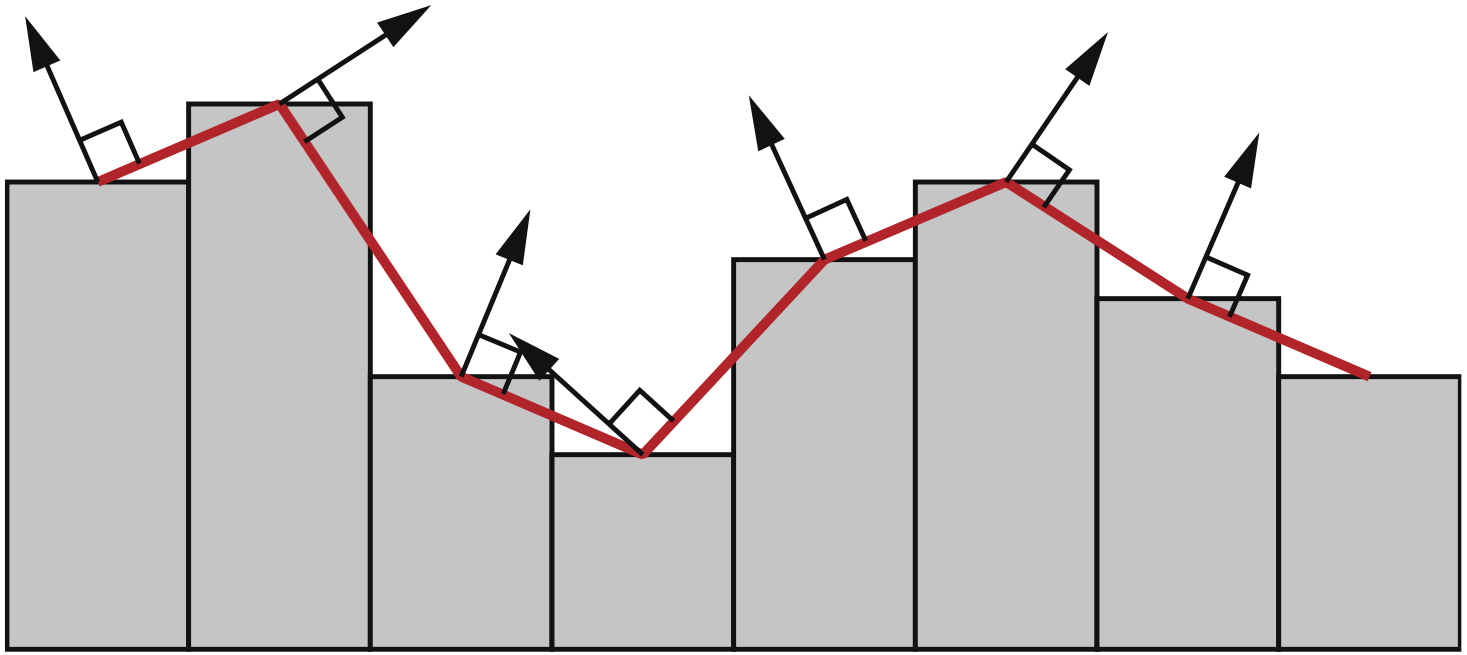


高さマップ



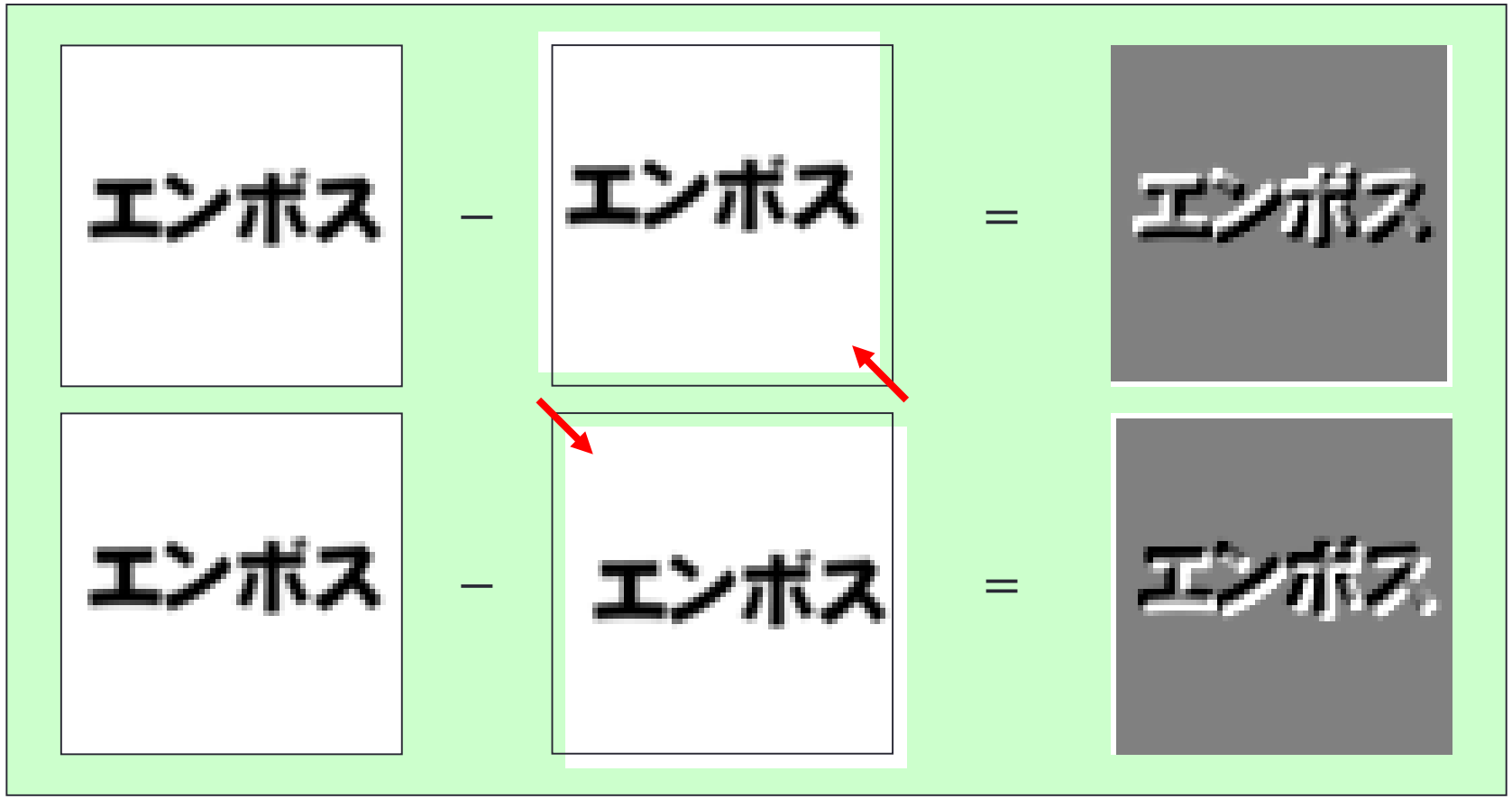
高さマップ (Height Field)

- 高さを濃淡で表したモノクロ画像を用いる
- 近傍の画素の差を求めて面の勾配として使う
- テクスチャは多少ぼかしておいたほうがいい



Emboss Bump Mapping

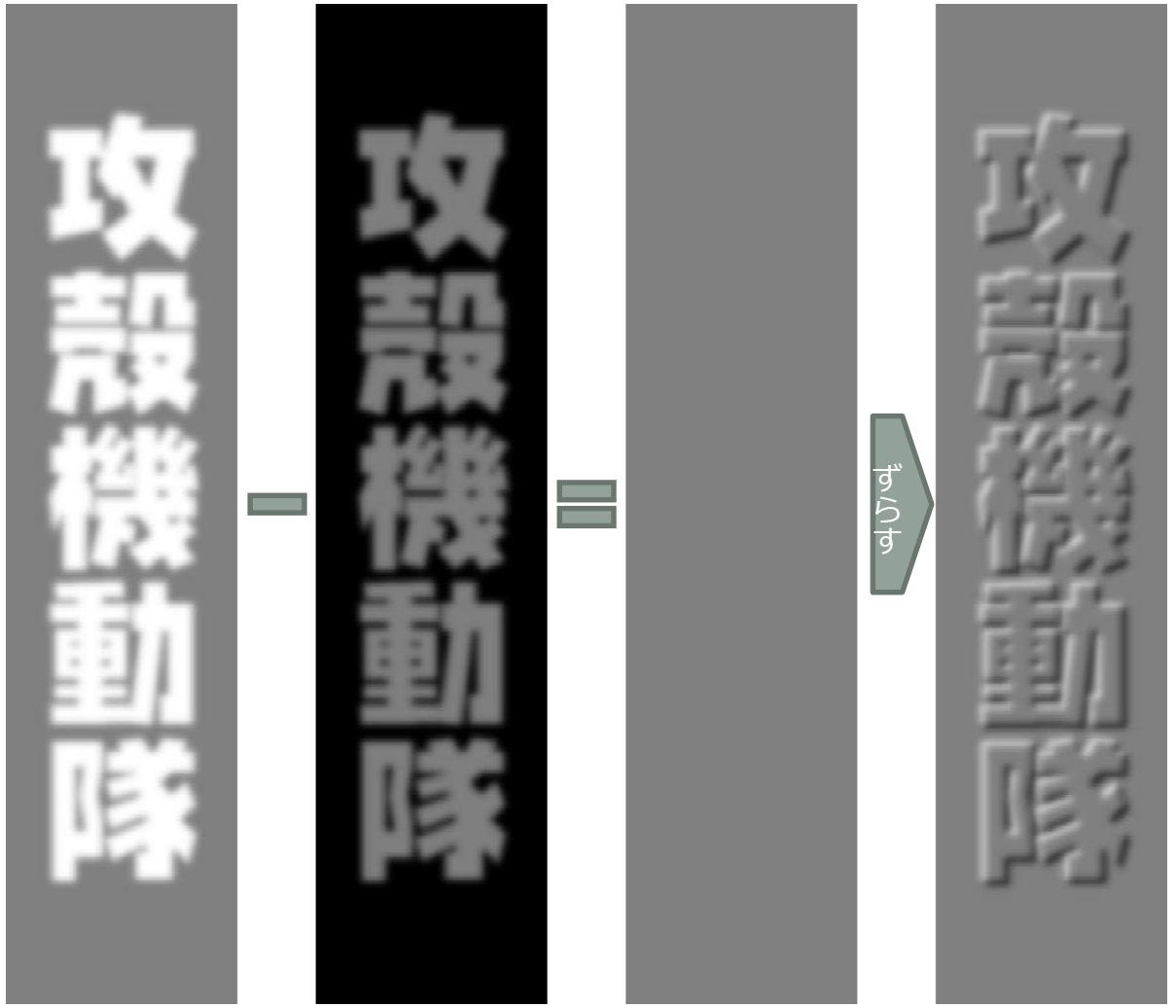
- ２次元画像処理のエンボス効果を使う方法



エンボス効果

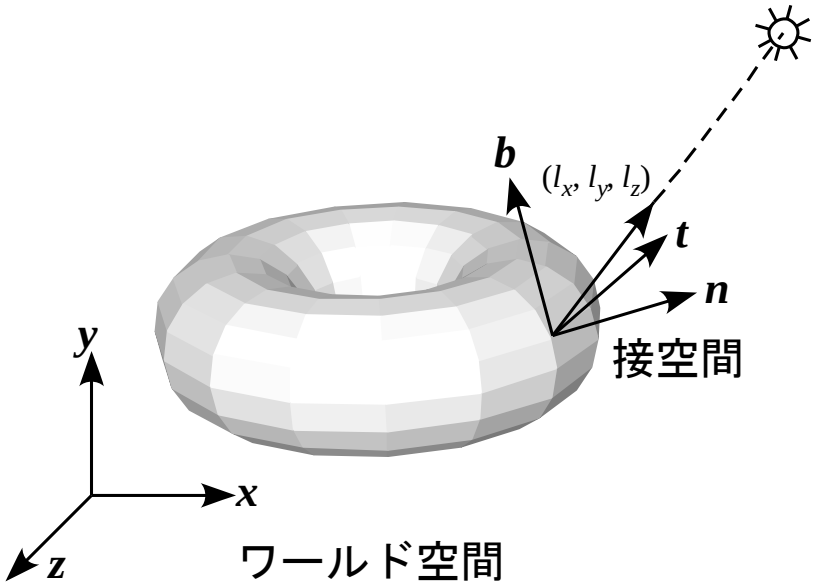
Emboss Bump Mapping の手順

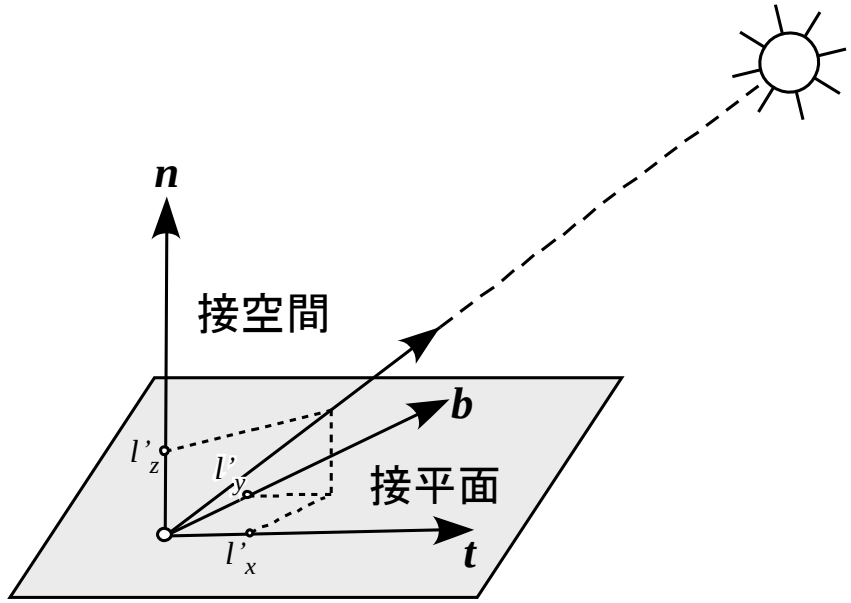
- 高さマップに用いるようなモノクロのテクスチャを貼り付けて面をレンダリングする
  - テクスチャの (u, v) 座標を光源の方向に従って少しずらす
  - ずらしたテクスチャを貼り付けて面をレンダリングし，それを最初のレンダリング結果から引く
- この面をテクスチャを貼らずにレンダリングして陰影を求め，先のレンダリング結果に加えて陰影を付ける



(u, v) 座標のずらし方

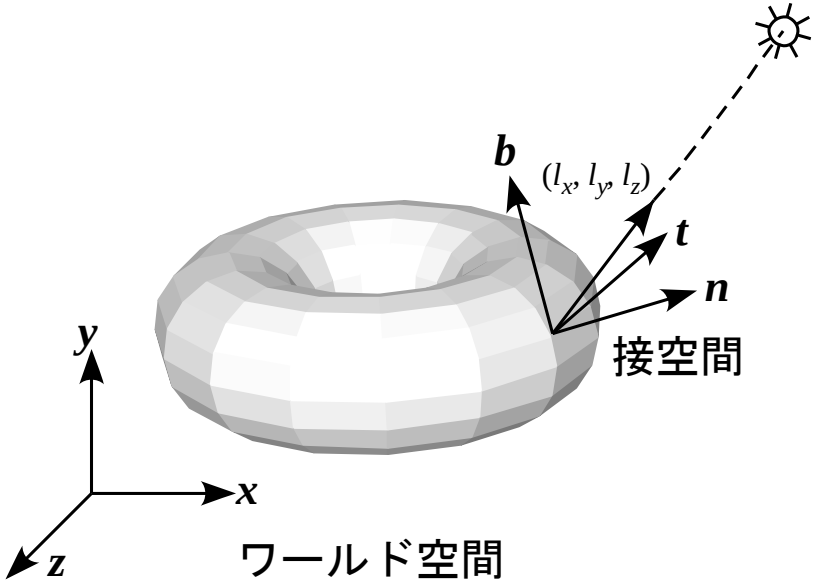
- 物体表面上の一点（この場合頂点）の接空間における光源ベクトル (l'_x, l'_y, l'_z) を求める
- テクスチャの解像度が r のとき，この点におけるテクスチャ座標を $(u + l'_x / r, v + l'_y / r)$ にずらす

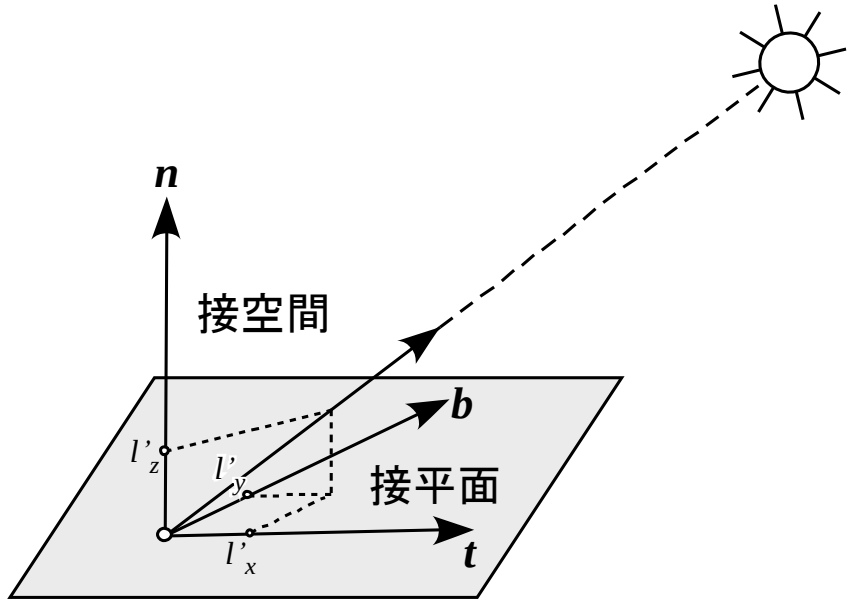




接空間 (Tangent Space)

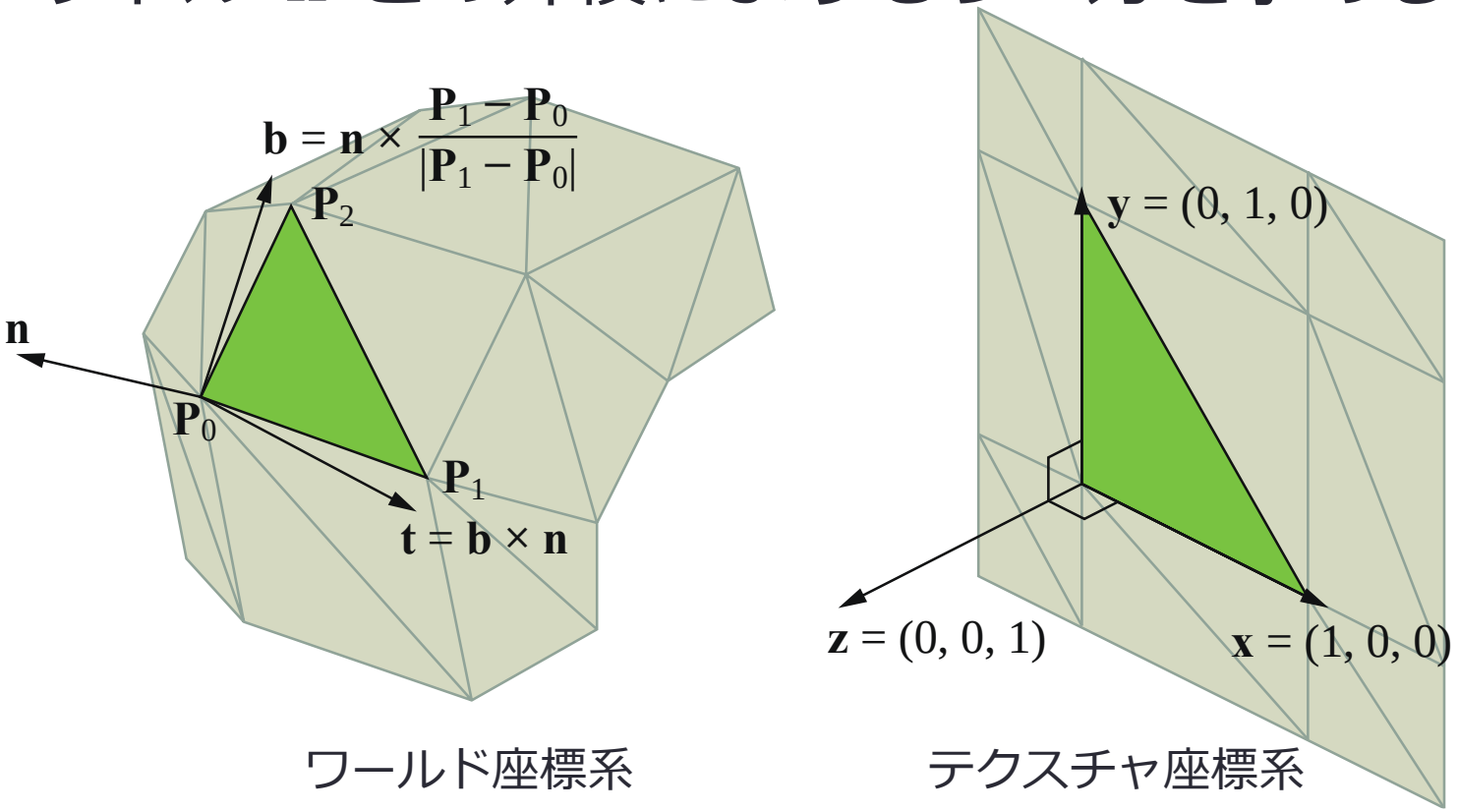
- 頂点ごとに保持される局所的な空間
  - **n**: 法線ベクトル
  - **t**: 接線ベクトル
  - **b**: 従接線ベクトル（従法線ベクトル）





接空間ベクトルの算出方法の例

- 接線ベクトル **t** あるいは従接線ベクトル **b** を決定する
- それと法線ベクトル **n** との外積によりもう一方を求める

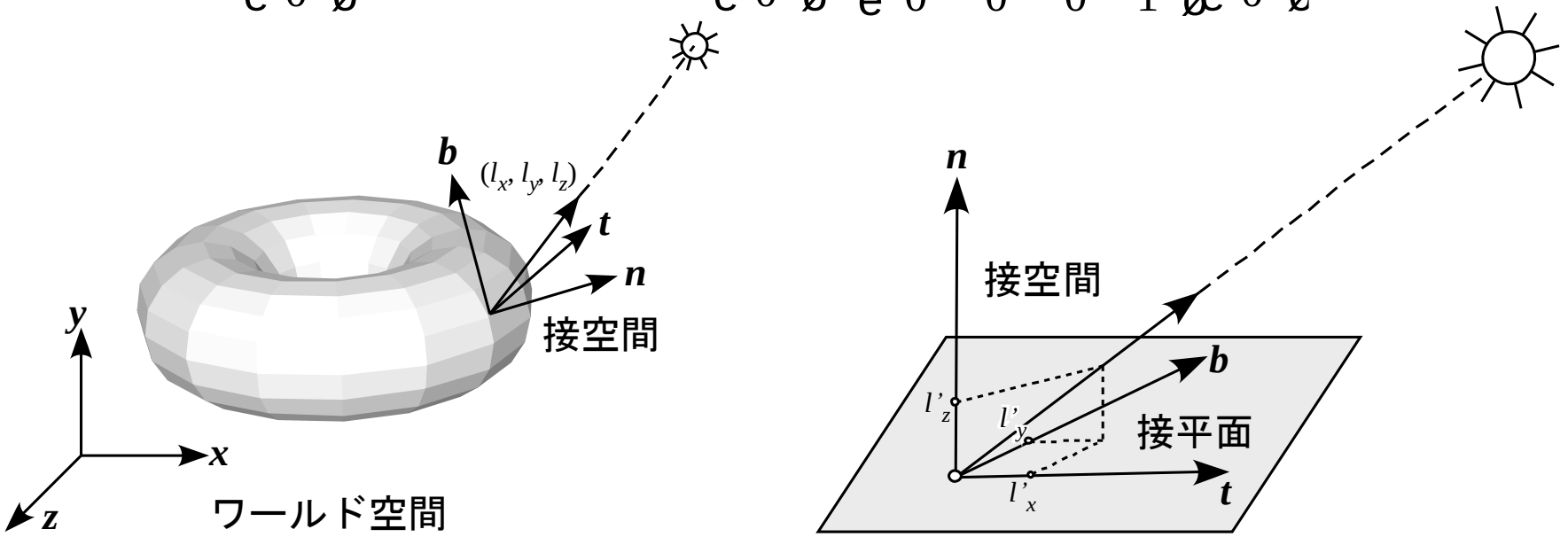


ワールド座標系　　テクスチャ座標系

接空間基底行列

- 光の方向をワールド空間から接空間に変換

$$\begin{pmatrix} l'_x \\ l'_y \\ l'_z \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{t} & \mathbf{b} & \mathbf{n} & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{pmatrix}^T \begin{pmatrix} l_x \\ l_y \\ l_z \\ 0 \end{pmatrix} = \begin{pmatrix} t_x & t_y & t_z & 0 \\ b_x & b_y & b_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} l_x \\ l_y \\ l_z \\ 0 \end{pmatrix}$$

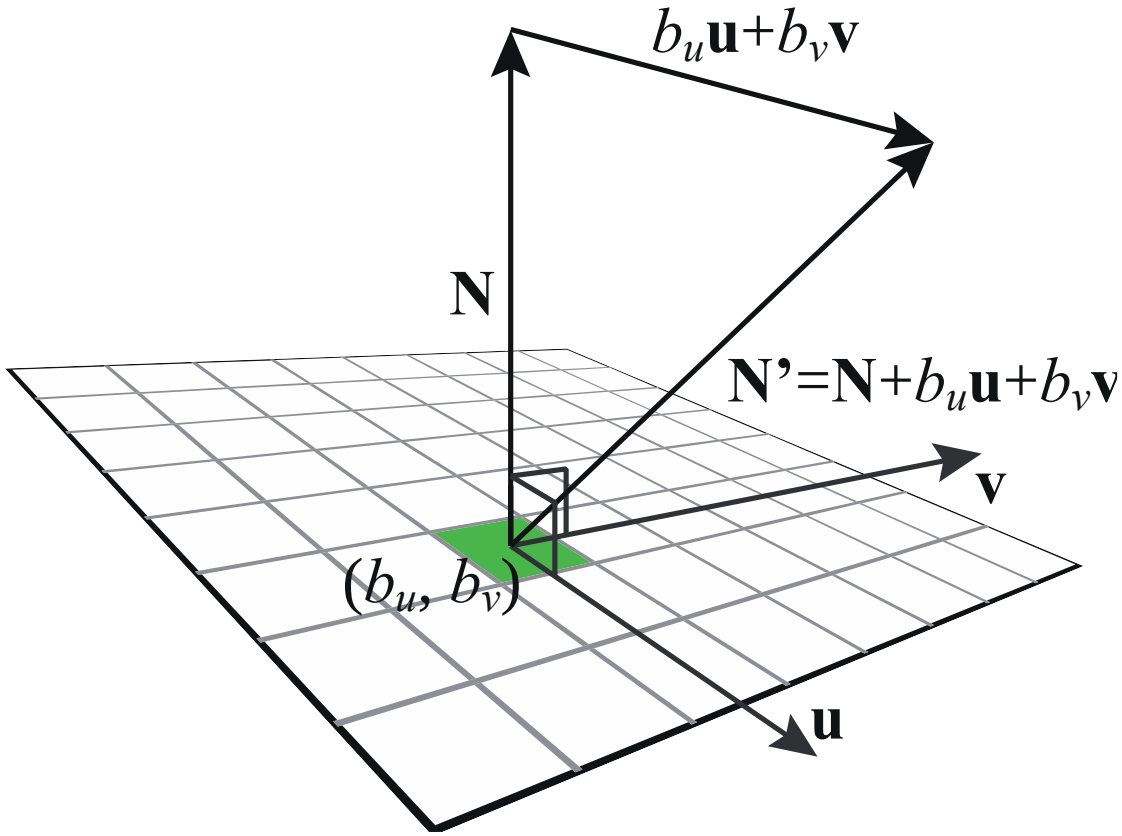


Emboss Bump Mapping の問題

- 拡散反射面にしか適用できない
- 光が正面から当たっているところでは凹凸が現れない
  - 光が正面から当たった場合にはずれが生じない
- 視点から遠く離れた面の凹凸は取り扱えない
  - ずれが小さくなってしまう
- Mipmap が利用できない
  - テクスチャの解像度に依存した手法
- 多分，もう使われない
  - んなら説明するな

Blinn の方法

- 面の法線ベクトル $\mathbf{N}$ と直交するベクトル $\mathbf{u}, \mathbf{v}$ を考える
  - これは接空間の基底ベクトル
- 画素ごとに (bu, bv) の 2 要素をもつテクスチャを用意する
- $\mathbf{N'} = \mathbf{N} + bu\mathbf{u} + bv\mathbf{v}$ で得られたベクトル $\mathbf{N'}$ を使って陰影計算を行う



法線マップ

- 高さマップから接空間での法線ベクトル (x, y, z) を求める
  - 法線ベクトル (x, y, z) を (r, g, b) に格納する
    - $(r, g, b) \leftarrow \{\ 0.5 * (x, y, z) + (0.5, 0.5, 0.5)\ \} * 255$

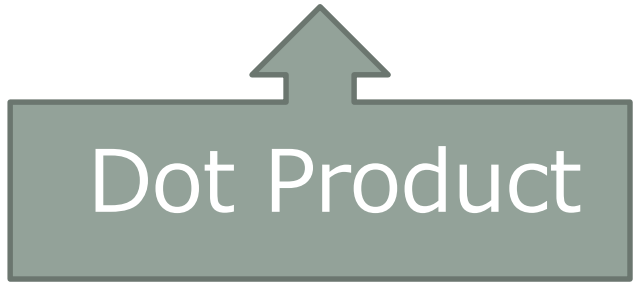
高さマップ

値を unsigned char で表す場合

法線マップ

Dot Product (Dot3) Bump Mapping

- 陰影は光線ベクトルと法線ベクトルの**内積**で求まる
- 法線ベクトルは**法線マップ**としてテクスチャに保持できる
- 光線ベクトルも**光線マップ**としてテクスチャに保持する
- 二つのテクスチャで画素ごとに**内積計算**を行う

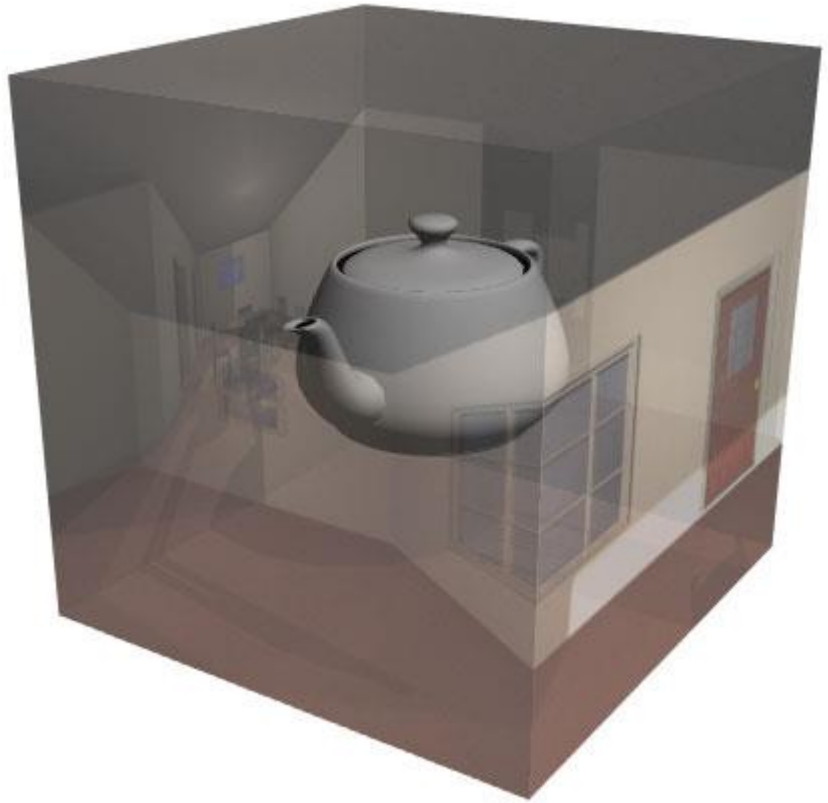


光源マップ（正規化マップ）

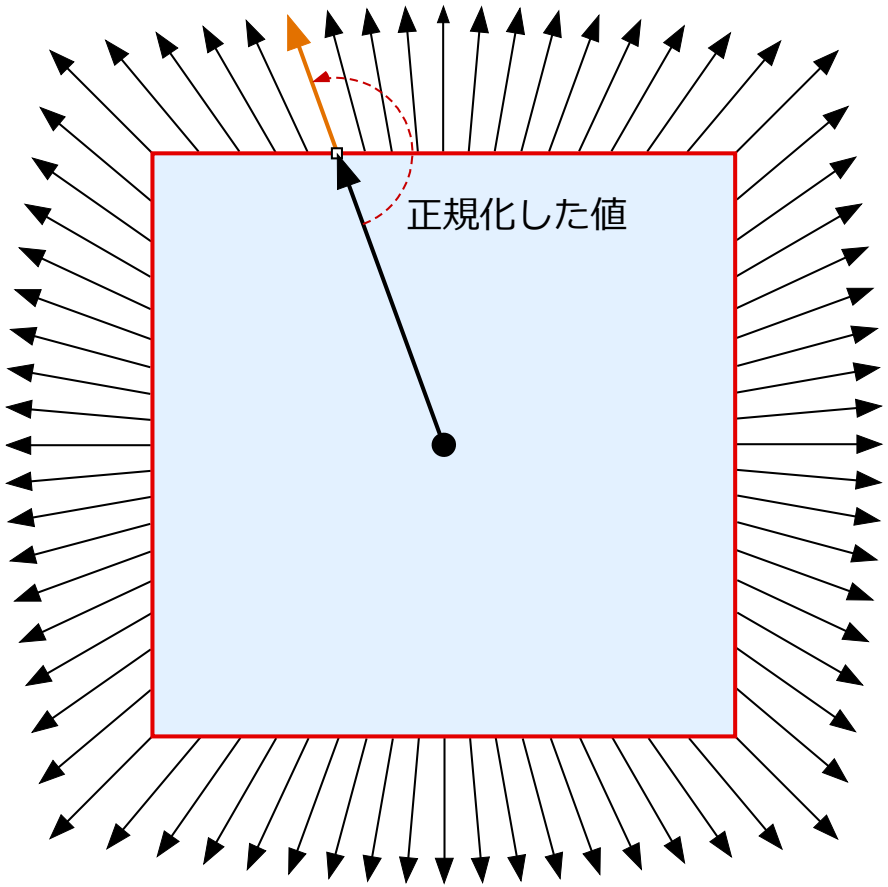
- 接空間における光線ベクトルをテクスチャから得る
  - 頂点における光線ベクトルは頂点属性として保持する
  - 画素における光線ベクトルは補間により求める
    - 補間された法線ベクトルは正規化されていない
- 補間した光線ベクトルをテクスチャを使って正規化する
  - テクスチャの各画素にその画素のテクスチャ座標を正規化したものを格納する
  - 光線ベクトルをテクスチャ座標に使ってそのテクスチャをサンプリングする
    - 環境マッピング（次回に説明）の機能を使う
      - 正規化されていない３次元ベクトルを使ってテクスチャをサンプリングできる

環境マッピングによる正規化

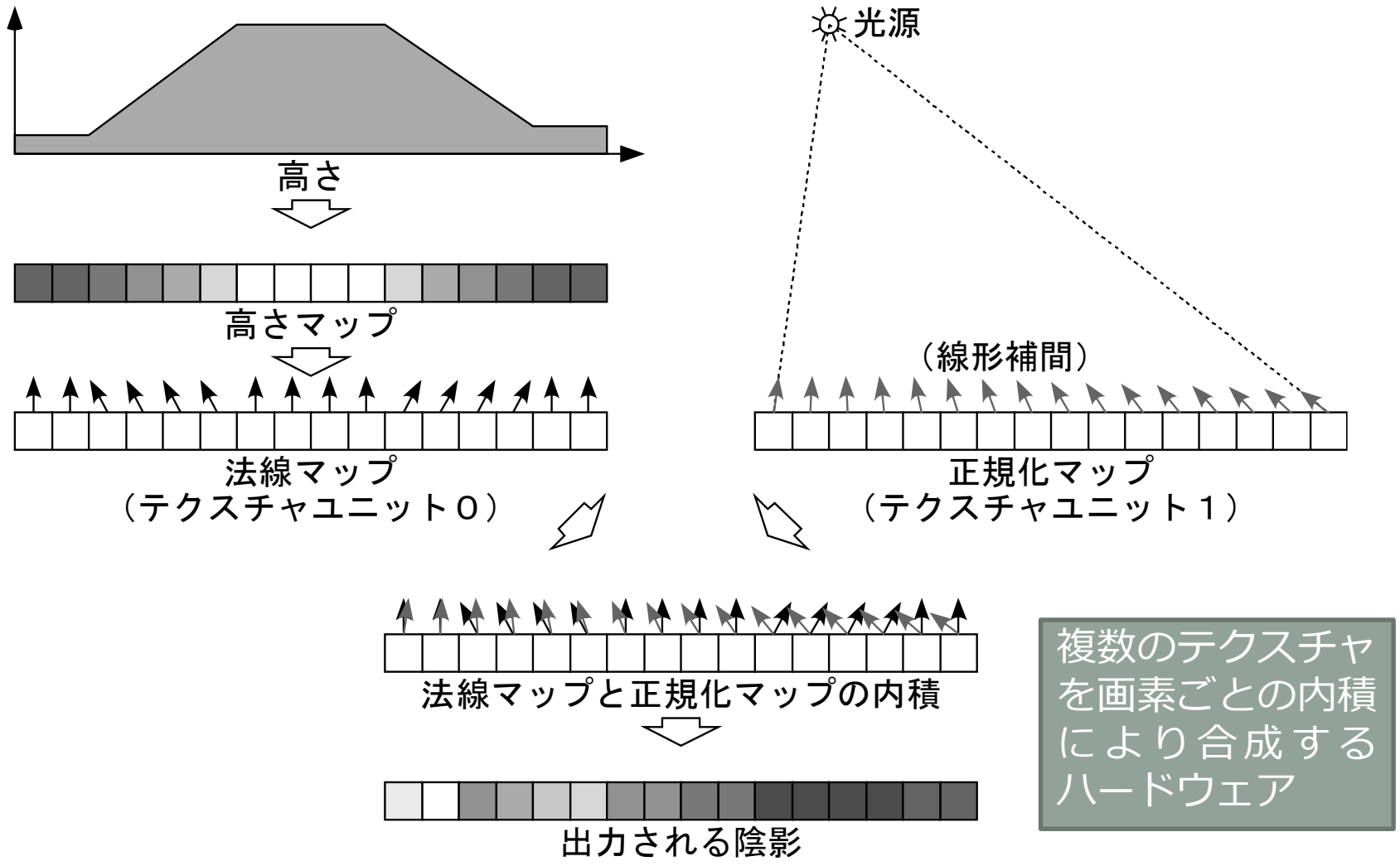
環境マッピング



正規化マップ



法線マップと光線マップの内積



複数のテクスチャを画素ごとの内積により合成するハードウェア

シェーダによる実装

- バーテックスシェーダ
  - 接線ベクトル $\mathbf{t}$ と従接線ベクトル $\mathbf{b}$ を求める
  - $\mathbf{t}, \mathbf{b}$ と法線ベクトル $\mathbf{n}$ を用いて接空間基底行列 $(\mathbf{t}\ \mathbf{b}\ \mathbf{n})^T$ を求める
  - 光線ベクトル $\mathbf{L}$ と中間ベクトル $\mathbf{H}$ を接空間基底行列で変換する
  - $\mathbf{L}$ と $\mathbf{H}$ を out 変数に格納してフラグメントシェーダに送る
    - $\mathbf{n}$ はフラグメントシェーダに送る必要は無い
- フラグメントシェーダ
  - 法線ベクトル $\mathbf{n}$ を法線マップのテクスチャを標本化して得る
    - テクスチャの値の範囲は [0,1] なので 2 倍して 1 引いて [-1, 1] に変換する
      - テクスチャの内部フォーマットを RGB16F や RGB32F にすればこれは不要
  - この $\mathbf{n}$ と in 変数により得た補間された $\mathbf{L}$ と $\mathbf{H}$ を用いて陰影を求める
    - $\mathbf{L}$ と $\mathbf{H}$ が接空間にあるので法線マップから得た $\mathbf{n}$ をそのまま使えば良い

シェーダによる実装では正規化マップは不要

バーテックスシェーダ

```
#version 410
…
out vec3 l;      // 接空間における光線ベクトル
out vec3 h;      // 接空間における中間ベクトル
out vec2 tc;     // フラグメントシェーダに送るテクスチャ座標
…
void main(void)
{
    vec4 p = mw * pv;                          // 視点座標系の頂点位
    vec3 v = normalize(p.xyz);                    // 視線ベクトル
    vec3 n = normalize((mg * nv).xyz);            // 法線ベクトル
    vec3 t = normalize(vec3(n.z, 0.0, -n.x));     // 接線ベクトル
    vec3 b = cross(n, t);                         // 従接線ベクト
    mat3 m = transpose(mat3(t, b, n));            // 接空間基底行列
    l = normalize(m * vec3(4.0, 5.0, 6.0));       // 光線ベクトル
    h = normalize(l + m * v);                     // 中間ベクトル
    tc = tv;
    …
```

法線ベクトル **n** は法線マップから獲得するのでフラグメントシェーダに送る必要はない

フラグメントシェーダ

```
#version 410
…
in vec3 l;                  // 補間された接空間における光線ベクトル
in vec3 h;                  // 補間された接空間における中間ベクトル
in vec2 tc;                 // 補間されたテクスチャ座標

uniform sampler2D dmap;  // diffuse color map
uniform sampler2D smap;  // specular color map
uniform sampler2D nmap;  // normal map

void main(void)
{
  vec3 nn = texture(nmap, tc).xyz * 2.0 - 1.0;  // 法線ベクトル
  vec4 iamb = kamb * lamb;
  vec4 idiff = max(dot(nn, l), 0) * kdiff * ldiff;
  vec4 ispec = pow(max(dot(nn, h), 0), kshi) * kspec * lspec;
  …
```

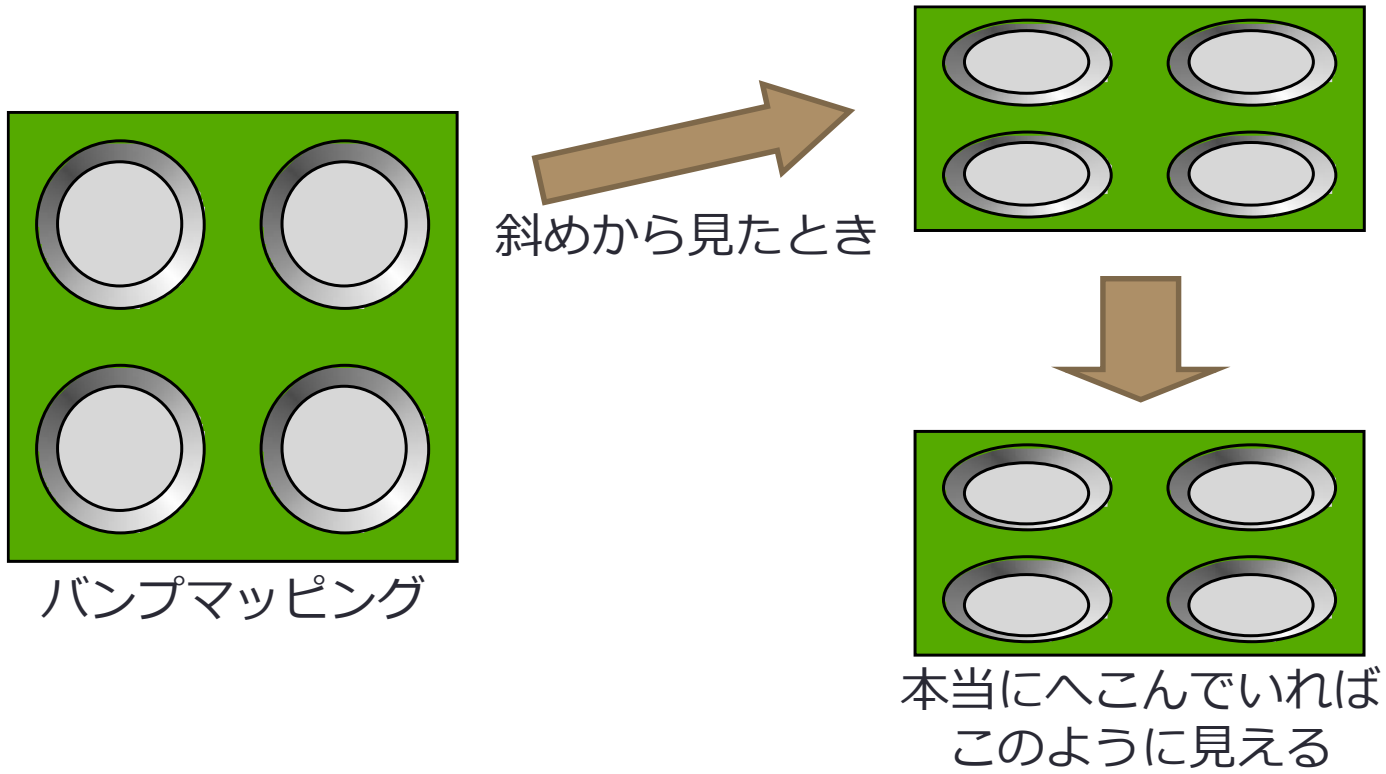
法線マッピングの例

バンプマッピングの拡張

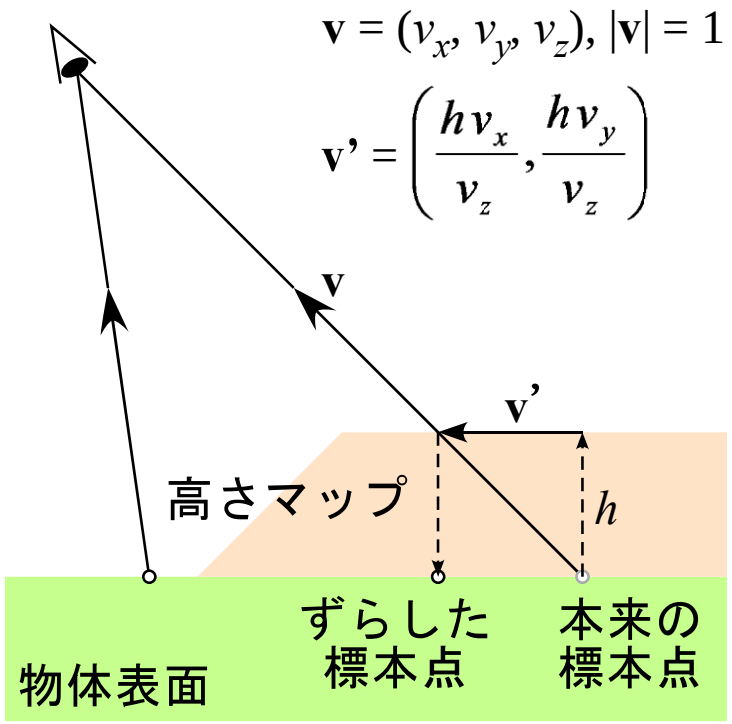
より精密なディティールの再現

視差マッピング (Parallax Mapping)

- バンプマッピングの問題



高さマップを参照



- テクスチャの標本点の高さマップを参照する
- 高さに比例してテクスチャの標本点を視線方向にずらす

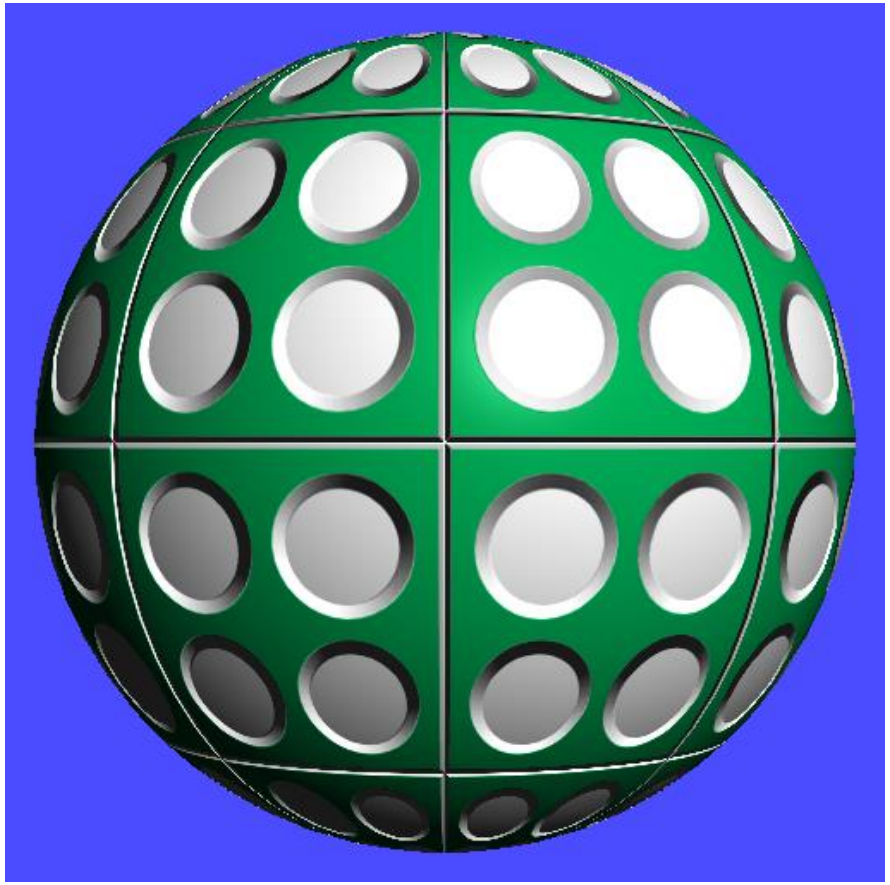
$$\mathbf{v}' = \left(\frac{hv_x}{v_z} \quad \frac{hv_y}{v_z}\right)$$

v_z で割ると変化が急なので

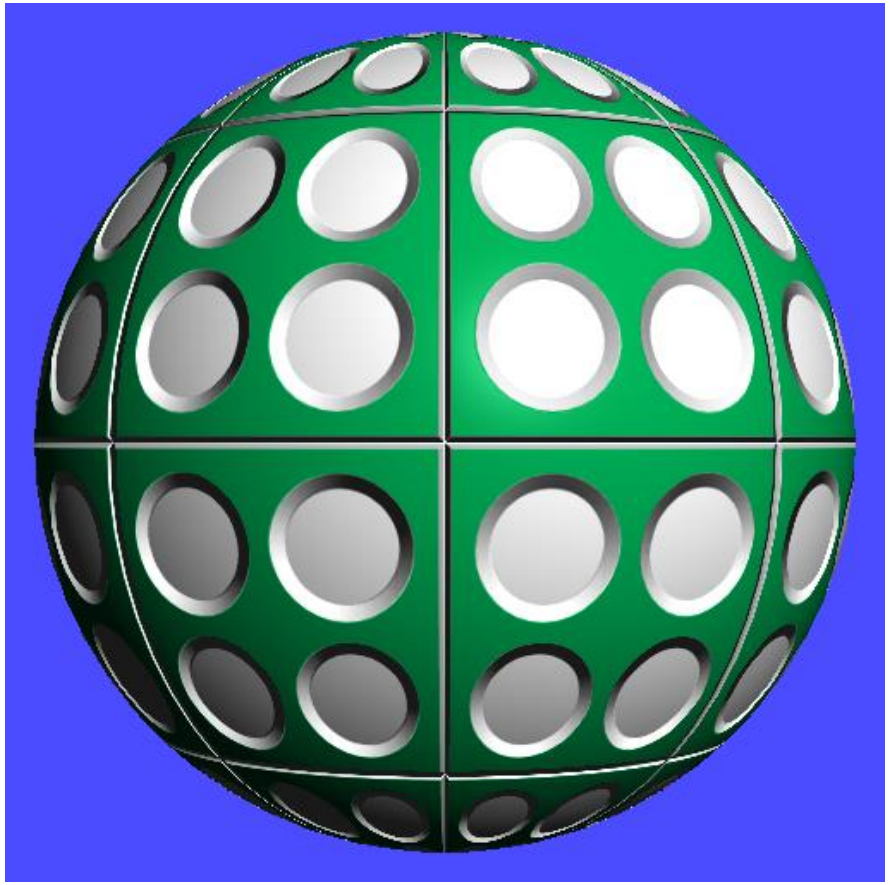
$$\mathbf{v}' = \left(hv_x \quad hv_y\right)$$

視差マッピングの効果

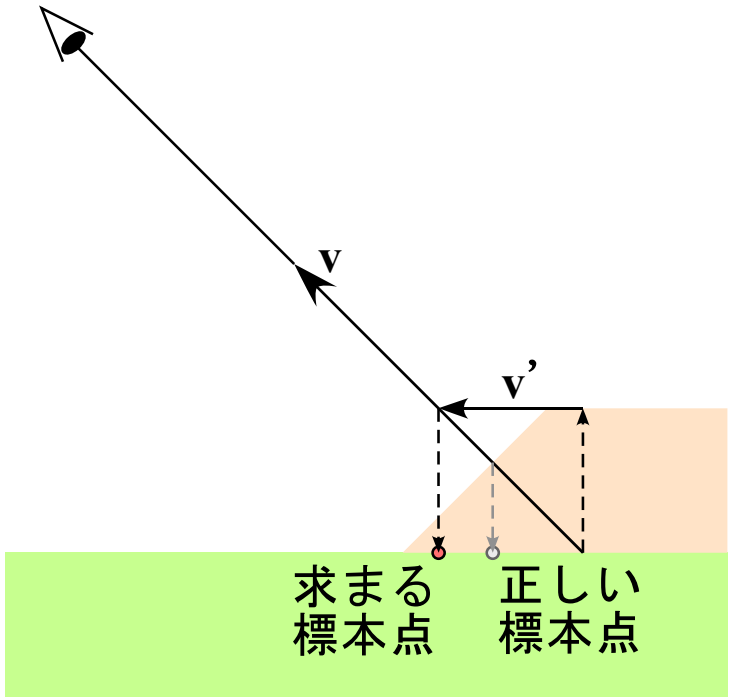
視差マッピングなし



視差マッピングあり

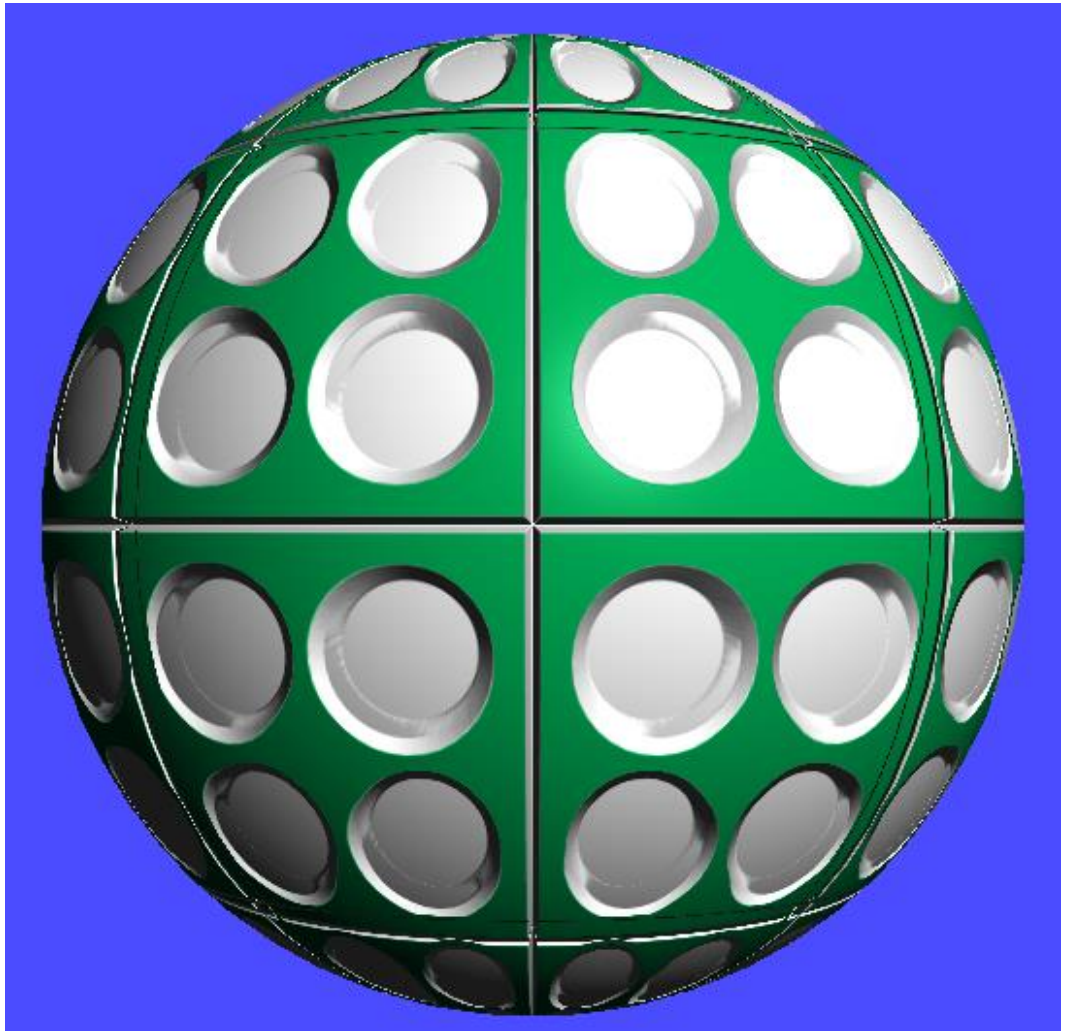


視差マッピングの限界



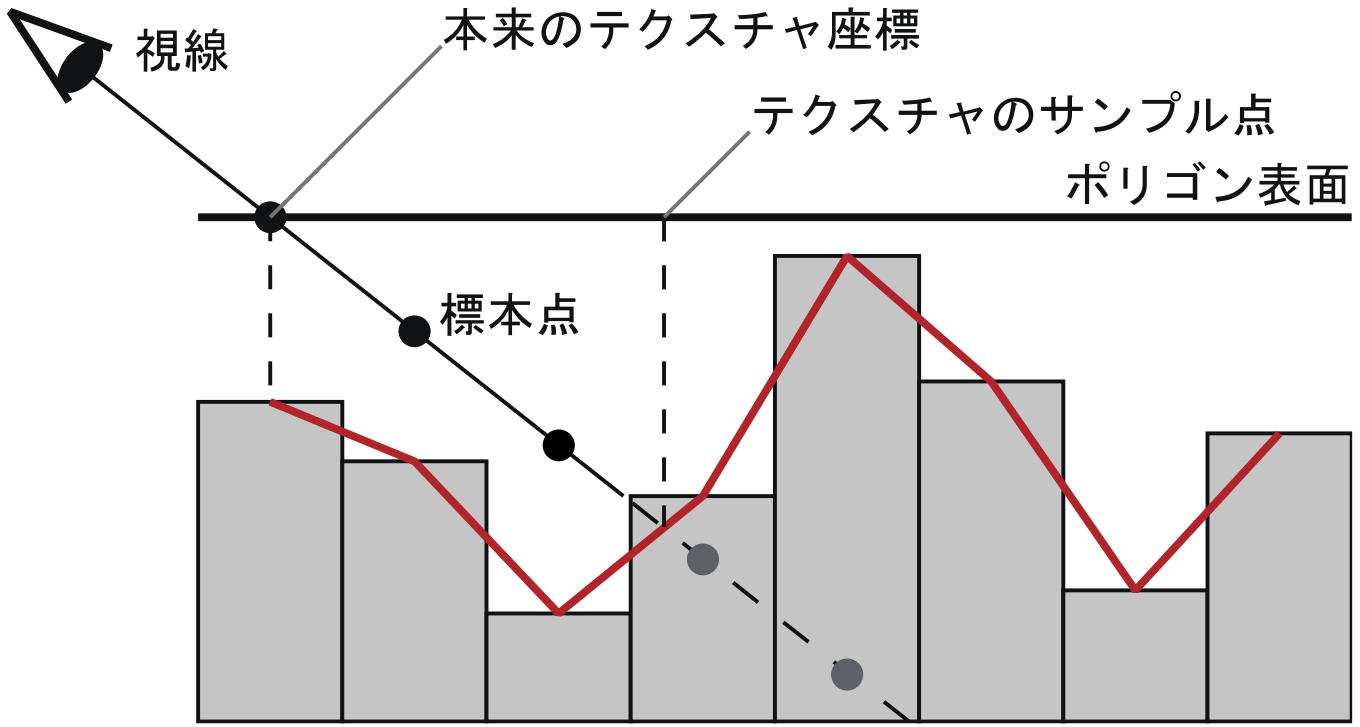
- 高さマップの変化が急だと求まる標本点の位置と正しい標本点の位置のずれが大きくなる
- 隠面消去を行っていないので面の凹凸によって本来隠される部分が見えてしまう

高さマップの変化を大きくしてみた



レリーフマッピング

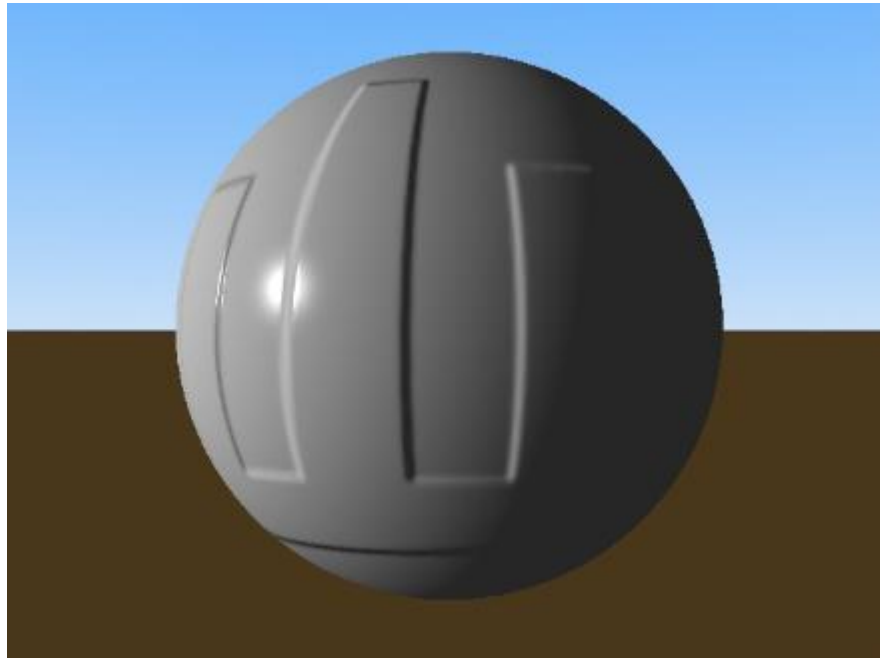
- 高さマップに対して隠面消去処理を行う
  - レイトレーシング的な手法を用いる
    - 視線上の標本点と高さマップを比較して交点を求める



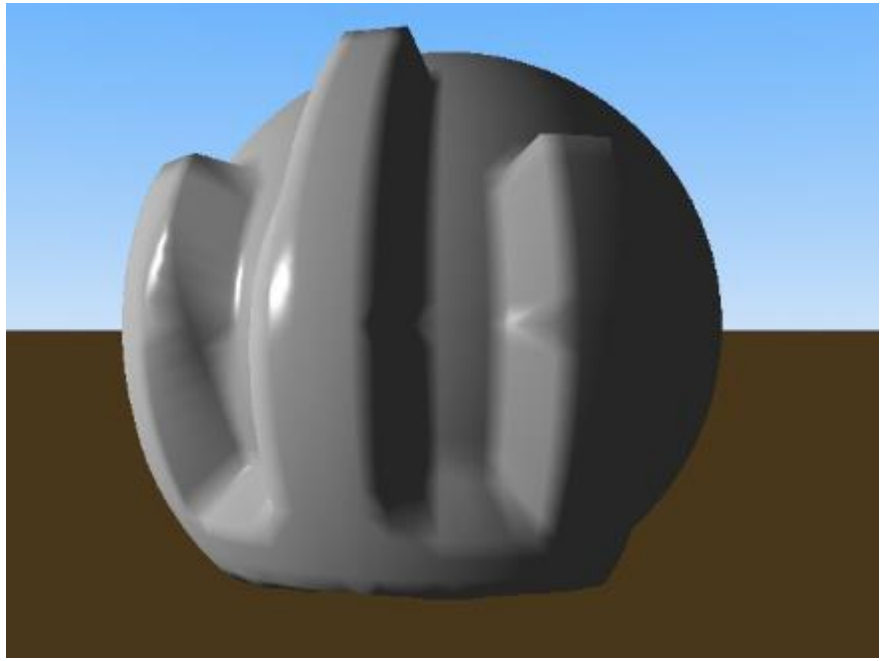
ATI Toy Shop Demo

Bump Mapping と Displacement Mapping

Bump Mapping



Displacement Mapping



小テスト－テクスチャマッピング

Moodle の小テストに解答してください

宿題

- バンプマッピングを実装してください.
  - 次のプログラムはテクスチャをマッピングした球の回転アニメーションを表示します.
    - https://github.com/tokoik/ggsample09
  - これにテクスチャユニット1に割り当てられたテクスチャを法線マップとして用いてバンプマッピングを追加してください.
  - このテクスチャの a 要素（アルファ値）には高さマップの値が入っています．これが 0 以下の時に鏡面反射光を加算するようにしています.
  - uniform 変数 normal に法線マップのテクスチャユニットが割り当てられています.
  - テクスチャの値は [0,1] の範囲なので，これを [-1,1] に変換したものを法線ベクトルとして使ってください.
- ggsample09.frag を**アップロード**してください

宿題プログラムの生成画像

バンプマッピングなし

バンプマッピングあり