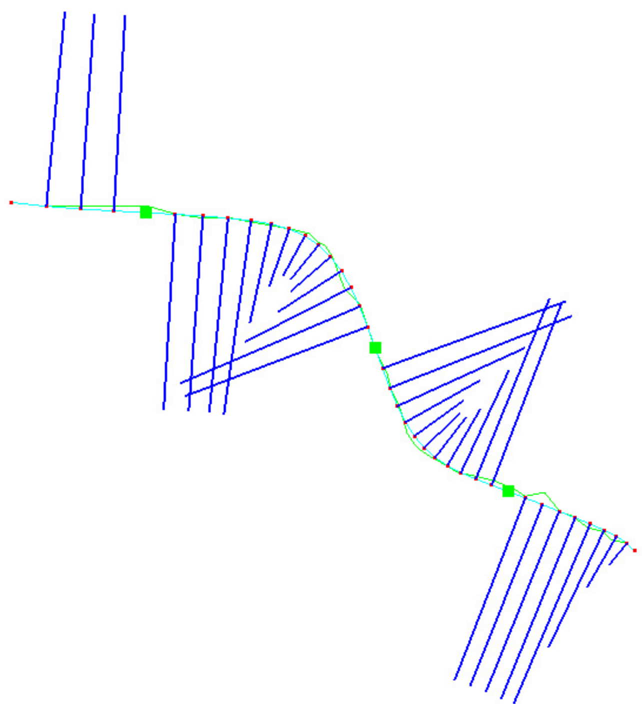
情報工学

2022年度後期　第4回 [10月26日]



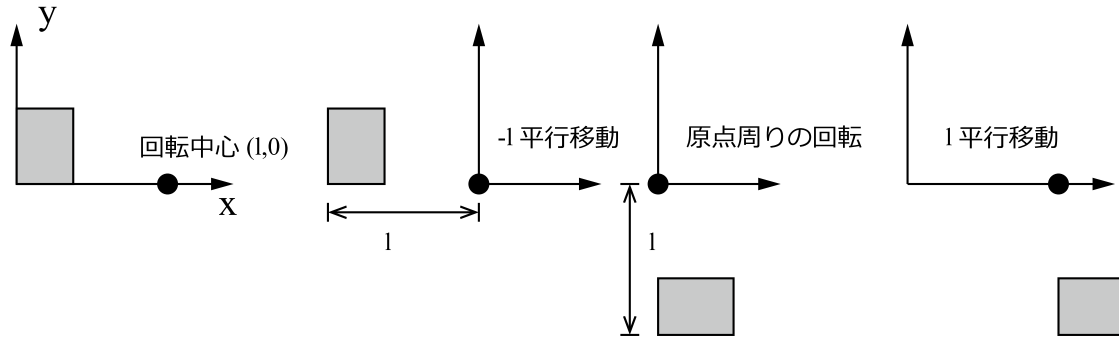
静岡大学

工学研究科機械工学専攻
　　ロボット・計測情報講座
創造科学技術大学院
　　情報科学専攻

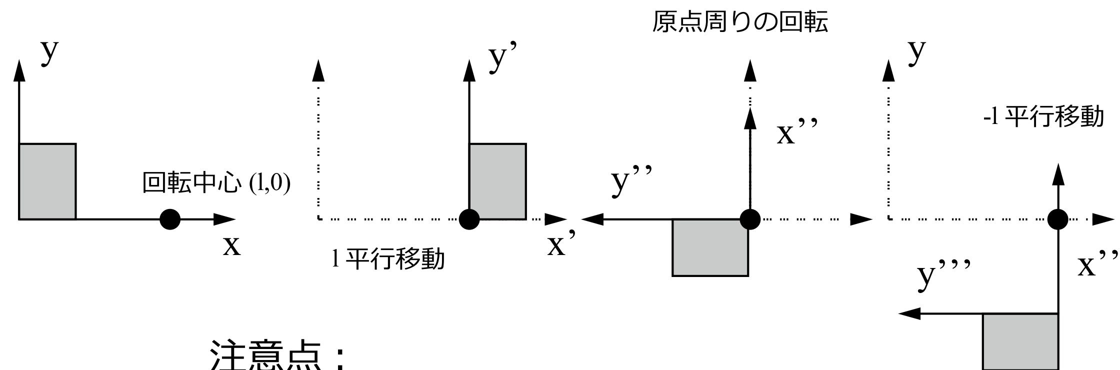
三浦 憲二郎

ローカル座標系による移動

グローバル座標を用いた回転 :(1,0) 周りの 90 度の回転



ローカル座標を用いた回転 :(1,0) 周りの 90 度の回転



注意点：

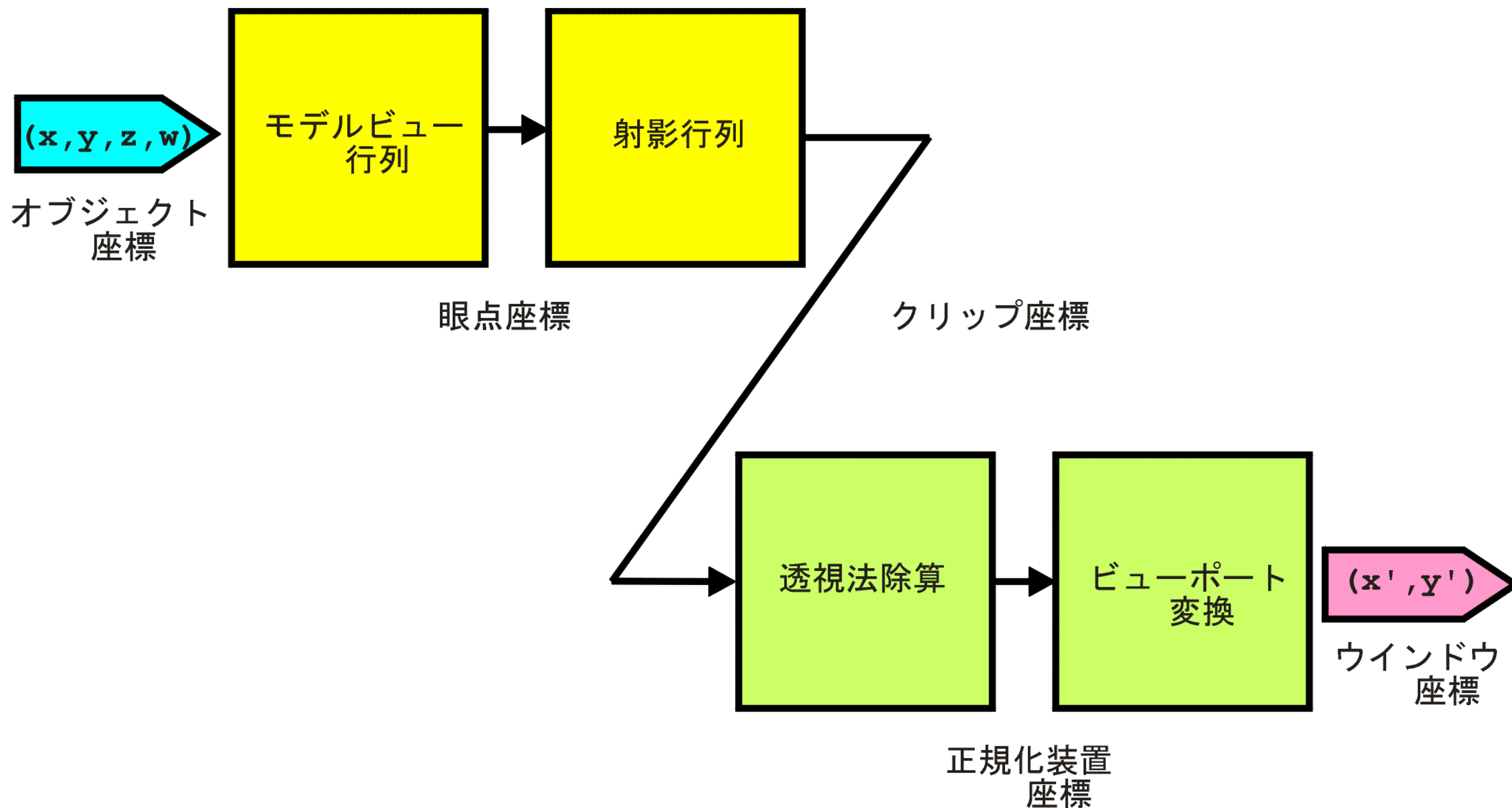
グローバル座標での行列の積の順序を

ローカル座標を用いる場合は反転する.

講義アウトライン [10月26日]

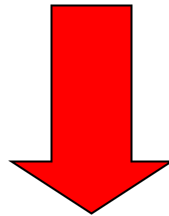
- ビジュアル情報処理
  - 1.3.4 投影変換
  - 1.3.5 いろいろな座標系と変換
- OpenGL
  - 投影変換
  - 曲線の描画
    - トロコイド(外トロコイドと内トロコイド)

頂点変換の手順



射影変換の種類

3D図形をそのままの形で2Dディスプレイに表示することはできない.



3D図形を2D図形に変換　射影変換

「正射影変換 (orthographic projection)」

「透視変換 (perspective projection)」

正射影変換と透視変換

「正射影変換 (orthographic projection)」

1. 無限の位置から立体を見た場合に相当する射影変換
2. 視点と図形との相対的な位置関係とは無関係

「透視変換 (perspective projection)」

1. 視点から図形が離れれば離れるほど小さく変換
2. 見える領域(視体積)がピラミッド型

正射影変換を表す行列

簡単な例として, xy平面への正射影する. この場合, 視点はz軸上のどの位置にあっても(どの位置から図形を見ても)同じように射影され(見れ)る.

変換行列T

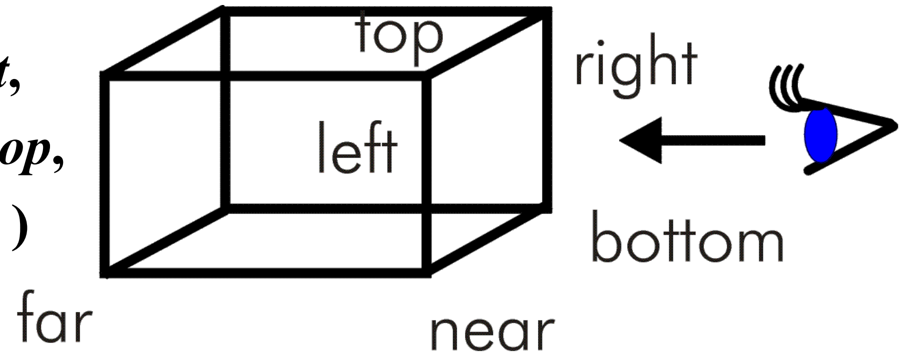
$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

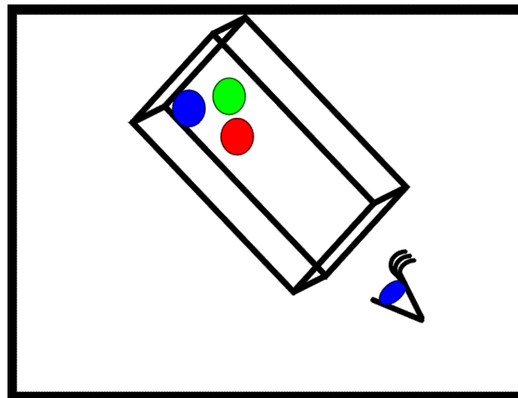
正射影変換:glOrtho()

コマンド glOrtho()

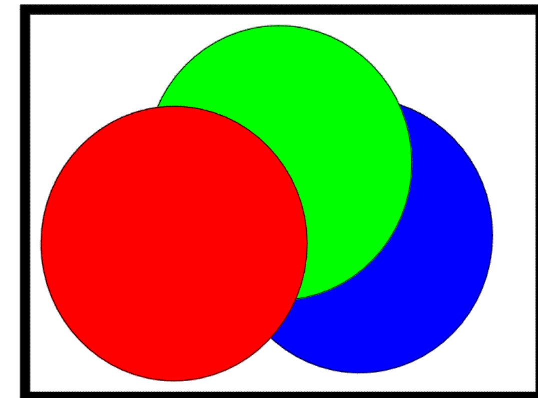
void

glOrtho(GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdouble *far*)





視体積の位置



視点からの図

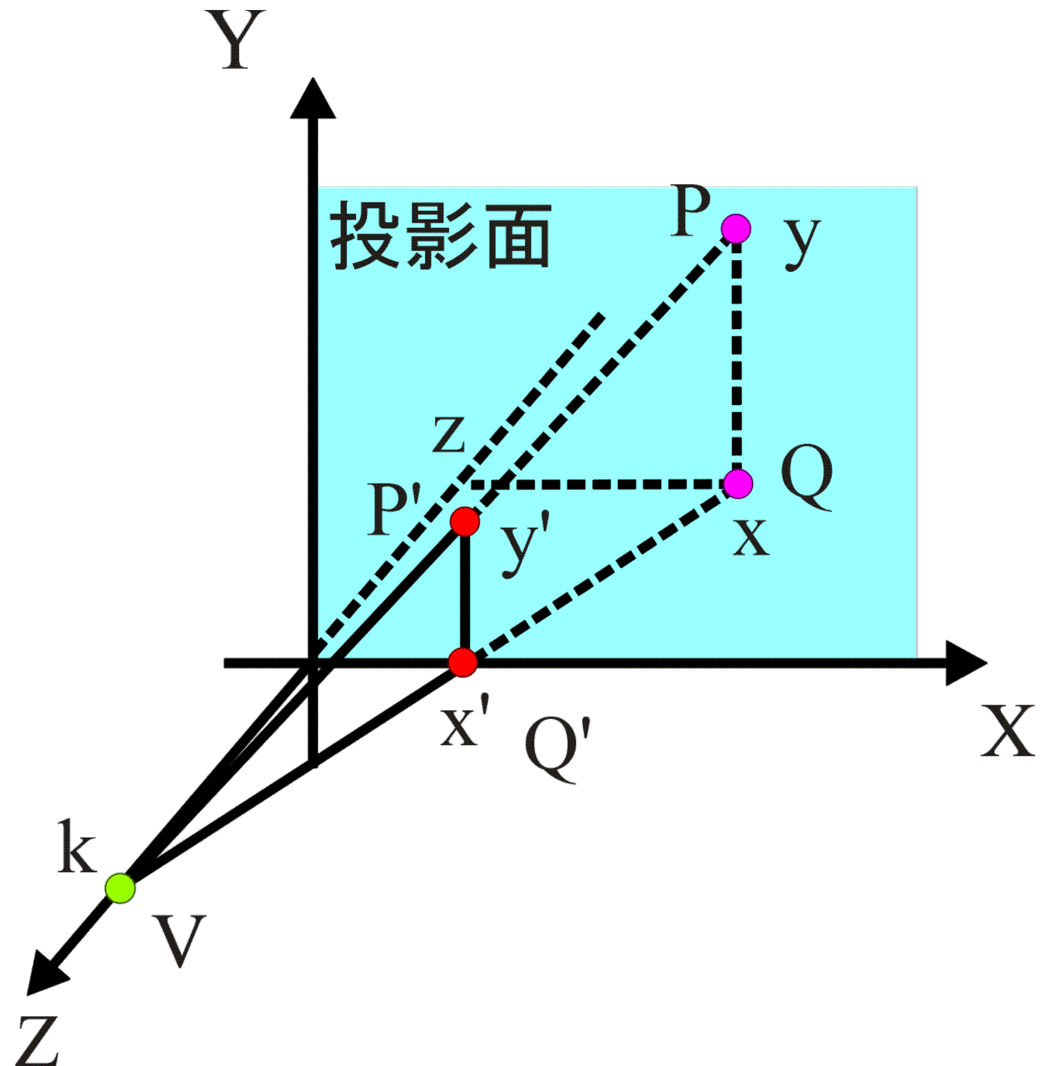
透視変換の計算

視点 V=(0,0,k)

$$\frac{x'}{k} = \frac{x}{-z+k}$$

$$x' = \frac{x}{1 - \frac{z}{k}}$$

$$y' = \frac{y}{1 - \frac{z}{k}}$$



透視変換を表す行列

視点 V=(0,0,k)

$$\frac{x'}{k} = \frac{x}{-z+k}$$

$$x' = \frac{x}{1-\frac{z}{k}}$$

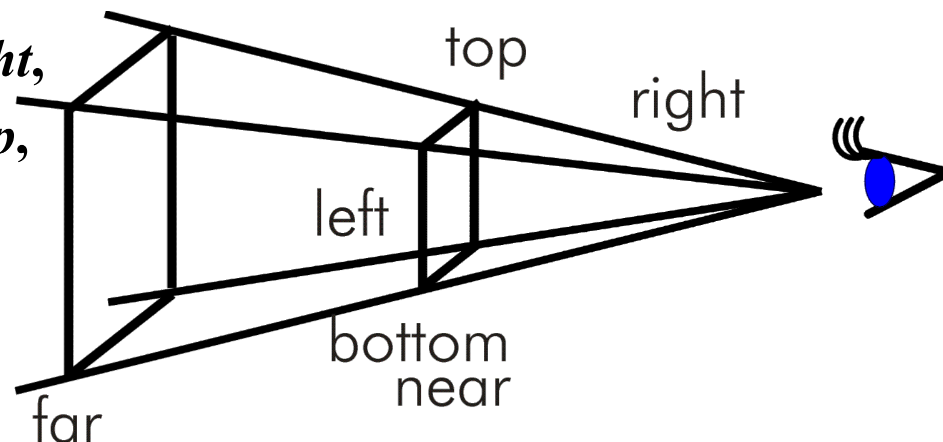
$$y' = \frac{y}{1-\frac{z}{k}}$$

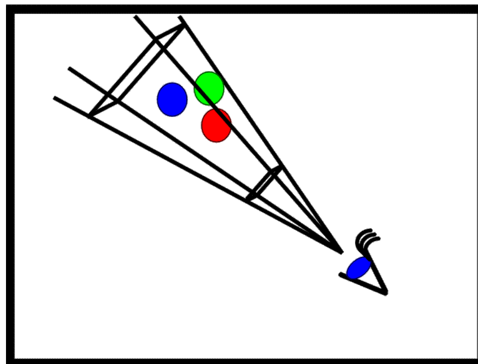
$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{k} & 1 \end{bmatrix}$$

透視変換:glFrustum()

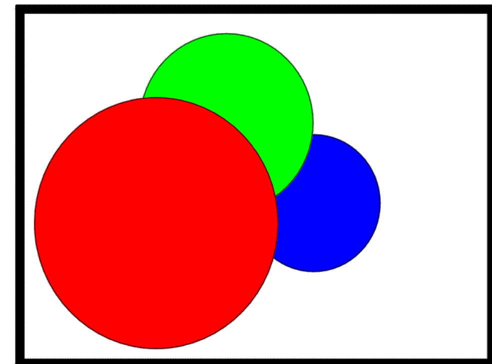
コマンド **glFrustum()**

void
glFrustum(GLdouble *left*, GLdouble *right*,
GLdouble *bottom*, GLdouble *top*,
GLdouble *near*, GLdouble *far*)





視体積の位置

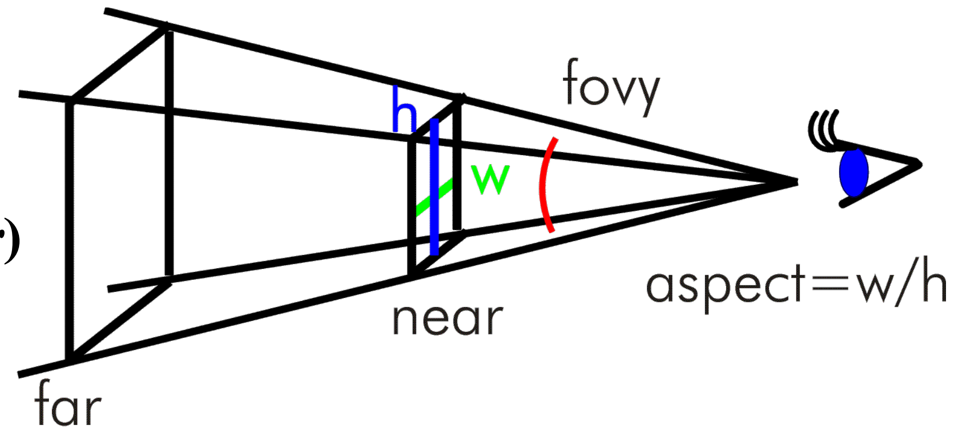


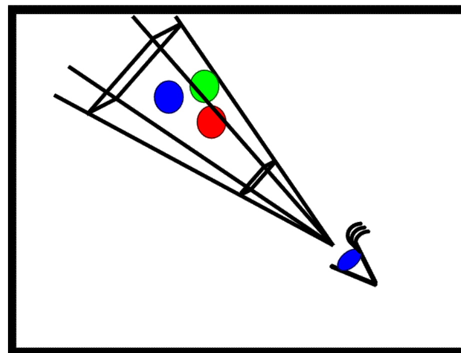
視点からの図

透視変換:gluPerspective()

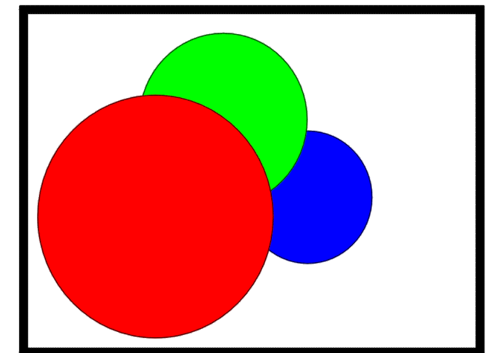
コマンド gluPerspective()

```
void
gluPerspective(GLdouble fovy,
               GLdouble aspect,
               GLdouble near, GLdouble far)
```





視体積の位置



視点からの図

視界変換:gluLookAt()

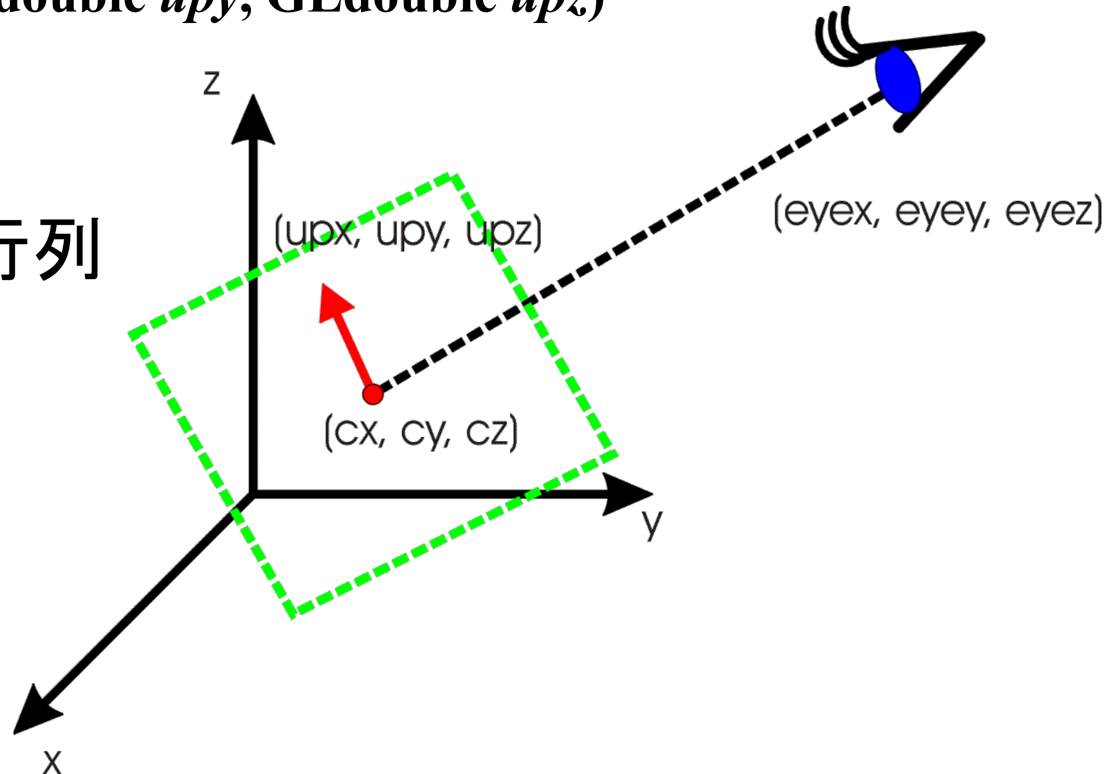
コマンド **gluLookAt()**

void

gluLookAt(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*,
GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*,
GLdouble *upx*, GLdouble *upy*, GLdouble *upz*)

注意：モデルビュー行列

を変更する．

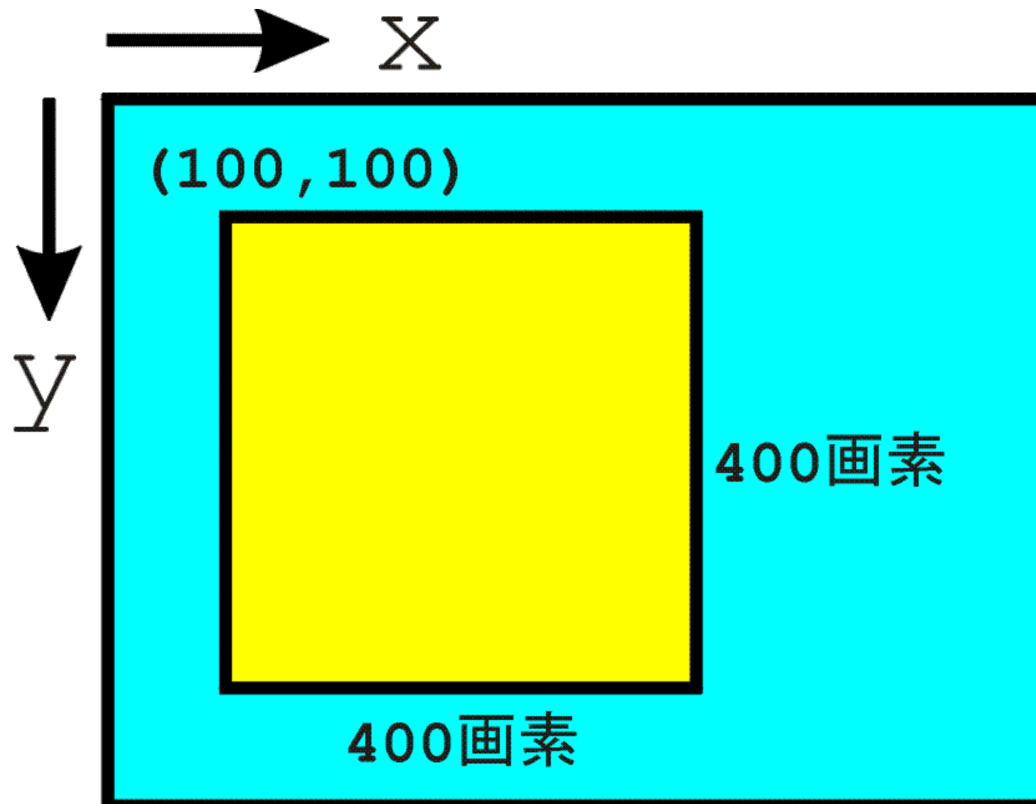


ビューポート変換

コマンド glViewport()

void

glViewport(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*)



ウインドウサイズの変更

```
int
void ourReshape(int width, int height)
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(30.0,
          (GLfloat)width/(GLfloat)height,
                   10.0, 1000.0);
}
```

C言語の基本的な知識と用語

#defineとは？

【説明】defineは文字列の置き換えを行う。次のような書式になる。

```
#define 文字列1 文字列2
```

たとえば、次のように定義する。

```
#define SIZE 500
```

こうするとプログラム中の“SIZE”という文字はコンパイルの時に“500”という数値に置き換わる。たとえば以下のように用いる。

```
#define SIZE 500
int a;
a = SIZE + 50;
```

こうすると、aには550が入る。一般にdefineで定義される名前(上の例ではSIZE)は普通の変数と区別するために大文字で書くのが慣例である。

C言語の解説

- 変数 － その２ (小数)
- 変数 － その３ (グローバル変数)
- 算術演算子 － その１(+ - * / %)
- 数学関数(sin , cos)
- 算術演算子 － その２ (優先順位と結合規則)

変数 - その2（小数）

【説明】

データ（数値）を記憶しておく箱を変数と言う。変数は使用する前に保持するデータの型を指定して宣言しなければならない。すでに整数を保存するデータ型として int を学んだが、小数を保存するデータ型として

float、double

というデータ型がある。C言語では小数のデータ型のことを「浮動小数点型」と呼んでいる。floatとdoubleの違いは小数を保存するサイズでdoubleの方がより多くの桁数を保持できる。絶対的なサイズ（何桁までデータを保持するか）は処理系に依存する。

【用例】

```
double x;  /* double型の変数 x を宣言する。*/
x = 0.05;  /* 変数に値を代入する。         */
```

変数 - その3（グローバル変数）

【説明】

Cでは関数の内部で宣言した変数は、その関数の中だけで有効である。逆に言えばある関数から、別の関数の内部で宣言されている変数をつかうことはできない。このような変数の機能はローカル (局所的) なのでこれを「ローカル変数」と呼んでいる。

それに対してどの関数からでも使うことのできる変数を「グローバル変数」と呼んでいる。グローバル変数は関数の外で定義される。

変数　その3（グローバル変数）

ローカル変数は特定関数の内部でしか使うことができない。このことは逆に言うと、他の関数の内部で同じ名前の変数を使っても衝突が起きない。下の例では、functionという関数とmain関数の中で変数 a を定義しているが、この2つの変数 a は全く関係のない独立した変数として働く。

```
#include<stdio.h>
int g;
function() {
  int a;        /* g も有効 */
}
main() {
  int a, b;    /* g も有効 */
}
```

算術演算子－その1(＋－＊/)

【説明】

演算子	説明	例
+	正符号	`a = +b;`
-	負符号	`a = -b;`
+	加算(足し算)	`a = b + c;`
-	減算(引き算)	`a = b - c;`
*	乗算(かけ算)	`a = b * c;`
/	除算(割り算)	`a = b / c;`
%	余り	`a = b % c;`

算術演算子は数値を計算する。結果として数値を得る。整数除算を行うと余りは切り捨てられる。また整数に対して % 演算子を使うと余りを求めることができる。0で除算すると結果は不定になる。

算術演算子－その1(＋－＊/)

【用例】

```
int a, b, c;
double d, e, f;
b = 5; c = 3;
e = 5.0; f = 3.0;

a = b + c; /* a = 8         */
a = b - c; /* a = 2         */
a = b * c; /* a = 15        */
a = b / c; /* a = 1         */
d = e / f; /* d = 1.6666…  */
a = b % c; /* a = 2         */
```

数学関数（cos　sin）

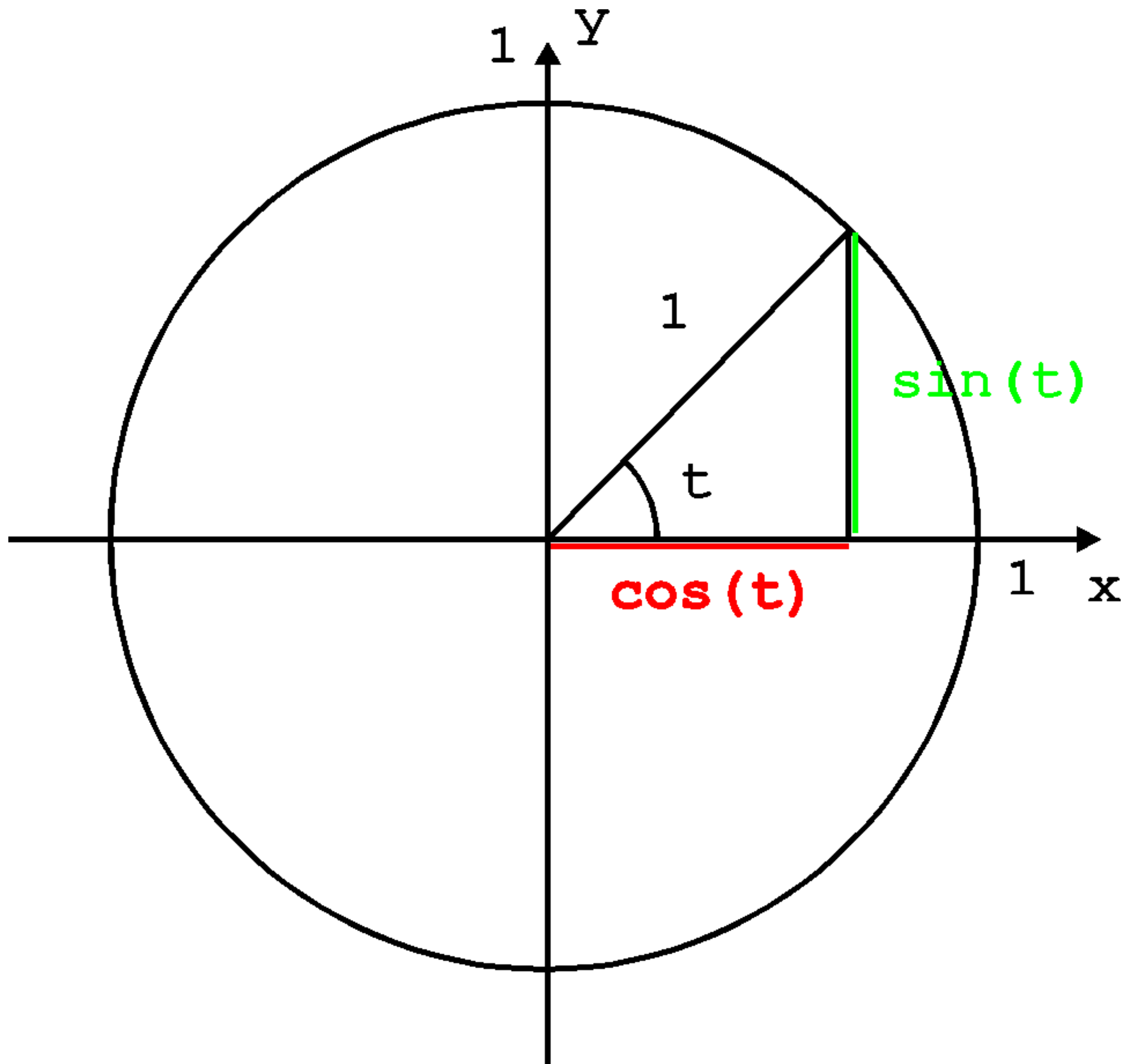
【説明】

Cにはさまざまな数学関数が用意されているが、ここでは sin と cos について説明する。

sin　は引数にラジアン値を与えると、サイン値(正弦値)を返す。
cos　は同様にコサイン値（余弦値）を返す。

引数が度数ではなくラジアン値であることに注意する。ラジアン、sin、cosについては高校の数学の教科書を参照して思い出す。簡単に書くと、
180度は 3.14159… （π）ラジアン
360度は 6.28318… （2π）ラジアン

数学関数（cos　sin）



数学関数（cos sin）

【用例】

```
#define P 3.141592                  /* P は π       */
main() {
  double a, b, c;  a = 75.0;        /* a に角度 75度 */
  b = sin(a/180*P);                 /* sinの計算     */
  c = cos(a/180*P);                 /* cosの計算     */
}
```

注意: a/180*P によって角度(degree)からラジアンに変換

算術演算子
– その2（優先順位と結合規則）

【説明】

これまでに説明した算術演算子（他にも演算子はあるが）には、優先順位がある。これはたとえば、「1+2*3」では

2*3が先に計算されて、結果は7になる

といったことである。もし「1+2」を優先して演算したいときは、優先順位がもっとも高い （ ） を用いて、順位を切り替える。すなわち「(1+2)*3」とする。

結合規則

もうひとつ演算子には結合規則というものがある。一つの式の中に同順位の演算子が存在した場合、結合法則に基づいて優先評価される。たとえば、

```
a = 10;
b = 20;
a = b = 30;
```

では変数 `a` と `b` の値は共に `30` になる。これは演算子 `'='` の結合規則が右結合的(右から左に結合する)となっているからである。このため式はまず右側の「`b = 30`」が実行され、そのあと「`a = b`」が実行される。

算術演算子
– その２（優先順位と結合法則）

また

```
8 / 4 * 2
```

という式では '/' と '*' の優先順位は等しいけれど左結合的であるため、

```
( 8 / 4 ) * 2
```

として働き、結果は 4 となる。

算術演算子 – その2（優先順位と結合法則）

種類	演算子	結合法則
カッコ	( )	→
乗除	* / %	→
加減	+ -	→
比較	< <= > >=	→
等値	== !=	→
論理積	&&	→
論理和	||	→
代入	= += -= *= /= %=	←
コンマ	,	→

第4回課題

```
#include <GL/glut.h>
#include <math.h>

#define PI 3.141592653589793          /* 円周率π                */

void display(void)
{
  double x, y, x1, y1, t;
  glClear(GL_COLOR_BUFFER_BIT);    /* 背景のクリア            */
  glBegin(GL_LINES);               /* 線分を描画する          */
  for(t = 0.0; t < PI; t += 0.3){
    x = cos(t);                    /* x座標を計算する         */
    y = sin(t);                    /* y座標を計算する         */
    if(t != 0.0){
          glVertex2f(x, y); glVertex2f(x1, y1); /* 始点、終点の指定 */
    }
    x1 = x; y1 = y;                /* 座標を保存しておく      */
  }
  glEnd();                         /* 線分の描画終了          */
  glFlush();                       /* 画面を再描画する        */
}
```

課題A

(a)　きれいな一周の円をかけ。

始点，終点の一致，線分の細かさ

(b) 半径を指定し、円をかけ。**EX.** 半径？

```
float r; double s;
scanf("%f", &r); scanf("%lf", &s);
```

(c) **n**重の円をえがけ。**Ex.** 半径，**n**？

(d) 楕円を**n**重にえがけ。**Ex.** 半径，**a**，**b**，**n?**

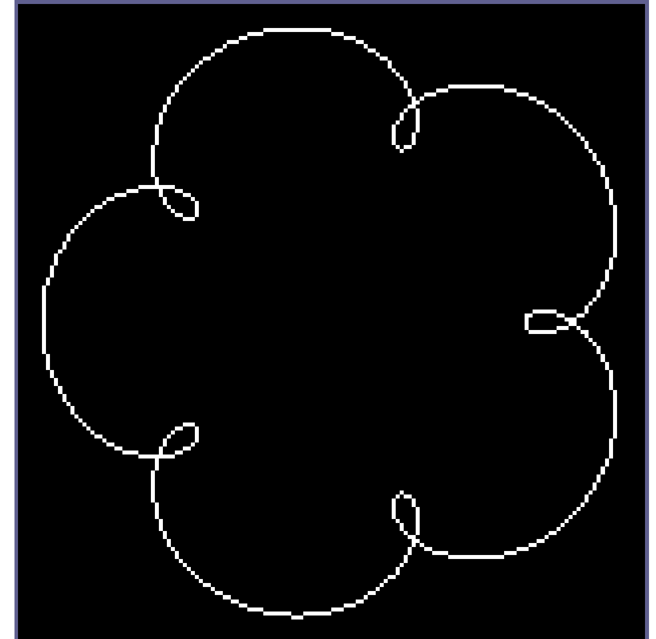
```
(x, y) = (a * cos(t), b * sin(t))
```

課題B

(a) エピトロコイド(**epitrochoid**：外トロコイド)をかけ。

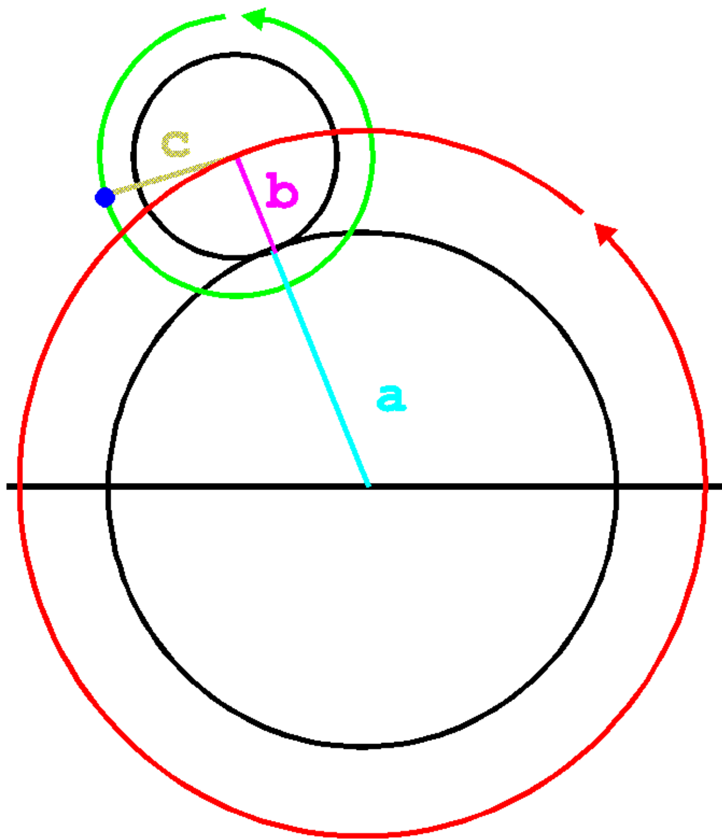
```
Ex. a b c?
x=(a+b)*cos(t)-c*cos((a/b+1.0)*t)
y=(a+b)*sin(t)-c*sin((a/b+1.0)*t)
Ex. a=1.0, b=0.2, c=0.3
```

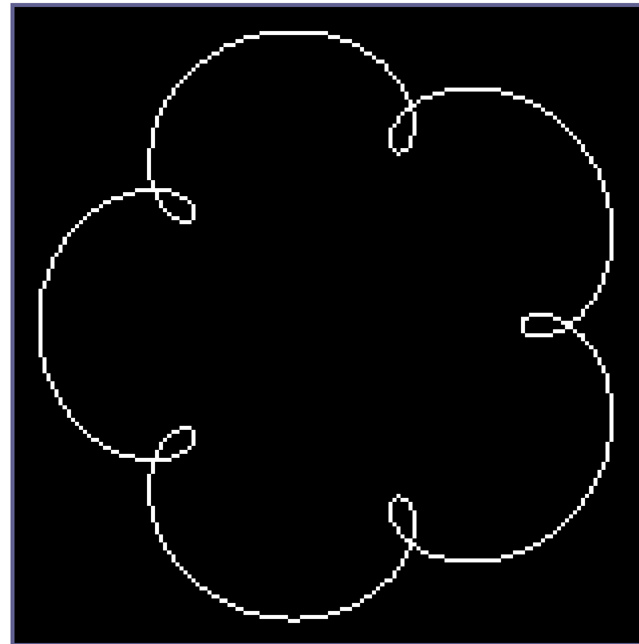
(b) cの値をかえて重ね書きをおこなえ。



課題B

```
x=(a+b)*cos(t)-c*cos((a/b+1.0)*t)
y=(a+b)*sin(t)-c*sin((a/b+1.0)*t)
```





課題C

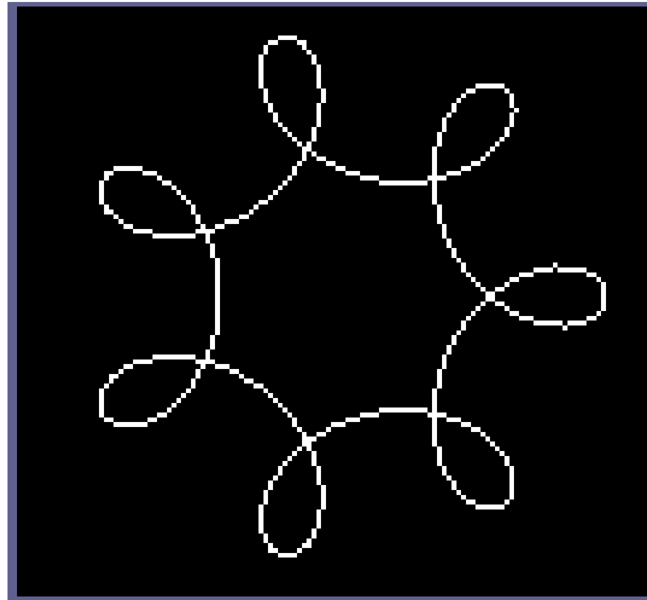
(a) ハイポトロコイド(**hypotrochoid**：内トロコイド)をかけ。

Ex. a b c?

x=(a-b)*cos(t)+c*cos((a/b-1.0)*t)

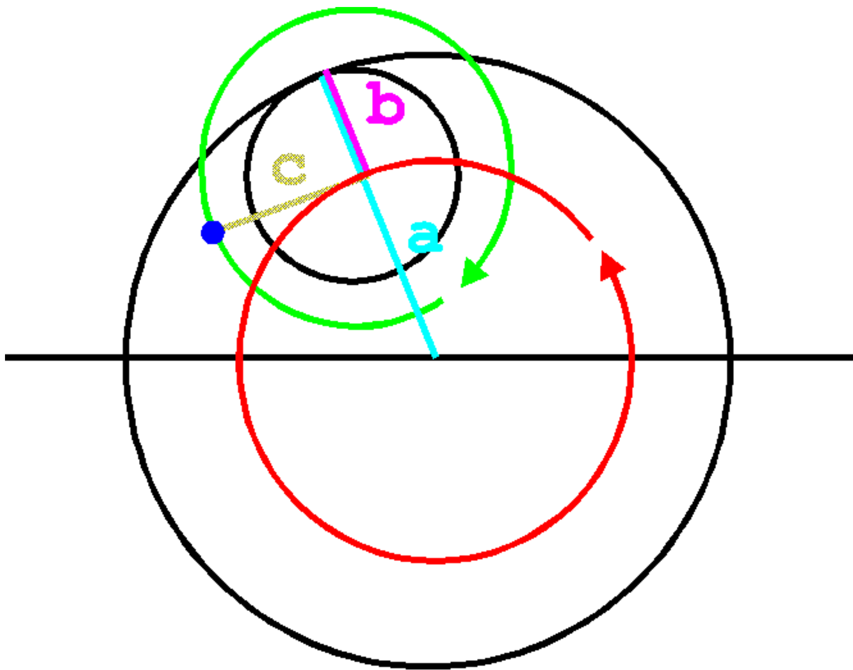
y=(a-b)*sin(t)-c*sin((a/b-1.0)*t)

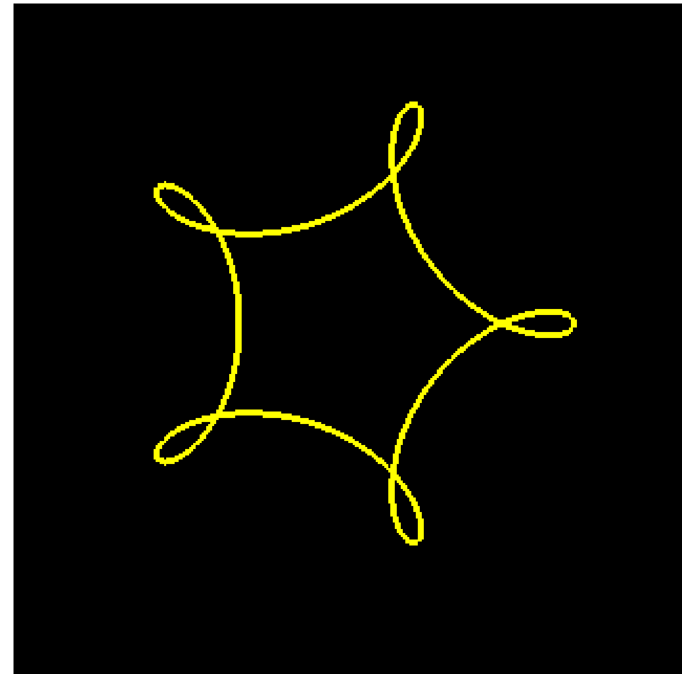
(b) **c**の値をかえて重ね書きをおこなえ。



課題C

```
x=(a-b)*cos(t)+c*cos((a/b-1.0)*t)
y=(a-b)*sin(t)-c*sin((a/b-1.0)*t)
```





まとめ

- ビジュアル情報処理
  - 1.3.4 投影変換
  - 1.3.5 いろいろな座標系と変換
- OpenGL
  - 投影変換
  - 曲線の描画
    - トロコイド(外トロコイドと内トロコイド)