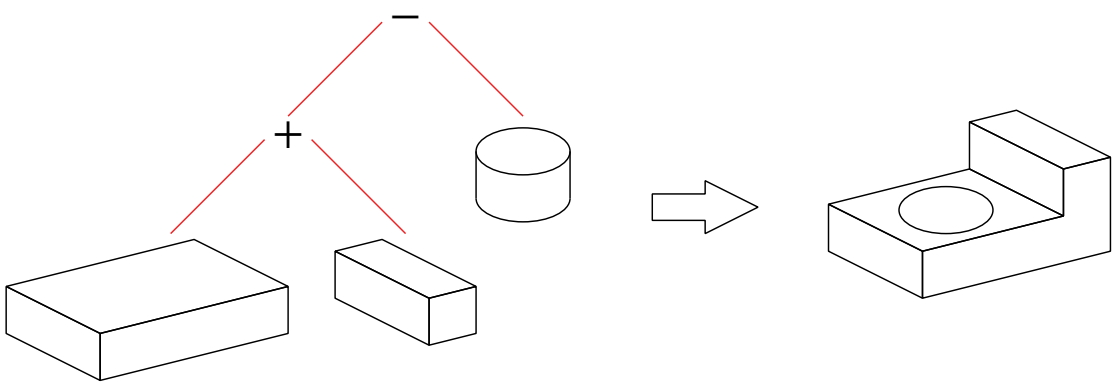
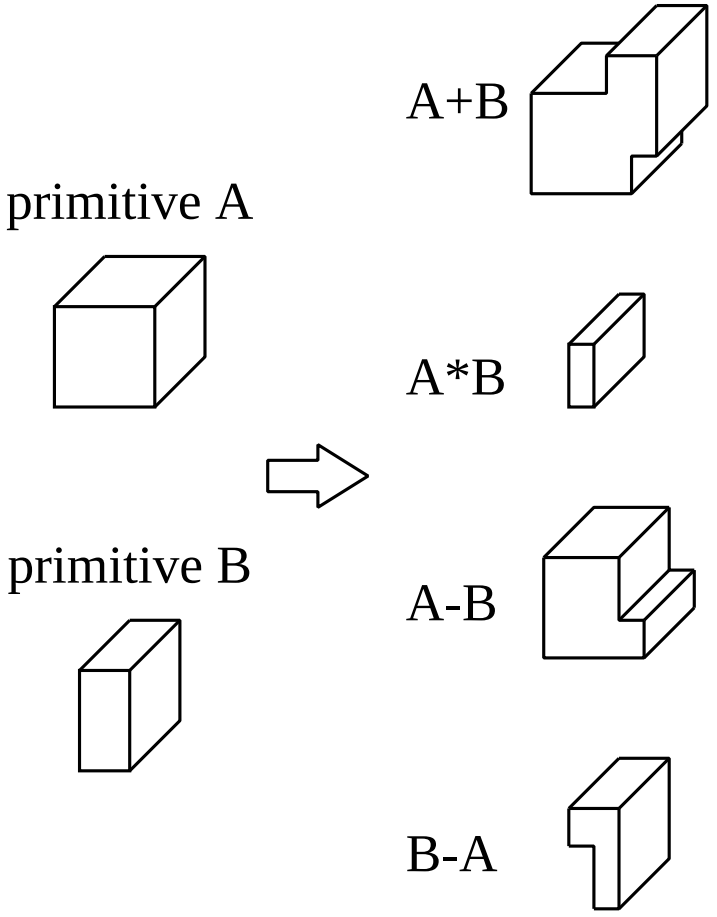
ソリッドモデルの表現法2

- **CSG表現**（Constructive Solid Geometry representations）

  基本立体（primitive）を**集合演算**（set operation）により組み合わせることで物体を表現.

  - 集合演算の例
    - **和**（union）
    - **積**（intersection）
    - **差**（difference）
  - 集合演算の組み合わせを木構造で表現.

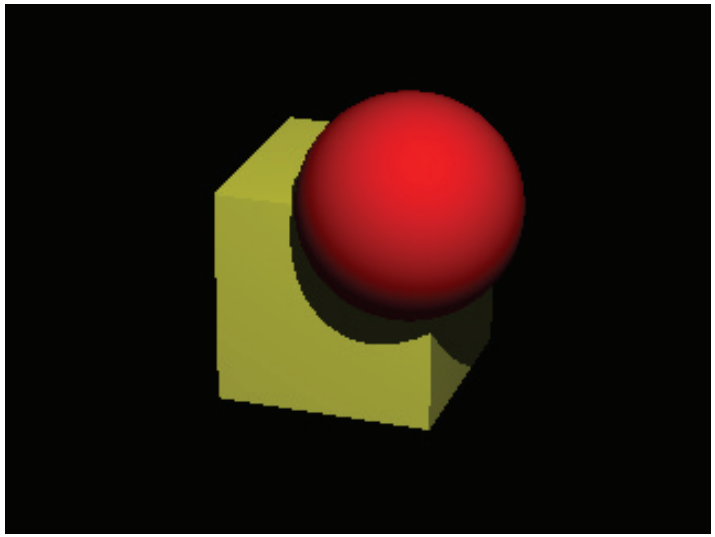




POV-Rayでの集合演算(1/3)

```
#include "colors.inc"
#include "shapes.inc"

camera {
  location <-10, 10, -20>
  look_at <0, 0, 0>
  angle 15
}

light_source { <-3, 10, -10> color 1.5*White }

union {
  object {
    Cube
    pigment { color Yellow }
    rotate 45*y
  }

  object {
    Sphere
    pigment { color Red }
    translate <0, 1, -1>
  }
}
```

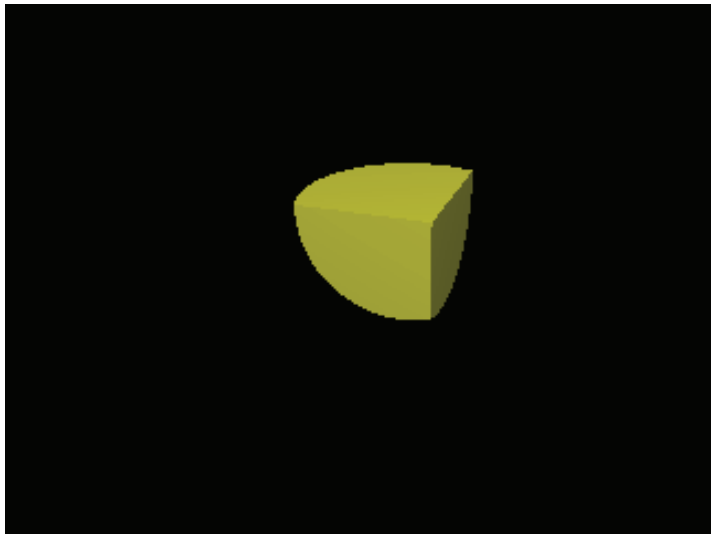
⇐ unionで2つのobjectの和



POV-Rayでの集合演算(2/3)

```
#include "colors.inc"
#include "shapes.inc"

camera {
  location <-10, 10, -20>
  look_at <0, 0, 0>
  angle 15
}

light_source { <-3, 10, -10> color 1.5*White }

intersection {
  object {
    Cube
    pigment { color Yellow }
    rotate 45*y
  }

  object {
    Sphere
    pigment { color Red }
    translate <0, 1, -1>
  }
}
```

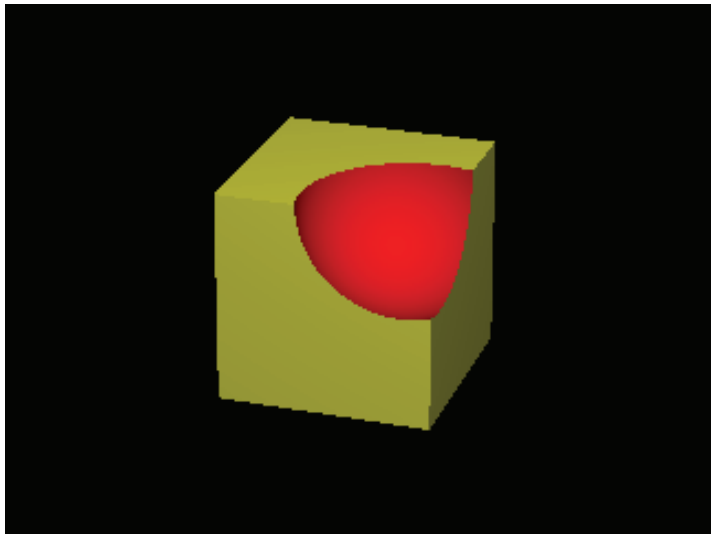
⇐ intersectionで2つのobjectの積



POV-Rayでの集合演算(3/3)

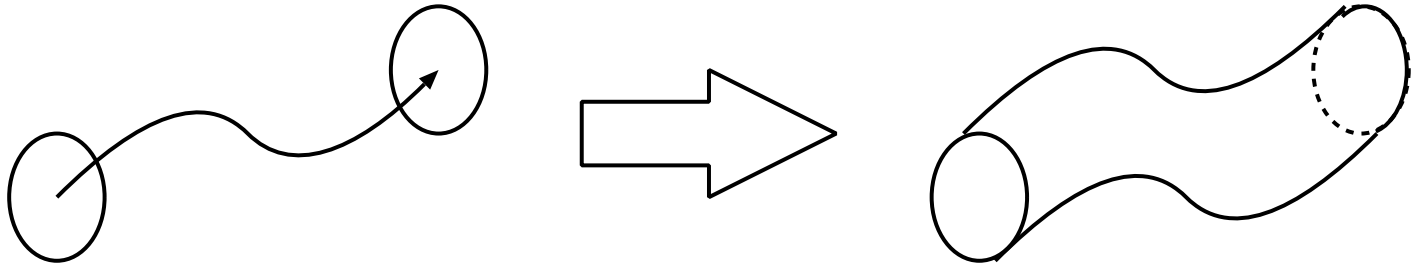
```
#include "colors.inc"
#include "shapes.inc"

camera {
  location <-10, 10, -20>
  look_at <0, 0, 0>
  angle 15
}

light_source { <-3, 10, -10> color 1.5*White }

difference {
  object {
    Cube
    pigment { color Yellow }
    rotate 45*y
  }

  object {
    Sphere
    pigment { color Red }
    translate <0, 1, -1>
  }
}
```

⇐ differenceで2つのobjectの差



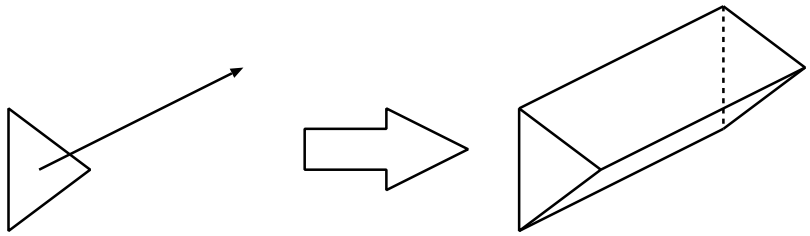
ソリッドモデルの表現法3

- **スイープ表現**（Sweep representations）
  物体の断面を表す2次元図形を，定めらた軌道（trajectory）に沿って移動させ，そのときの軌跡で物体形状を表現.

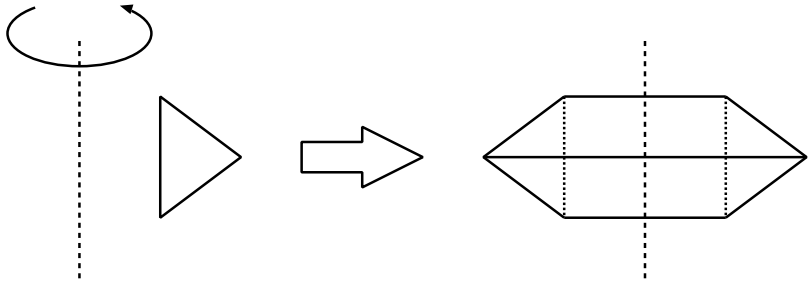


  - 作成される形状を直感的に把握できる.
  - 軌道を変化させれば様々な形状を作成可能.

平行移動スイープ



回転移動スイープ



POV-Rayでのスイープ表現

```
#include "colors.inc"
#include "shapes.inc"

camera {
  location <0, 0, -20>
  look_at <0, 0, 0>
  angle 15
}

light_source {
  <0, 0, -10>
  color 1.3*White
}

sphere_sweep {
  linear_spline
  5,
  <-2, -1, 0>, 0.1
  <-1,  1, 0>, 0.1
  < 0, -0.5, 0>, 0.1
  < 1,  1, 0>, 0.2
  < 2, -1, 0>, 0.2
  pigment { color White }
}
```

⇐ 折れ線に沿って球（sphere）を移動．途中で球の半径を0.1から0.2に変更．

ソリッドモデルの表現法4 (1/2)

- **ボクセル表現**（Voxel representations）

  単位立方体（voxel: volume elementからの造語）の集合として物体を表現.

  – **利点:**
    * データ構造が簡単
    * 多様な形状（自然物の不規則な形状）を表現可能
    * 集合演算が容易
    * 各ボクセルに透過率等のデータを与えれば，物体の内部構造も表現可

      → **ボリュームデータ**（volume data）（例: CT，MRIデータ）

  – **欠点:**
    *（一つの物体を多数のボクセルの集合として表現するため）

      データ量が膨大，形状の表現精度が低い，変換に手間がかかる.

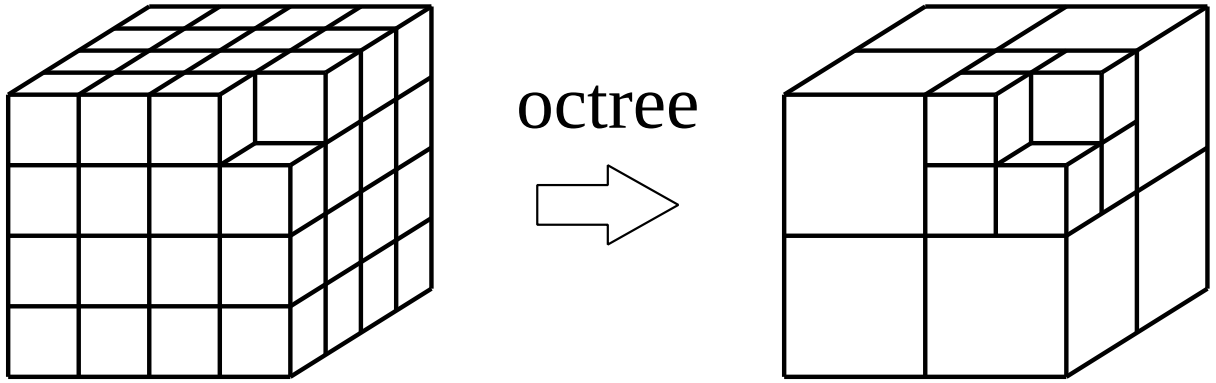
      ↓

  サイズの異なるボクセルを用いることでボクセル数を削減し欠点へ対処

  → **八分木**（octree）.

ソリッドモデルの表現法4 (2/2)

- **八分木**
  1. 物体全体を含む立方体（ボクセル）を考える.
  2. ボクセルが，物体と物体以外の両方を含む場合，そのボクセルを8分割.（同じ大きさ（元のボクセルの1/8の大きさ）のボクセルを8つ作成）
  3. 各ボクセルについて，2を再帰的に繰り返す.
  4. 物体を含むボクセルの集合により，物体の形状を表現.



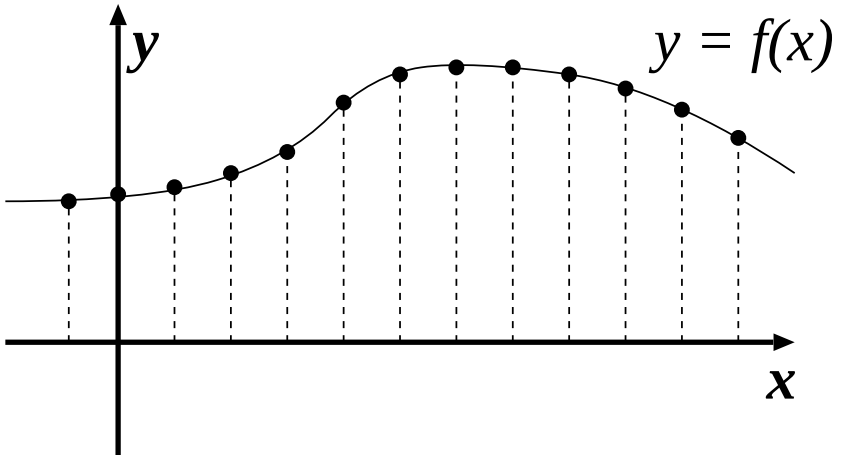
- **補足:** ボクセル以外にも，以下を単位に用いた手法などがある.
  - **メタボール(metaball)** → 濃度分布をもつ球の集合として物体を表現
  - **パーティクル(particle)** → 一定の規則に従って生成した粒子の集合として物体を表現

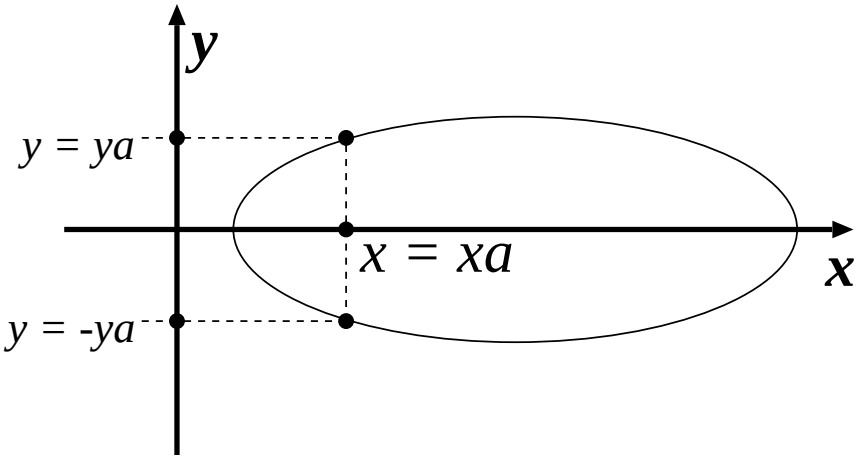
曲線・曲面の表現形式1

- **陽関数形式**（explicit function form）
  1つの座標値を他の座標値の関数として表現.

  2次元空間内の曲線：$y = f(x)$ （任意のxに対し，yが1つだけ存在）
  3次元空間内の曲面：$z = f(x, y)$ （任意のx, yに対し，zが1つだけ存在）

  - 曲線・曲面上の点を等間隔にサンプリングする場合には好都合.
  - 任意の座標値の組に対して，1つの座標値のみが対応.
    （閉曲線・閉曲面の表現には不都合）





曲線・曲面の表現形式2

- **陰関数形式**（implicit function form）

  関数を陰に用いて曲線や曲面を定義.

  – 陰関数形式による曲線・曲面表現の考え方

    * 1つの等式で1自由度が拘束される.
    * 空間の自由度:　2次元空間→2自由度，3次元空間→3自由度.
    * 図形上の点の自由度:　曲線上の点→1自由度，曲面上の点→2自由度.

    ⇓

    2次元空間で1つの等式を定義（例: $f(x,y)=0$）

    ⇒ 2自由度中の1自由度が拘束，残りは1自由度 → 曲線

    3次元空間で2つの等式を定義（例: $f(x,y,z)=0,\ g(x,y,z)=0$）

    ⇒ 3自由度中の2自由度が拘束，残りは1自由度 → 曲線

    3次元空間で1つの等式を定義（例: $f(x,y,z)=0$）

    ⇒ 3自由度中の1自由度が拘束，残りは2自由度 → 曲面

  – 曲線・曲面上の点を適当な間隔でサンプリングする場合には不都合.

曲線・曲面の表現形式3

- **パラメトリック形式**（parametric form）
  個々の座標値を**パラメータ**（媒介変数）の関数として表現.

$$\text{2次元空間内の曲線} \begin{cases} x = f(t) \\ y = g(t) \end{cases}$$

$$\text{3次元空間内の曲線} \begin{cases} x = f(t) \\ y = g(t) \\ z = h(t) \end{cases}, \quad \text{3次元空間内の曲面} \begin{cases} x = f(s,t) \\ y = g(s,t) \\ z = h(s,t) \end{cases}$$

  - 曲線・曲面上で点列を求めることが容易.
  - 曲線・曲面の交点の計算は複雑.

曲線・曲面の表現形式4

- **各形式の比較**
  - **2次元空間内での円（曲線）**

$$\text{陽関数形式}: y = f(x) = \pm\sqrt{r^2 - x^2}$$

$$\text{陰関数形式}: f(x,y) = x^2 + y^2 - r^2 = 0$$

$$\text{パラメトリック形式}: x = f(t) = r\sin(t),\quad y = g(t) = r\cos(t)$$

  - **3次元内での直線（曲線）** $\frac{x-x_0}{a} = \frac{y-y_0}{b} = \frac{z-z_0}{c}$

$$\text{陽関数形式}: y = f(x,z) = \frac{b}{a}(x - x_0) + y_0,\quad z = g(x,y) = \frac{c}{a}(x - x_0) + z_0$$

$$\text{陰関数形式}: f(x,y,z) = b(x - x_0) - a(y - y_0) = 0,\quad g(x,y,z) = c(x - x_0) - a(z - z_0) = 0$$

$$\text{パラメトリック形式}: x = f(t) = at + x_0,$$

$$y = g(t) = bt + y_0,$$

$$z = h(t) = ct + z_0$$

  - **3次元空間内での球（曲面）**

$$\text{陽関数形式}: z = f(x,y) = \pm\sqrt{r^2 - x^2 - y^2}$$

$$\text{陰関数形式}: f(x,y,z) = x^2 + y^2 + z^2 - r^2 = 0$$

$$\text{パラメトリック形式}: x = f(s,t) = r\sin(s)\sin(t),$$

$$y = g(s,t) = r\sin(s)\cos(t),$$

$$z = h(s,t) = r\cos(s)$$

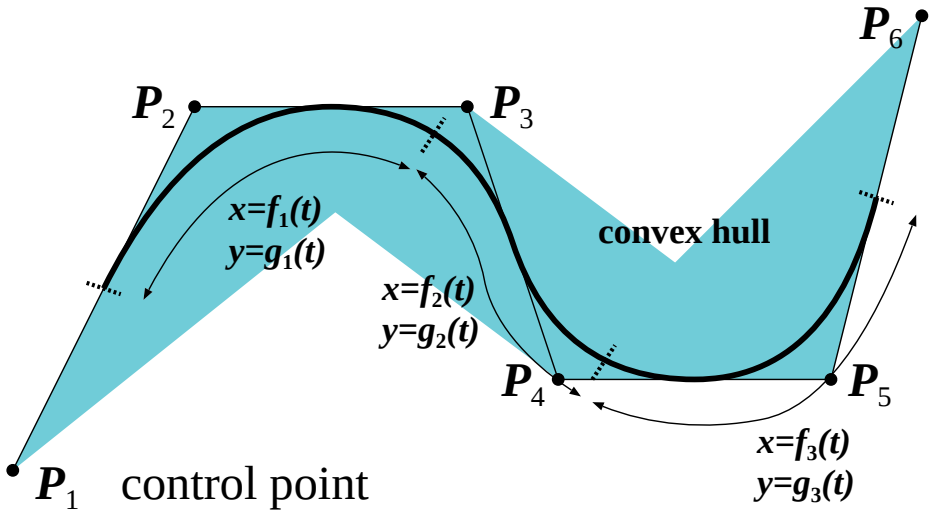
スプライン曲線・曲面

- **スプライン曲線・曲面**（spline curve, spline surface）
  曲線全体の何個かの区間に分割し，各区間での形状をパラメトリック形式で表現（区間の境界で滑らかとなるような多項式の関数を用いる）
  - ファーガソン曲線（Ferguson curve）
  - ベジエ曲線（Bézier curve）
  - Bスプライン曲線（B-spline curve）
  - 有理ベジエ曲線（Rational Bézier curve）
  - NURBS曲線（Non-Uniform Rational B-spline curve）

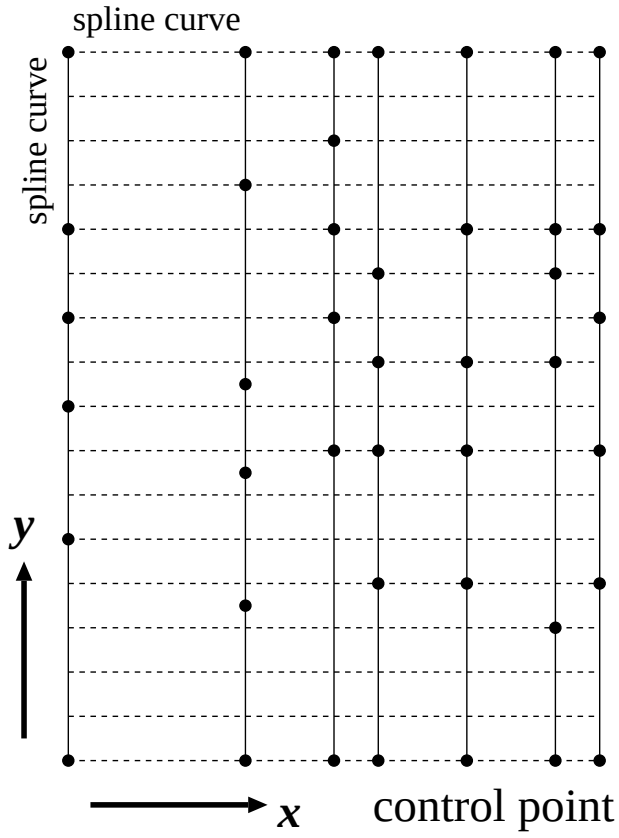
    他にも色々ある...

スプライン曲線・曲面のイメージ

- **制御点**（control point）に基づき，パラメータ t の関数（多項式）として座標値を表現.
  - 関数が t に関する n 次の多項式なら **n 次のスプライン曲線**.
  - 生成される曲線は，制御点で定義される **凸包**（convex hull）内に含まれる.
    （曲線は必ずしも制御点を通らない）
    * 凸包: 全制御点を含む最小の多角形



- スプライン曲面の生成
  - 制御点に基づきスプライン曲線を生成.
  - 生成された曲線の各点を制御点と見做し，スプライン曲線を再度生成.



スプライン曲線の例（3次のBスプライン曲線）

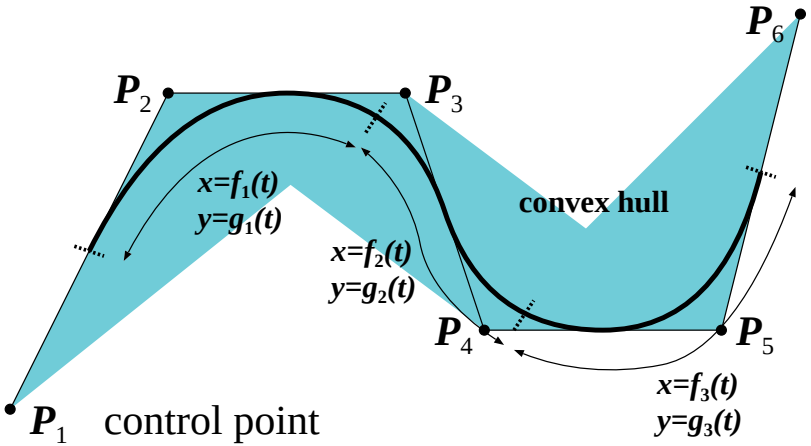
- 制御点の位置を $\boldsymbol{P}_n = (x_n, y_n)$ とすると，3次のBスプライン曲線は

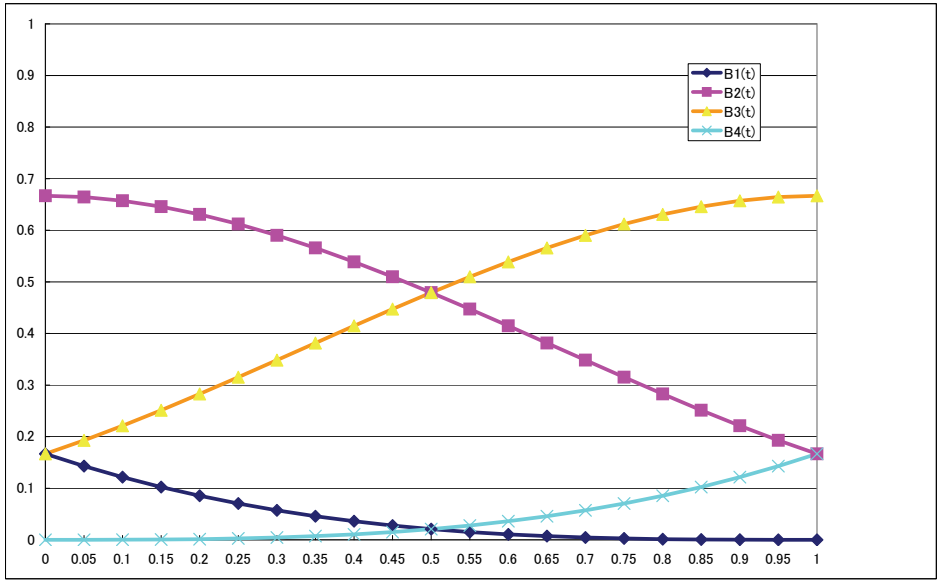
$$\begin{cases} x &= f_n(t) = x_n B_1(t) + x_{n+1} B_2(t) + x_{n+2} B_3(t) + x_{n+3} B_4(t) \\ y &= g_n(t) = y_n B_1(t) + y_{n+1} B_2(t) + y_{n+2} B_3(t) + y_{n+3} B_4(t) \end{cases} \tag{37}$$

隣接した4個の制御点に重み $B_i(t)$ を付け足し合せることで曲線を生成.
（$B_i(t)$ は，**3次のBスプライン基底関数**（B-spline basis function））

$$\begin{aligned} B_1(t) &= \tfrac{1}{6}(-t^3 + 3t^2 - 3t + 1) \\ B_2(t) &= \tfrac{1}{6}(3t^3 - 6t^2 + 4) \\ B_3(t) &= \tfrac{1}{6}(-3t^3 + 3t^2 + 3t + 1) \\ B_4(t) &= \tfrac{1}{6}(t^3) \end{aligned} \tag{38}$$

ただし，$0 \le t \le 1$





ポリゴン曲面（polygonal surface）

- 多様な形状を表現するために必要なデータ量
  - **ボクセル等の集合として形状を表現**

    → 多様な形状が表現可能であるるが，必要なデータ量は大.
  - **関数を用いた形状の表現**

    → 必要なデータ量は小，ただし，作成・変更が面倒（面同士の接続が大変）.

⇓　ハードウェアの進歩

- 必要なデータ量の問題の大きさは，さして問題ではなくなった.
  - **小さい平面（ポリゴン（polygon），パッチ（patch），メッシュ（mesh））**の集合として，**境界表現**により物体の形状を表現.
  - 関数表現の場合よりも，面同士の接続が楽.
  - ポリゴンとしては三角形が良く用いられる（3点を指定すれば平面が定まる）.