

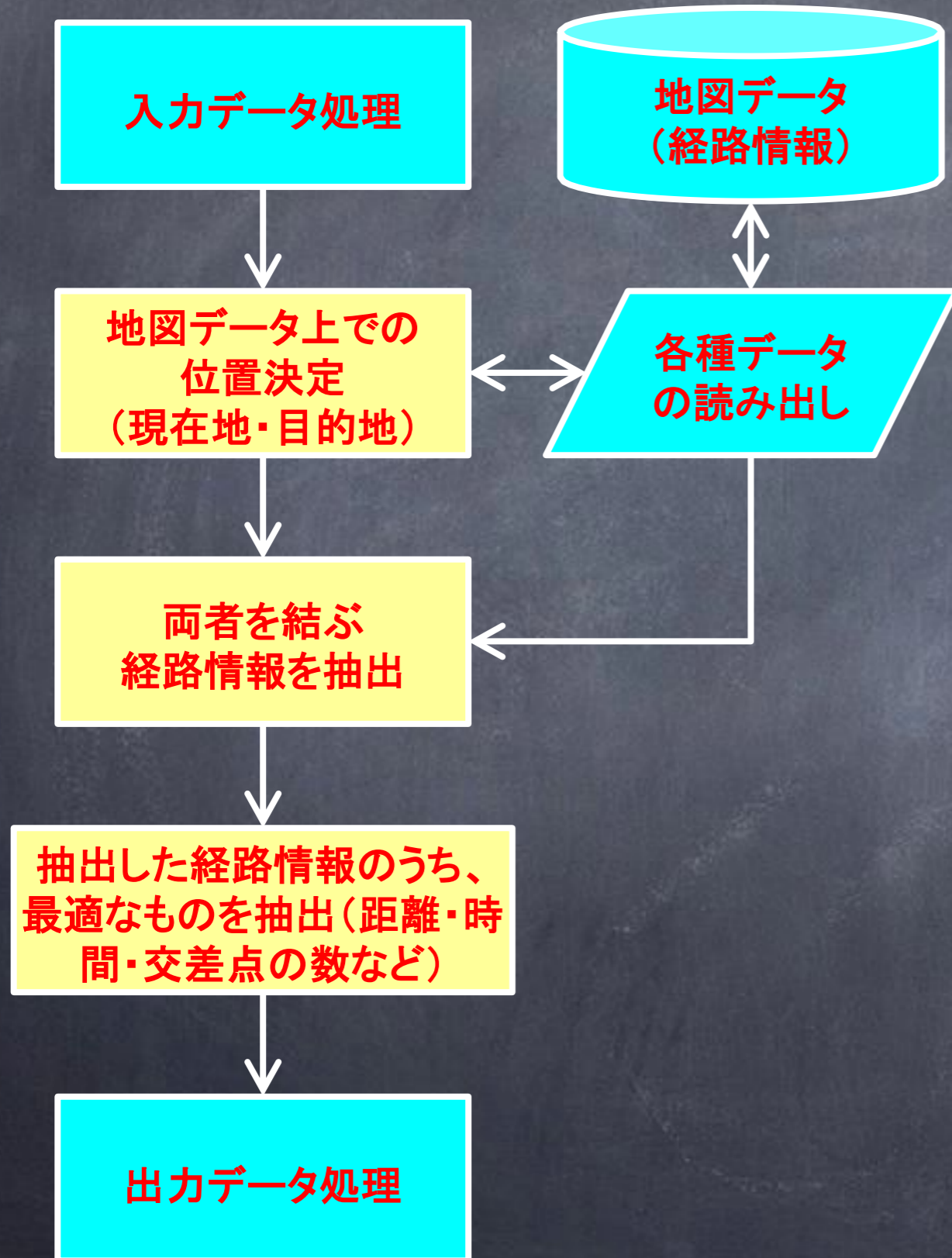
コンピュータ実習
第9回 2025年12月4日

カーナビゲーション3
グラフィックスによる地図表示

担当: 相澤 直人, 宍戸 博紀

コンピュータ実習(第9回)

カーナビ作成に向けて



地図データの検索と抽出
→ 入出力・変数・構造体

抽出したデータを元に経路を割り出す
→ 条件検索(サーチ)

割り出した経路のうち、最適なものを選ぶ
→ 条件並び替え(ソート)

コンピュータ実習(第8回)

サーチングとソーティング

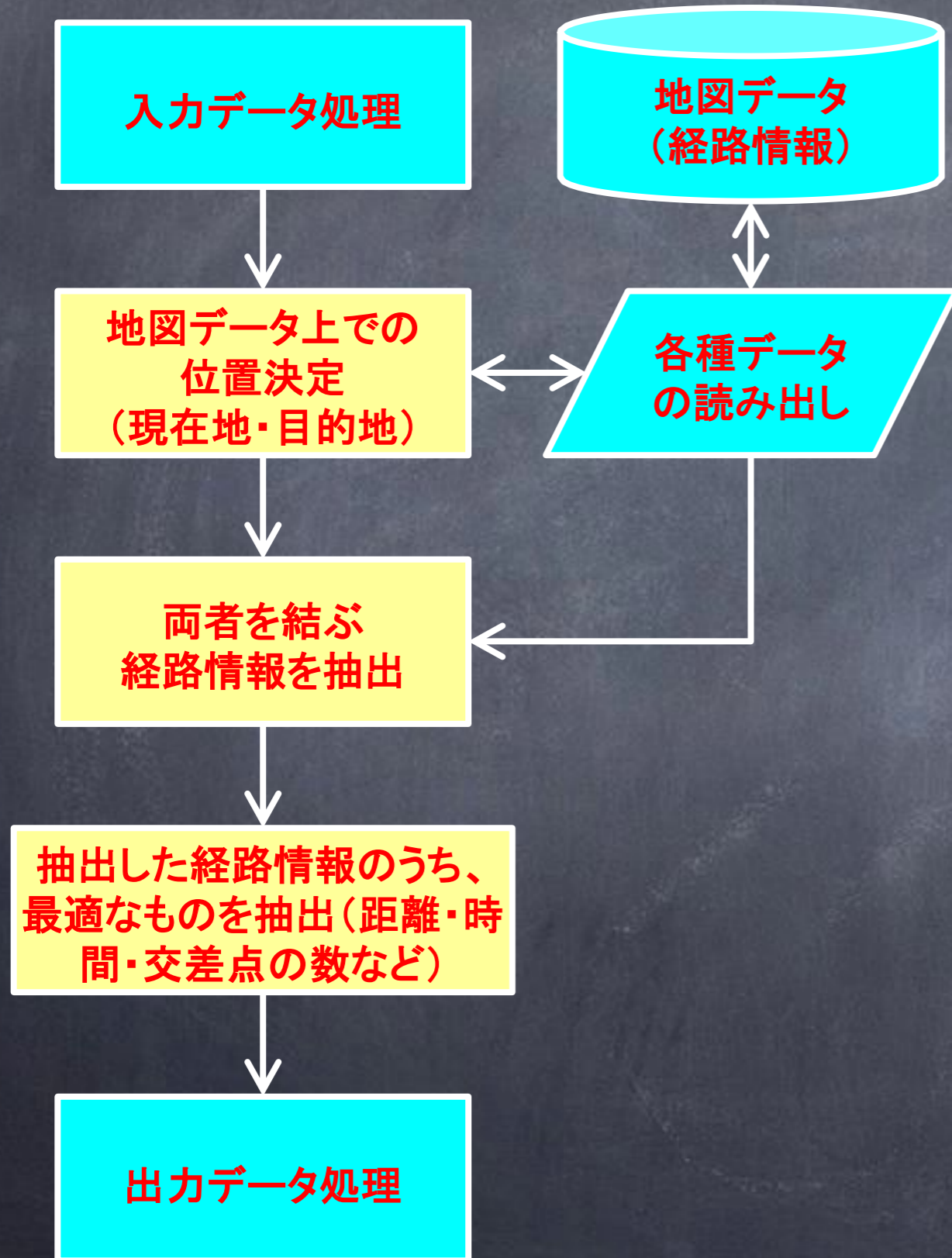
演習2:ソート演習(ex8-2)

- map2.datのデータに対して入力された座標から近い交差点順にデータを並べ替えて出力させよ.

※アルゴリズムはバブルソートに限定しない.

コンピュータ実習(第9回)

カーナビ作成に向けて



地図データの検索と抽出
→ 入出力・変数・構造体

抽出したデータを元に経路を割り出す
→ 条件検索(サーチ)

割り出した経路のうち、最適なものを選ぶ
→ 条件並び替え(ソート)

コンピュータ実習(第9回)

今後の授業の進行

1. 地図データを元に、目的地を決定
 - 地図データの読み込み (11/20)
 - 目的地(地図データ)の検索 (11/27)
2. 経路を探索
 - 条件に基づく経路の設定 (12/4: 簡易版)
 - 最適経路の計算 (12/11)
3. 経路をドライバーに分かりやすく提示
 - 画面上で車をアニメーション表示 (12/4)



本日の講義

コンピュータ実習(第9回)

今回の内容: 地図表示



カーナビ作成に必要な要素

- データの用意
- データの読み込み
- データの検索(並べ替え)
現在地・目的地・最短経路
- **グラフィックス表示**
- 使い勝手・見栄えの改良

コンピュータ実習(第9回)

今回は情報を見やすくする**可視化**を行います

カーナビの画面



方角

現在地

地図の拡大・縮小

目的地までの距離

縮尺

現在地名

経路と混雑状況

コンピュータ実習(第9回)

Web教材 地図のグラフィックス表示

まずは地図を画面に表示させます。Web教材 map.c を完成させてください。

ヒント

- map_show関数以外はこれまでの演習で作成してきていますので、活用して下さい。
- 構造体にデータを読み込み、各交差点のX,Y座標を使って隣接交差点との間に線を引いていきます。
- グラフィックスを扱うソースファイルのコンパイル方法は覚えていますね？

```
cc map.c -g -O2 -Wall -I/usr/include/freetype2  
-lftgl -lglfw -IGLU -IGL -IX11 -IXrandr -o map
```

コンピュータ実習(第9回)

エイリアスの設定

前ページにあるように、今回の地図を表示させるソースファイルのコンパイルを行うためには、

```
cc map.c -g -O2 -Wall -Wno-unused-result  
-I/usr/include/freetype2 -lftgl -lglfw -lGLU -lGL  
-lX11 -lXrandr -lm -o map
```

となり、一回一回すべて入力するのは面倒。。。

そこで、**エイリアスを設定することをおススメします。**

エイリアスの設定方法は、第6回の講義参考資料
にありましたよね・・・？

コンピュータ実習(第9回)

エイリアスの設定の例

- ホームディレクトリに、`glcc.sh`というファイルを作成。
ファイル中に以下を書き込み、保存。

```
#!/bin/bash
```

```
cc $1.c -g -O2 -Wall -Wno-unused-result
```

```
-I/usr/include/freetype2 -lftgl -lglfw -lGLU -lGL -lX11
```

```
-lXrandr -lm -o $1
```

- `glcc.sh`に操作権限を与えるために以下を入力

```
chmod +x glcc.sh
```

- ホームディレクトリにある`.bashrc`に以下の一文を追加

```
alias glcc='/home/gakusei2017/glcc.sh'
```

- 上記の変更をh反映させるために以下を入力

```
source ~/.bashrc
```

cc ~ \$1は
1行で書く!

コンピュータ実習(第9回)

関数map_show()の考え方

関数 map_show()の中で行う処理は以下の通りです。

```
for(交差点数分の繰り返し){  
  ・各交差点の位置に円を描く  
  ・各交差点の位置に交差点名を表示する  
    for(隣接交差点簿分の繰り返し){  
      ・隣接する交差点へ直線を引く  
        (交差道路の表示)  
    }  
}
```

コンピュータ実習(第9回)

map.cのイメージ

1. 交差点位置に丸を書く
(円を書く関数を使う
関数の中身は第6回参照)



コンピュータ実習(第9回)

map.cのイメージ

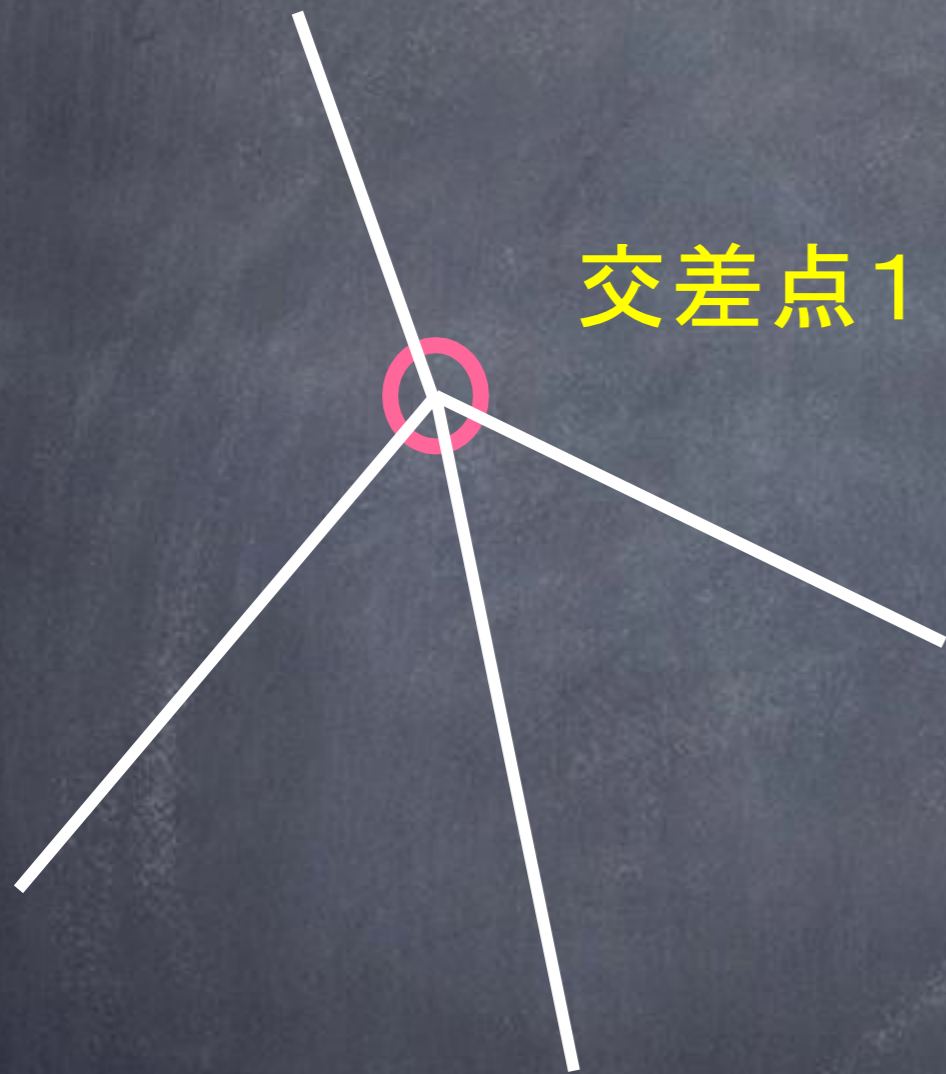


交差点1

1. 交差点位置に丸を書く
(円を書く関数を使う
関数の中身は第6回参照)
2. 交差点の名前を書く
(サンプルにある
draw_outtextxy関数を使う)

コンピュータ実習(第9回)

map.cのイメージ

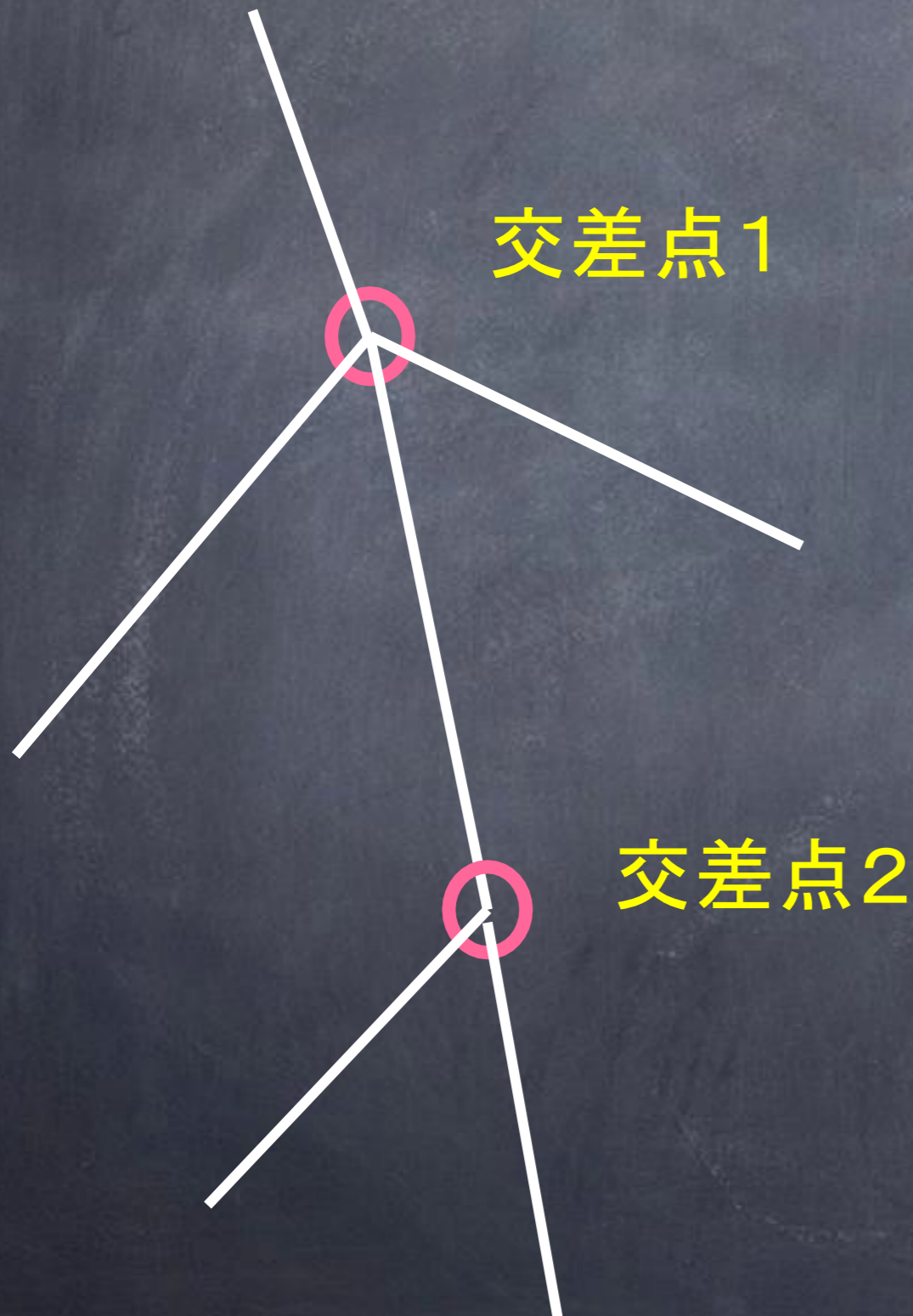


1. 交差点位置に丸を書く
(円を書く関数を使う
関数の中身は第6回参照)
2. 交差点の名前を書く
(サンプルにある
draw_outtextxy関数を使う)
3. 隣接交差点の数だけ、隣接交
差点の座標位置まで直線を
引く
(for文を使う/OpenGLの
機能で直線を引く)

コンピュータ実習(第9回)

map.cのイメージ

次の



1. 交差点位置に丸を書く
(円を書く関数を使う
関数の中身は第6回参照)
2. 交差点の名前を書く
(サンプルにある
draw_outtextxy関数を使う)
3. 隣接交差点の数だけ、隣接交
差点の座標位置まで直線を
引く
(for文を使う/OpenGLの
機能で直線を引く)

コンピュータ実習(第9回)

関数map_show()について

```
/* 道路網の表示(新しく作成する部分) */
```

```
void map_show(int crossing_number) {
```

```
    /* 指定した交差点の数だけ、ウィンドウに交差点(地図)を表示する。  
       まず各交差点には赤丸を描き、隣接交差点への道路を直線で描く */
```

```
    int i, j;
```

```
    /* 交差点ごとのループ */
```

```
    for (i = 0; i < crossing_number; i++) {
```

```
        double x0 = cross[i].pos.x;
```

```
        double y0 = cross[i].pos.y;
```

```
        /* 交差点を表す円を描く */
```

```
        /* 交差点の名前を書く */
```

```
        /* 交差点から伸びる道路を描く */
```

```
    }
```

```
}
```

コンピュータ実習(第9回)

おまけ: glColor3dで表示する色について

OpenGLでのグラフィックス表示では、

`glColor3d(1.0, 1.0, 0.0)` (この場合は黄色)

のように、カッコ内に3つの数字を入れて、色を設定する。

その中の数字の意味は、

1つ目: 赤の割合

2つ目: 緑の割合

3つ目: 青の割合

を示す。すなわち、光の3原色のそれぞれの割合を与えているわけです。

以下のサイトでは、色と数字の組み合わせの目安が載っていますので、参考にして下さい。

<http://www.motohasi.net/html/ColorOpenGL/Color3.php>

コンピュータ実習(第9回)

演習1 アニメーションによる可視化

交差点トレースプログラム

交差点を順にマーカーが移動する、アニメーション・プログラムを作成して下さい。

まず、先ほどの`map.c`を`mobile.c`に名前を変えて保存しておいて下さい。これを改造していきます。

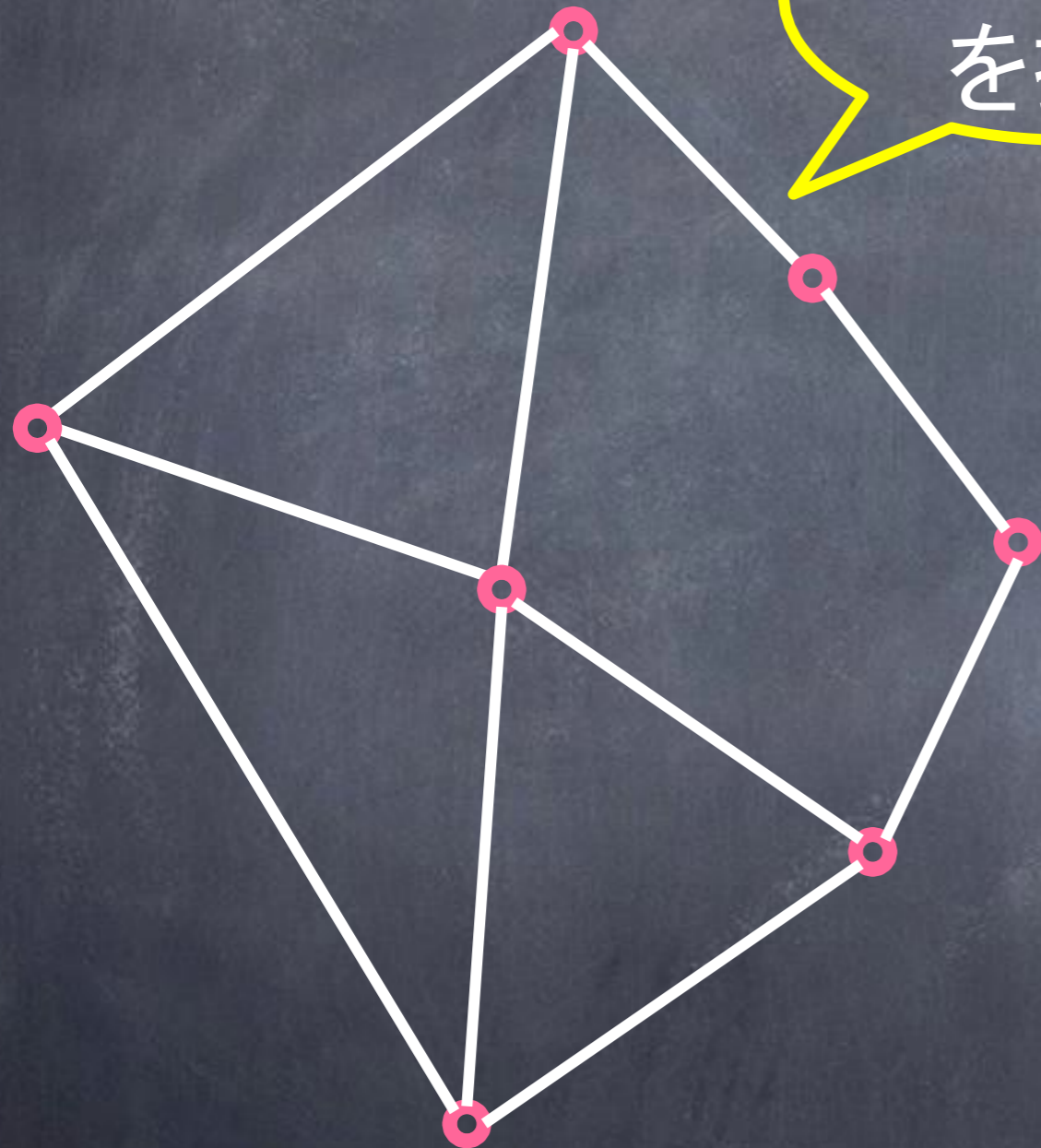
追加が必要な関数

1. 目的地までの経路を決める関数
2. 隣接する2つの交差点間をマーカーが移動するアニメーション表示をする関数
3. 2の処理を目的地まで繰り返させる関数

コンピュータ実習(第9回)

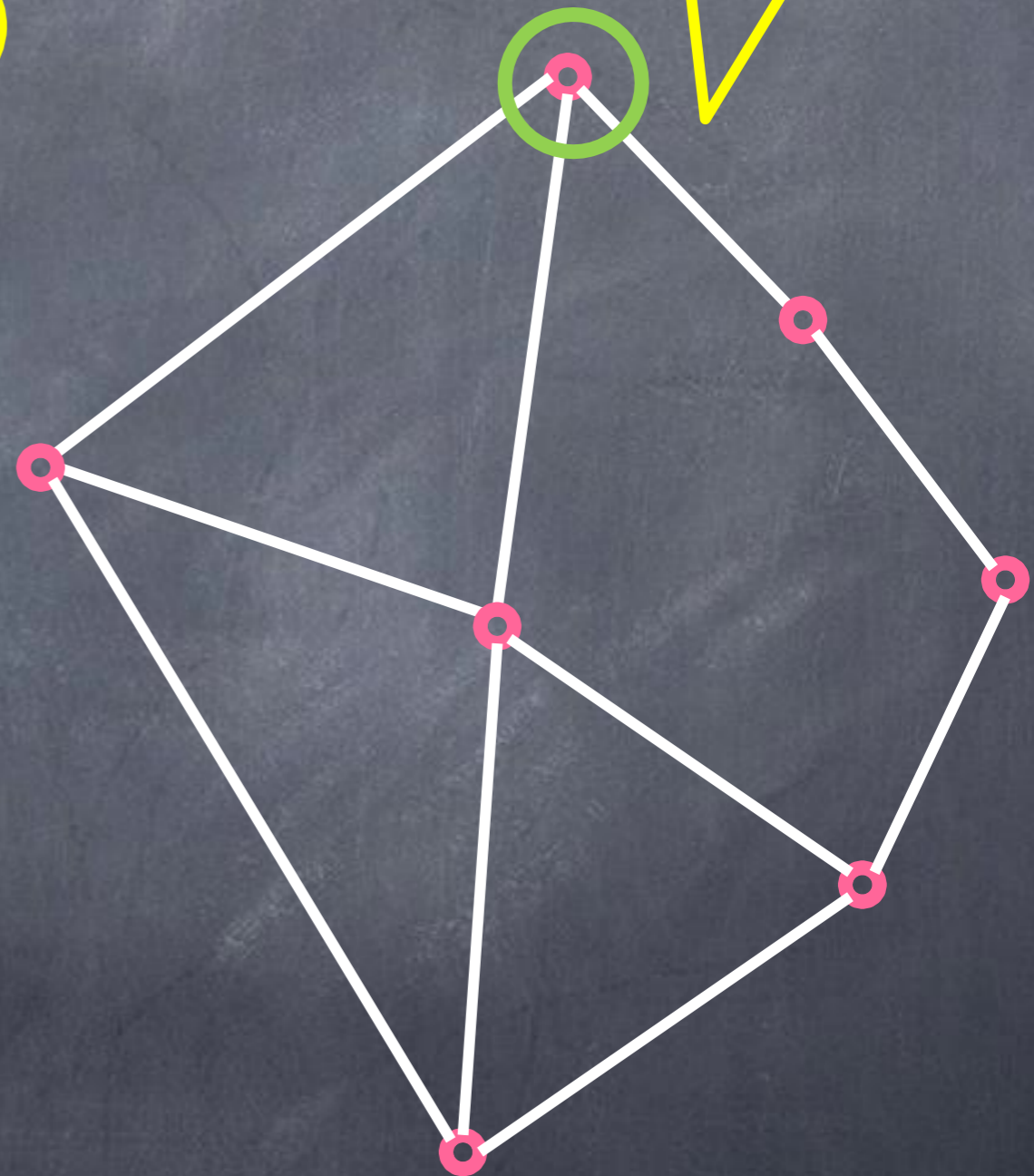
車の移動アニメーションのイメージ

1. 地図を描く



@フロントバッファ

2. 車を描く

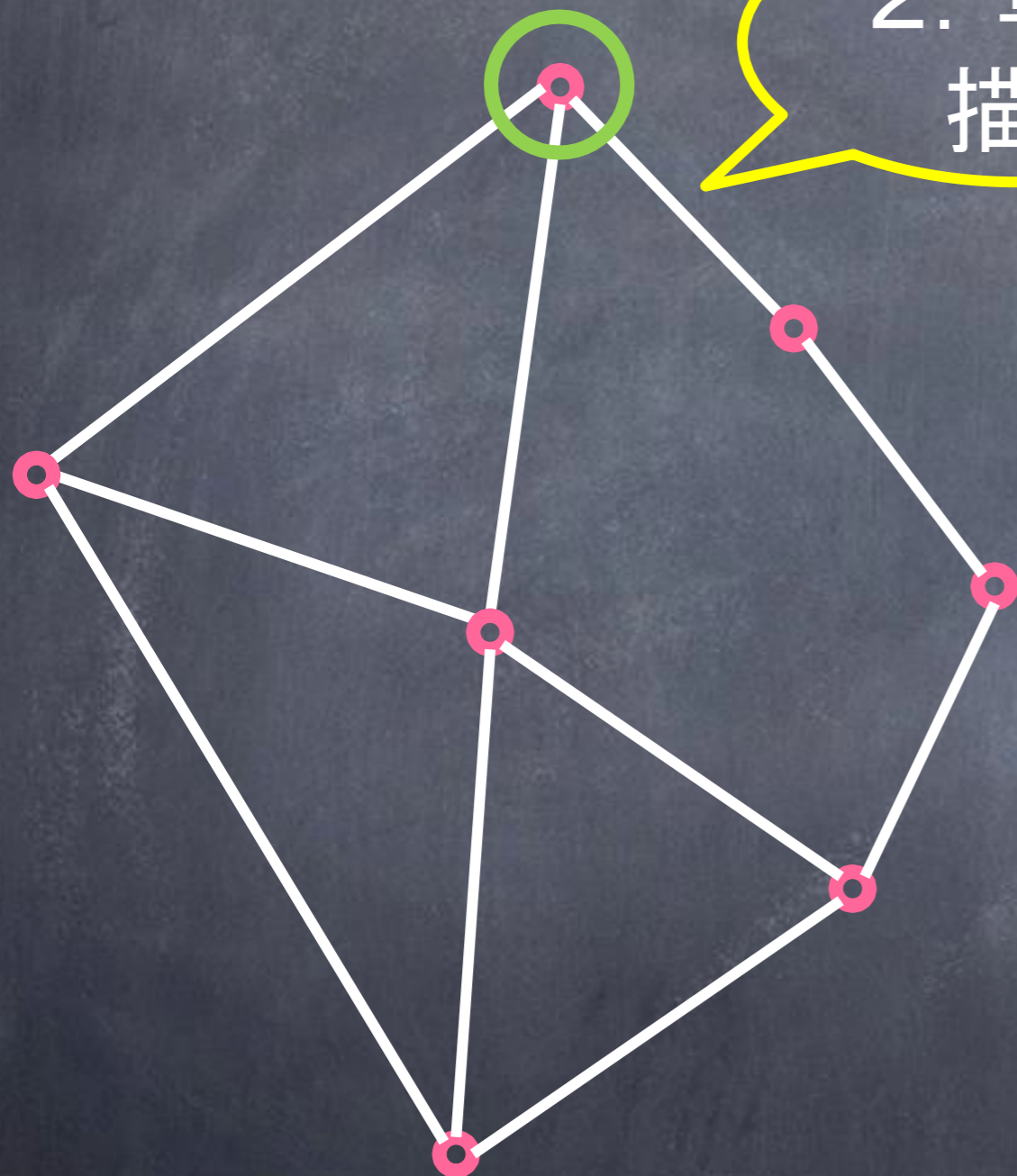


@フロントバッファ

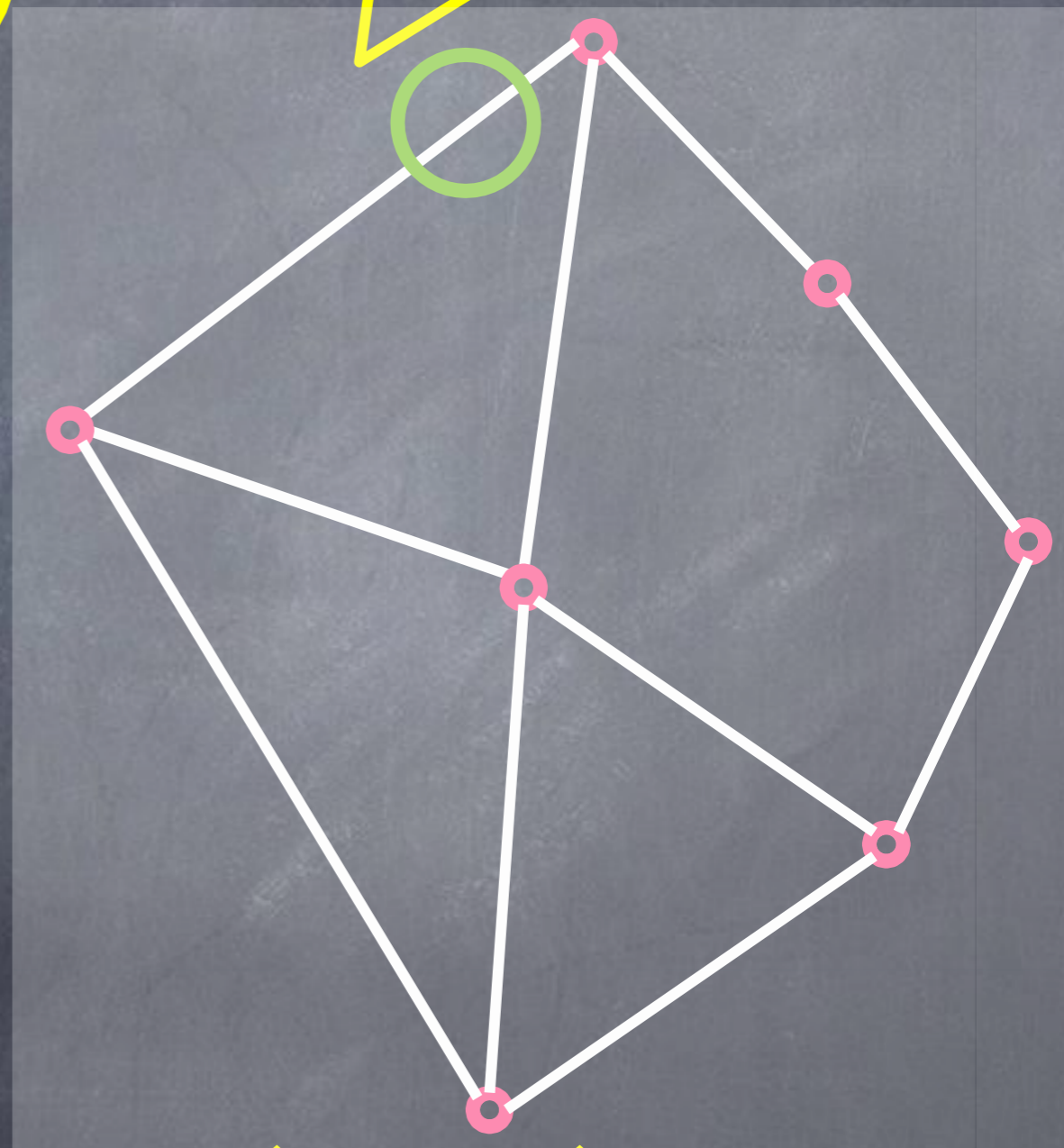
コンピュータ実習(第9回)

車の移動アニメーションのイメージ

2. 車を描く



3. 地図&少し移動した車を描く

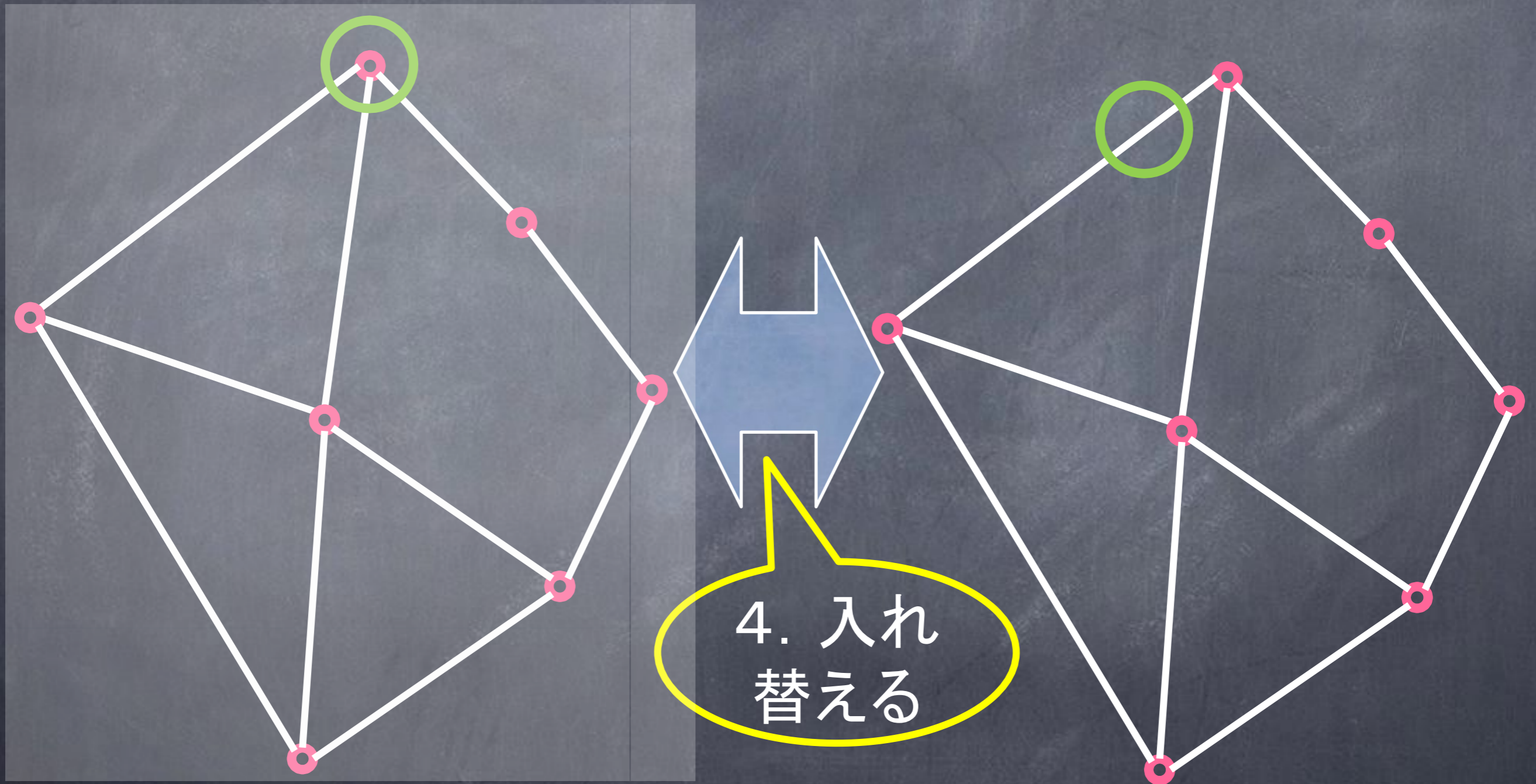


@フロントバッファ

@バックバッファ

コンピュータ実習(第9回)

車の移動アニメーションのイメージ



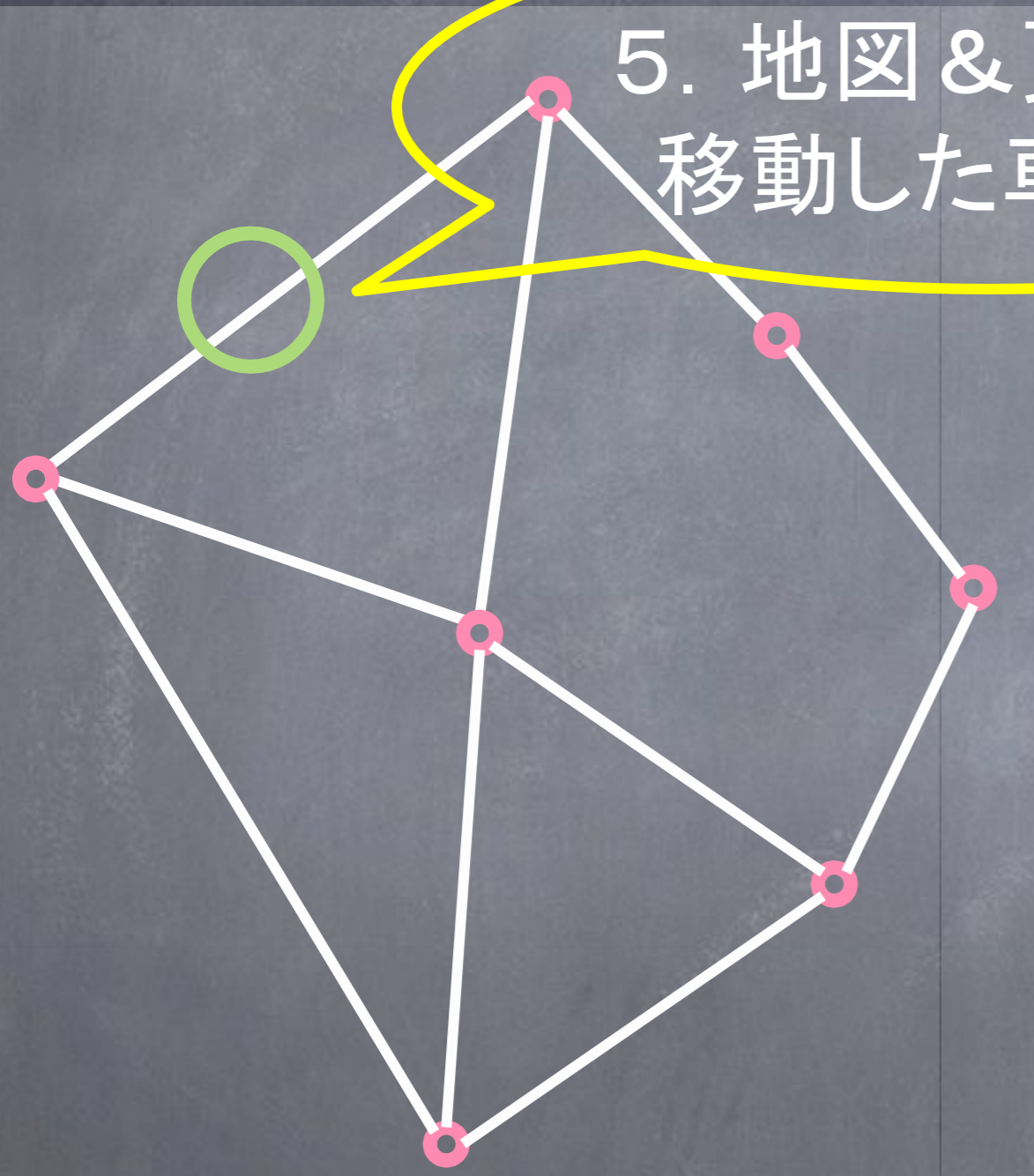
@フロントバッファ
→@バックバッファ

@バックバッファ
→@フロントバッファ

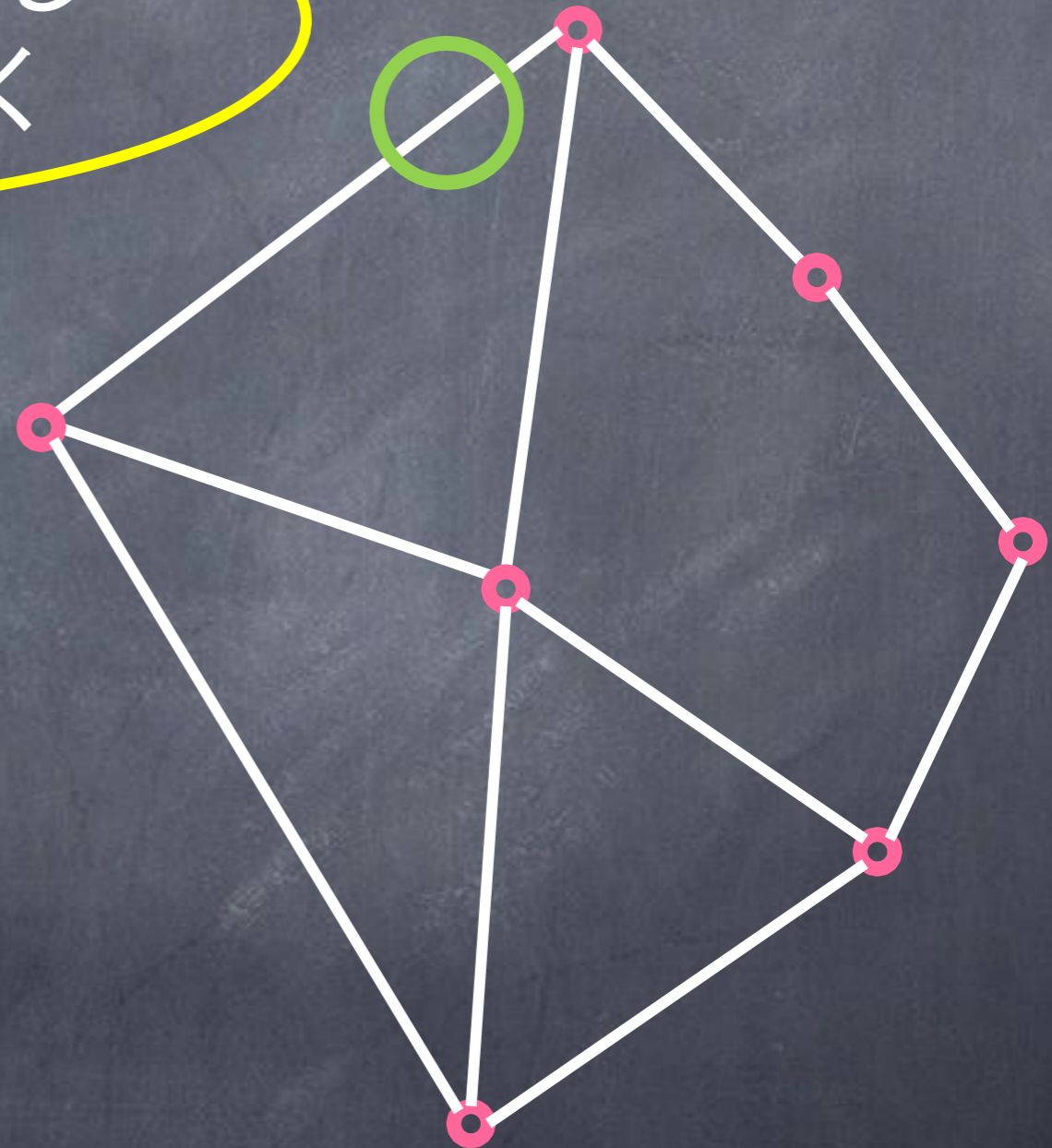
コンピュータ実習(第9回)

車の移動アニメーションのイメージ

5. 地図&更に少し
移動した車を描く



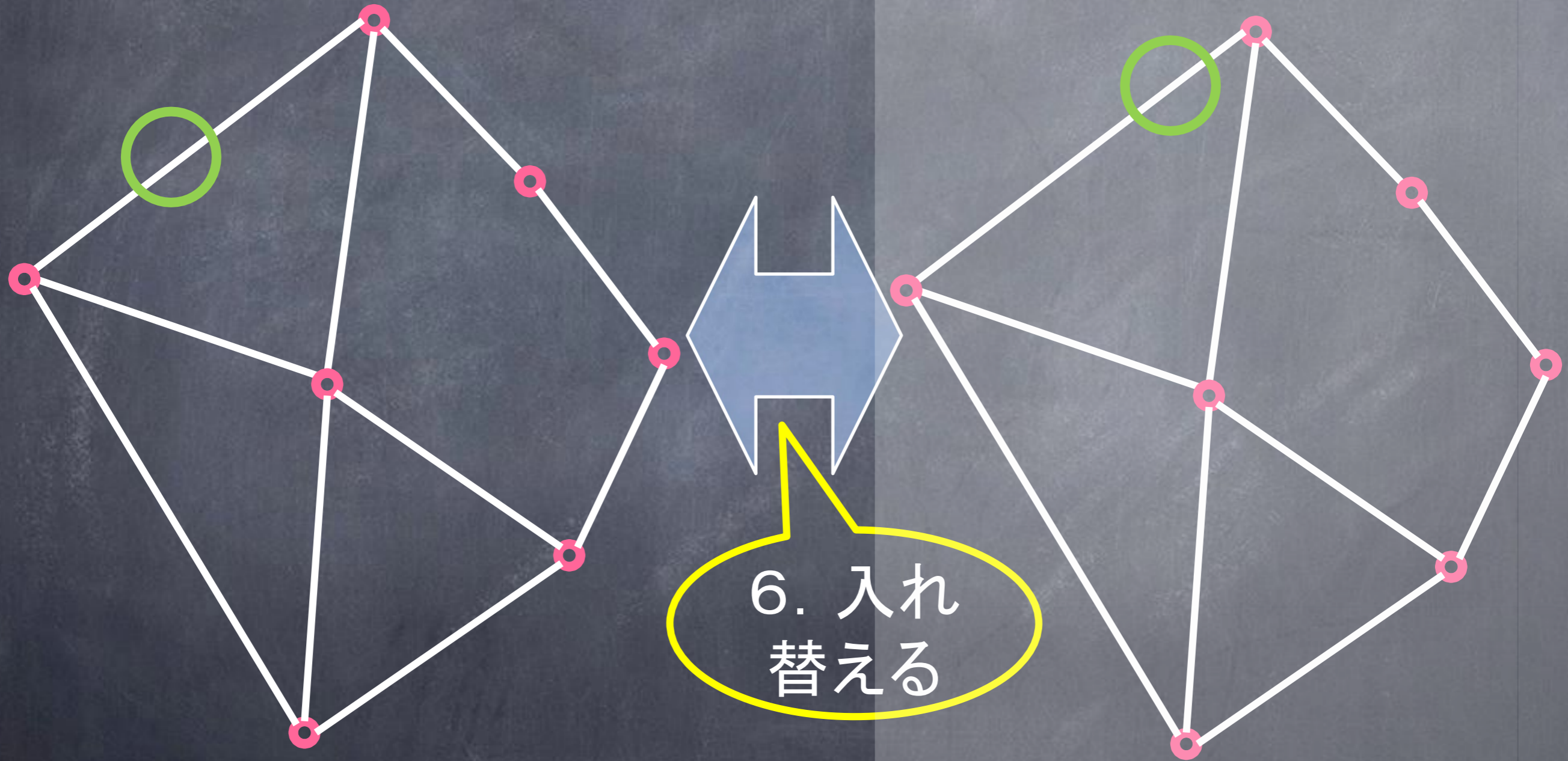
@バックバッファ



@フロントバッファ

コンピュータ実習(第9回)

車の移動アニメーションのイメージ



@バックバッファ
→@フロントバッファ

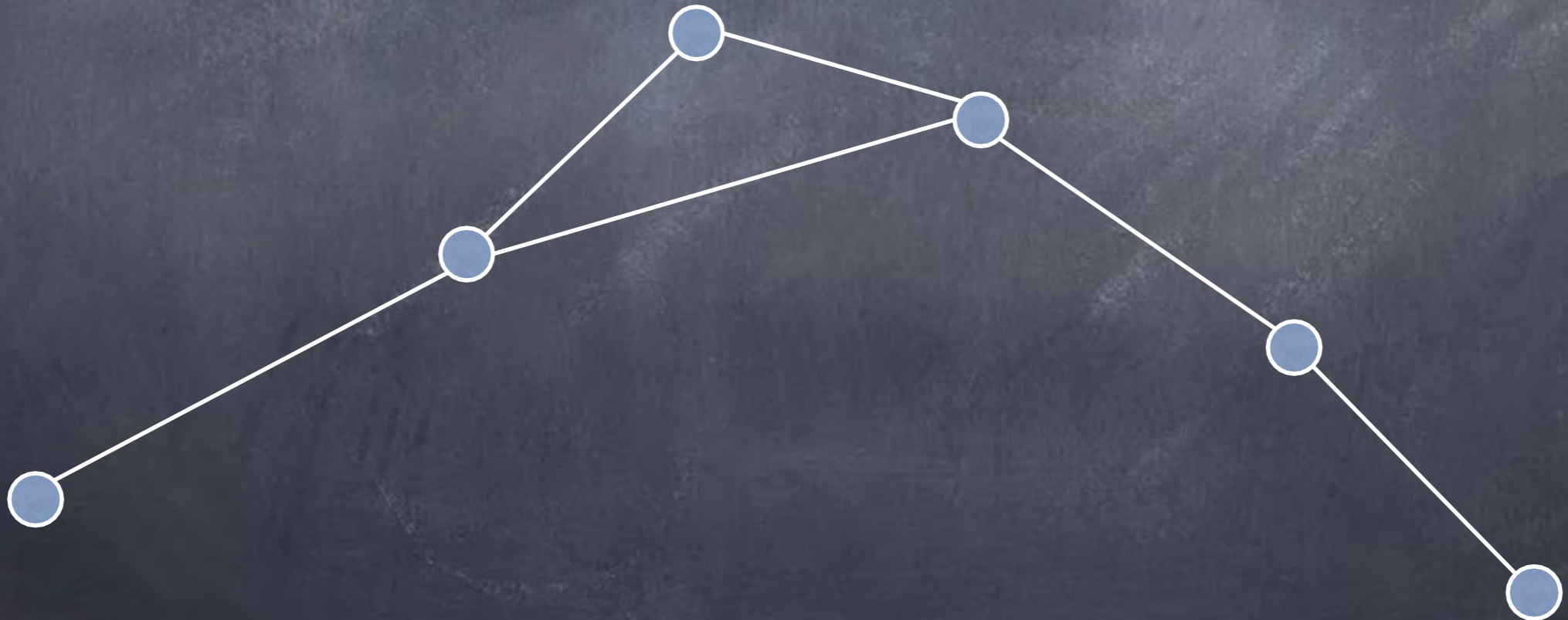
@フロントバッファ
→@バックバッファ

コンピュータ実習(第9回)

演習1 アニメーションによる可視化

1. 目的地までの経路を決める関数 `void path_set(void)`

今回は予め自分で経路を決める



コンピュータ実習(第9回)

演習1 アニメーションによる可視化

1. 目的地までの経路を決める関数 `void path_set(void)`

今回は予め自分で経路を決める

隣接する交差点の設定(移動経路の設定)

交差点のID(数字)を配列 `path[]` に入れる。

例えば、経路上の交差点IDが: 1, 5, 3, 2, 7 だとすると、

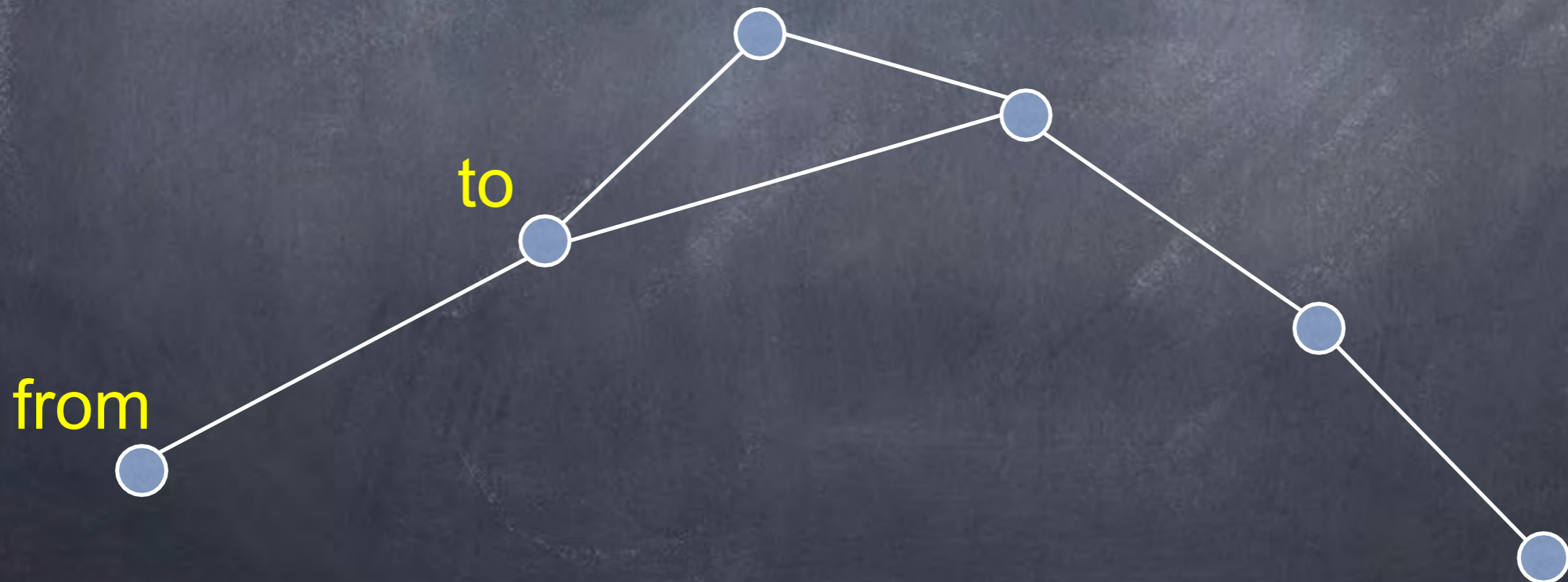
```
path[0] = 1;  
path[1] = 5;  
path[2] = 3;  
path[3] = 2;  
path[4] = 7;  
path[5] = -1;
```

とする。最後の -1 は、経路が終ったことを示すダミーのID。

コンピュータ実習(第9回)

演習1 アニメーションによる可視化

- 隣接する2つの交差点間をマーカが移動するアニメーション表示をする



コンピュータ実習(第9回)

演習1 アニメーションによる可視化

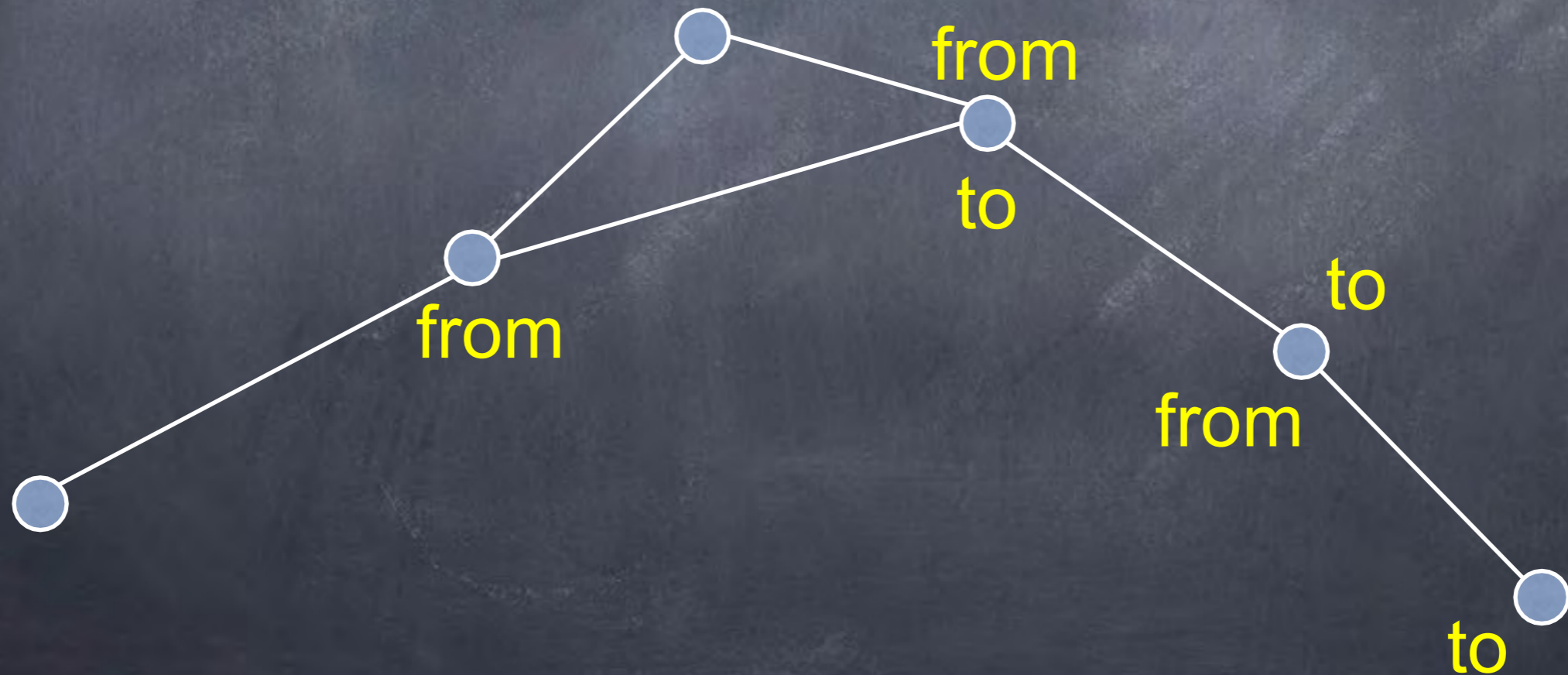
2. 隣接する2つの交差点間をマーカが移動する アニメーション表示をする

```
/* 現在の交差点と次の交差点を (x0, y0),(x1, y1)とする */
    x0 = (現在の交差点のx座標)
    y0 = (現在の交差点のy座標)
    x1 = (次の交差点のx座標)
    y1 = (次の交差点のy座標)
    distance = hypot(x1 - x0, y1 - y0); /* 2点間の距離を計算 */
    steps = (int)(distance / 0.1); /* 距離に応じてステップ数を決める */
/* 交差点の間で現在の位置(vehicle_x, vehicle_y)を少しずつ進める */
    vehicle_stepOnEdge++;
    vehicle_x = (車両マーカーのx座標位置の更新)
    vehicle_y = (車両マーカーのy座標位置の更新)
/* 現在の位置に移動体(円形)を表示 */
    glColor3d(1.0, 1.0, 1.0);
    draw_circle(vehicle_x, vehicle_y, MARKER_RADIUS);
```

コンピュータ実習(第9回)

演習1 アニメーションによる可視化

3. 2の処理を目的地まで繰り返させる



コンピュータ実習(第9回)

演習1 アニメーションによる可視化

3. 2の処理を目的地まで繰り返させる

```
/* 今の交差点と次の交差点のIDがどちらもダミーでない時 */  
if (path[ /* パス上の現在の交差点を表すインデクス */ ] != -1 &&  
    path[ /* パス上の次の交差点を表すインデクス */ ] != -1) {  
  
    /* まだゴールに達していないので、移動体の位置を進める */  
    /* 現在の交差点と次の交差点を (x0, y0),(x1, y1)とする */  
    /* 交差点の間で現在の位置(vehicle_x, vehicle_y)を少しずつ進める */
```

2のアルゴリズムがここに入る！！！！

```
if( /* 現在の位置が次の交差点を過ぎていたら */ ){  
    /* パス上の現在の交差点を表すインデクスを進める */  
}  
}  
/* 現在の位置に移動体を表示 */
```

コンピュータ実習(第9回)

演習2 サーチを利用したマーカの移動

出発地と目的地を指定するとその間の経路(上の交差点)を見つけ出す簡単なプログラムを作成してください。

まず、演習1で作成した`mobile.c`を`mobile_s.c`に名前を変えて保存しておいて下さい。これを改造していきます。

演習1では経路をプログラム中で指定しました。

一連の実習で最終的に使用するサーチアルゴリズムは次回の演習で勉強しますので、

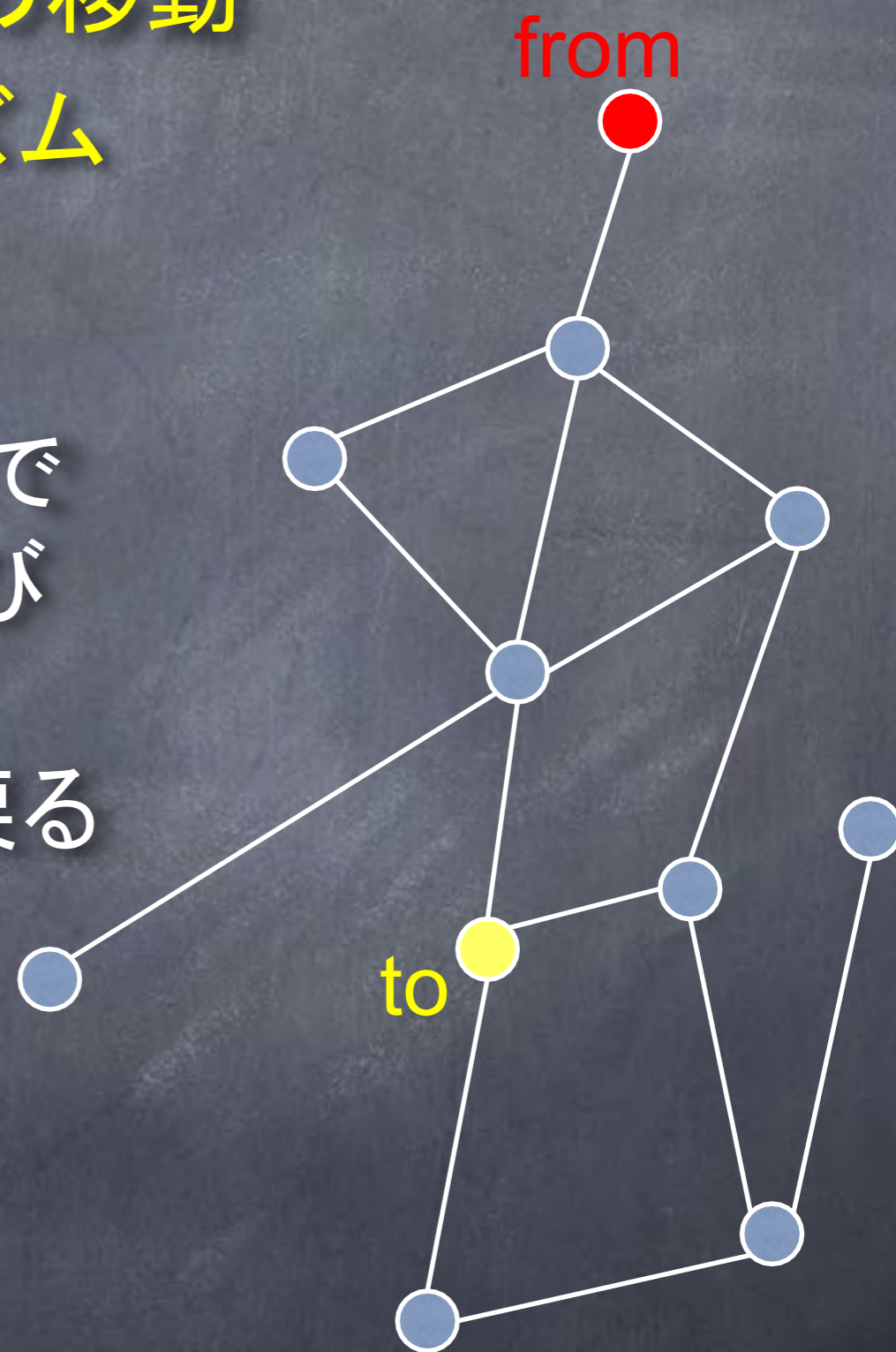
今回の演習では一番簡単なサーチアルゴリズムをコーディングします。

コンピュータ実習(第9回)

演習2 サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

1. 現在地を出発点として
2. 出発地に隣接する交差点の中で最も目的地に近い交差点を選び
3. そこに移動して
4. そこが目的地でなければ1に戻る
5. 目的地であれば探索終了



まずはこれを使って演習2を完成させて下さい

コンピュータ実習(第9回)

演習2 サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

目的地に最も近い隣接交差点のサーチアルゴリズム

現在地交差点ID : id と 目的地交差点ID : goal
が与えられているとき、goal に最も近い隣接交差点の id は、次のようにして求められる。

```
min_distance = 1e100; /* 最短距離を暫定的に大きな値としておく */
for (i = 0; i < cross[id].points; i++)
{
    distance = (cross[goal] と cross[id]のi番目の隣接交差点との距離を計算);
    /* もしこれまでの最短距離よりも短いものが見つかったら */
    if (min_distance > distance)
    {
        min_distance = distance; /* 新たな最短距離をセット */
        nearest = cross[id].next[i]; /* 最も近い隣接交差点IDを更新 */
    }
}
```

コンピュータ実習(第9回)

演習2 サーチを利用したマーカの移動

最近傍隣接交差点検索アルゴリズム

「交差点毎に次の隣接交差点を調べてそこに移動する」
という一番素直なやり方

ヒント

• `int search_nearest(int id, int goal)` の追加

- 交差点 `id` の隣接交差点の中から交差点 `goal` に最も近いものを探し、`return`文により返す

• `void path_set(void)` の削除

- この関数はもう要りません

• `int main(void)` の修正

弱点

同じ経路を行ったり来たりする場合がある
→行き止まりや袋小路

コンピュータ実習(第9回)

演習2 サーチを利用したマーカの移動 最近傍隣接交差点検索アルゴリズム

同じ経路を行ったり来たりする袋小路問題の解決策を考える
例えば、

さっきいた交差点は計算の際の候補に加えない
今まで通ってきた交差点は計算の際の候補に加えない
など、が考えられます。

しかし、これだけでは不十分ですので、考えてみて下さい。

次回の授業では、この問題を解決可能な最適経路探索アルゴリズムとして「**ダイクストラ法**」を取り上げます。