http://www.takagi.inf.uec.ac.jp/mmip/

マルチメディア処理

Multimedia Information Processing

第6回 ２次元図形の表現と描画

髙木一幸

講義概要

- マルチメディアデータ（文書、音声、画像、映像など）の表現方法と処理技術について、基礎的な内容を紹介・解説する。
- 授業時間中および宿題として、計算機を使った演習を行うことにより実際のデータの扱い方を学び、理解を深める。

第1回：授業の概要説明、
人間の感覚とマルチメディア処理（4/11）

髙木

第2回：文字・テキストの表現と処理（4/18）

第3回：音声のディジタル表現と処理（4/25）

第4回：画像・映像のディジタル表現（5/2）

第5回：マルチメディアデータの符号化とファイル形式（5/9）

第6回：2次元図形の表現と描画（5/16）

第7回：画像処理(1)画素ごとの濃淡変換（5/23）

第8回：画像処理(2)空間フィルタリング（5/30）

廣田

第9回：カメラと写真撮影（6/13）

第10回：3次元コンピュータグラフィックス（6/20）
(1)形状表現と透視投影

第11回：3次元コンピュータグラフィックス（6/27）
(2)照明効果とシェーディング

第12回：アニメーションと映像制作（7/4）

第13回：シミュレーションと可視化（7/11）

第14回：グラフィックパイプラインとシェーダ（7/18）

第15回：マルチメディアの応用例I（7/25）

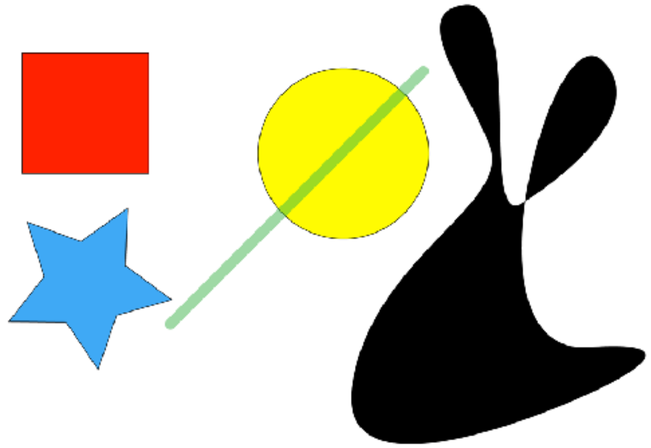
２次元図形の表現と描画
ラスタ形式 vs ベクタ形式

ラスタ形式（raster format）

LENNA （SIDBA標準画像データベース）

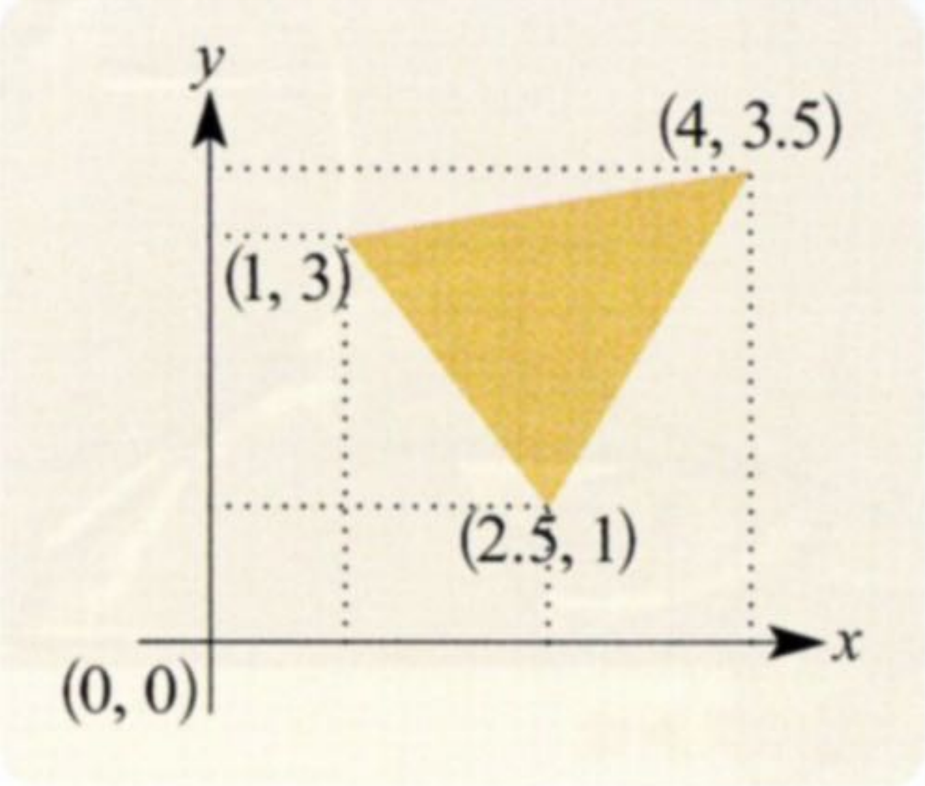
- 画素値の集合
- 写真などの画像の表現に向く。
- 画素値演算により、平滑化や鮮鋭化などの画像処理を容易に施すことが可能。
- 斜めの線にギザギザが現れたり、画像の拡大・縮小や回転などの幾何変換を施すと情報が失われる。

ベクタ形式（vector format）



- 図形の集合
- 情報を失わずに幾何変換を施すことが可能。
- 線や図形の組み合わせで表現することが難しい写真のような画像の表現には向かない。

図形の数値表現



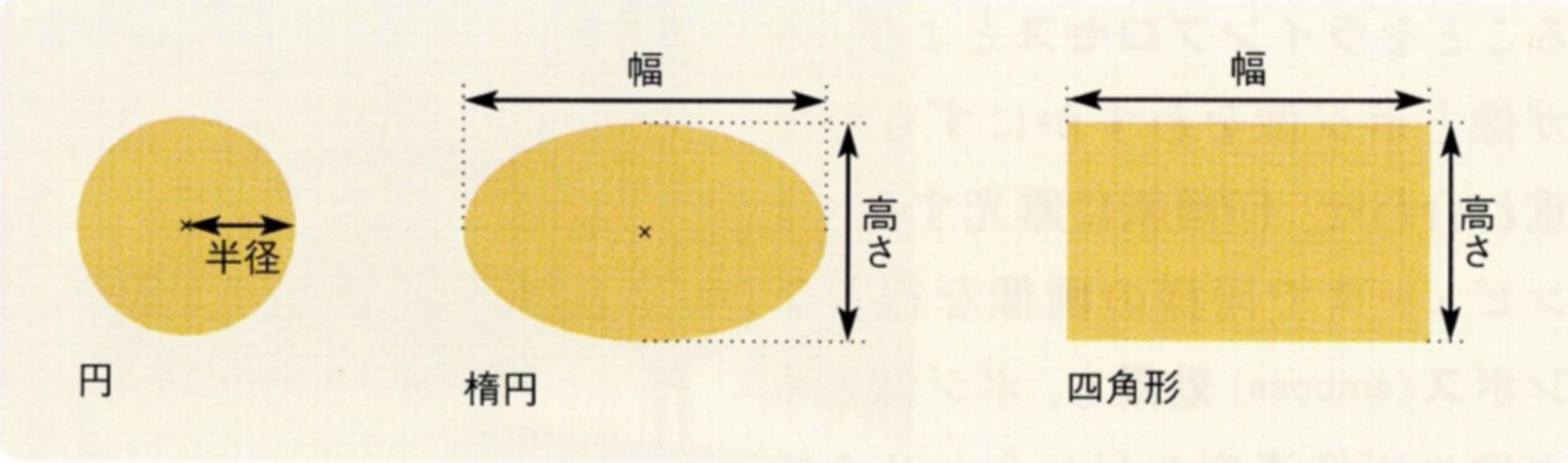
■図2.37——ベクタ形式による表現

・ベクタ形式の基本的な方法として、描画しようとする図形の形状を、2次元座標系のなかでの座標値として表す。

・形状を線分で表し、それぞれの線分の両端の頂点を x、y の座標値で表現することで数値化を行う。

©実践マルチメディア[改訂新版]画像情報教育振興協会

図形の数値表現



■図 2.38——基本図形の例

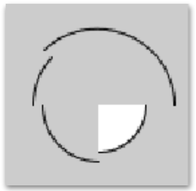
・四角形や円などの基本的な幾何学形状は、縦、横の大きさや中心と半径などで形状を指定。

・連続した直線をつないで多角形を描くことによって自由な形状を指定することができる。

©実践マルチメディア[改訂新版]画像情報教育振興協会

Processingの２次元基本図形

円弧 arc()



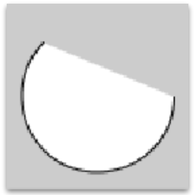
```
arc(50, 55, 50, 50, 0, HALF_PI);
noFill();
arc(50, 55, 60, 60, HALF_PI, PI);
arc(50, 55, 70, 70, PI, PI+QUARTER_PI);
arc(50, 55, 80, 80, PI+QUARTER_PI, TWO_PI);
```

中心(50, 55)、直径50の円上の[0, π/2]の 円弧（白色）

中心(50, 55)、直径60の円上の[π/2, π] の円弧

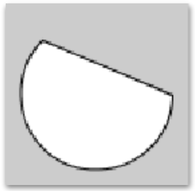
中心(50, 55)、直径70の楕円上の[π, 5π/4] の円弧

中心(50, 55)、直径80の楕円上の[5π/4, 2π] の円弧



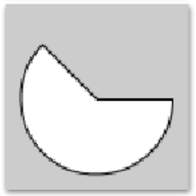
```
arc(50, 50, 80, 80, 0, PI+QUARTER_PI, OPEN);
```

OPEN：弦に線を引かない



```
arc(50, 50, 80, 80, 0, PI+QUARTER_PI, CHORD);
```

CHORD：弦に線を引く



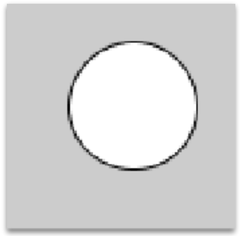
```
arc(50, 50, 80, 80, 0, PI+QUARTER_PI, PIE);
```

PIE：弦に線を引く（180°以上の場合）

https://processing.org/reference/ による

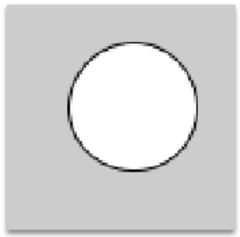
Processingの２次元基本図形

円 circle()　楕円 ellipse()



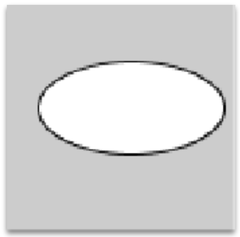
```
circle(56, 46, 55);
```

中心 (56, 46) 、直径55の円



```
ellipse(56, 46, 55, 55);
```

中心 (56, 46) 、X幅55、Y幅55の楕円



```
ellipse(56, 46, 80, 40);
```

中心 (56, 46) 、X幅80、Y幅40の楕円

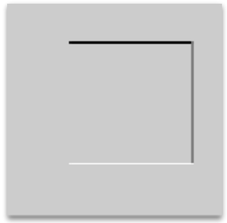
https://processing.org/reference/ による

Processingの２次元基本図形

線分 line()

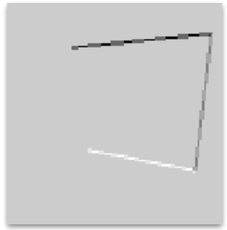
```
line(30, 20, 85, 75);
```

始点(30, 20) 、終点(85,75) の線分



```
line(30, 20, 85, 20);
stroke(126);
line(85, 20, 85, 75);
stroke(255);
line(85, 75, 30, 75);
```

`stroke()`で線の色を指定する



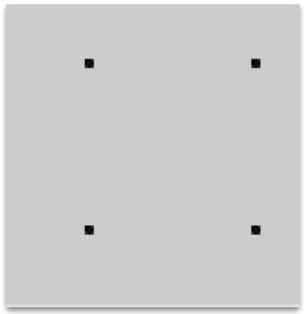
```
size(100, 100, P3D);
line(30, 20, 0, 85, 20, 15);
stroke(126);
line(85, 20, 15, 85, 75, 0);
stroke(255);
line(85, 75, 0, 30, 75, -50);
```

三次元空間に線を引く場合は、まず、
`size(*,*,P3D)`を指定する

https://processing.org/reference/ による

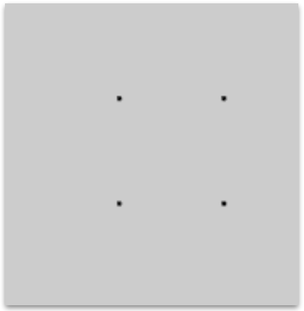
Processingの２次元基本図形

点 point()



```
noSmooth();
point(30, 20);
point(85, 20);
point(85, 75);
point(30, 75);
```

point()関数で２次元空間に４つの点を描く
点の大きさは１ピクセル



```
size(100, 100, P3D);
noSmooth();
point(30, 20, -50);
point(85, 20, -50);
point(85, 75, -50);
point(30, 75, -50);
```

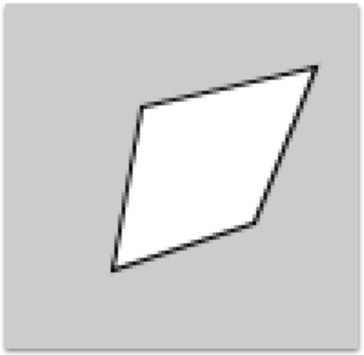
point()関数で3次元空間に４つの点を描く

この説明図では点の位置を明示するために実際の点より大きくしています

https://processing.org/reference/ による

Processingの２次元基本図形

四角形 quad()



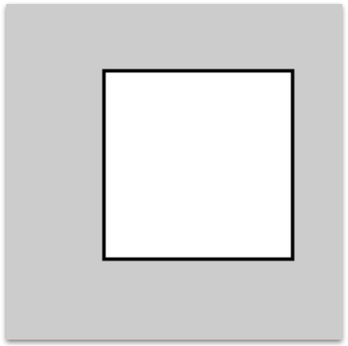
```
quad(38, 31, 86, 20, 69, 63, 30, 76);
```

(38,31),(86,20),(69,63),(30,76)の4点を頂点とする四角形を描く

https://processing.org/reference/ による

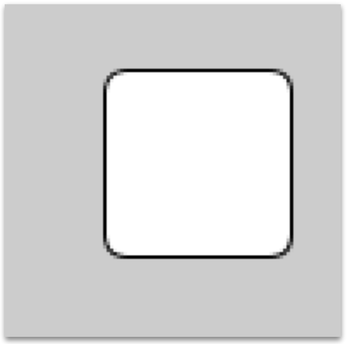
Processingの２次元基本図形

矩形 rect()



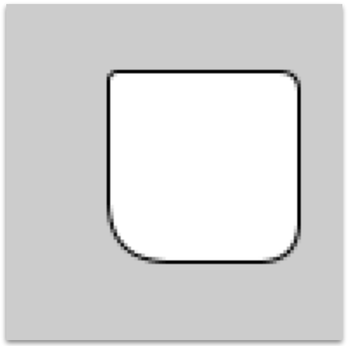
```
rect(30, 20, 55, 55);
```

(30,20)から55×55の矩形を描く



```
rect(30, 20, 55, 55, 7);
```

(30,20)から四隅の角の丸みが
半径7の円である55×55の矩形を描く



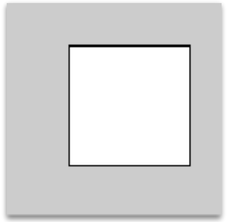
```
rect(30, 20, 55, 55, 3, 6, 12, 18));
```

(30,20)から55×55の矩形を描く
角の丸みは左上から時計回りに3、6、12、18

https://processing.org/reference/ による

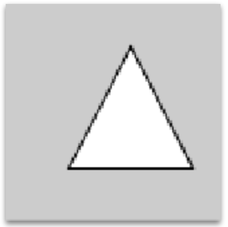
Processingの２次元基本図形

正方形 square() 三角形 triangle()



```
square(30, 20, 55);
```

(30,20)から55×55の正方形を描く



```
triangle(30, 75, 58, 20, 86, 75);
```

(30,75),(58,20),(86,75)の３点を頂点とする三角形を描く

https://processing.org/reference/ による

曲線の表現　Bスプライン曲線　B-spline curve

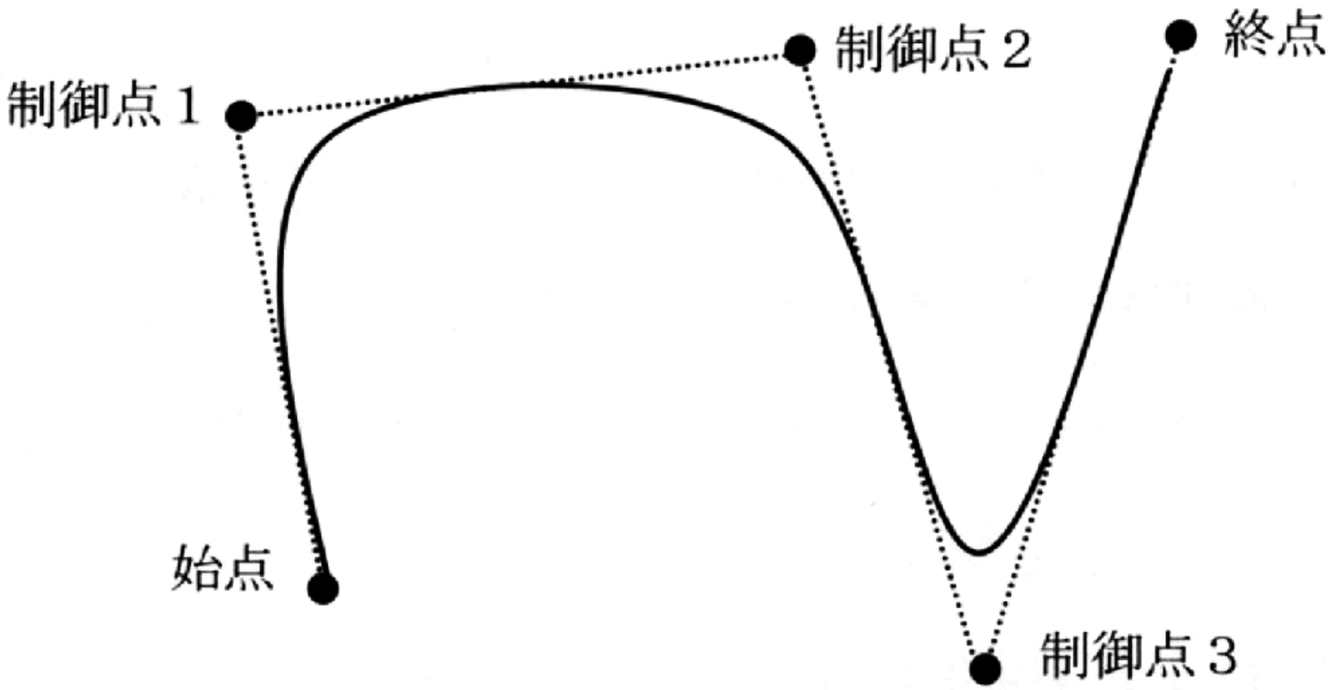


図 3-19　B-スプライン曲線

制御点によって指定された
多角形に内接する曲線

©田上博司、マルチメディア情報学概論、阪南大学叢書76、2006年1月16日

曲線の表現

ベジェ曲線　Bezier curve

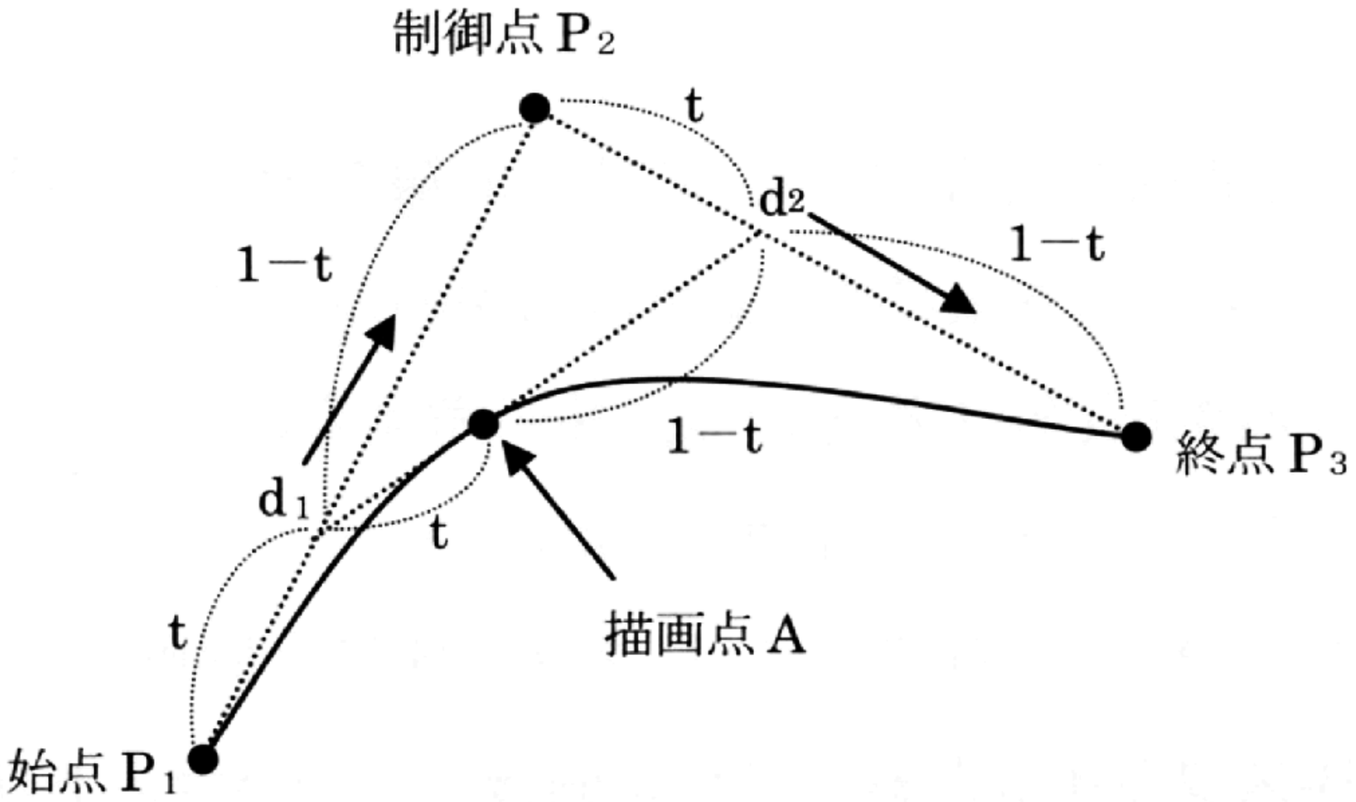


図 3-20 ベジエ曲線

始点P_1、制御点P_2、終点P_3が与えられたとき、

P_1とP_2を結ぶ線分を t : (1 - t) に内分する点をd_1、

P_2とP_3を結ぶ線分を t : (1 - t) に内分する点をd_2とし、

t が 0 から1まで変化するときに

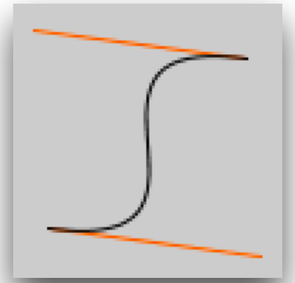
d_1とd_2を結ぶ線分を t : (1 - t)に内分する

点Aが描く曲線。

©田上博司、マルチメディア情報学概論、阪南大学叢書76、2006年1月16日

Processingの曲線

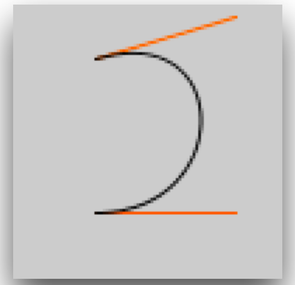
Bezier曲線 bezier()



```
noFill();
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
```

第1制御点 曲線通過　第2制御点　第3制御点　第4制御点 曲線通過

この例では、制御点を両端とする線分と
Bezier曲線の両方を図示している。



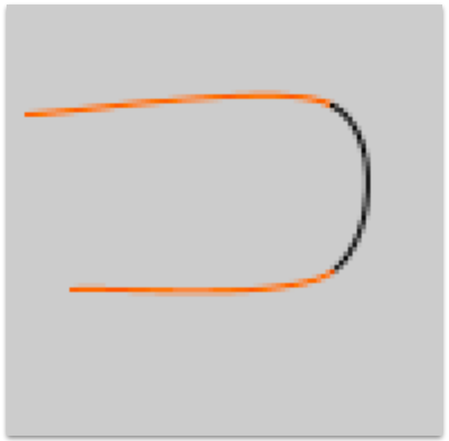
```
noFill();
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20,  80, 5,  80, 75,  30, 75);
```

https://processing.org/reference/ による

Processingの曲線

スプライン曲線 curve()

この例では、3本のスプライン曲線を
2色に色分けして表示している。



```
noFill();
stroke(255, 102, 0);                                    上側
curve(5, 26, 5, 26, 73, 24, 73, 61);
```

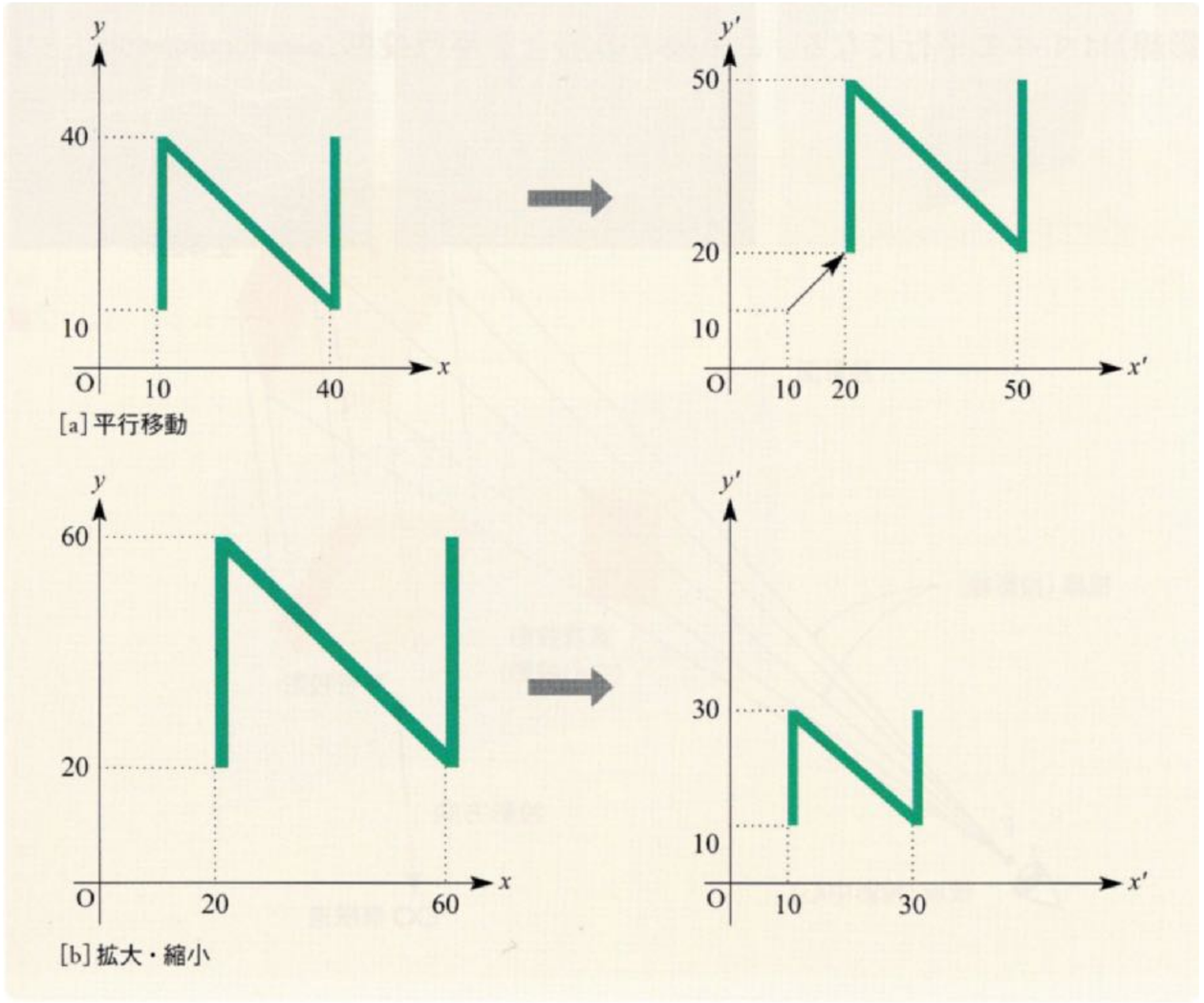
開始制御点　第1制御点　第2制御点　終了制御点

```
stroke(0);
curve(5, 26, 73, 24, 73, 61, 15, 65);

stroke(255, 102, 0);                                    下側
curve(73, 24, 73, 61, 15, 65, 15, 65);
```

https://processing.org/reference/ による

幾何変換

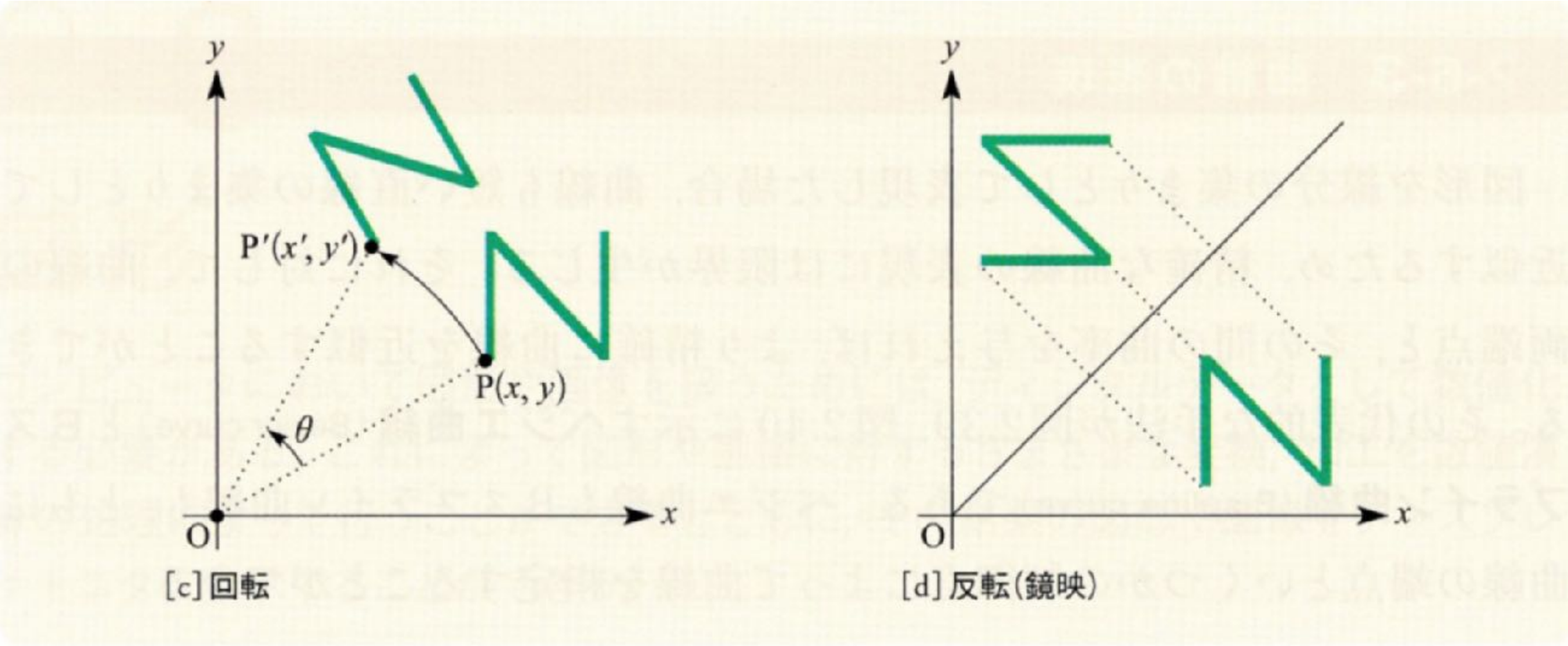
2次元や3次元の座標において定義された図形に位置や形状などの変化を与える処理



©実践マルチメディア[改訂新版]画像情報教育振興協会

幾何変換

2次元や3次元の座標において定義された図形に位置や形状などの変化を与える処理

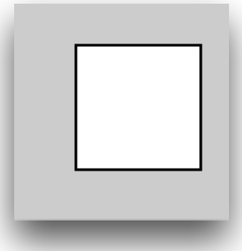


■図 2.41——幾何変換（前頁続き）

©実践マルチメディア[改訂新版]画像情報教育振興協会

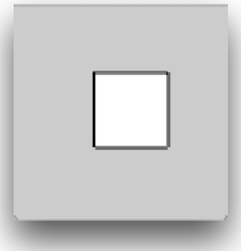
Processingの幾何変換

平行移動　translate()



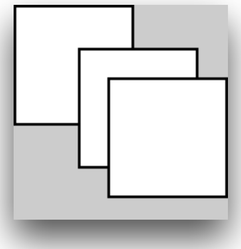
```
translate(30, 20);
rect(0, 0, 55, 55);
```

原点を(30,20)に移動し55×55の矩形を描く



```
size(100, 100, P3D);
translate(30, 20, -50);
rect(0, 0, 55, 55);
```

3D空間で原点を(30,20,-50)に移動し
55×55の矩形を描く



```
rect(0, 0, 55, 55);
translate(30, 20);
rect(0, 0, 55, 55);
translate(14, 14);
rect(0, 0, 55, 55);
```

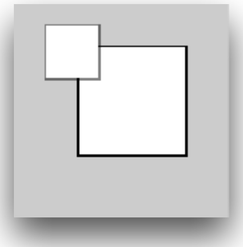
translate()の結果は累算される

(30,20)+(14,14)=(44,34)

https://processing.org/reference/ による

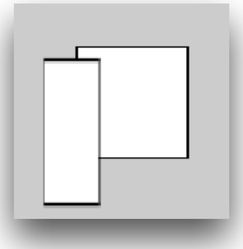
Processingの幾何変換

拡大・縮小　scale()



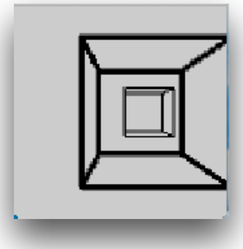
```
rect(30, 20, 50, 50);
scale(0.5);
rect(30, 20, 50, 50);
```

表示倍率を**0.5**にして矩形を描く
座標系が0.5倍になることに注意
したがって線の太さも**0.5**倍になる



```
rect(30, 20, 50, 50);
scale(0.5, 1.3);
rect(30, 20, 50, 50);
```

縦横比を変えた倍率を設定した例



```
size(100, 100, P3D);
noFill();
translate(width/2+12, height/2);
box(20, 20, 20);
scale(2.5, 2.5, 2.5);
box(20, 20, 20);
```

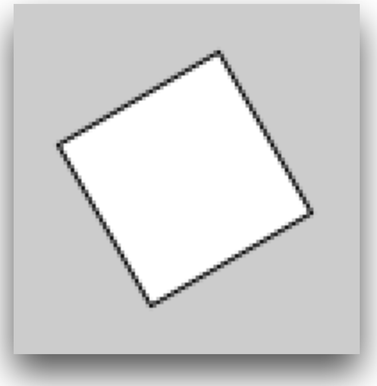
3D空間に倍率を設定する

一辺20の立方体
XYZ軸を**2.5**倍にする

https://processing.org/reference/ による

Processingの幾何変換

回転　rotate()



```
translate(width/2, height/2);
rotate(PI/3.0);
rect(-26, -26, 52, 52);
```

原点を画面中央に移動

π/3回転

(-26, -26)から52×52の矩形を描く

https://processing.org/reference/ による

投影 projection

平面上に立体を表現するための変換処理

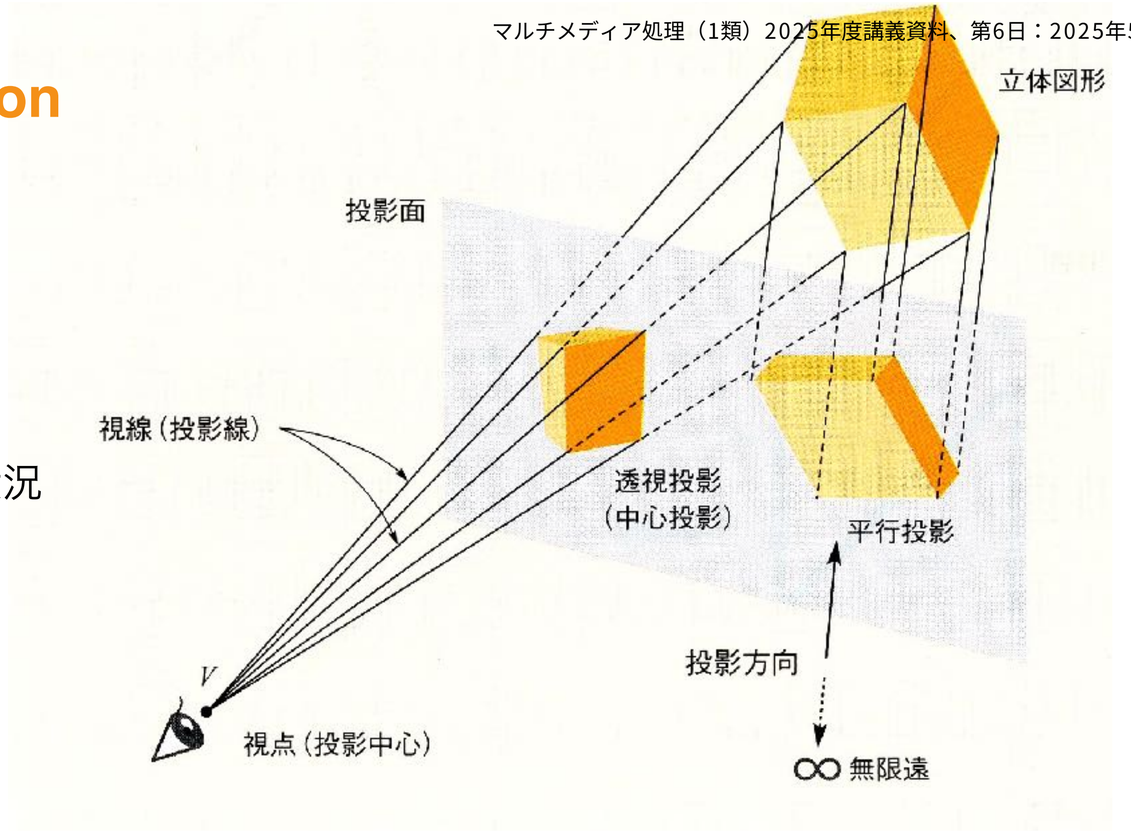


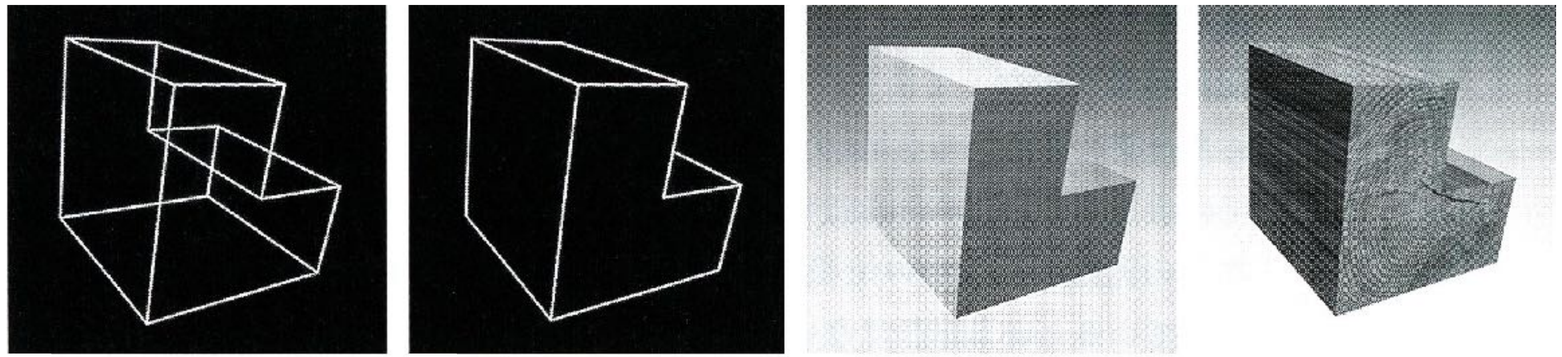
図 2.42——投影変換の原理

©実践マルチメディア[改訂新版]画像情報教育振興協会

- 3次元空間に置かれた立体図形を**視点V** から見ている状況
- 2次元図形が描かれる平面は「**投影面**」
- 視点から立体上の点へ引いた半直線を「**視線（投影線）**」と呼ぶ。

- 視線が投影面と交差する点を求めることを「**投影**」という。これらの交点によって生成される図形を「**透視図**」と呼び、立体の2次元的表現である。
- 透視法には、視点の位置や物体と投影面の相対的位置などによって様々な種類のものがある。
- 視線は視点を中心に放射上になっていて、これを「**透視投影**」または「**中心投影**」と呼ぶ。
- 視点が投影面から無限の遠方にある特別な場合を想定すると、視線（投影線）は全て平行になる。このような場合を「**平行投影**」と呼ぶ。

隠線消去・隠面消去



[a] [b] [c] [d]

図2.44——3次元モデルの表示法

隠線消去：線分で構成される面を仮定し、辺の見えない部分を描かないようにする

隠面消去：見える面がどこなのかを判定し、可視面だけを表示する

[a] 隠線消去を行わないで全ての面の境界の線を表示した立体

[b] 隠線・隠面消去して表示した立体

[c] 見えている面に陰影を与えた立体（光源は斜め上）

[d] 材質感を出すための模様を付加した立体

使用している幾何形状は同じだが隠線消去した画像や隠面消去した画像のほうが立体感が表現でき、位置関係や奥行きもよくわかる。

©実践マルチメディア[改訂新版]画像情報教育振興協会

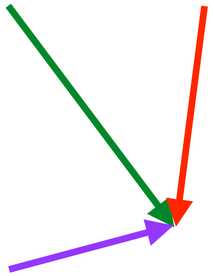
Processingの3D表現

描画モードの変更　size()　　立方体の描画 box()

size()関数の第3引数に "P3D" を指定するとOpenGLの3次元描画を行う

```
size(150, 200, P3D);
background(153);
```
３次元空間を扱う150 × 200ピクセルの画面を定義

３次元空間に３本の座標軸を引く
```
line(0, 0, 0, width, height, -100);
line(width, 0, 0, width, height, -100);
line(0, height, 0, width, height, -100);
```

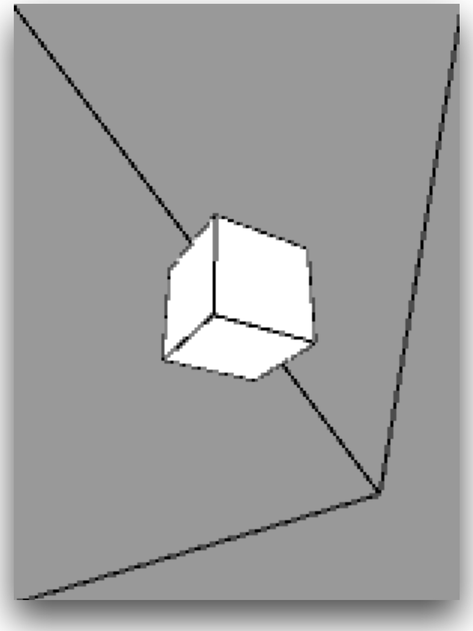


描画の基点を画面の中央に移動
```
translate(width/2, height/2);
```

```
rotateX(PI/6);
rotateY(PI/6);
```
X軸とY軸をπ/6回転

```
rotateX()
rotateY()
```

```
box(35);
```
一辺35の立方体を描く



https://processing.org/reference/ による

Processingの3D表現　図形

sphere(), lights()

```
size(400, 400, P3D);
noStroke();
lights();
translate(200, 200, 0);
sphere(150);
```

３次元空間を扱う400×400の画面を定義

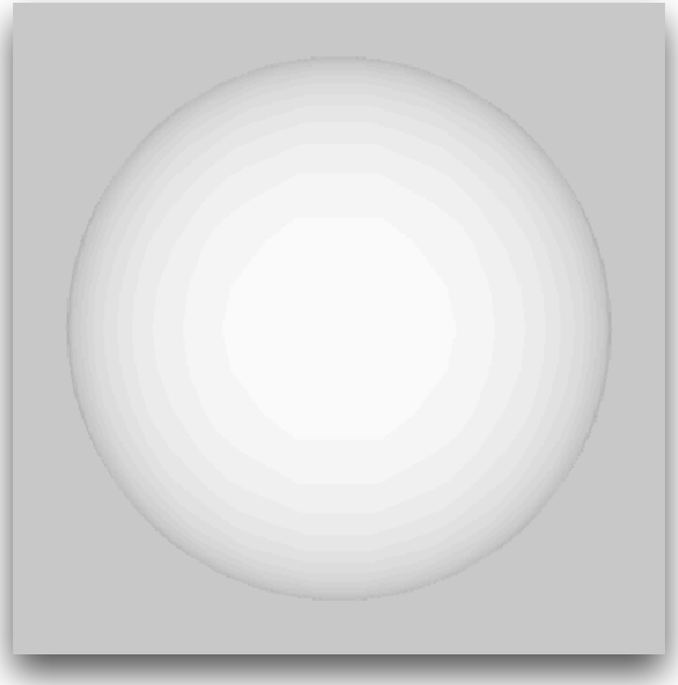
環境光を設定する

球の中心は画面中央

半径150ピクセルの球を描く

光線に関する
Processingの関数には
右のようなものがある。

```
ambientLight()
directionalLight()
lightFalloff()
lights()
lightSpecular()
noLights()
normal()
pointLight()
spotLight()
```



Processingの3D表現　図形

陰線消去された立方体を描く

directionalLight(), ambientLight()

```
size(100, 100, P3D);
background(0);
noStroke();
directionalLight(126, 126, 126, 0, 0, -1);
ambientLight(102, 102, 102);
translate(50, 50, 0);
rotateX(PI/5);
rotateY(PI/5);
box(40);
```

3次元空間を扱う100 × 100ピクセルの画面を定義

平行光源を設定

光源のRGB値

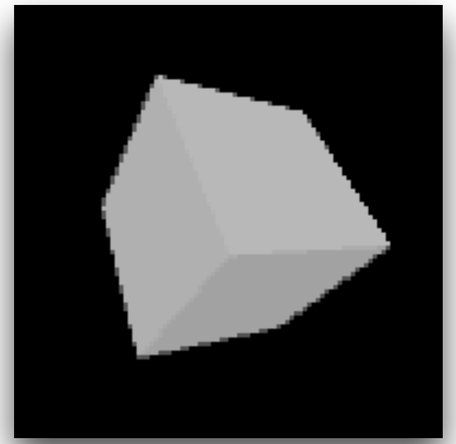
光の方向（"**-1**" は上向きの光）

環境光を設定

図形描画の基点を画面の中心に移動

X方向とY方向にπ/5 回転

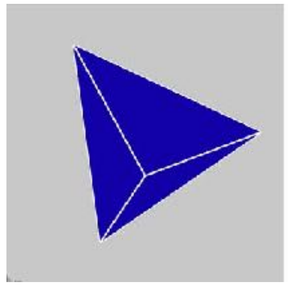
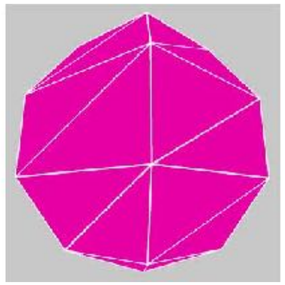
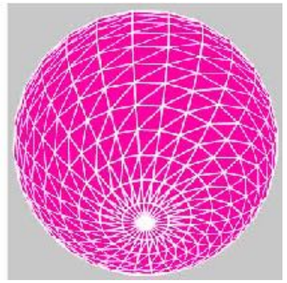
一辺40の立方体を描く



Processingの3D表現　図形

sphere()で描く3次元球面の細かさを指定する sphereDetail()

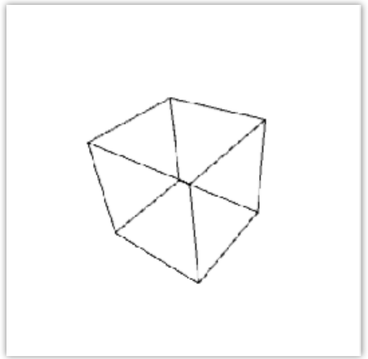
```
void setup() {
  size(600, 600, P3D);
  strokeWeight(4);
}

void draw() {
  background(200);
  stroke(255);
  translate(300, 300, 0);
  rotateX(mouseY * 0.05);    マウスのXY座標に応じて球を回転させる
  rotateY(mouseX * 0.05);
  fill(mouseX * 2, 0, 160); マウスのX座標によって球の色を決める
  int res = mouseX/20;    球面を構成する頂点数をマウスのX座標によって決定　値の範囲は[0, 30]（省略時値は30）
  sphereDetail(res);
  sphere(250);          半径250ピクセルの球を描く
}
```

https://processing.org/reference/ のサンプルを改訂

Processingの3D表現　投影

camera() 視点の設定



```
size(300, 300, P3D);
noFill();
background(255);

camera(70.0, 35.0, 120.0, 50.0, 50.0, 0.0, 0.0, 1.0, 0.0);

translate(50, 50, 0);
rotateX(-PI/6);
rotateY(PI/3);
box(45);
```

視点の座標　　風景中心の座標　　カメラの上方向の向き

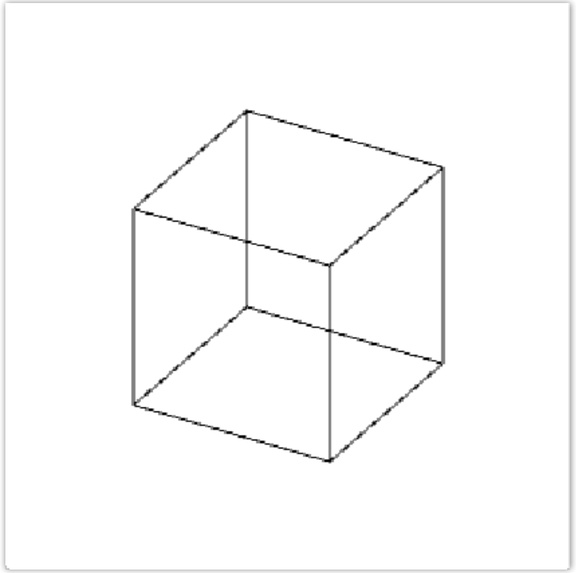
https://processing.org/reference/ による

Processingの3D表現　投影

ortho()　正投影図を作る　orthographic projection

遠近を付けない投影法

```
size(300, 300, P3D);
background(255);
noFill();

ortho();     正投影を設定（省略時設定）

translate(width/2, height/2, 0);
rotateX(-PI/6);
rotateY(PI/3);
box(120);
```



`ortho()`の引数の数は 0、4、あるいは 6 である。

引数の意味はProcessingの説明（以下）を参照のこと。

https://processing.org/reference/ のサンプルを改訂

Processingの3D表現　投影

perspective() 透視投影図を作る

perspective projection

```
size(300, 300, P3D);
noFill();
background(255);

float fov = PI/3.0;
```

画角はπ/3

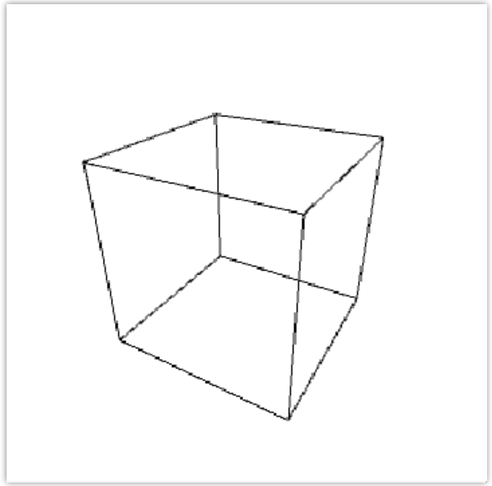
```
float cameraZ = (height/2.0) / tan(fov/2.0);

perspective(fov, float(width)/float(height),
```

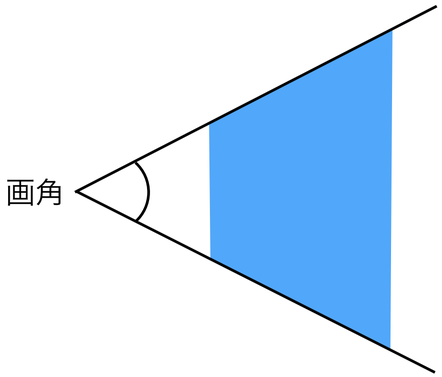
画角　　アスペクト（縦横比）

```
            cameraZ/10.0, cameraZ*10.0);
```

最も近いクリッピング平面のZ座標　　最も遠いクリッピング平面のZ座標

```
translate(150, 150, 0);
rotateX(-PI/6);
rotateY(PI/3);
box(120);
```





https://processing.org/reference/ のサンプルを改訂

Processingの3D表現　テクスチャマッピング

PShape setTexture shape

```
PImage img;
PShape box;

void setup(){
  size(600, 600, P3D);
  smooth();
  img = loadImage("wood.jpg");
}

void draw() {
  background(255);
  pushMatrix();
  translate(300,300,100);
  box = createShape(BOX, 200);
  box.setTexture(img);
  rotateX(map(mouseY, 0, width, -PI, PI));
  rotateY(map(mouseX, 0, width, -PI, PI));
  shape(box);
  popMatrix();
}
```

PShape：ベクタ画像を格納するデータ型

anti-aliasing処理をして滑らかな縁にする

変数**img**に貼り付ける画像を読み込む

一辺200ピクセルの"BOX"（立方体）を生成

立方体のテクスチャとして**img**に読み込まれている画像を設定

画面に画像を表示

２次元図形の表現と描画

本日の要点

1. 図形の表現方法にはラスタ形式とベクタ形式がある。
2. ラスタ形式は画素値の集合であり、写真などの画像の表現用。
3. ベクタ形式では図形の形状を座標系の中の座標値として表現。
4. ベクタ形式では自由曲線の正確な表現にはベジェ曲線や
   スプライン曲線が用いられる。
5. 幾何変換：定義された図形に位置や形状の変化を与える処理
   平行移動、回転、拡大・縮小、反転（鏡映）

. 平面上に立体を表現するためには、投影と隠線・隠面消去を行う。

Processingで力学シミュレーション

運動を表現する目的で PVector クラスを使う

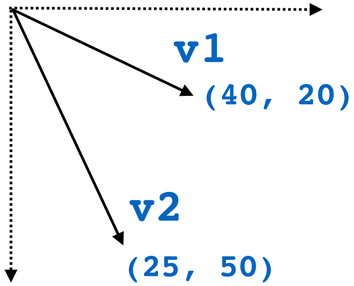
最も簡単なPVectorの使用例（力学シミュレーションではない）

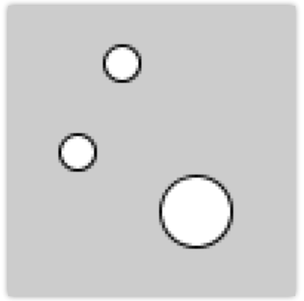
```
PVector v1, v2;

void setup() {
  noLoop();
  v1 = new PVector(40, 20);
  v2 = new PVector(25, 50);
}

void draw() {
  ellipse(v1.x, v1.y, 12, 12);
  ellipse(v2.x, v2.y, 12, 12);
  v2.add(v1);
  ellipse(v2.x, v2.y, 24, 24);
}
```

PVectorクラスのデータv1とv2を宣言

new：新しいオブジェクトを生成

v2.add(v1)：v2+v1





そのほかベクトル演算に必要な
メソッドが用意されている

https://processing.org/reference/ による

Processingで力学シミュレーション

壁にバウンドする球 1/4

```
PVector force;
PVector acceleration;
PVector location;
PVector velocity;
PVector gravity;
float mass;
float friction;
PVector min;
PVector max;

void setup() {
  size(800, 600);
  frameRate(60);

  location = new PVector(0.0, 0.0);
  velocity = new PVector(0.0, 0.0);
  gravity = new PVector(0.0, 1.0);
  force = new PVector(12.0, 8.0);
```

force：球に働く力
acceleration：球に働く加速度
location：球の位置
velocity：球の速度
gravity：重力
mass：球の質量
friction：球に働く摩擦力（空気抵抗）
min：位置ベクトルの最小値
max：位置ベクトルの最大値

© 田所潤、Processingクリエイティブ・コーディング入門―コードが生み出す創造表現、2017年、技術評論社　による

各種初期値の設定
球の初期位置は原点
初速は０
重力は下向きに１
球を投げる速度は（12, 8）

Processingで力学シミュレーション

壁にバウンドする球 2/4

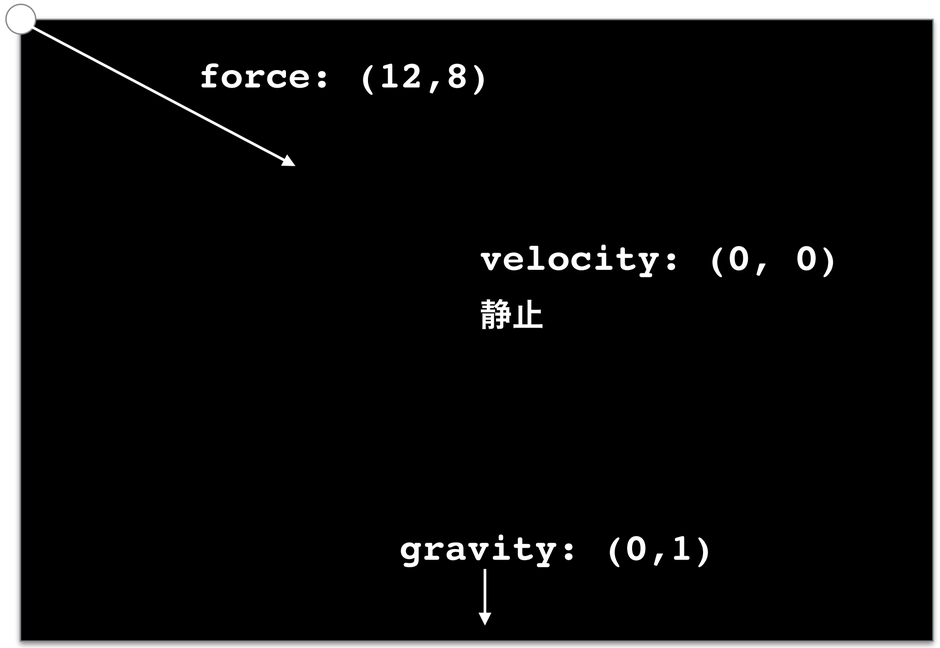
```
  min = new PVector(0.0, 0.0);
  max = new PVector(width, height);
  mass = 1.0;
  friction = 0.01;
  acceleration = force.div(mass);
}
```

空間の寸法

球の質量は１

摩擦は0.01

加速度=力／質量



初期状態

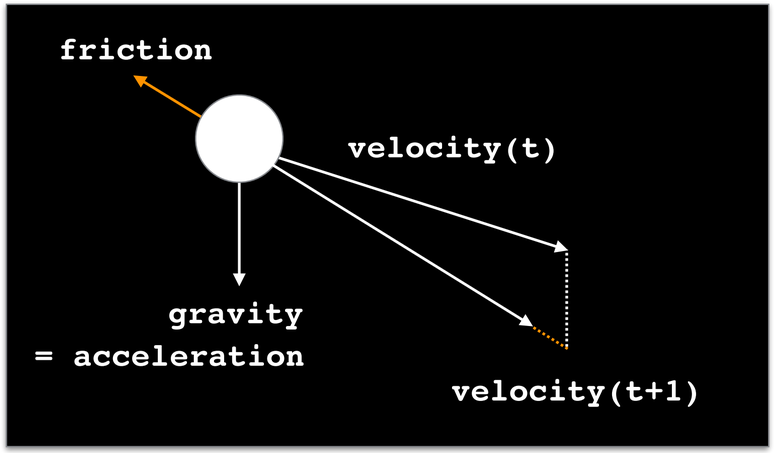
© 田所潤、Processingクリエイティブ・コーディング入門—コードが生み出す創造表現、2017年、技術評論社　による

Processingで力学シミュレーション

壁にバウンドする球 3/4

```
void draw() {
  fill(0, 31);
  rect(0, 0, width, height);
```
軌跡が残って見えるようにする
```
  fill(255);
```
球の色は白
```
  noStroke();
```
球の縁線は描かない
```
  acceleration.add(gravity);
  velocity.add(acceleration);
  velocity.mult(1.0-friction);
  location.add(velocity);
  acceleration.set(0, 0);
```
加速度を0に再設定
```
  ellipse(location.x, location.y, 20, 20);
  bounceOffWalls();
}
```
新しい位置に直径20の円（球）を描く

壁の衝突判定（ユーザ定義関数：説明は次頁）



次の時刻の球の位置の計算

1. 加速度に重力加速度を加える
2. 速度に加速度を加える
3. 速度から摩擦分を割り引く
4. 直前位置に速度を加える

© 田所潤、Processingクリエイティブ・コーディング入門―コードが生み出す創造表現、2017年、技術評論社　による

Processingで力学シミュレーション

壁にバウンドする球 4/4

```
void bounceOffWalls() {
  if(location.x > max.x) {  右壁
    location.x = max.x;
    velocity.x *= -1;
  }
  if(location.x < min.x) {  左壁
    location.x = min.x;
    velocity.x *= -1;
  }
  if(location.y > max.y) {  床
    location.y = max.y;
    velocity.y *= -1;
  }
  if(location.y < min.y) {  天井
    location.y = min.y;
    velocity.y *= -1;
  }
}
```

bounceOffWalls()：壁への衝突判定

x 位置が最大値を超えたら
x 位置を最大値に設定し、
速度の **x** 成分を逆向きにする

以下同様

© 田所潤、Processingクリエイティブ・コーディング入門—コードが生み出す創造表現、2017年、技術評論社　による

おまけ

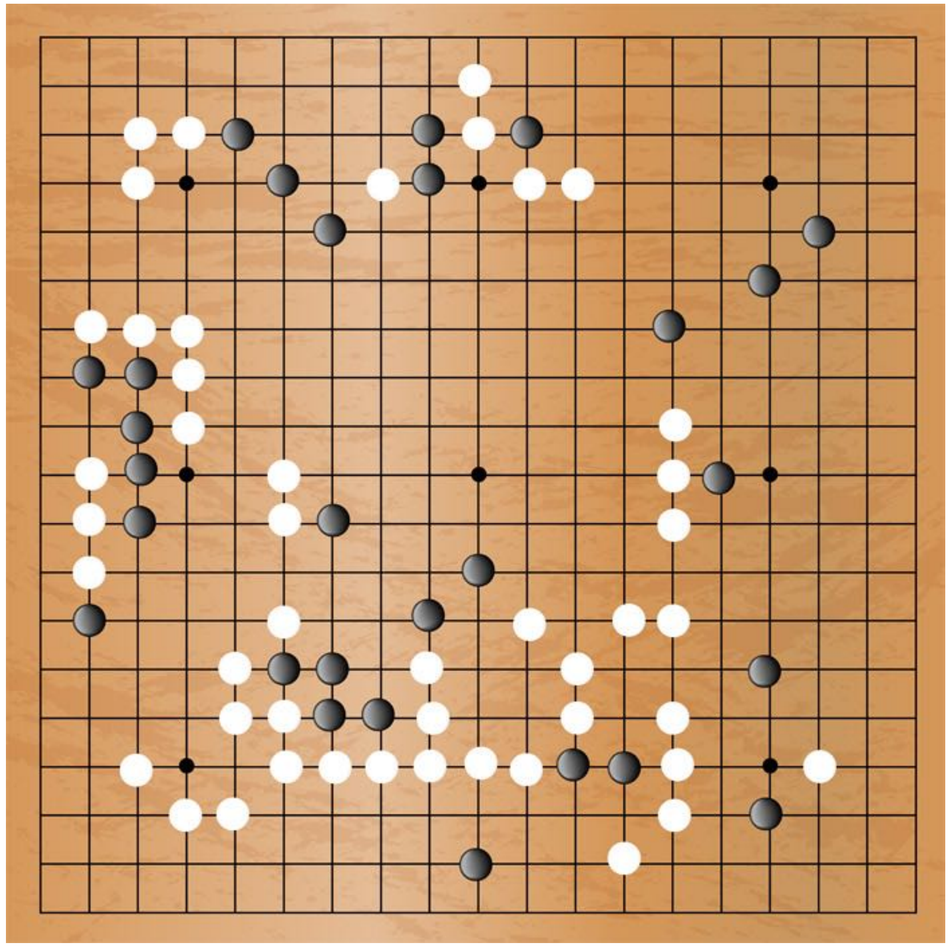
チコちゃんに叱られる！

「▽とび箱って何▽ハマグリの謎▽QRコードの秘密」

[NHK総合] 2022年6月17日 19:57 ~ (45分)

「QRコードの秘密」

QRコードは囲碁



講義では技術仕様を説明してませんが

誤り訂正符号（リードソロモン符号）など興味深い。

日本発で世界で広く便利に使われている技術。