

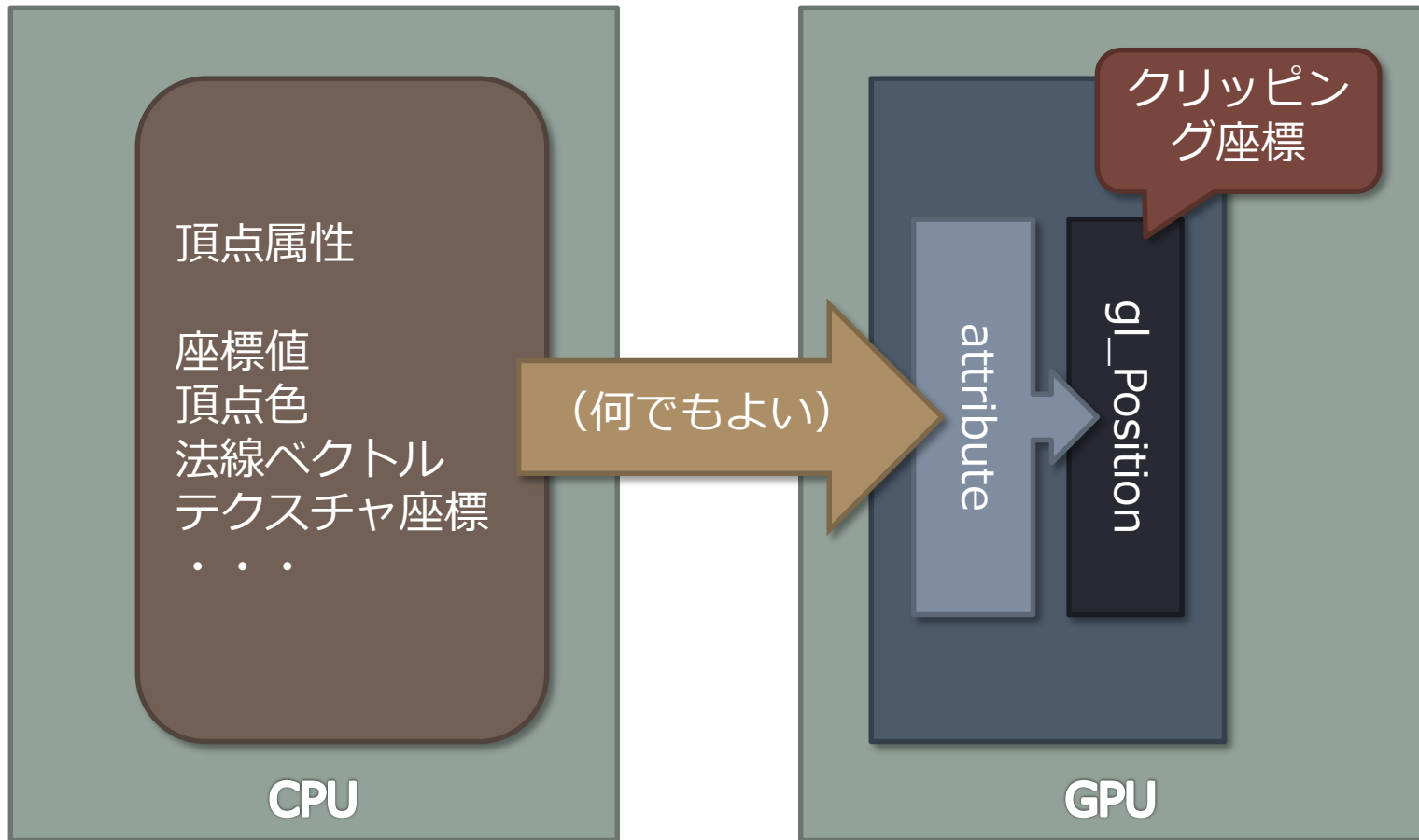
ゲームグラフィックス特論

第5回 変換 (3)

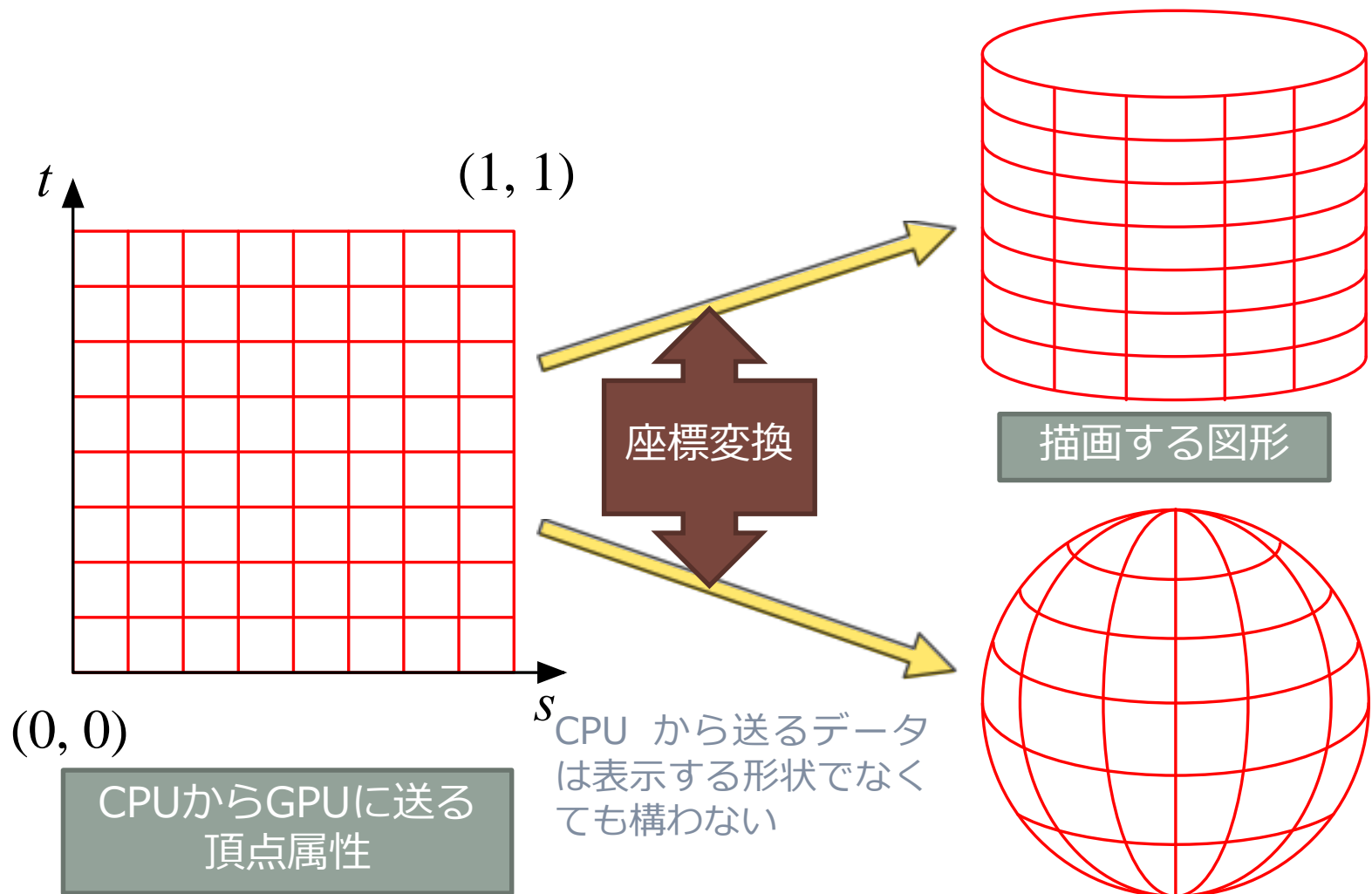
パラメータによる図形描画

バーテックスシェーダで座標を生成する

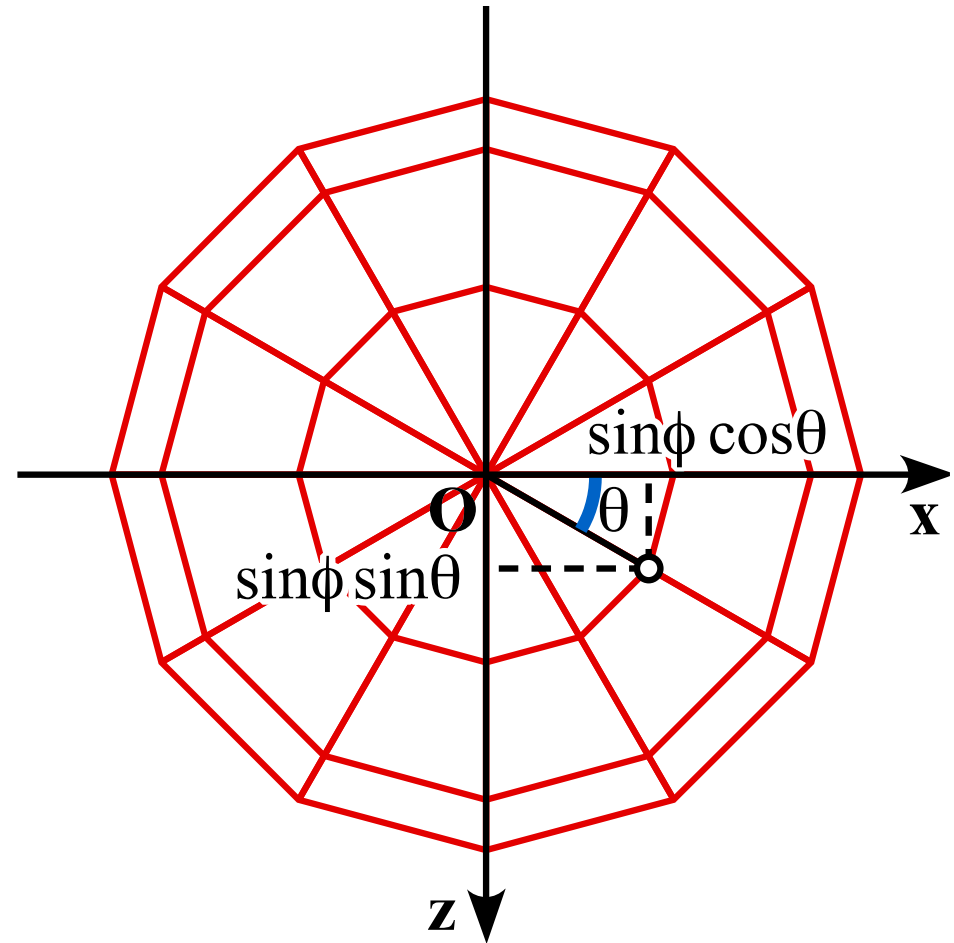
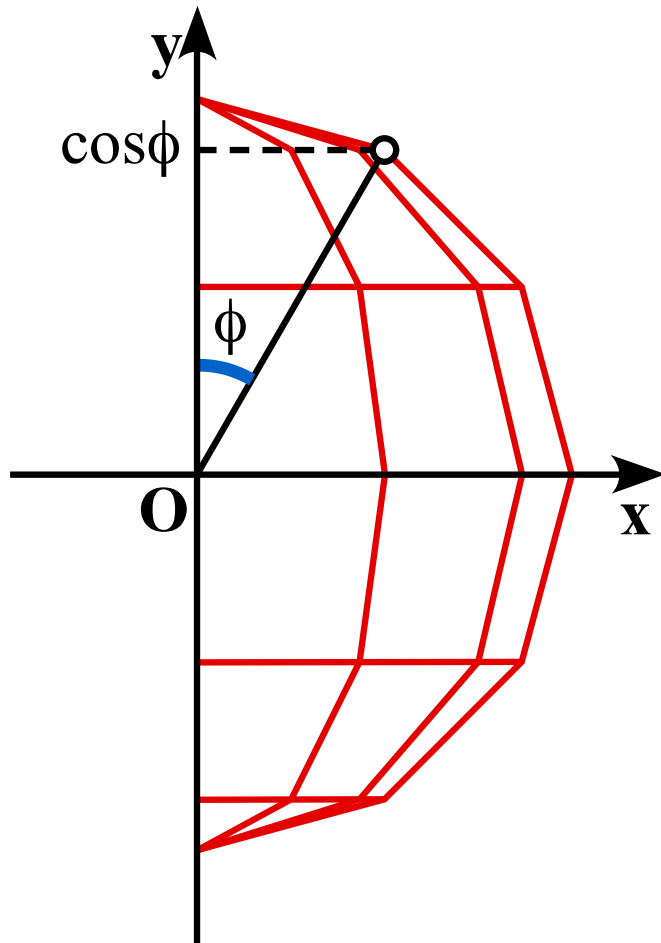
頂点属性の情報



パラメータを頂点属性として使って描画



方位角・仰角と球面上の点の位置



バーテックスシェーダーで球の座標生成

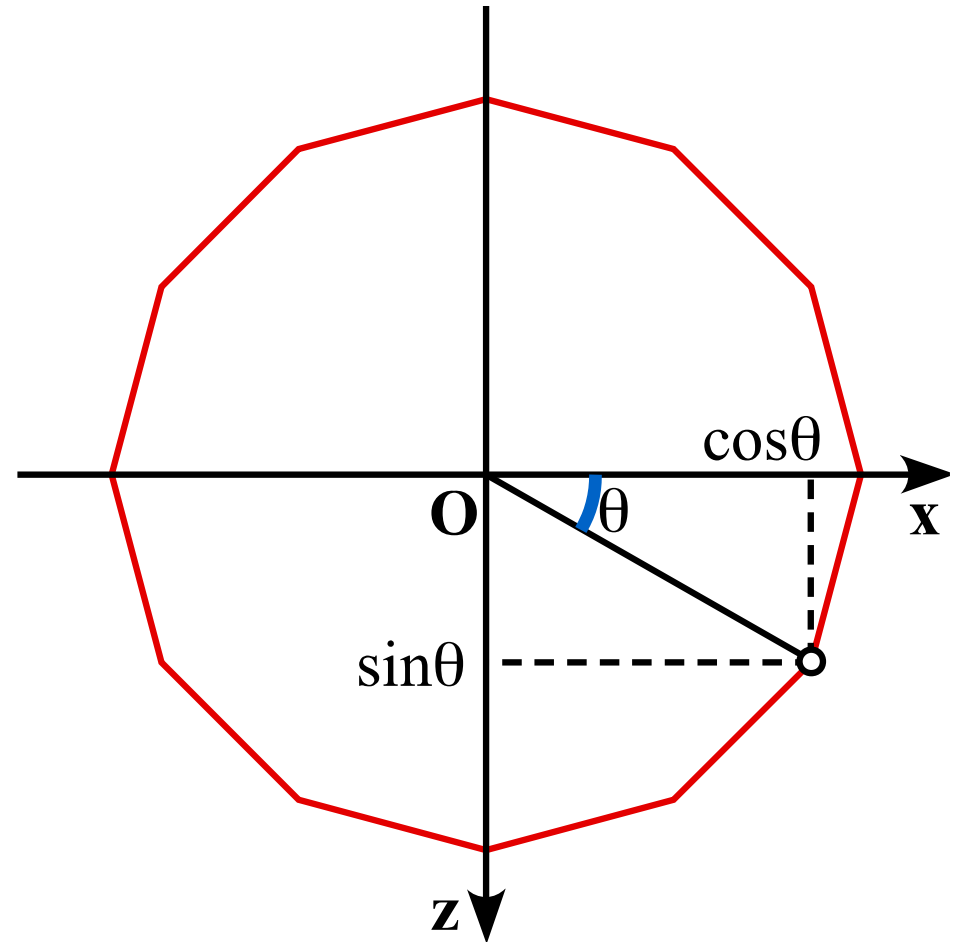
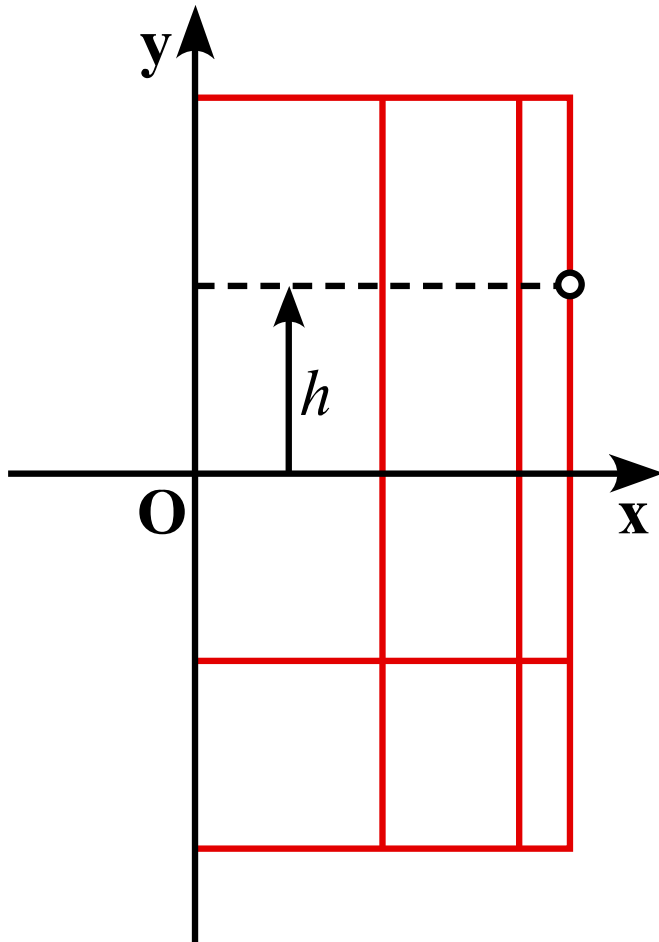
```
#version 150 core
in vec2 parameter;           // CPU から与えられる頂点属性
uniform mat4 mc;              // 変換行列

void main(void)
{
    float th = 6.283185 * parameter.s;           //  $[0,1] \rightarrow [0,2\pi]$ 
    float ph = 3.141593 * parameter.t;           //  $[0,1] \rightarrow [0,\pi]$ 
    float r = sin(ph);

    vec4 p = vec4(r * cos(th), cos(ph), r * sin(th), 1.0);

    gl_Position = mc * p;
}
```

方位角・高さと円柱上の点の位置



バーテックスシェーダーで円柱の座標生成

```
#version 150 core
in vec2 parameter;           // CPU から与えられる頂点属性
uniform mat4 mc;              // 変換行列

void main(void)
{
    float th = 6.283185 * parameter.s;           // [0,1]→[0,2π]
    float y = parameter.t * 2.0 - 1.0;           // [0,1]→[-1,1]

    vec4 p = vec3(cos(th), y, sin(th), 1.0);

    gl_Position = mc * p;
}
```


変形する

点ごとに異なる変換

座標変換を行う対象

描画単位全体

- 剛体変換
 - 形状の変形を伴わない
- 拡大縮小・せん断
 - 描画単位全体を均一に変形



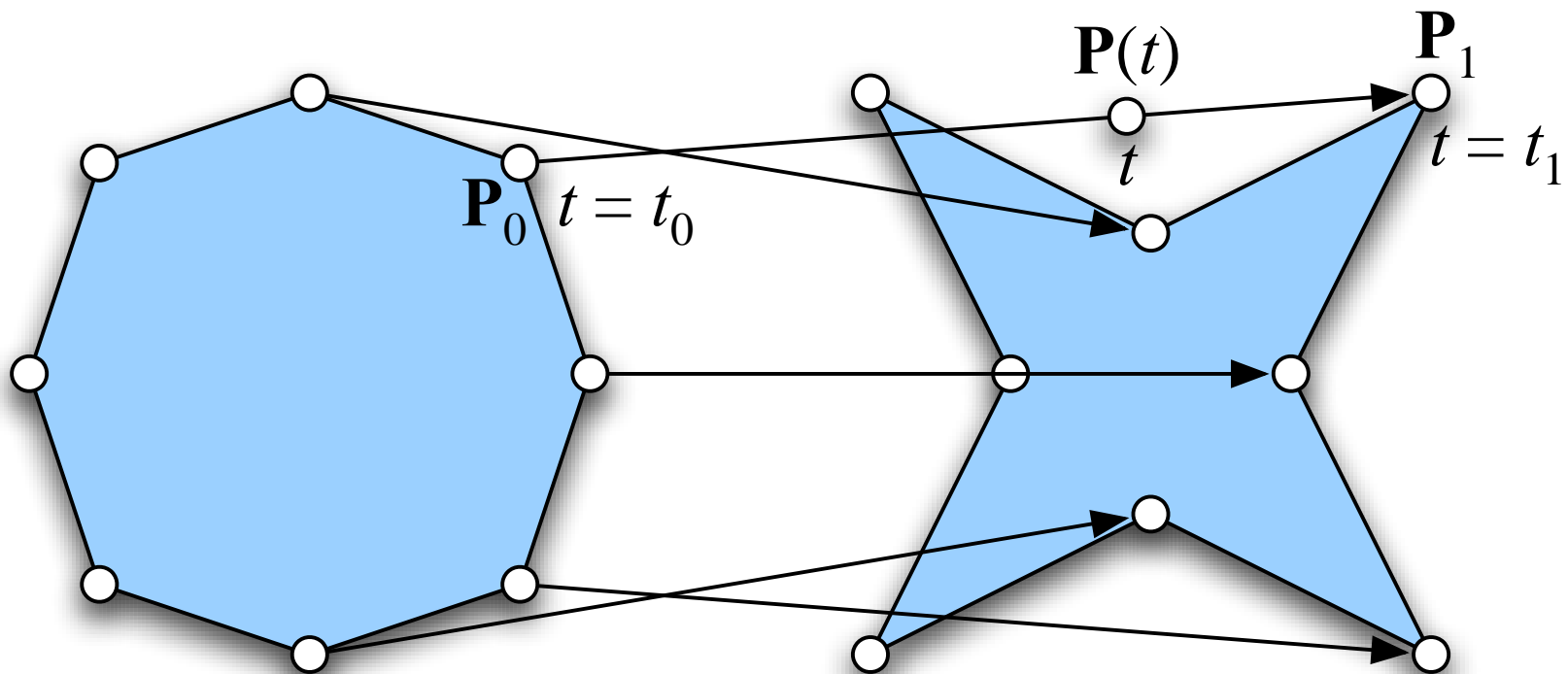
描画前に変換行列を設定する

頂点ごと

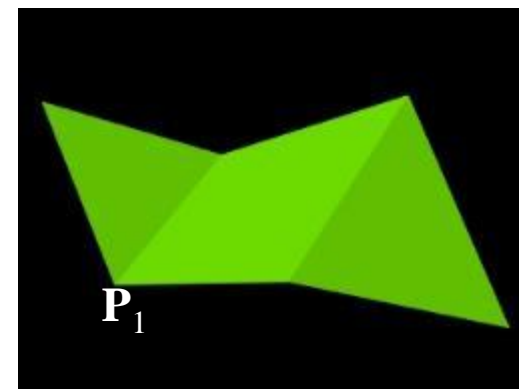
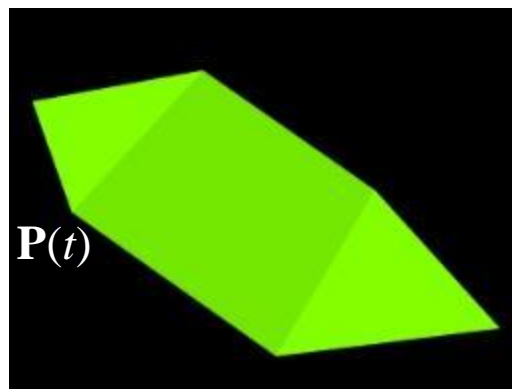
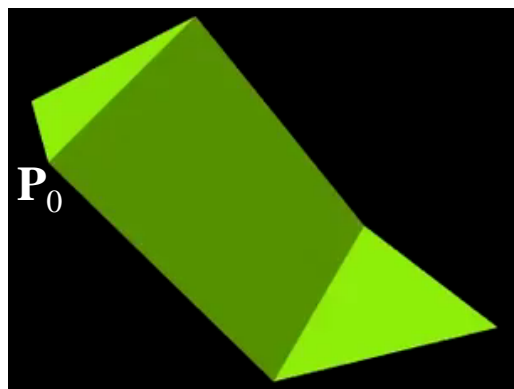
- 局所的な変形
 - キャラクタの口パク
 - 手足の曲げのぼし
- 頂点ごとに処理が異なる
 - すべての頂点をCPUで変換してから描画する
 - 元になる頂点属性 (attribute) だけをGPUに送りシェーダプログラムで変換する

モーフィング

- すべての頂点に対して変形前と変形後の位置を指定する
 - 中間の頂点の位置を補間により求める
- 頂点の順序／インデックスはそのまま



頂点位置の線形補間



t_0 ————— t ————— t_1

$$t \in [t_0, t_1]$$

$$0 \leq s(t) \leq 1$$

$$s(t) = \frac{t - t_0}{t_1 - t_0}$$

$$\mathbf{P}(t) = \{1 - s(t)\}\mathbf{P}_0 + s(t)\mathbf{P}_1$$

一つの頂点に複数の頂点属性 (attribute)

バーテックスシェーダによる補間

```
#version 150 core
in vec4 p0, p1;           // 変形前と変形後の点の位置
uniform float t;           // 時刻
uniform mat4 mc;           // 座標変換行列
void main(void)
{
    gl_Position = mc * mix(p0, p1, t);
}
```

$(1 - t)P_0 + tP_1$ を計算する
組み込み関数がある！


attribute 変数のインデックスの取り出し

```
// シェーダプログラムの読み込み
program = glCreateProgram();

... (ソースプログラムの読み込み, コンパイル, 取り付け等)

// in (attribute) 変数 p1 のインデックスの検索 (見つからなければ -1)
GLint p1Loc = glGetAttribLocation(program, "p1");

// p1 の頂点バッファオブジェクトの作成
GLuint p1Buf;
glGenBuffers(1, &p1Buf);
glBindBuffer(GL_ARRAY_BUFFER, p1Buf);
glBufferData(GL_ARRAY_BUFFER,
             sizeof (GLfloat[3]) * vertices, p1, GL_STATIC_DRAW);
```



GPU 側の attribute 変数名
と CPU 側の配列変数名を
意図的に同じにしている

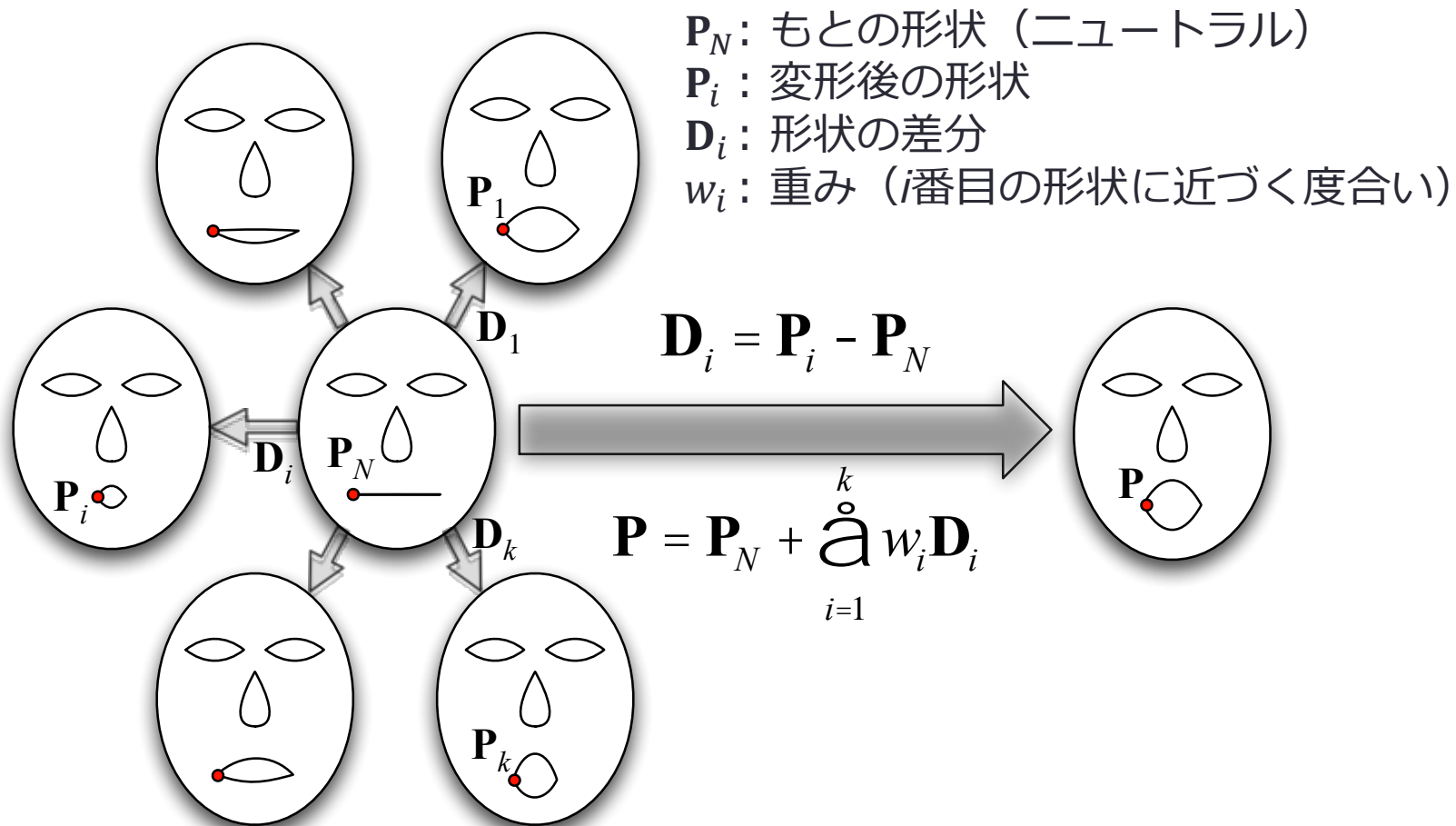
頂点バッファオブジェクトの追加

```
// 描画に使う頂点配列オブジェクトの指定
glBindVertexArray(vao);

// 頂点バッファオブジェクトを in (attribute) 変数 p1 で参照する
glBindBuffer(GL_ARRAY_BUFFER, p1Buf);
glVertexAttribPointer(p1Loc, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(p1Loc);

// 図形の描画
glDrawElements(GL_LINES, lines, GL_UNSIGNED_INT, 0);
```

モーフトarget

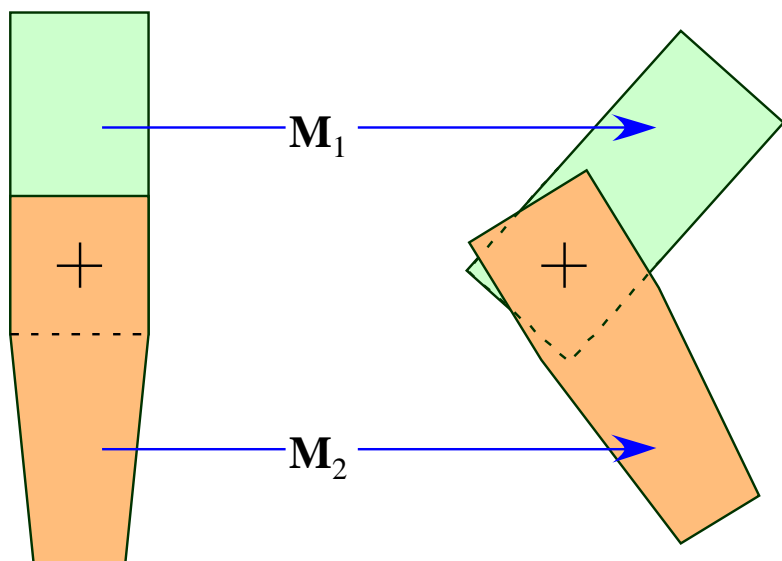


関節を動かす

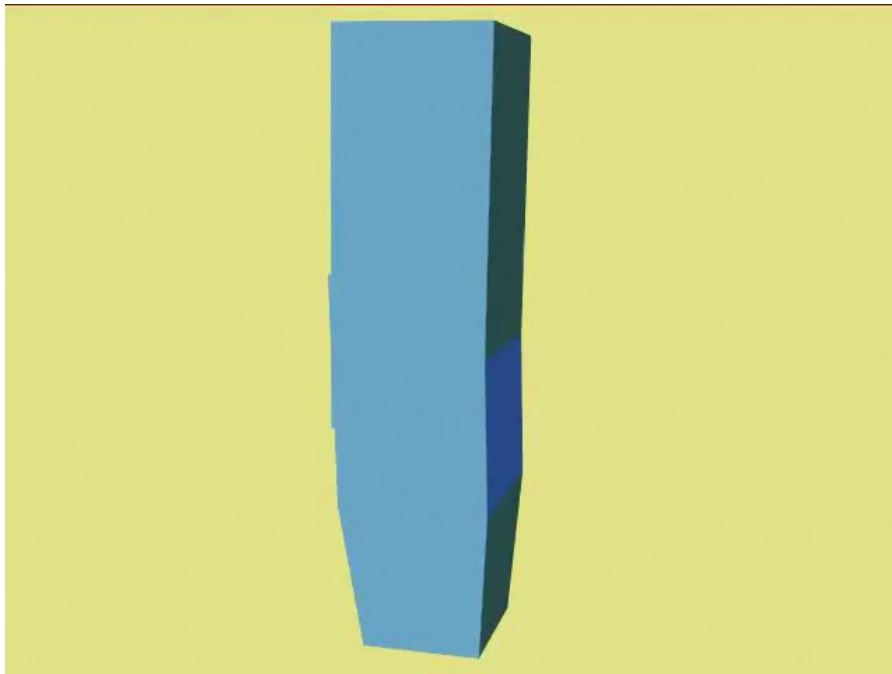
骨格による変形

関節を動かす

- 2つの独立したパーツのそれぞれに対して剛体変換を用いる

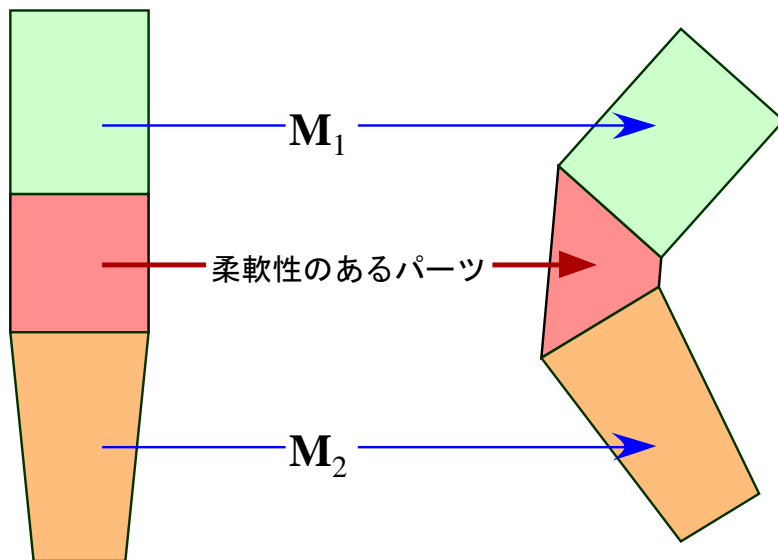


独立したパーツを用いる



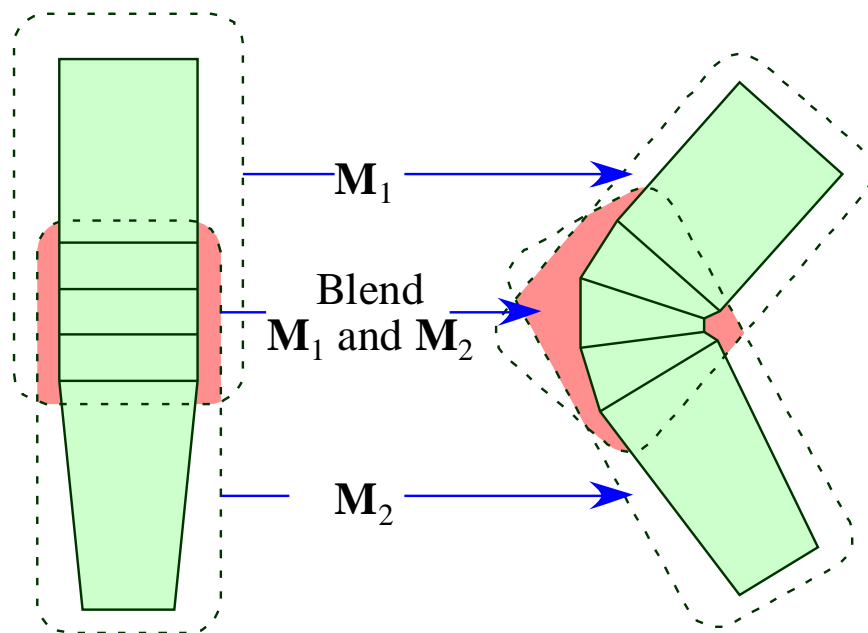
- 接合部分は実際の「ひじ」
のようには見えない
 - 接合部分は2つのパーツが重な
り合っている
- このような部分は単一の
パーツで表現すべき
 - 静的なモデルであらわしたパー
ツでは解決できない

柔軟性のあるパーツを用いる



- 2つのパーツを柔軟性のあるパーツで接合する
- 柔軟性のあるパーツの頂点の座標は、2つのパーツのそれぞれの変換の影響を受ける
 - 近いほうのパーツの変換の影響が大きい

バーテックスブレンディング

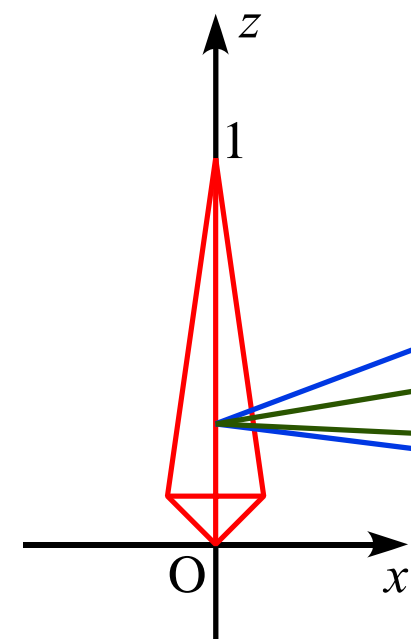


- 全体を柔軟性のある単一のパーツで表現する
- 2つの変換 M_1 , M_2 の影響範囲を決める
- 影響範囲が重なる部分の頂点の座標値は, M_1 , M_2 による変換の結果を合成して求める

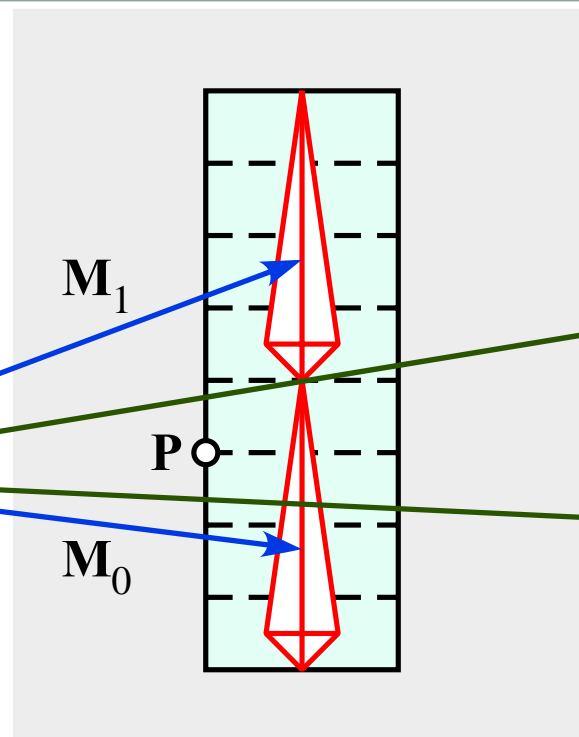
ボーンの設定と移動

M_0, M_1 はオブジェクト内に配置したボーンのローカル座標系からの変換行列

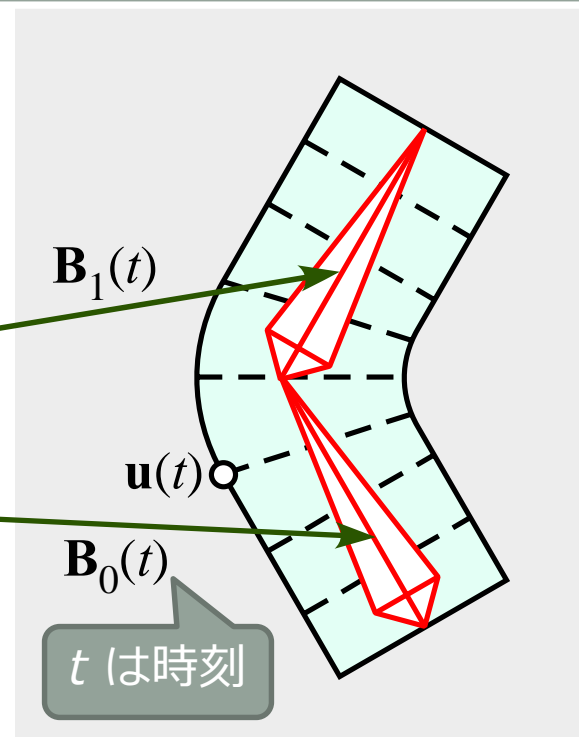
$B_0(t), B_1(t)$ は配置したボーンの移動後のローカル座標系からの変換行列



ボーン
(ローカル座標系)

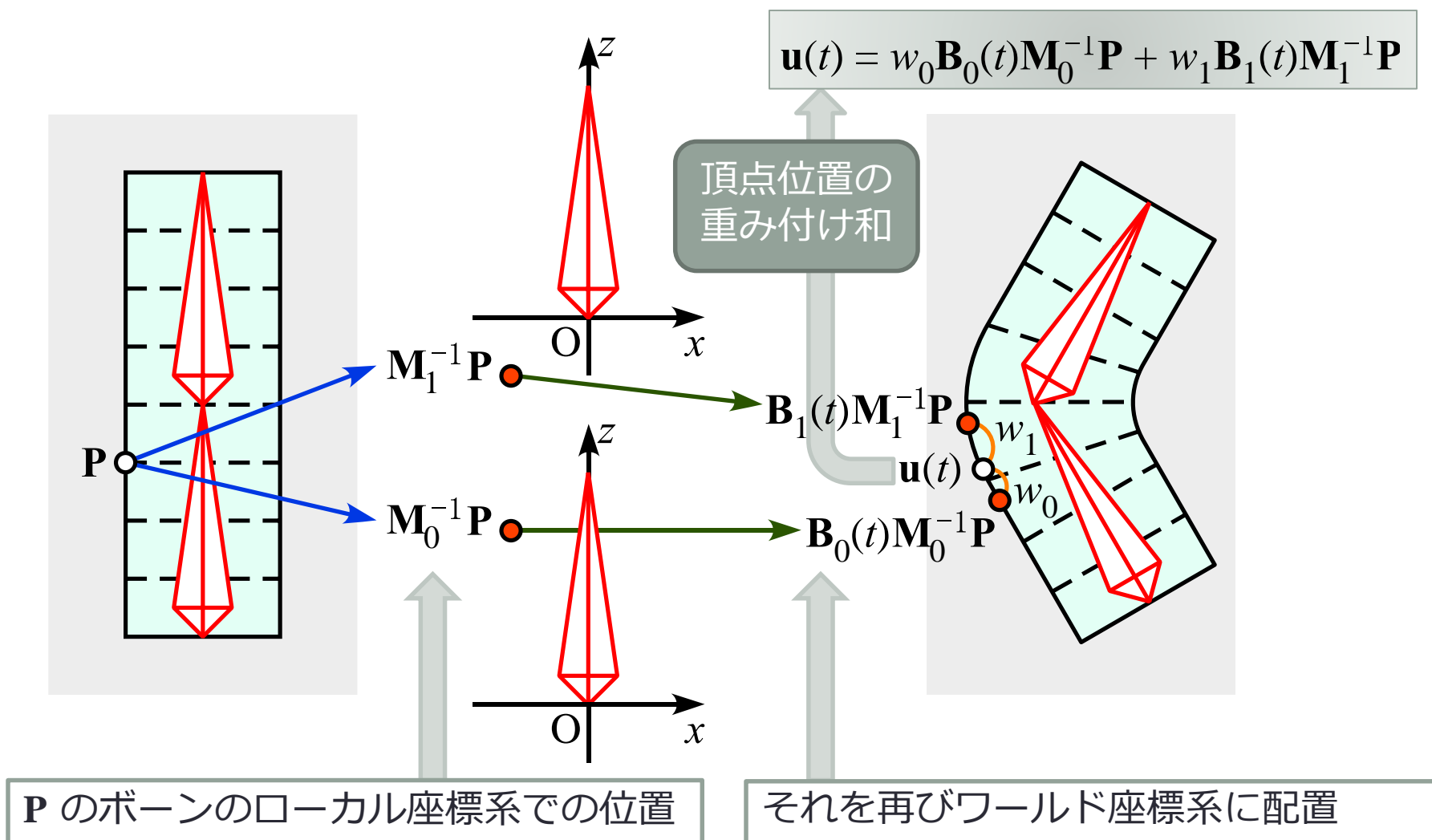


ボーンの設定
(ワールド座標系)

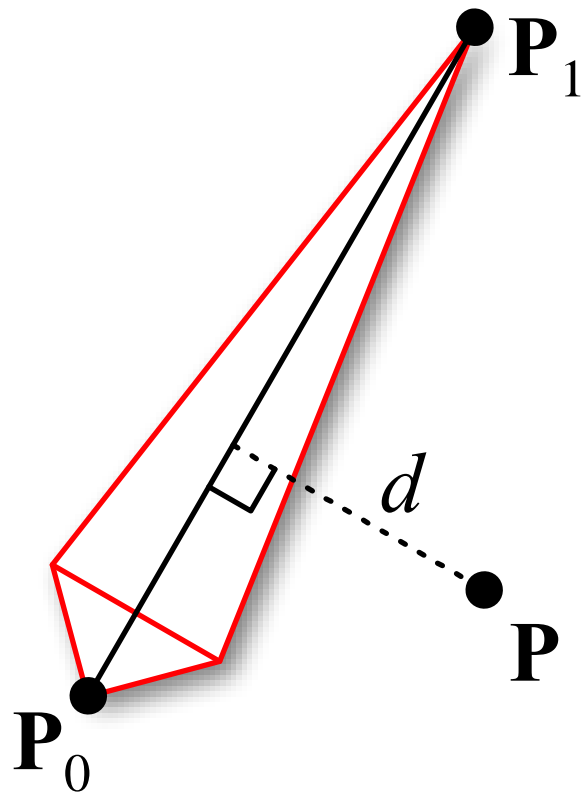


ボーンの変形

頂点位置の合成



頂点のボーンからの距離を重みに使う



- 重み

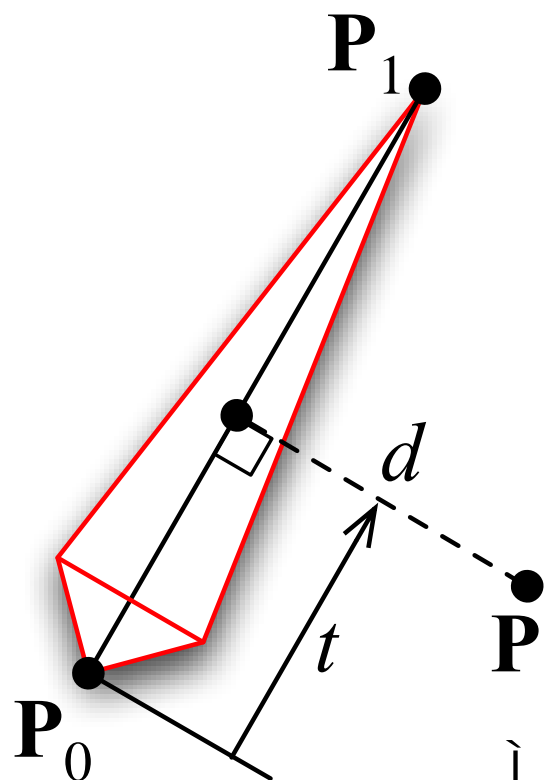
$$w = \frac{1}{(d + 1)^c}$$

- c が大きい場合
 - d の増加に対して重み w が急激に小さくなる
 - すなわち近い方のボーンの影響が支配的になる
- n 個のボーンに対して

$$\mathbf{u}(t) = \sum_{i=0}^{n-1} w_i \mathbf{B}(t) \mathbf{M}_i^{-1} \mathbf{P}$$

$$\sum_{i=0}^{n-1} w_i = 1, w_i \geq 0$$

ボーンと頂点の距離



$$\mathbf{V}_1 = \mathbf{P}_1 - \mathbf{P}_0$$

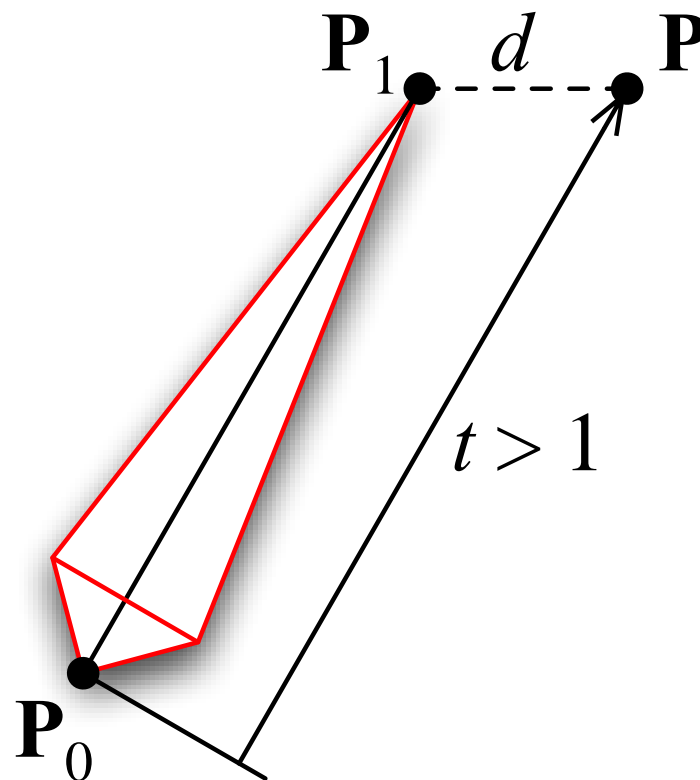
$$\mathbf{V}_2 = \mathbf{P} - \mathbf{P}_0$$

$$t = \frac{\mathbf{V}_1 \times \mathbf{V}_2}{\mathbf{V}_1^2}$$

$$d = |\mathbf{V}_2 - \mathbf{V}_1 t|$$



$$d = \begin{cases} |\mathbf{V}_2| & (t \leq 0) \\ |\mathbf{V}_2 - \mathbf{V}_1 t| & (0 < t < 1) \\ |\mathbf{V}_2 - \mathbf{V}_1| & (t \geq 1) \end{cases}$$



t を $[0,1]$ の範囲で
クランプすればよい

バーテックスシェーダによるブレンディング

```
#version 150 core
in vec4 position;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
```

```
// 点の位置 (ローカル座標)
// モデルビュー変換行列
// 投影変換行列
```

```
// ボーンのデータ
```

```
const int MAXBONES = 8;
uniform int numberOfBones;
uniform vec4 p0[MAXBONES];
uniform vec4 p1[MAXBONES];
uniform mat4 blendMatrix[MAXBONES];
const float exponent = -16.0;
```

```
// ボーンの数
// ボーンの根元の位置 (ワールド座標)
// ボーン先端の位置 (ワールド座標)
//  $B_i * \text{inverse}(M_i)$  (CPUで計算しておく)
// 重みの指数
```

```
void main()
```

```
{
```

```
    vec4 p = modelViewMatrix * position, u = vec4(0.0);
```

```
    for (int i = 0; i < numberOfBones; ++i) {
```

```
        vec4 v1 = p1[i] - p0[i], v2 = p - p0[i];
```

```
        float l = dot(v1, v1);
```

```
        if (l > 0.0) v2 -= v1 * clamp(dot(v1, v2) / l, 0.0, 1.0);
```

```
        u += pow(length(v2) + 1.0, exponent) * blendMatrix[i] * p; // 重み付け和
```

```
    }
```

```
    gl_Position = projectionMatrix * u;
```

```
}
```

OpenGL 3.2 (GLSL 1.5) 以降なら
inverse という組み込み関数がある

[0,1] でクランプ

重み w_i

p が同次座標なら重み w_i を
正規化する必要はない

今回出てきた GLSL の組み込み関数

- `sin(x)`, `cos(x)`
 - 三角関数, x (radian) の正弦, 余弦
- `pow(x, y)`
 - 指数関数, x^y
- `mix(v1, v2, t)`
 - $v1$ と $v2$ を t で比例配分, $v1 * (1 - t) + v2 * t$
- `dot(v1, v2)`
 - ベクトル $v1$, $v2$ の内積, 外積は `cross(v1, v2)`
- `length(v)`
 - v の絶対値／長さ
- `clamp(v, min, max)`
 - クランプ, $v < \min$ なら \min , $v > \max$ なら \max , それ以外は v

課題

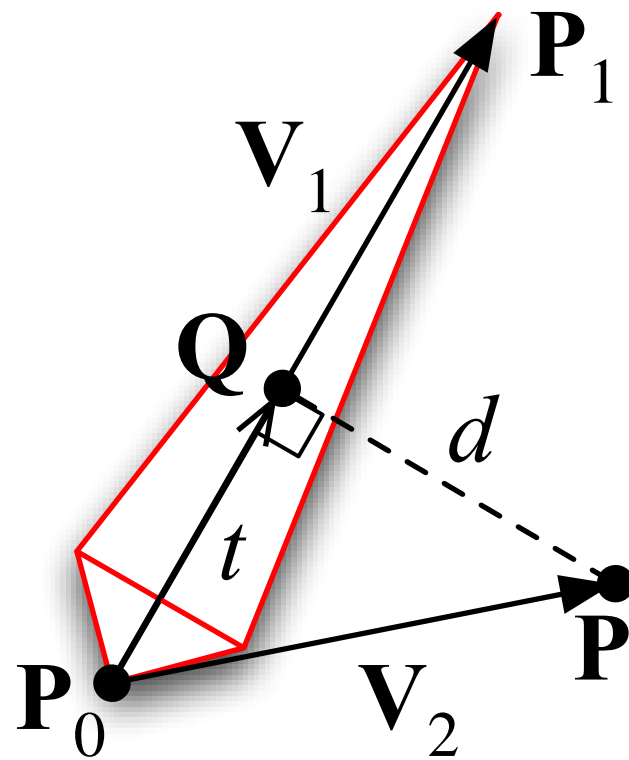
- 2点 $\mathbf{P}_0, \mathbf{P}_1$ を通る直線と点 \mathbf{P} との距離 d が次式により求められることを示しなさい

$$\mathbf{V}_1 = \mathbf{P}_1 - \mathbf{P}_0$$

$$\mathbf{V}_2 = \mathbf{P} - \mathbf{P}_0$$

$$t = \frac{\mathbf{V}_1 \times \mathbf{V}_2}{\mathbf{V}_1^2}$$

$$d = |\mathbf{V}_2 - \mathbf{V}_1 t|$$



宿題

- アニメーションにモーフィングによる変形を加えてください
 - 次のプログラムは線画の多角柱を回転しながら平行移動するアニメーション（前回の宿題の実装による）を表示します
 - <https://github.com/tokoik/ggsample05>
 - この点データは cylinder.h で定義されている p0 に格納されています
 - これをアニメーションにともなって p1 の点データの形に変形するようにしてください
 - ggsample05.cpp で p1 も GPU に送り, ggsample05.vert でモーフィングを実現してください
- ggsample05.cpp と ggsample05.vert を**アップロード**してください

結果

このような画像が表示されれば、多分、正解です。

