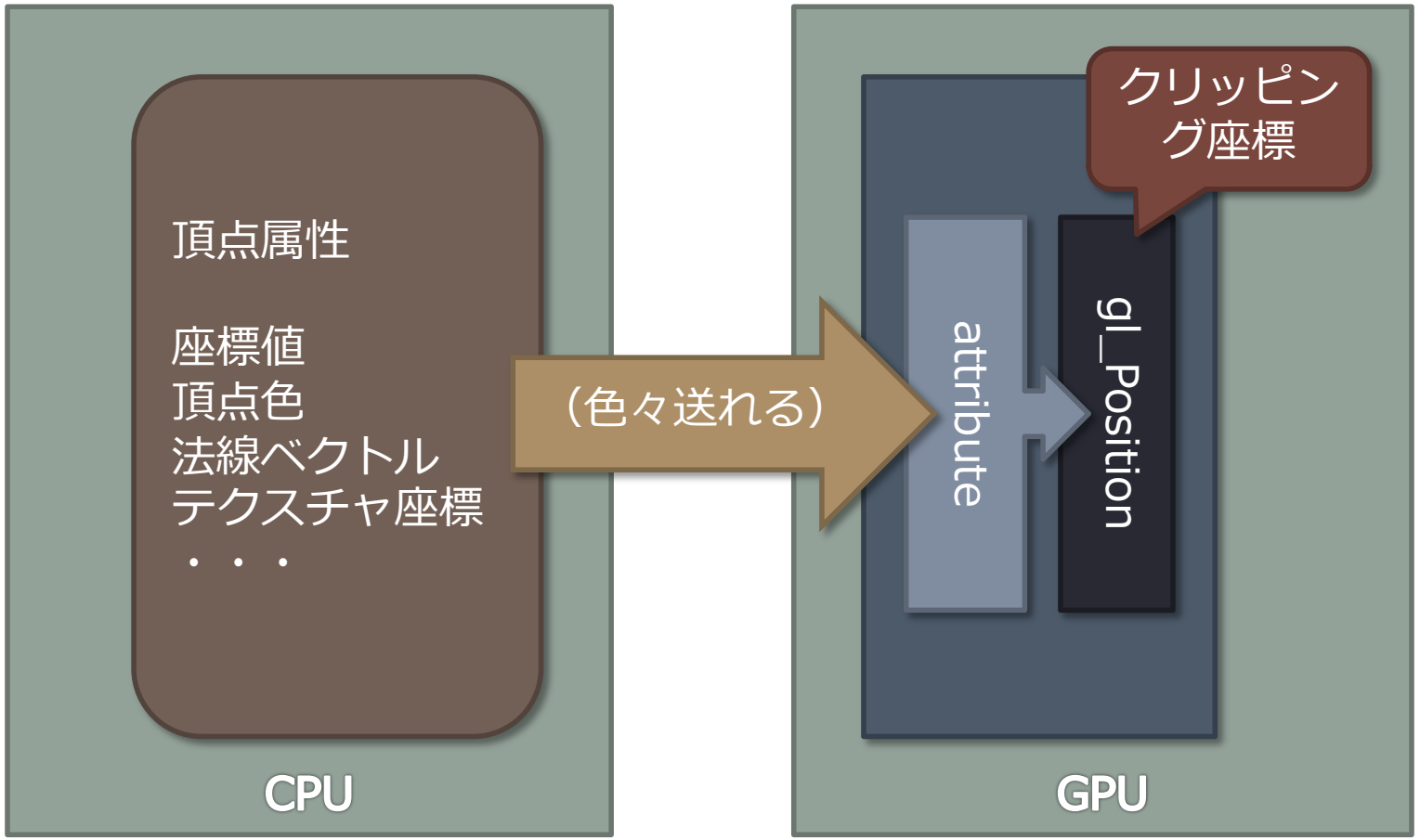
ゲームグラフィックス特論

第5回 変換 (3)

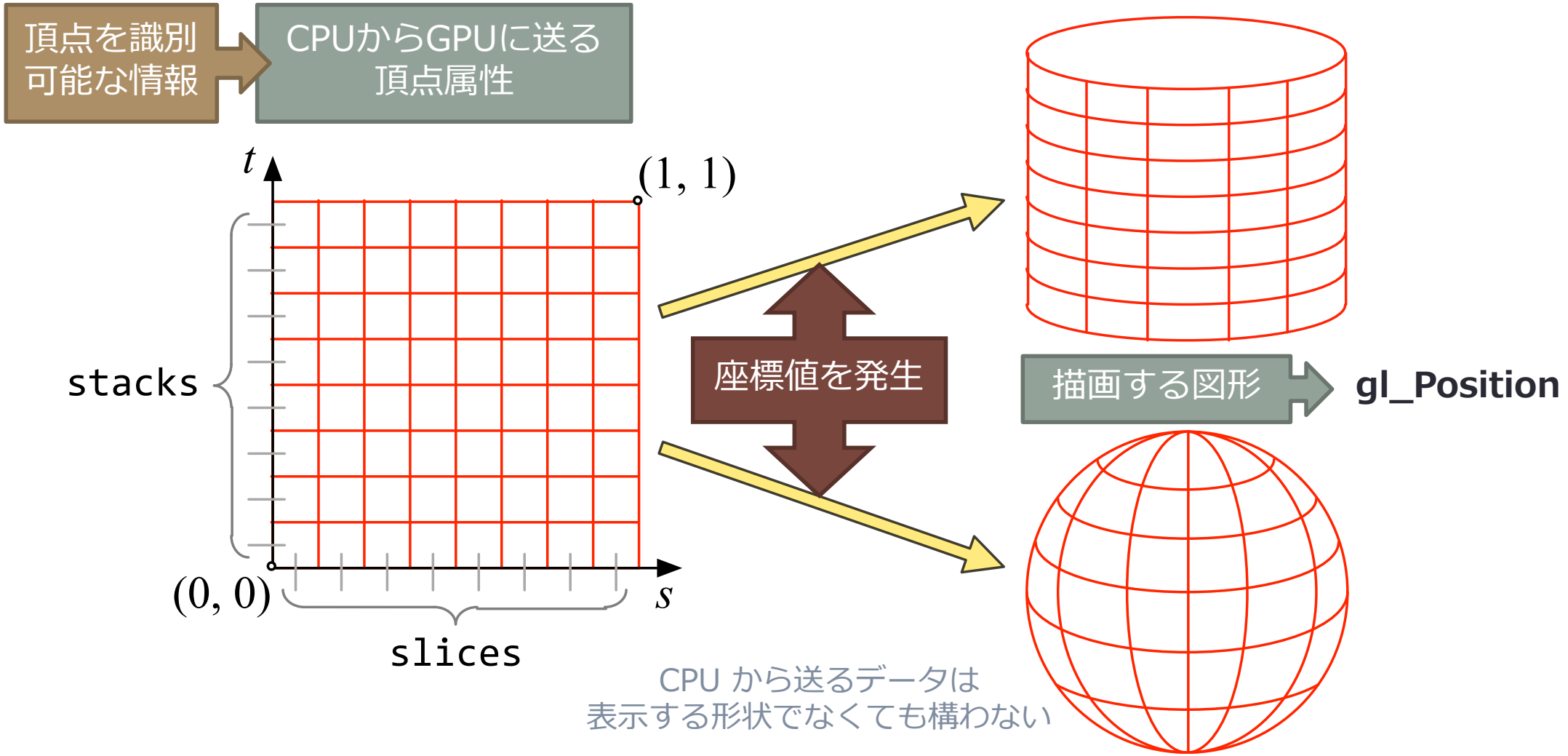
パラメータによる図形描画

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


頂点属性
座標値
頂点色
法線ベクトル
テクスチャ座標
・・・
CPU
（色々送れる）
attribute
gl_Position
クリッピング座標
GPU

パラメータを頂点属性として使って描画



パラメータの頂点属性の作成

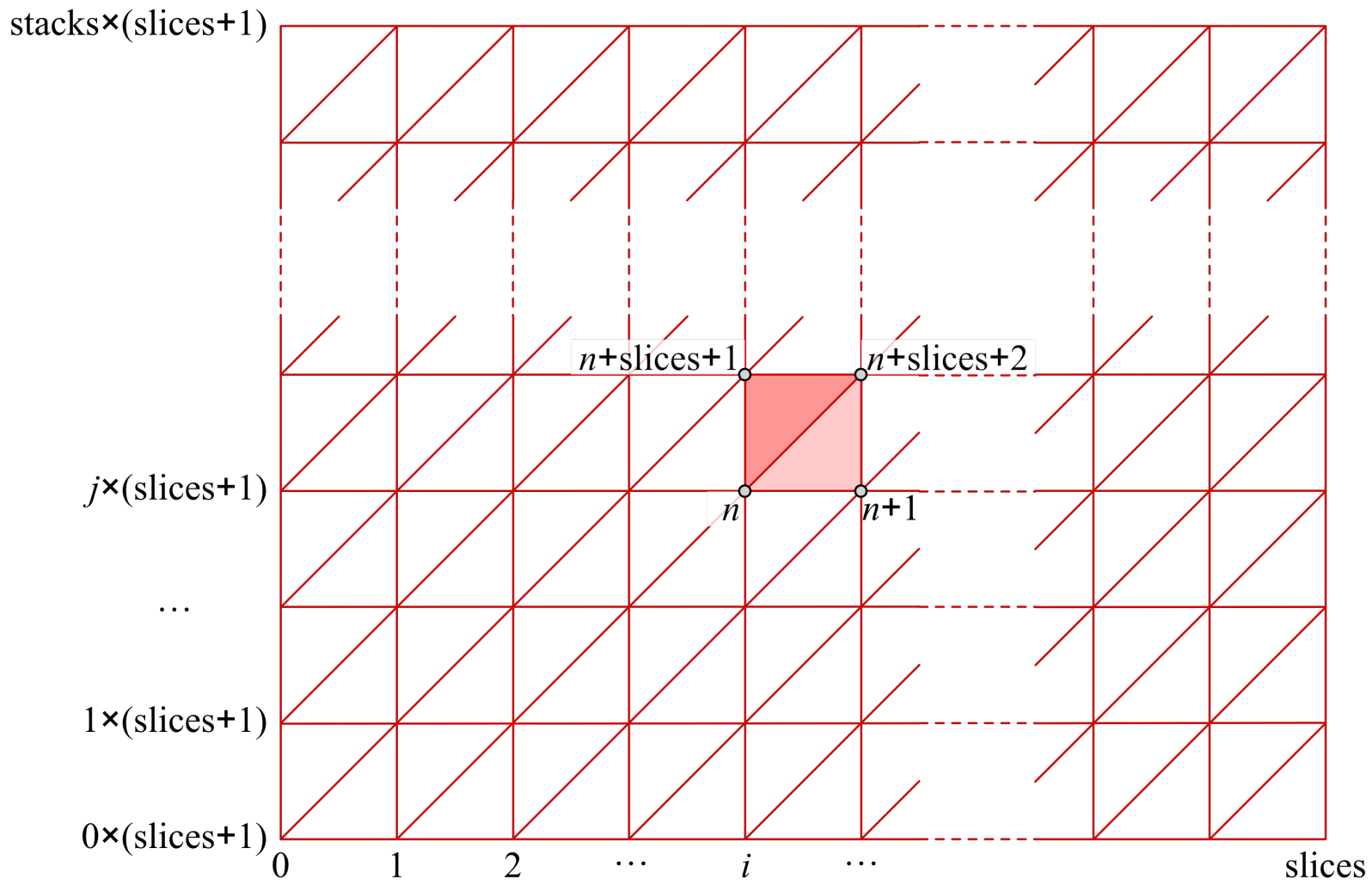
```
// パラメータの配列
GLfloat parameter[stacks + 1][slices + 1][2];

for (int j = 0; j <= stacks; ++j)
{
  // t∈[0, 1]
  const auto t{ static_cast<GLfloat>(j) / static_cast<GLfloat>(stacks) };

  for (int i = 0; i <= slices; ++i)
  {
    // s∈[0, 1]
    const auto s{ static_cast<GLfloat>(i) / static_cast<GLfloat>(slices) };

    parameter[j][i][0] = s;
    parameter[j][i][1] = t;
  }
}
```

頂点配列オブジェクトの準備

第２回と同じ

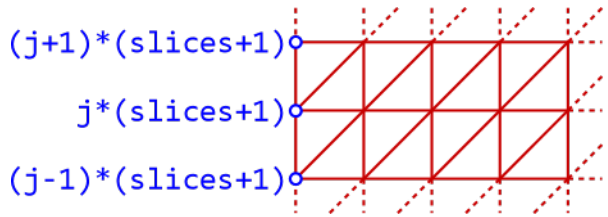
```
// 頂点配列オブジェクトを作成する
GLuint vao;
glGenVertexArrays(1, &vao);
glBindVertexArray(vao);

// 頂点バッファオブジェクトを作成する
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof parameter, parameter, GL_STATIC_DRAW);

// 結合されている頂点バッファオブジェクトを in 変数から参照する
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
```

頂点インデックス $n = j \times (\text{slices} + 1) + i$

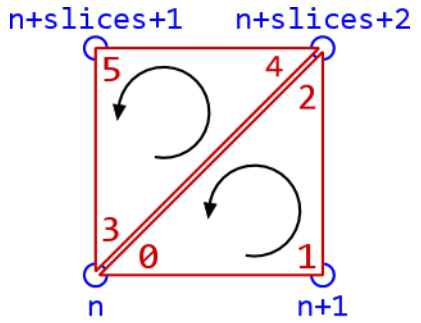


GL_TRIANGLES 用の頂点インデックスの作成

```
// 頂点インデックスの数（＝描画する頂点の数）
const auto size{ slices * stacks * 3 * 2 };
                 マス目の数        三角形が２枚

// 頂点インデックスの配列
GLuint index[size];

// 頂点インデックスの格納先
int count = 0;

for (int j = 0; j < stacks; ++j)
{
  // 左端の頂点インデックス
  const auto m{ j * (slices + 1) };
```




```
  for (int i = 0; i < slices; ++i)
  {
    // 頂点インデックス
    const auto n{ m + i };

    // 下側の三角形
    index[count++] = n;
    index[count++] = n + 1;
    index[count++] = n + slices + 2;

    // 上側の三角形
    index[count++] = n;
    index[count++] = n + slices + 2;
    index[count++] = n + slices + 1;
  }
}
```



頂点インデックスのバッファオブジェクト (IBO) の追加

```
// 頂点インデックスのバッファオブジェクト
GLuint ibo;
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof index, index, GL_STATIC_DRAW);
```

glDrawElements() による描画

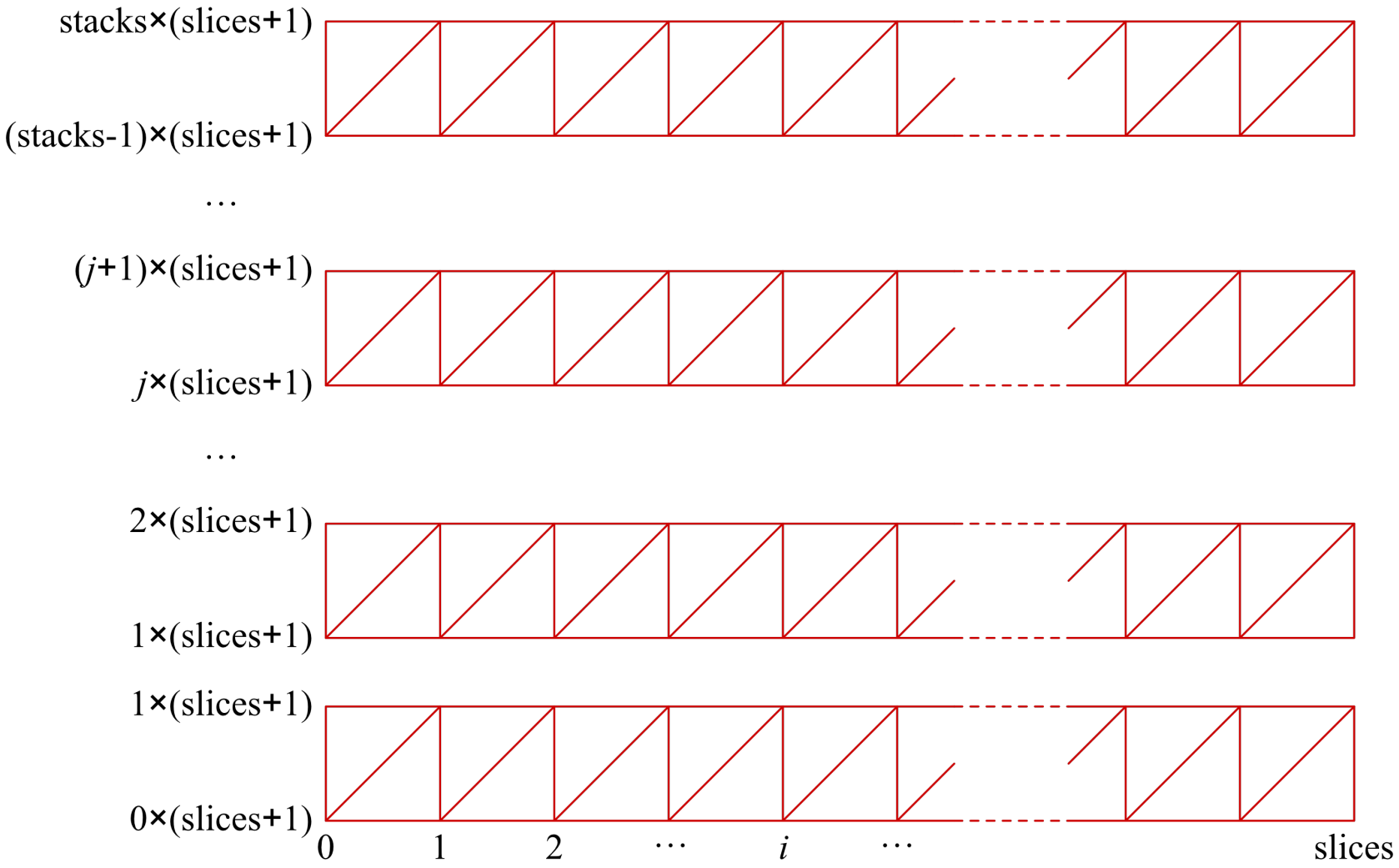
```
// シェーダプログラムの選択
glUseProgram(program);

// 図形を描画する
glBindVertexArray(vao);
glDrawElements(GL_TRIANGLES, size, GL_UNSIGNED_INT, 0);
```

IBO の要素数

IBO のデータ型

IBO の先頭

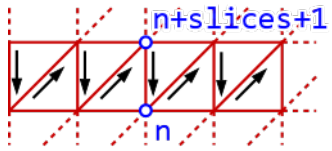
GL_TRIANGLE_STRIP で描くときの頂点インデックス



GL_TRIANGLE_STRIP 用の頂点インデックスの作成

```
// １つの STRIP の頂点数
const auto strip_count{ (slices + 1) * 2 };

// 頂点インデックスの数
const auto size{ strip_count * stacks };

// 頂点インデックスの配列
GLuint index[size];

// 頂点インデックスの格納先
int count = 0;

for (int j = 0; j < stacks; ++j)
{
  // 左端の頂点インデックス
  const auto m{ j * (slices + 1) };
```

```
  for (int i = 0; i <= slices; ++i)
  {
    // 頂点インデックス
    const auto n{ m + i };

    // STRIP の上側
    index[count++] = n + slices + 1;

    // STRIP の下側
    index[count++] = n;
  }
}
```



頂点インデックスのバッファオブジェクト (IBO) の追加

```
// 頂点インデックスのバッファオブジェクト
GLuint ibo;
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof index, index, GL_STATIC_DRAW);
```

glDrawElements() による描画

```
// シェーダプログラムの選択
glUseProgram(program);

// 図形を描画する
glBindVertexArray(vao);
for (int j = 0; j < stacks; ++j)
{
  auto offset{ static_cast<const GLuint*>(0) + strip_count * j };
  glDrawElements(GL_TRIANGLE_STRIP, strip_count, GL_UNSIGNED_INT, offset);
}
```

1 つの STRIP の頂点数

IBO のデータ型

IBO の先頭からのオフセットをポインタに変換したもの

glMultiDrawElements() による描画

```
// 各ストリップの頂点数と頂点インデックスのバッファオブジェクト（IBO）上の位置
GLsizei counts[stacks];
GLuint* indices[stacks];
for (int j = 0; j < stacks; ++j)
{
  counts[j] = strip_count;
  indices[j] = static_cast<const GLuint*>(0) + strip_count * j;
}

// シェーダプログラムの選択
glUseProgram(program);

// 図形を描画する
glBindVertexArray(vao);
glMultiDrawElements(GL_TRIANGLE_STRIP, count, GL_UNSIGNED_INT, indices, stacks);
```

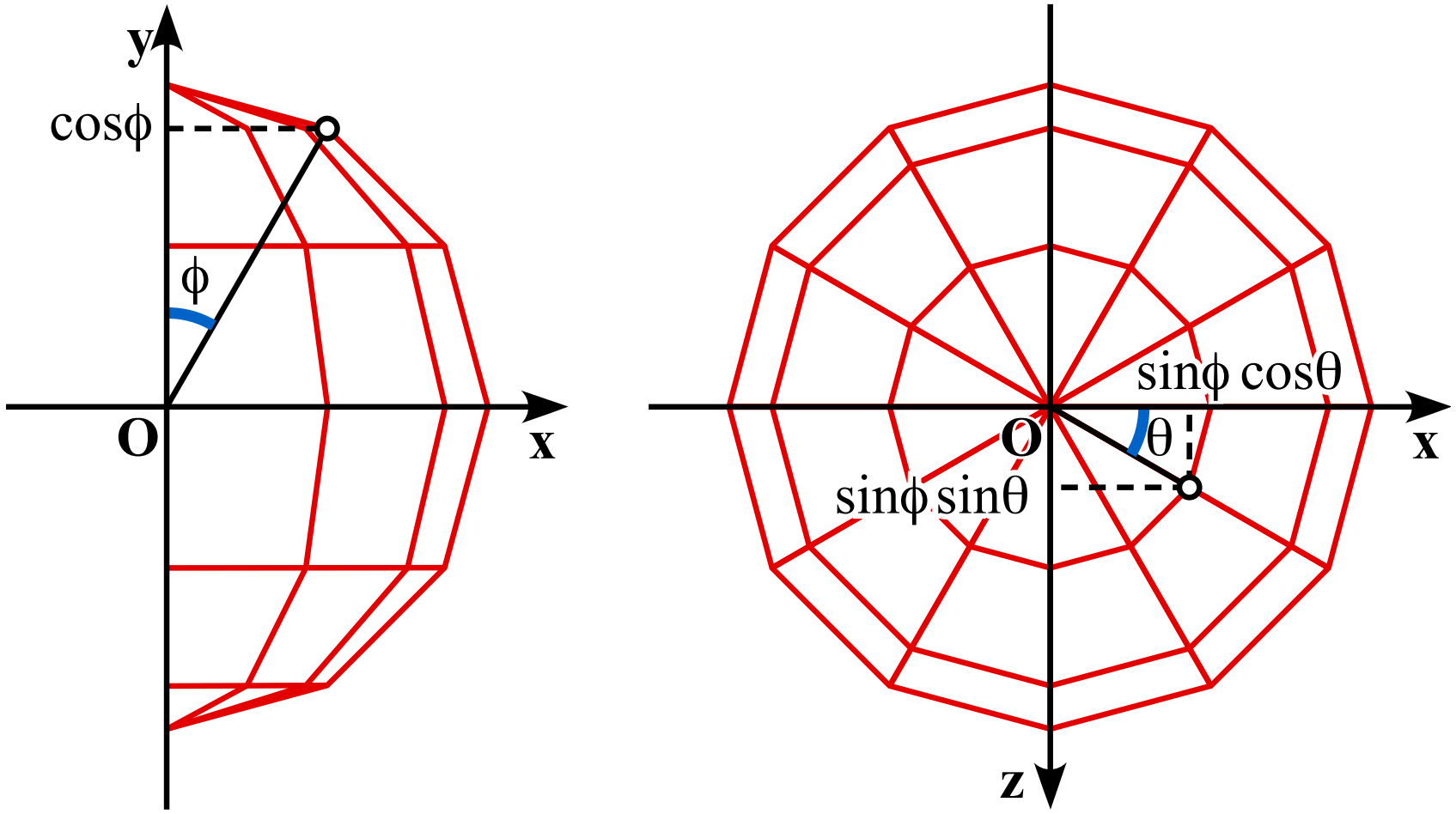
1 つの STRIP の頂点数

IBO の 1 つの STRIP のデータの格納場所

各 STRIP の頂点数

IBO のデータ型

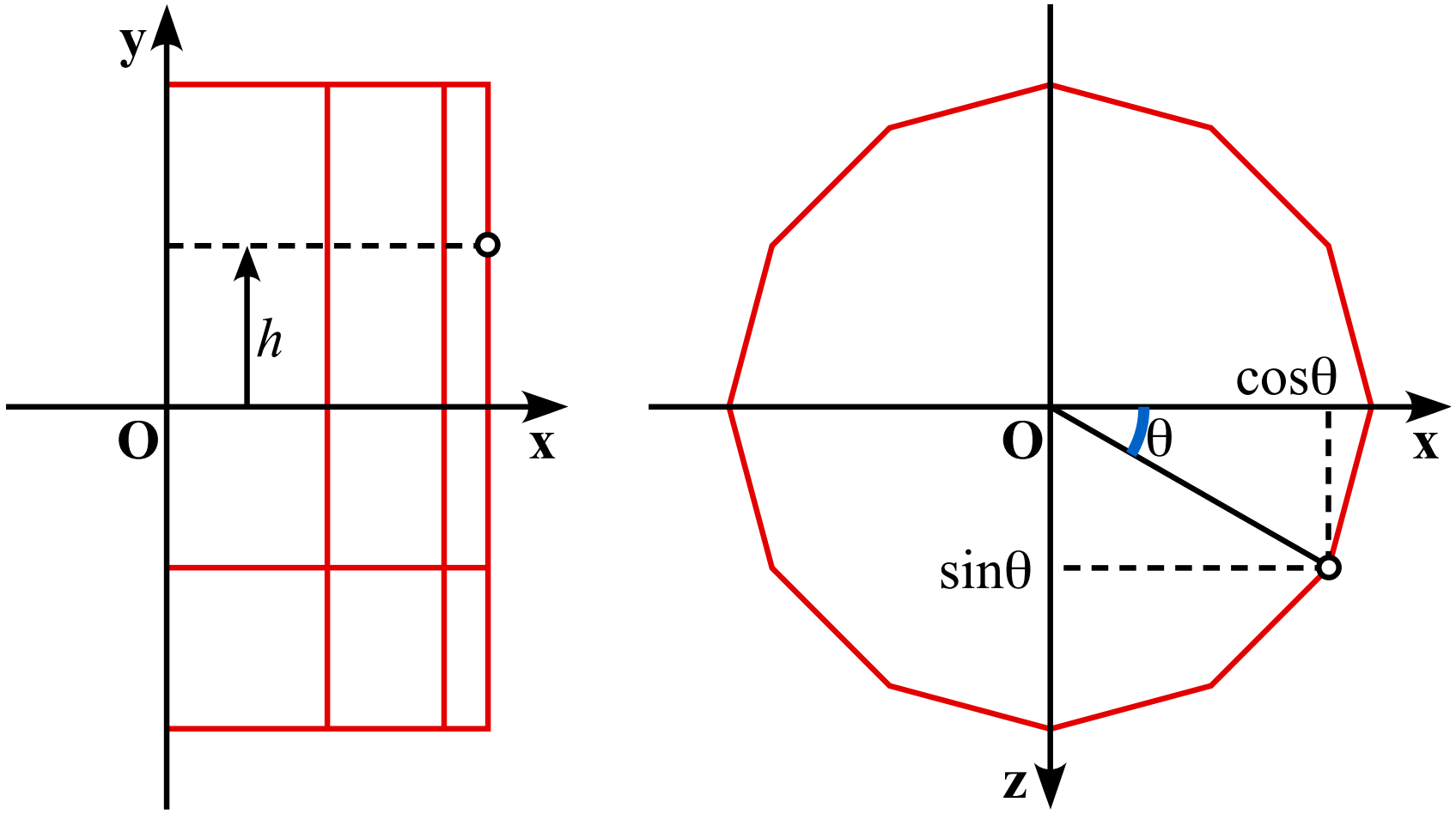
方位角・仰角と球面上の点の位置



バーテックスシェーダで球の座標生成

```
#version 410
in vec2 parameter;                          // CPU から与えられる頂点属性
uniform mat4 mc;                            // 変換行列

void main()
{
  float th = 6.283185 * parameter.s;        // 経度 [0,1]→[0,2π]
  float ph = 3.141593 * parameter.t;        // 緯度 [0,1]→[0,π]
  float r = sin(ph);                        // 緯度 ph における単位球の断面の半径

  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で円柱の座標生成

```
#version 410
in vec2 parameter;                          // CPU から与えられる頂点属性
uniform mat4 mc;                            // 変換行列

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

  vec4 p = vec4(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 uniform 変数に設定する

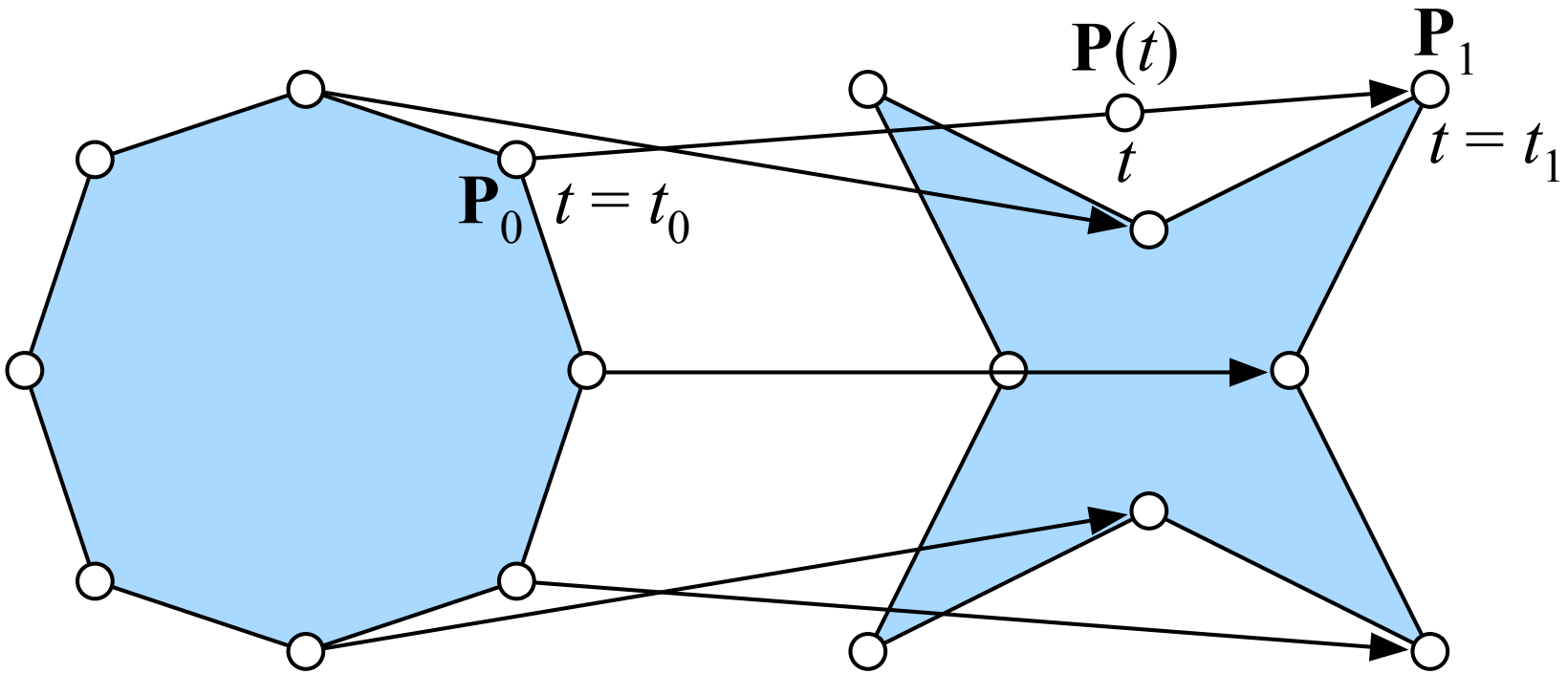
頂点ごと

- 局所的な変形
  - キャラクタの口パク
  - 手足の曲げのばし
- 頂点ごとに処理が異なる
  - 全部の頂点をCPUで変換してから描画する

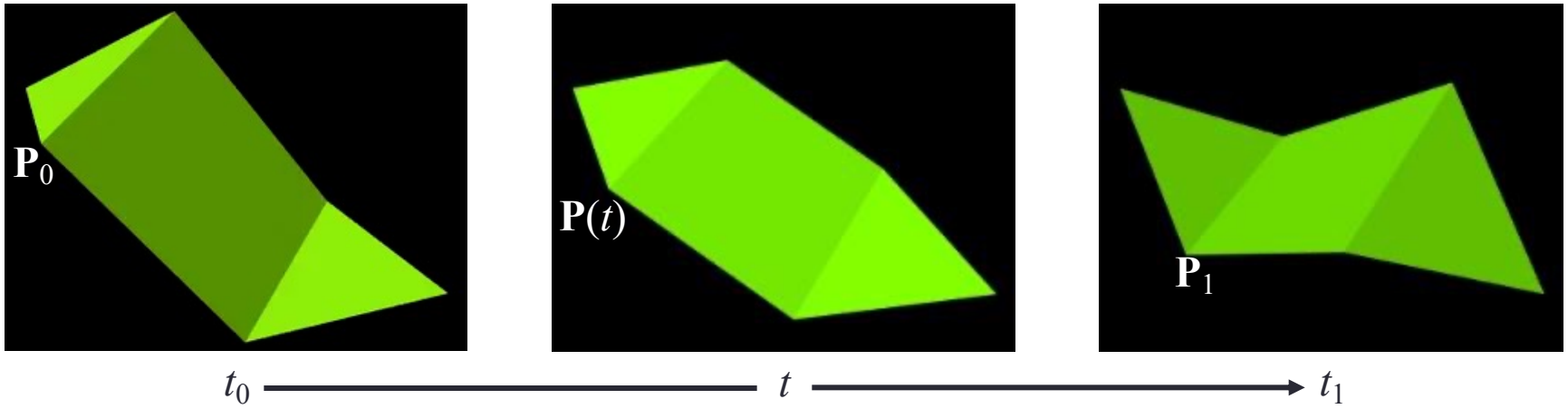
    （または）
  - 元になる頂点属性 (attribute) だけをGPUに送りシェーダプログラムで変換する

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／頂点インデックスはそのまま



頂点位置の線形補間



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

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

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

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

一つの頂点に複数の頂点属性 (attribute)

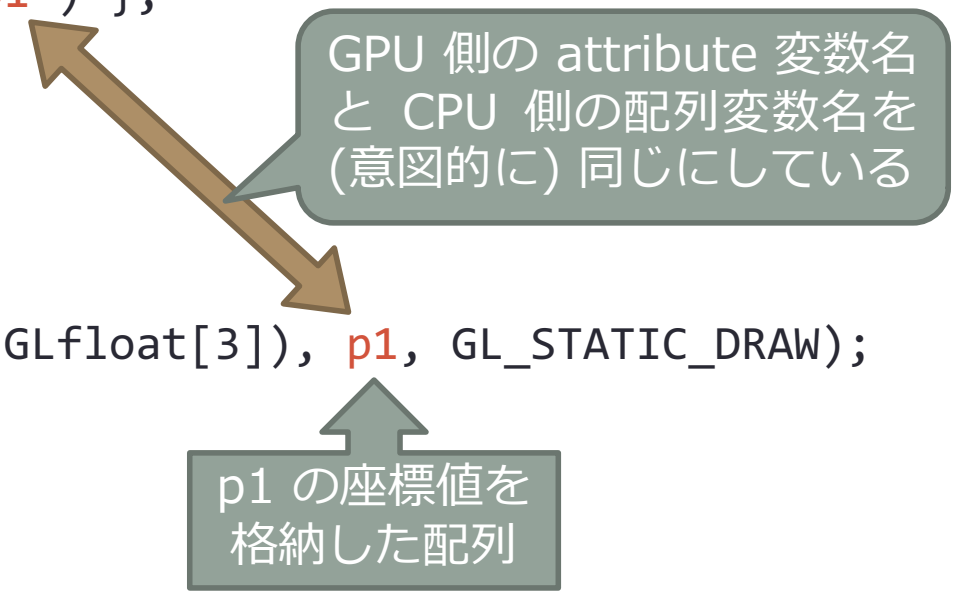
バーテックスシェーダによる補間

```
#version 410
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()
{
  gl_Position = mc * mix(p0, p1, t);
}
```

$(1-t)\mathbf{P}_0 + t\mathbf{P}_1$ 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 変数（attribute 変数）p1 のインデックスの検索（見つからなければ -1）
const auto 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, vertices * sizeof (GLfloat[3]), p1, GL_STATIC_DRAW);
```



頂点バッファオブジェクトの追加

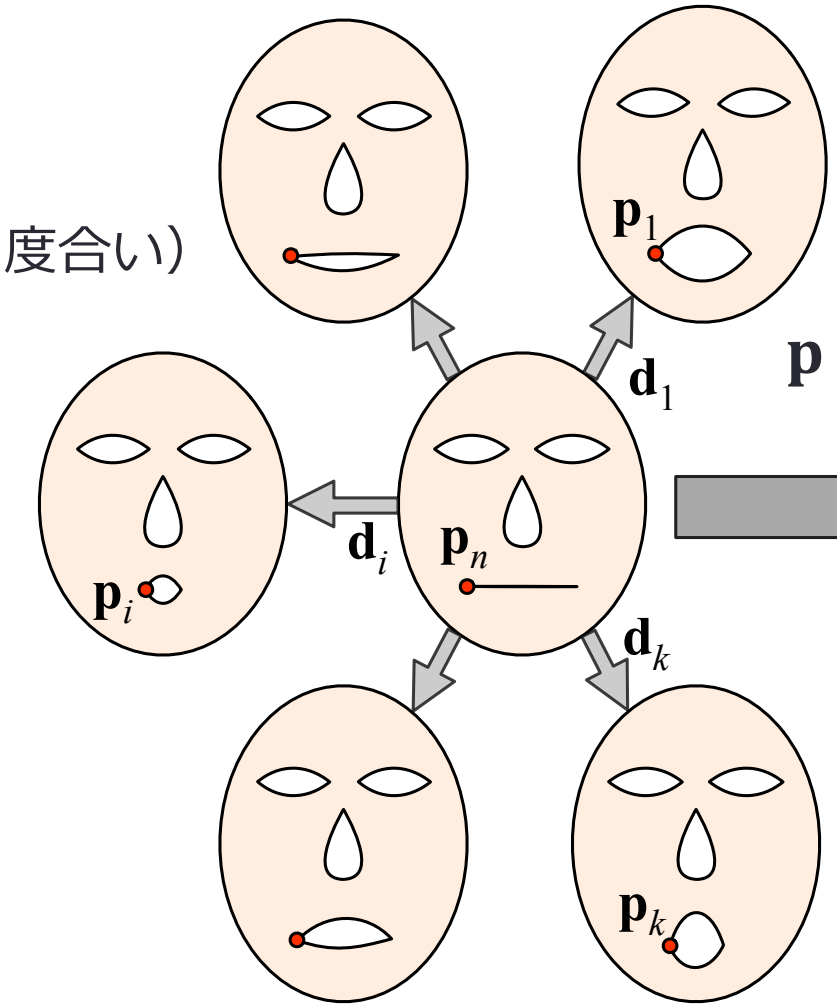
```
// 描画に使う頂点配列オブジェクトの指定
glBindVertexArray(vao);

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

// プログラムオブジェクトの使用開始
glUseProgram(program);

// 時刻の設定（tLoc は uniform 変数 t のインデックス）
glUniform1f(tLoc, t);

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

モーフターゲット

$\mathbf{p}_N$：元の形状（ニュートラル）
$\mathbf{p}_i$：変形後の形状
$\mathbf{d}_i$：形状の差分
w_i：重み（i番目の形状に近づく度合い）

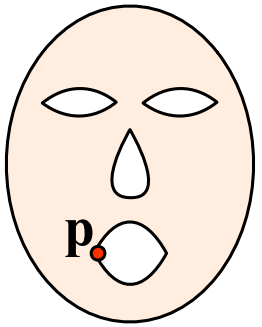
$$\mathbf{d}_i = \mathbf{p}_i - \mathbf{p}_N$$

変形後の各形状の元の形状からの差分を求めておく



元の形状に形状の差分の重み付け和を求める

$$\mathbf{p} = \mathbf{p}_N + \sum_{i=1}^{k} w_i \mathbf{d}_i$$



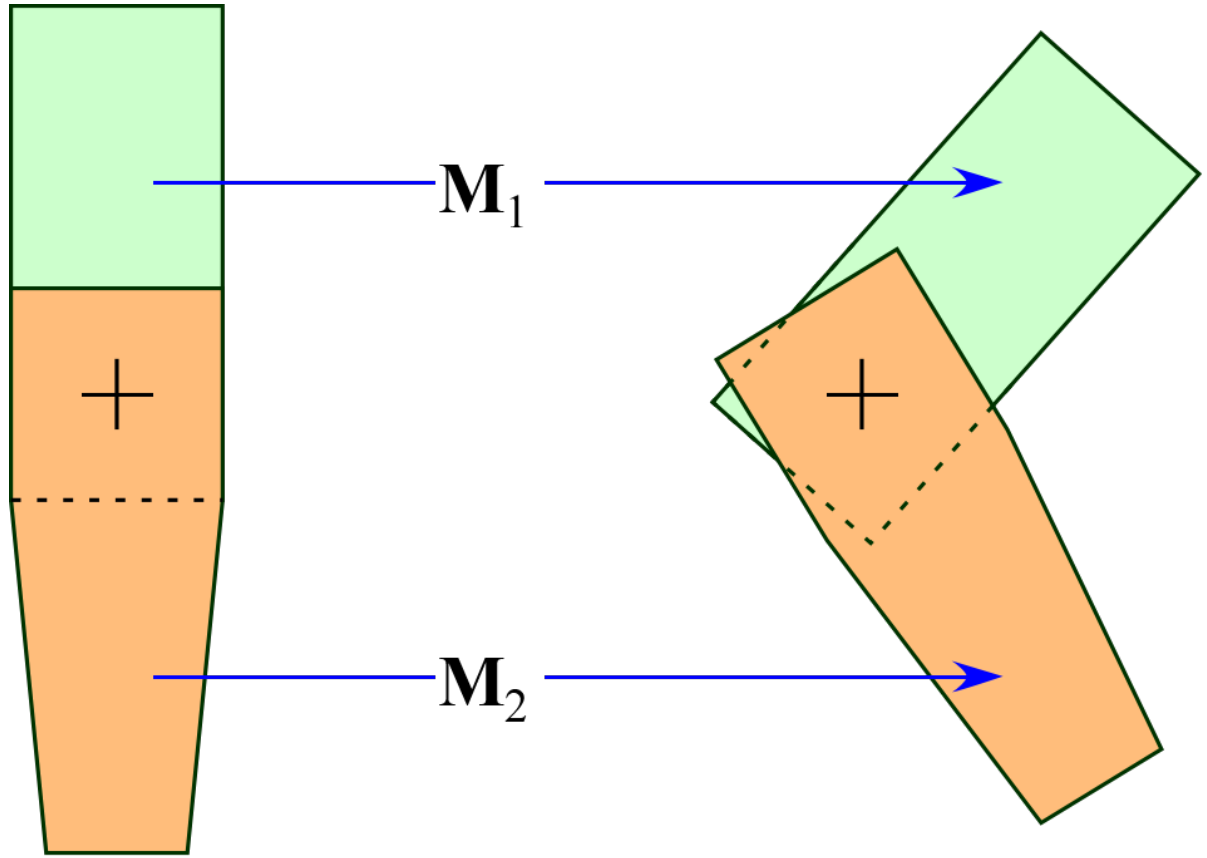
$$w_1 = 1, w_2 = w_3 = w_4 = w_5 = 0$$
$$\Downarrow$$
$$\mathbf{p} = \mathbf{p}_N + \mathbf{d}_1 = \mathbf{p}_1$$

バーテックスブレンディング

骨格による変形

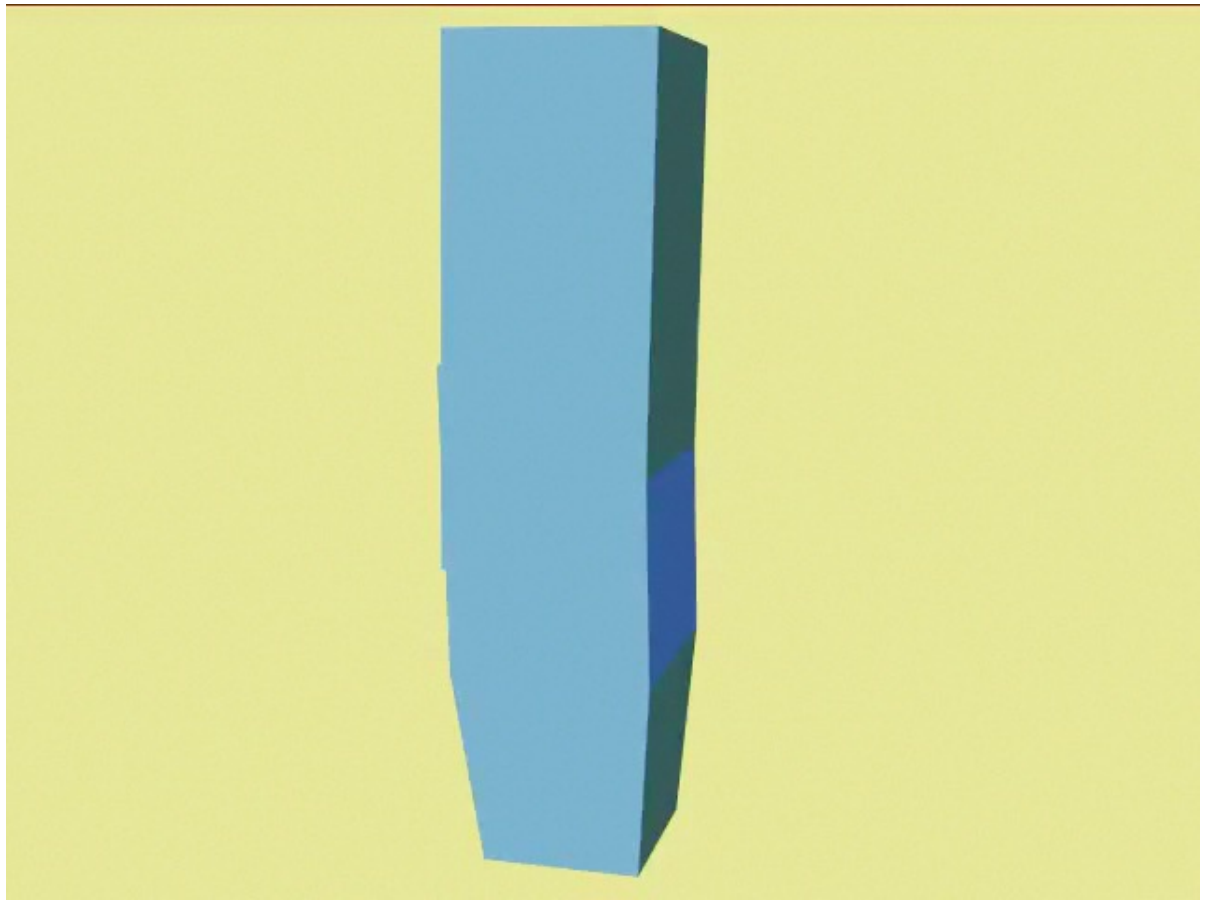
関節を動かす

- 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



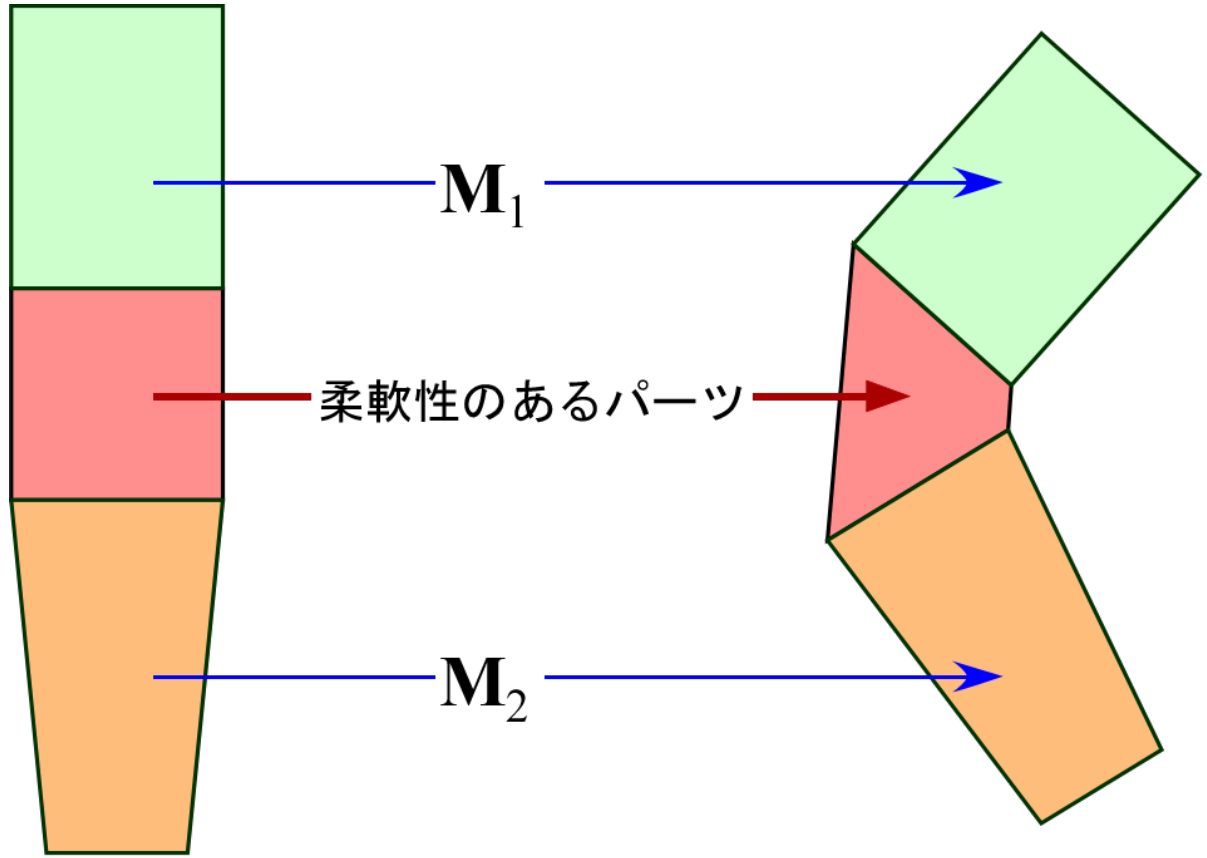
独立したパーツを用いる

- 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



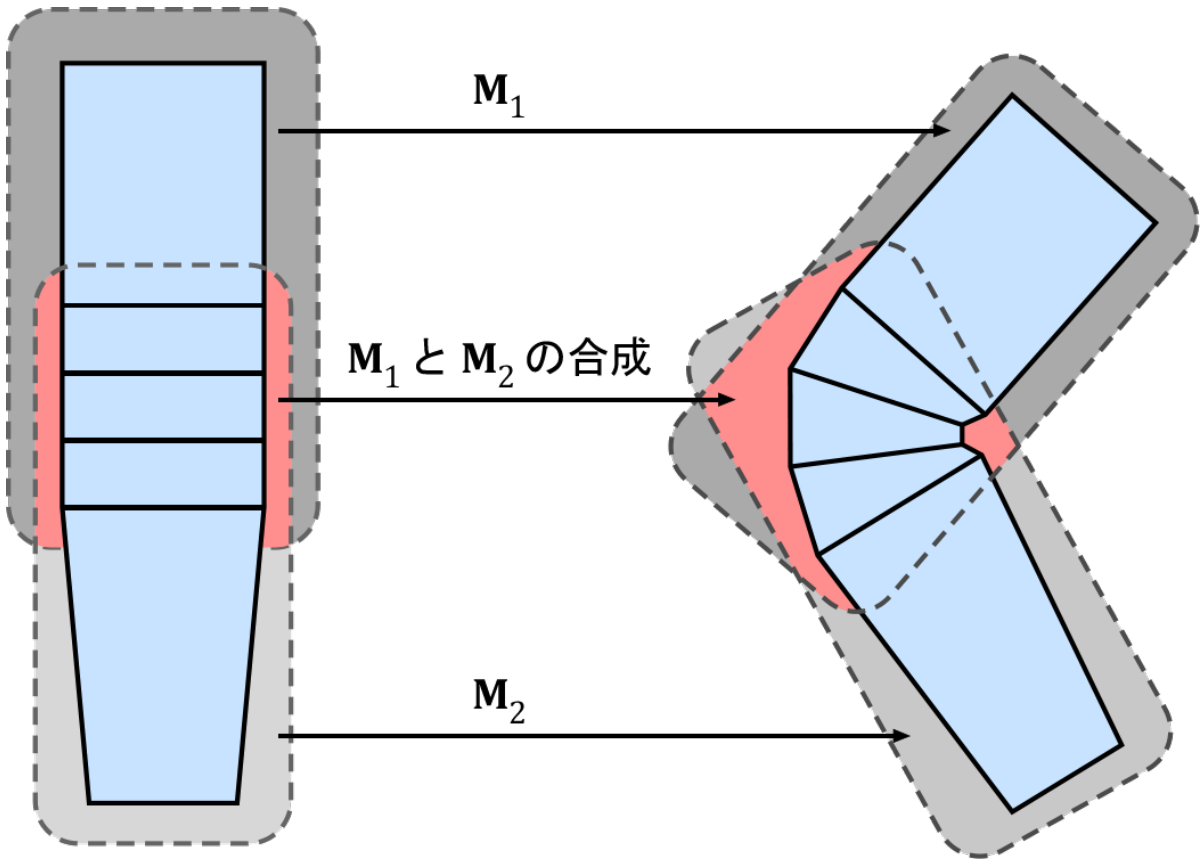
柔軟性のあるパーツを用いる

- 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

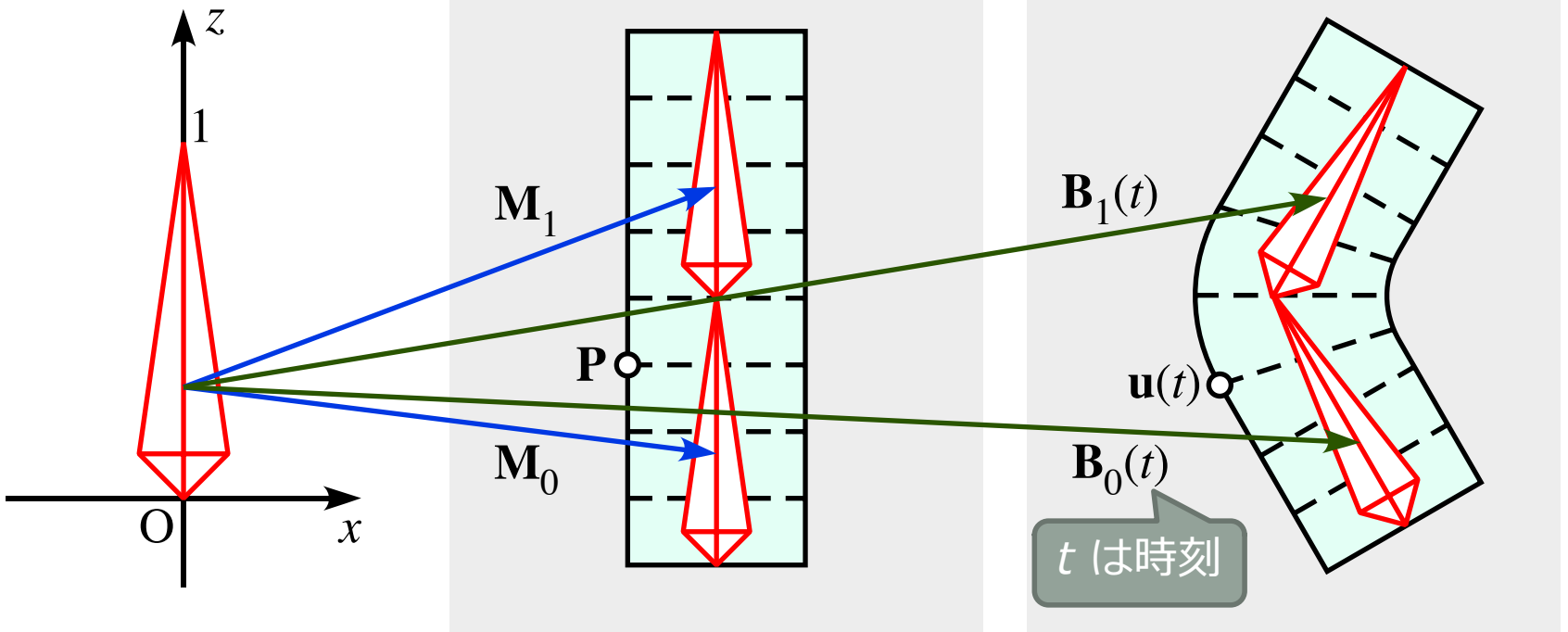
- 全体を柔軟性のある単一のパーツで表現する
- ２つの変換 $\mathbf{M}_1$，$\mathbf{M}_2$ の影響範囲を決める
- 影響範囲が重なる部分の頂点の座標値は $\mathbf{M}_1$，$\mathbf{M}_2$ による変換の結果を合成して求める



ボーンの配置と移動

$\mathbf{M}_0$, $\mathbf{M}_1$ 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

$\mathbf{B}_0(t)$, $\mathbf{B}_1(t)$ 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

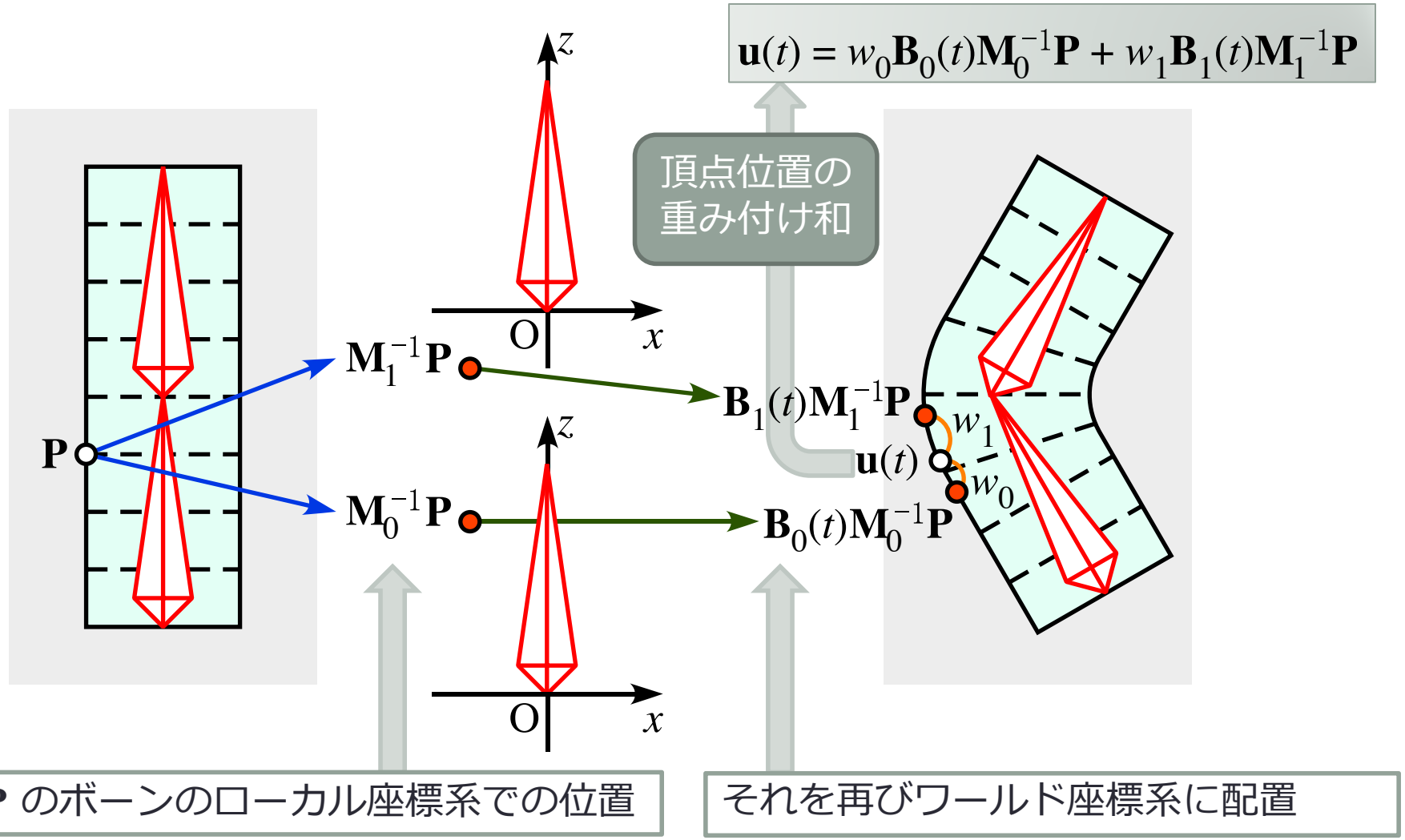


ボーン
（ローカル座標系）

ボーンの配置
（ワールド座標系）

ボーンの変形

頂点位置の合成


$\mathbf{u}(t) = w_0\mathbf{B}_0(t)\mathbf{M}_0^{-1}\mathbf{P} + w_1\mathbf{B}_1(t)\mathbf{M}_1^{-1}\mathbf{P}$
頂点位置の
重み付け和
P
$\mathbf{M}_1^{-1}\mathbf{P}$
$\mathbf{M}_0^{-1}\mathbf{P}$
$\mathbf{B}_1(t)\mathbf{M}_1^{-1}\mathbf{P}$
$\mathbf{B}_0(t)\mathbf{M}_0^{-1}\mathbf{P}$
$\mathbf{u}(t)$
w_1
w_0
z
x
O
P のボーンのローカル座標系での位置
それを再びワールド座標系に配置

頂点のボーンからの距離を重みに使う

- 重み

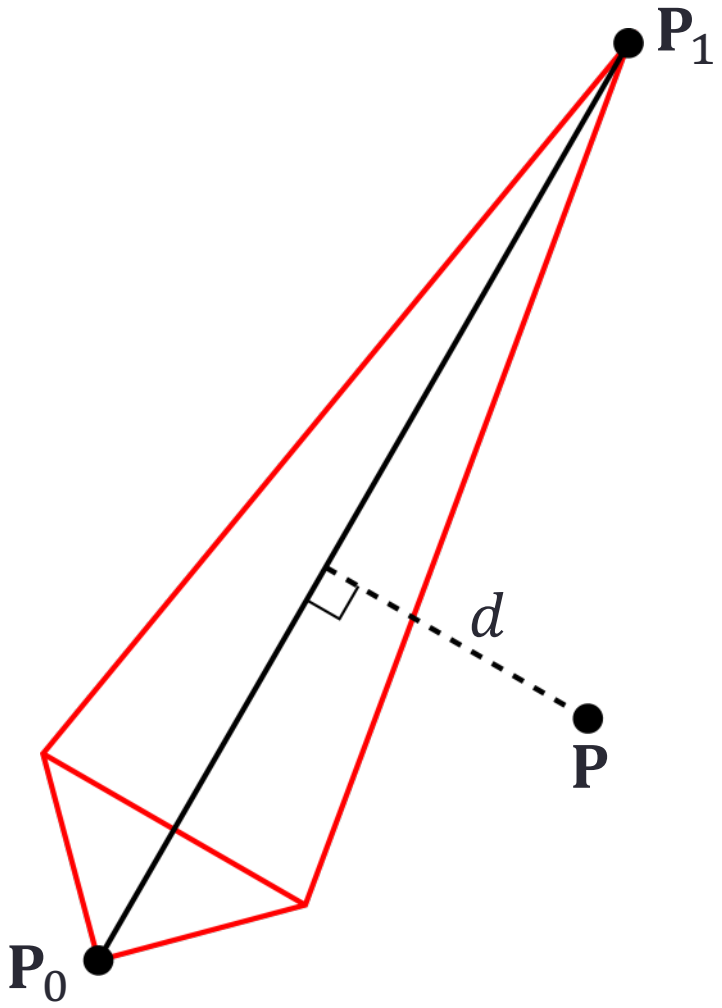
$$w = \frac{1}{(d+1)^c}$$

  - 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
    - そのため近いボーンの影響が支配的になる
  - n 個のボーンに対して

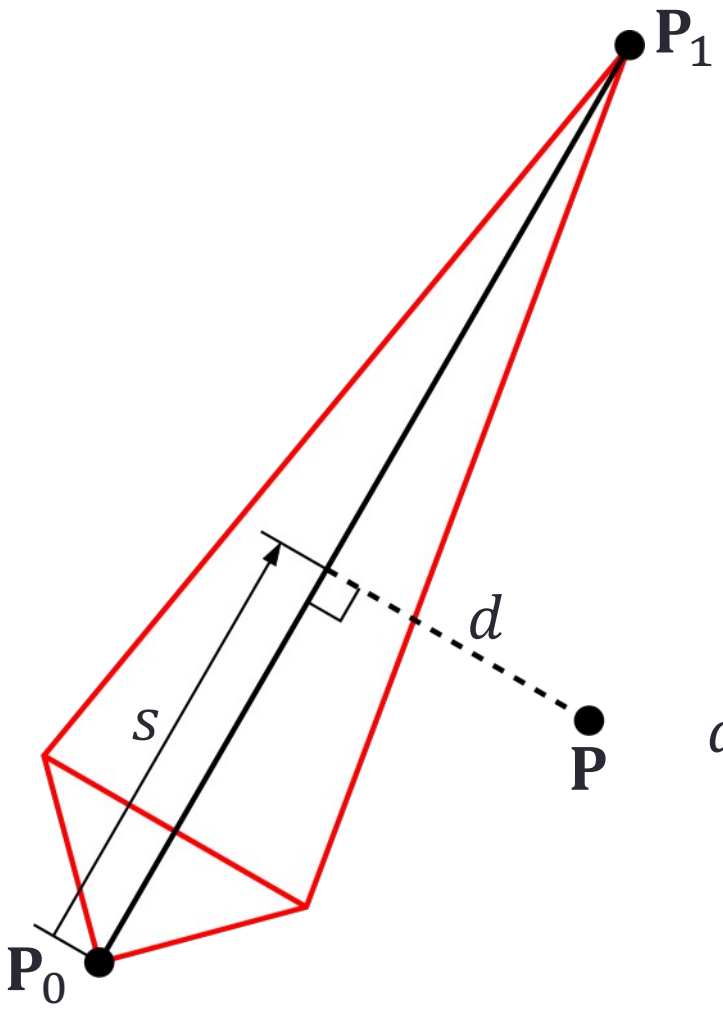
$$\mathbf{u}(t) = \sum_{i=0}^{n-1} w_i \, \mathbf{B}(t)\mathbf{M}_i^{-1}\mathbf{P}$$

    - ここで

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

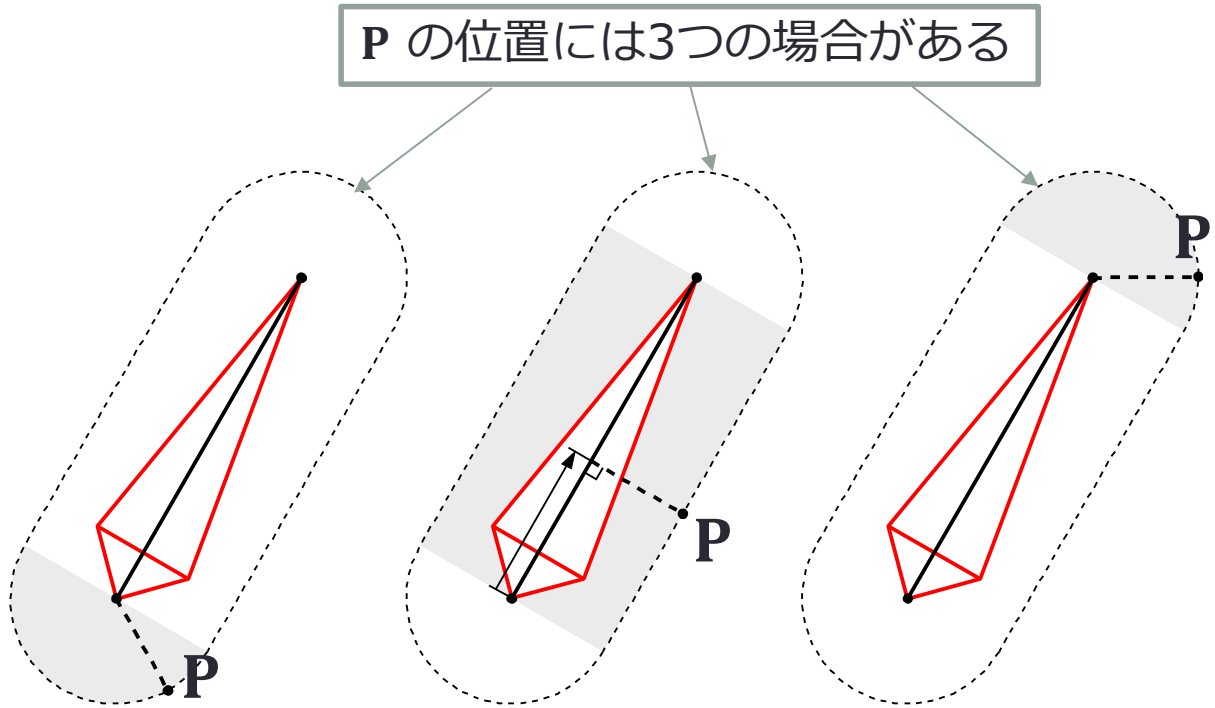


ボーンと頂点の距離



$$\mathbf{V}_1 = \mathbf{P}_1 - \mathbf{P}_0$$
$$\mathbf{V}_2 = \mathbf{P} - \mathbf{P}_0$$
$$s = \frac{\mathbf{V}_1 \cdot \mathbf{V}_2}{{\mathbf{V}_1}^2}$$

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



$(s < 0)$	$(0 \le s \le 1)$	$(s > 1)$
$\lvert\mathbf{V}_2\rvert$	$\lvert\mathbf{V}_2 - \mathbf{V}_1 s\rvert$	$\lvert\mathbf{V}_2 - \mathbf{V}_1\rvert$

s を $[0,1]$ の範囲でクランプすれば場合分け不要

バーテックスシェーダによるブレンディング

```
#version 410
in vec4 position;                       // 点の位置（ローカル座標）
uniform mat4 modelViewMatrix;           // モデルビュー変換行列
uniform mat4 projectionMatrix;          // 投影変換行列

// ボーンのデータ
const int MAXBONES = 8;                 // このプログラムで扱えるボーンの最大数
uniform int numberOfBones;              // 実際に処理するボーンの数
uniform vec4 p0[MAXBONES];              // ボーンの根元の位置（ワールド座標）
uniform vec4 p1[MAXBONES];              // ボーンの先端の位置（ワールド座標）
uniform mat4 blendMatrix[MAXBONES];     // Bi(t) * Mi-1 （CPU で計算しておく）
uniform float exponent = -16.0;         // 重みの指数，-c
```

uniform 変数に CPU から値を設定しなかった時の既定値

OpenGL 3.2 (GLSL 1.5) 以降なら GLSL に逆行列を求める組み込み関数 inverse がある

```
void main()
{
  vec4 p = modelViewMatrix * position;                    // ワールド座標系の頂点位置
  vec4 u = vec4(0.0);                                     // 重み付け和を求める変数

  // 全てのボーンについて
  for (int i = 0; i < numberOfBones; ++i) {
    vec4 v1 = p1[i] - p0[i];                              // v1 を求める

    // v1 が 0 ベクトルでなければ
    if (v1 != vec4(0.0)) {
      float s = dot(v1, p - p0[i]) / dot(v1, v1);         // s = v1・v2 / v1^2
      float d = length(v2 - v1 * clamp(s, 0.0, 1.0));     // d = |v2 - v1 * s|
      float wi = pow(d + 1.0, exponent);                  // wi = 1 / (d + 1)^-exponent
      u += wi * blendMatrix[i] * p;                       // u(t) = wi * Bi(t) * Mi^-1 * p
    }
  }

  // ブレンドした結果を頂点の座標値に使う
  gl_Position = projectionMatrix * u;
}
```

[0,1] でクランプ

重み付け和

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

小テスト – バーテックスブレンディング

Moodle の小テストに解答してください

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．