コンピューターグラフィックスS

第3回　演習（1）：OpenGL&GLUT入門

システム創成情報工学科　尾下 真樹

2019年度 Q2

今回の内容

- 前回の復習
- 演習環境
  - OpenGLとGLUTの概要
- サンプルプログラムの解説
  - サンプルプログラムの概要
  - 変換行列の設定、描画、光源情報の設定、等
- プログラミング演習
  - コンパイルと実行、演習課題

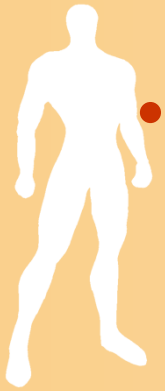
前回の復習

コンピュータグラフィックスの応用

- 映画
- コンピュータゲーム
- CAD
- シミュレーション
- 仮想人間(ヴァーチャル・ヒューマン)
- 可視化（ビジュアライゼーション）
- ユーザインターフェース

3次元グラフィックスの要素技術

- モデリング
- レンダリング
- 座標変換
- シェーディング
- マッピング
- アニメーション



オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

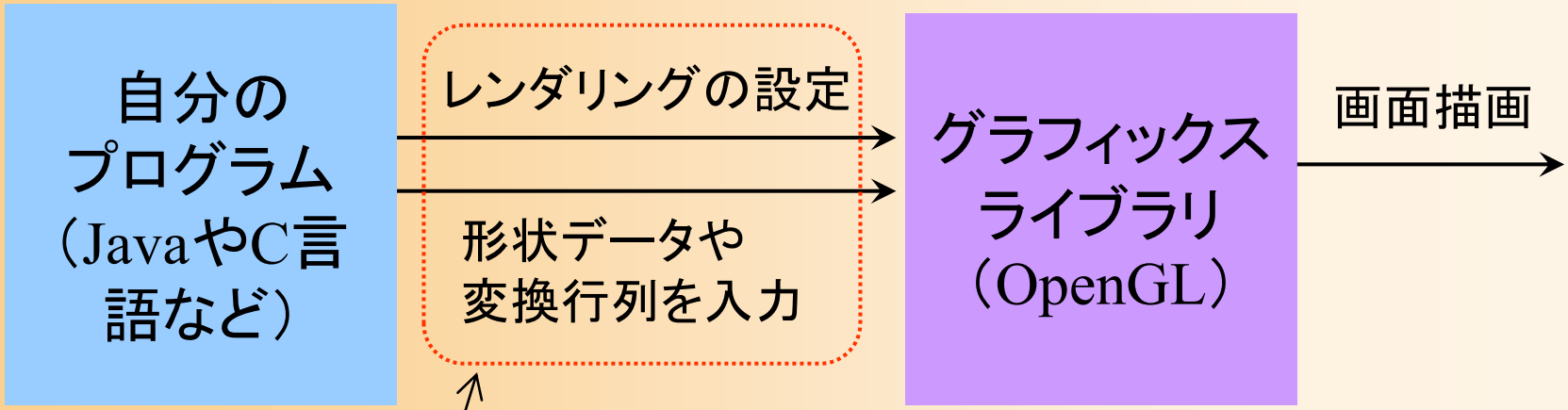
光源

カメラから見える画像を計算

光の効果の表現

グラフィックスライブラリの利用

- 自分のプログラム と OpenGL の関係



最低限、これらの方法だけ学べば、
プログラムを作れる

これらの処理は、自分でプログラ
ムを作る必要はないが、しくみは
理解しておく必要がある

レンダリング（＋座標変換、
シェーディング、マッピング）
などの処理を行ってくれる

演習概要

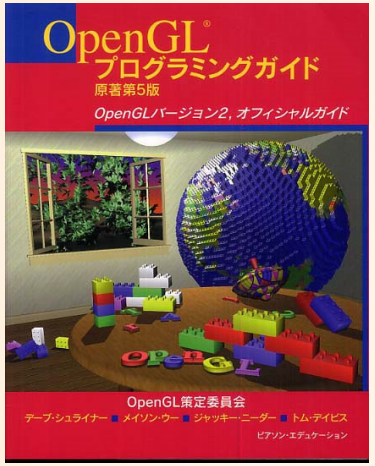
本講義の演習内容

- OpenGL + GLUT による演習
  - 簡単な物体描画、マウスによる視点操作、アニメーションなど
  - 具体的な描画処理はOpenGLが行ってくれる

- 演習の目的
  - 実際にプログラムを作成することで、3次元CGの仕組みをより深く理解する
  - 将来、3次元CGのプログラミングが必要になる時のために、とりあえず最低限使えるようになる

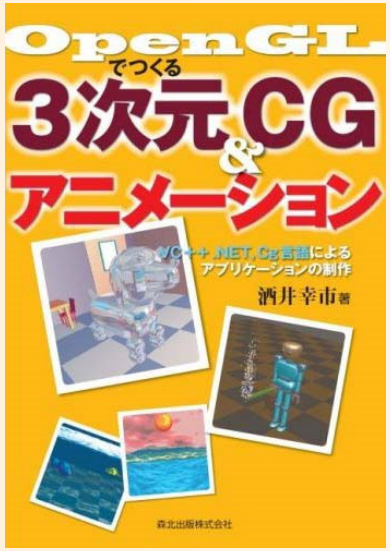
演習の参考書(1)

- 最低限の使い方は資料で説明
  - 特に参考書を買う必要はない
- OpenGLの定番の本(高い)
  - OpenGLプログラミングガイド(赤本), 12,000円
  - OpenGLリファレンスマニュアル(青本), 8,300円
    - 共に、ピアソン・エデュケーション出版
    - OpenGLの使い方が、詳しく解説されている
    - 本格的にOpenGLを使ったプログラムを作りたい人は、買うと良い



演習の参考書(2)

- 他の参考書
  - OpenGLでつくる 3次元CG & アニメーション（3600円）
    - 酒井 幸市 著
    - OpenGL・GLUTの使い方 ＋ 最新技術
    - 興味がある人は、買ってみると良い
  - OpenGL入門（3,000円）
    - エドワード・エンジェル 著、滝沢 徹・牧野 祐子 訳
    - ピアソン・エデュケーション出版
    - OpenGL・GLUTの使い方

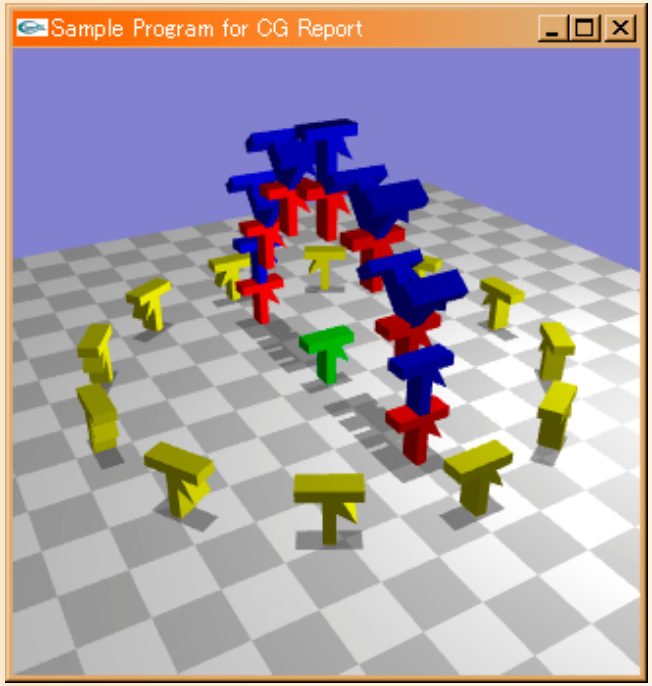


演習の流れ

- OpenGLの使い方を講義で説明
- 資料に従って、各自、プログラムを拡張
  - 資料に従ってサンプルプログラムに少しずつ修正を加えながら、講義で学習した内容を、実際に作成してみて確認
  - 一部のプログラムは、穴埋めになっている
  - プログラムで困ったときには、TAが補助
- 各回の演習が終わったらプログラムを提出

レポート課題

- レポート課題
  - 与えられた課題を実現するように、演習で作成したプログラムを拡張する
  - レポート課題は、全員、違うものを与える
  - プログラムとレポートの両方を提出



演習資料(3種類)

- 演習資料(OpenGL演習)
  - この資料に従って、プログラムを拡張していく
    (次回以降の説明は、逐次追加)
- コンパイル方法の説明資料
  - コンパイル方法の詳しい説明
  - CL端末や自宅でのコンパイル方法も一応説明
- OpenGL関数 簡易リファレンス
  - OpenGLの関数を簡単に説明した資料
  - 次回以降に使用する関数の説明も含む

演習環境

OpenGL

- OpenGL
  - 現在、最も広く使われている3次元API
    - API ＝ Application Programming Interface
    - C言語を始め、いろんな言語から使える
  - ポリゴンの描画、Zバッファなどの3次元グラフィックス描画に必要な機能を提供
  - ウィンドウ生成やマウス・キーボード入力などの処理の機能は持たない
    - これらは、OSやウィンドウシステム固有の機能なので、各環境に応じたAPIを使って記述する必要がある
    - 実装が大変、環境ごとに実装する必要がある

GLUT

- OpenGL Utility Toolkit（GLUT）
  - ウィンドウ生成やイベント処理などの環境依存の部分を共通化したライブラリ
    - Mark Kilgard 氏が開発
    - OpenGL標準ではないがかなり広く普及している
  - 内部に各OS用のコードを含んでいるため、一度プログラムを作ればいろんな環境で動く
  - 機能が限定されている代わりに非常にシンプル
  - とりあえずOpenGLを使いたい場合に適している

DirectXとの比較

- DirectX
  - Windowsのみでしか動かない
  - Windowsと密接に関連している
    - WindowsやCOMなどの仕組みを理解する必要がある
    - プログラミングが必要以上に面倒
  - 最新のハードウェアの機能を使えるという利点もある
  - 他のマルチメディア機能も持っている
    - DirectSound, DirectPlay, DirectInput
  - 基本な描画処理の考え方はOpenGLと同じ

Java3Dとの比較

- Java3D
  - シーングラフ API（高レベルAPI）
    - カメラや物体などのシーンの階層構造を設定してやると、細かい描画は自動的に行ってくれる
    - 高機能で便利、CGの原理をよく知らなくても使える
  - デメリット
    - 独自のライブラリなので、Java3Dの使い方だけ覚えても使い回しが利かない
    - クラスライブラリになっているので、本当にきちんと理解しようとすると全体像を把握する必要があり、大変

他のライブラリとの比較

- 携帯端末（iOS/Androidなど）用アプリ
  - OpenGL ES が採用されている
    - OpenGL のサブセット（機能限定版）
    - 描画処理の基本は OpenGL と基本的に同様
- ゲームエンジン
  - Unity, Unreal など
  - Java 3D 同様、シーン情報やアニメーションを設定すれば、細かい描画処理は自動的に行ってくれる

演習環境

- 主に使う環境
  - OpenGL
  - C言語
  - Windows + Visual Studio（or Linux + gcc ）
- もし希望があれば、各自のやりたい環境でやっても構わない
  - DirectXやJava3Dなど
  - 同じ内容ができていればレポートは受け付ける
    - ただし必ず低レベルAPIを使うこと

演習環境の説明(1)

- OpenGLを使う理由
  - OSとは独立しているので、3次元処理だけを勉強しやすい
  - Windows以外の環境でも広く使える
  - 基本的な考え方はDirectXなどでもほぼそのまま通用する

演習環境の説明(2)

- C言語を使う理由
  - OpenGLを使うのに適している
  - C++との比較
    - オブジェクト指向言語という点では、C++の方がC言語よりもJavaに近いが、
    - C++は高機能な分 JavaやCよりもかなり複雑
    - OpenGLの使い方を学ぶのには、オブジェクト指向言語よりも構造化言語の方が単純で良い
  - C言語は社会で広く使われているので慣れておいて損はない

演習環境の説明(3)

- 社会では C/C++ が広く使われている
  - Javaよりも広く使われている
- 本学科の他の科目でも C/C++ を使用
  - システム創成プロジェクトIII
  - 卒業研究に、C/C++ を使う研究室もある
- プログラミングの基本は同じなので、Javaを十分に使いこなすことが出来れば、基本的には、慌ててC/C++を勉強する必要はない
  - Java の方が勉強しやすい面があるので、本学部では、主に Java を勉強するようになっている

C言語とJavaの違い

C言語 と Java の比較

- Java と C言語は文法などはかなり近い
  - Javaはもともと C言語（C++）が祖先
  - ただし、プログラミングの考え方はかなり異なる
    - オブジェクト指向言語（Java）と構造化言語（C言語）
  - Javaが使えれば C言語はすぐに使えるはず
    - 本来はオブジェクト指向言語の方がより高度な概念
- Java と C言語（C++）の主な違い
  - コンパイルと実行の仕組み
  - オブジェクト指向言語と構造化言語の違い
  - 文法やクラスライブラリの違い

プログラミングで重要なポイント

- Java も C/C++ も、文法自体はかなり近い
- ソフトウェア設計が重要
  - 与えられたアルゴリズムをもとに、プログラムを書くのは比較的簡単
  - 大きなプログラムを作成するときに、どのようにクラスを作るべきか、どのように機能を細分化するか、といった設計を考える能力が重要
- コンパイルや実行の仕組みを理解する
- 本演習の範囲ではそれほど問題はない

C言語とJavaの違い(1)

- C言語にはクラスがなく、関数(Javaのメソッドと同様)の集まりとしてプログラムを作成
- 関数の外側で定義した変数は、グローバル変数となり、全ての関数からアクセス可能
  - Javaでクラスのプロパティ(メンバ変数)に全てのメソッドからアクセスできるのと同様
  - Java同様、関数内部で定義した変数はローカル変数となり、その関数内でしかアクセスできない

C言語とJavaの違い(2)

- 関数を呼び出すためには関数定義が必要
  - 呼び出しよりも前(上)に、関数の定義が記述されている必要がある
  - 呼び出しより前に、関数本体を記述するか、プロトタイプ宣言(関数名・引数・戻り値)を記述する
- C言語では多次元配列の扱いが異なる
  - 通常の方法で多次元配列を使う場合は、2番目以降の次元の要素数がコンパイル時に確定している必要がある
    - 例: float vertex[ ][3]

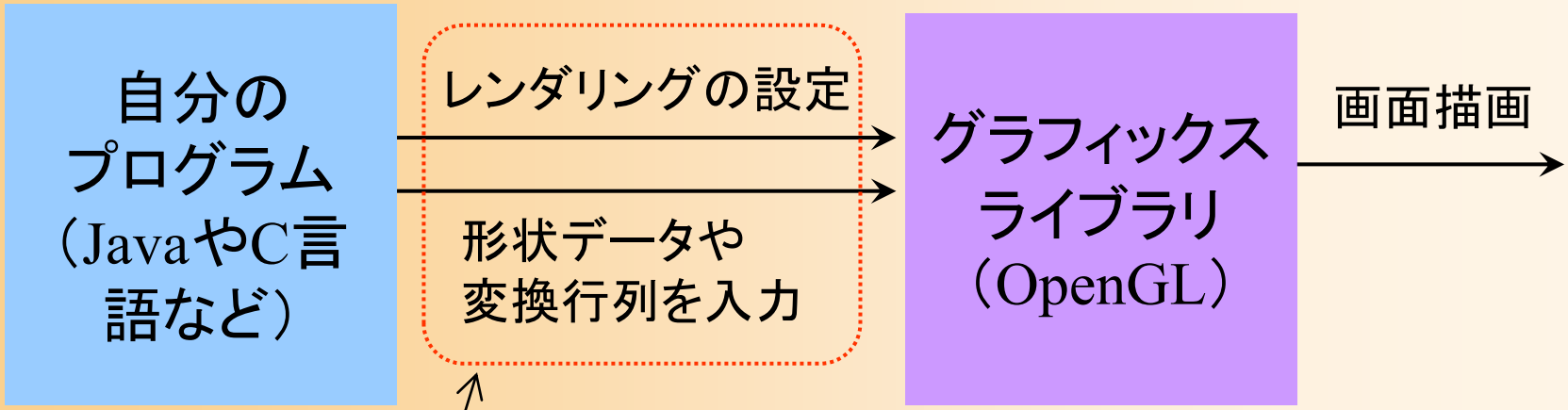
より詳しい説明

- 補足資料を参照
  - C言語 と Java の違い
  - C言語

OpenGL & GLUT入門

OpenGLの利用（復習）

- 自分のプログラム と OpenGL の関係



最低限、これらの方法だけ学べば、
プログラムを作れる

これらの処理は、自分でプログラ
ムを作る必要はないが、しくみは
理解しておく必要がある

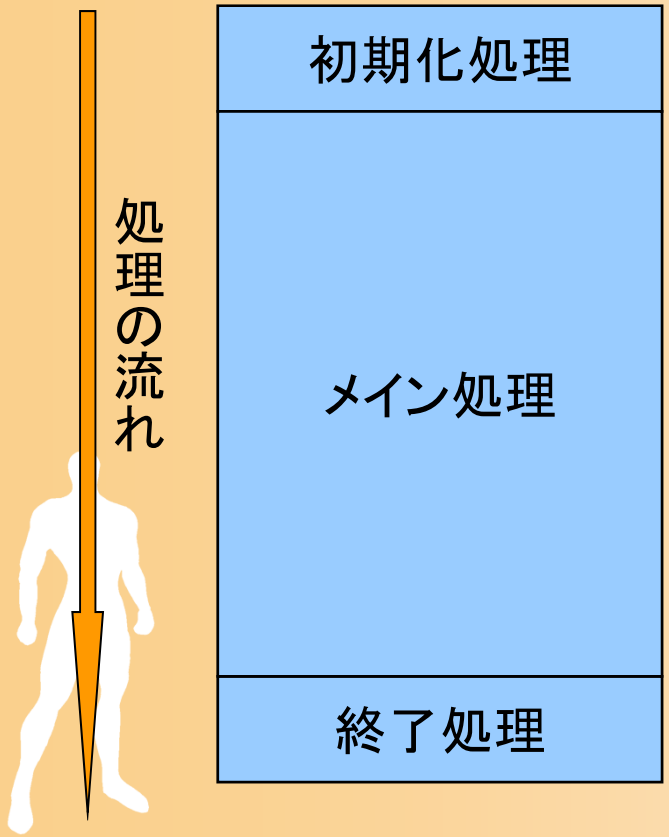
レンダリング（＋座標変換、
シェーディング、マッピング）
などの処理を行ってくれる

GLUTのイベントモデル

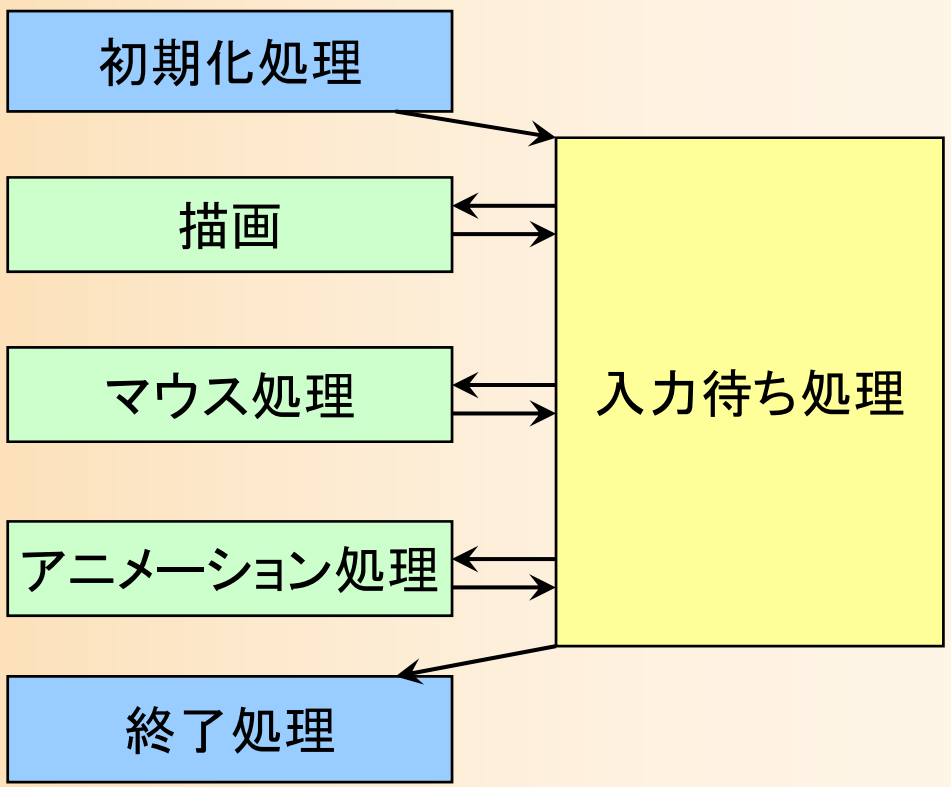
- ウィンドウシステムでのプログラミング
  - Windows や X Window などの一般的なウィンドウシステム
  - ウィンドウ管理やマウス操作などはシステムがまとめて処理するため、ユーザプログラムは扱わない
  - ユーザプログラムは初期化処理を行った後は処理をウィンドウシステムに移す
  - ウィンドウシステムは、画面の再描画やマウスの操作などのイベントが起こるたびにユーザプログラムに処理を一時的に戻す

イベントドリブン型プログラム

コンソール・プログラム



ウィンドウ・プログラム（イベントドリブン）

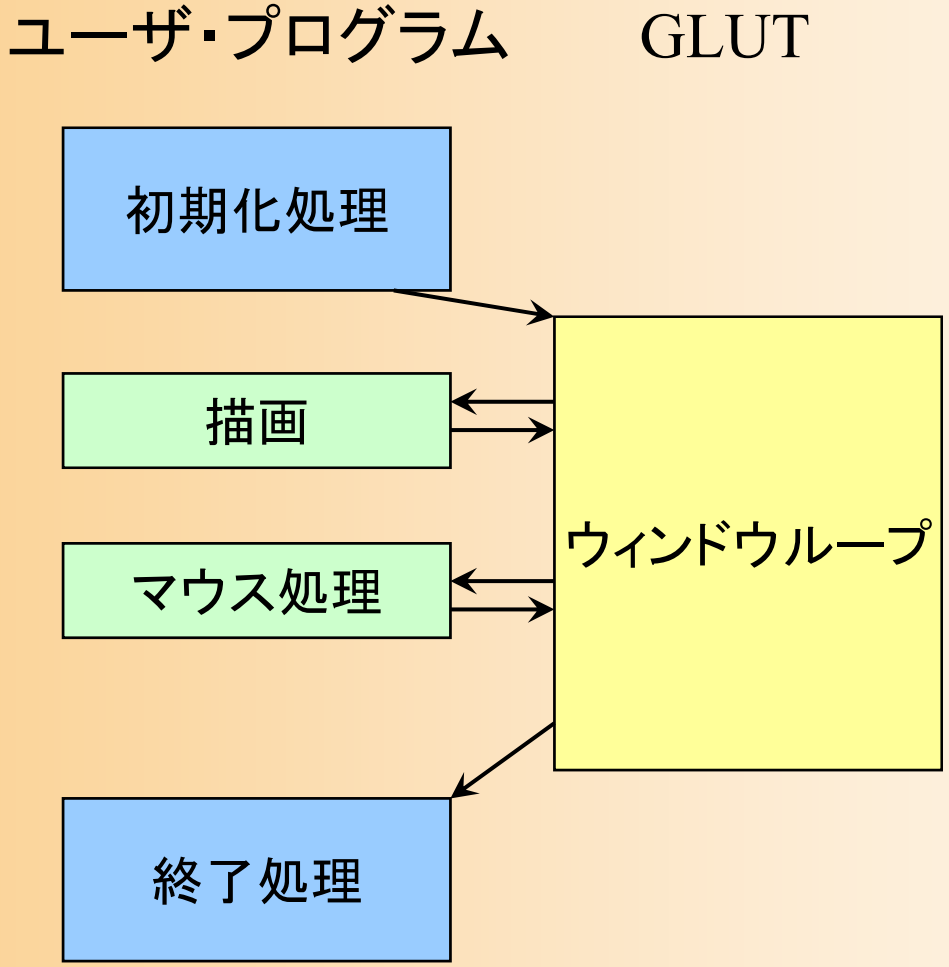


GLUTのイベントモデル

- イベントループとコールバック
  - イベントが起こった時にそのイベントを処理する関数をあらかじめ登録しておく
  - プログラムは初期化が終わったら、GLUTに処理を移す
  - マウス操作などのイベントが起こったらあらかじめ登録した関数が呼ばれる(コールバック)

- Javaの AWT や Swing などでは、同様の機能をリスナクラスを使って実現している

GLUTのイベントモデル



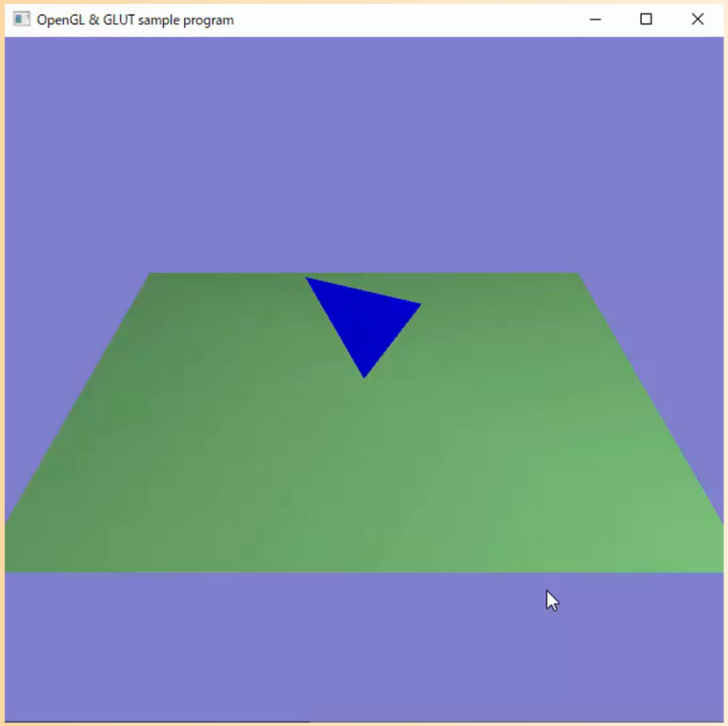
GLUTのコールバック関数の種類

- 描画コールバック関数
  - 画面の再描画が必要な時に呼ばれる
- サイズ変更コールバック関数
  - ウィンドウサイズ変更時に呼ばれる
- マウスクリック・コールバック関数
  - マウスのボタンが押されたとき、離されたときに呼ばれる
- マウスドラッグ・コールバック関数
  - マウスがウィンドウ上でドラッグされたときに呼ばれる
- キーボード・コールバック関数
  - キーボードのキーが押されたときに呼ばれる
- アイドル・コールバック関数
  - 処理が空いた時に定期的に呼ばれる

サンプルプログラムの解説

サンプルプログラム

- opengl_sample.c
  - 地面と1枚の青い三角形が表示される
  - マウスの右ボタンドラッグで、視点を上下に回転



サンプルプログラム

- opengl_sample.c

opengl_sample.c

```
//
//  コンピュータグラフィックスＳ
//  OpenGLによる３次元グラフィックス演習 サンプルプログラム
//


// 基本的なヘッダファイルのインクルード
#ifdef _WIN32
    #include <windows.h>
#endif
#include <stdio.h>
#include <math.h>

// GLUTヘッダファイルのインクルード
#include <GL/glut.h>


// 視点操作のための変数
float  camera_pitch = -30.0;  // Ｘ軸を軸とするカメラの回転角度

// マウスのドラッグのための変数
int  drag_mouse_r = 0;  // 右ボタンをドラッグ中かどうかのフラグ (0:非ドラッグ中,1:ドラッグ中)
int  last_mouse_x;      // 最後に記録されたマウスカーソルのＸ座標
int  last_mouse_y;      // 最後に記録されたマウスカーソルのＹ座標


//
//  画面描画時に呼ばれるコールバック関数
//
void  display( void )
{
    // 画面をクリア (ピクセルデータとＺバッファの両方をクリア)
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    // 変換行列を設定 (ワールド座標系→カメラ座標系)
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0,  0.0, -15.0 );
    glRotatef( -camera_pitch, 1.0, 0.0, 0.0 );

    // 光源位置を設定 (モデルビュー行列の変更にあわせて再設定)
    float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );

```

サンプルプログラムの解説

- ここでは、プログラム全体を眺めて、大まかに、各部分でどのような処理を行っているかを確認する
- 各自、実際にコンパイルをしてみて、動作を確認する
- 各処理の詳しい内容は、今後の講義で徐々に説明

OpenGLの関数

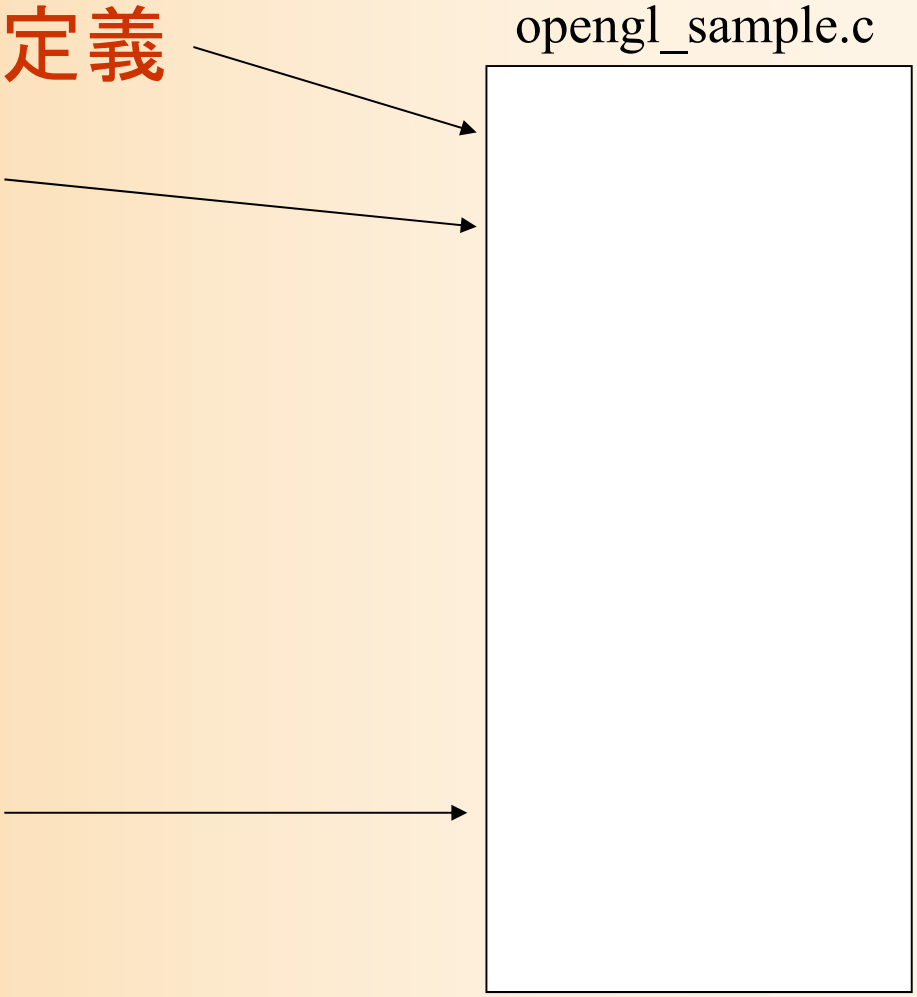
- gl～ で始まる関数
  - OpenGLの標準関数
- glu～ で始まる関数
  - OpenGL Utility Library の関数
  - OpenGLの関数を内部で呼んだり、引数を変換したりすることで、使いやすくした補助関数
- glut～ で始まる関数
  - GLUT（OpenGL Utility Toolkit）の関数
  - 正式にはOpenGL標準ではない

OpenGLの関数名

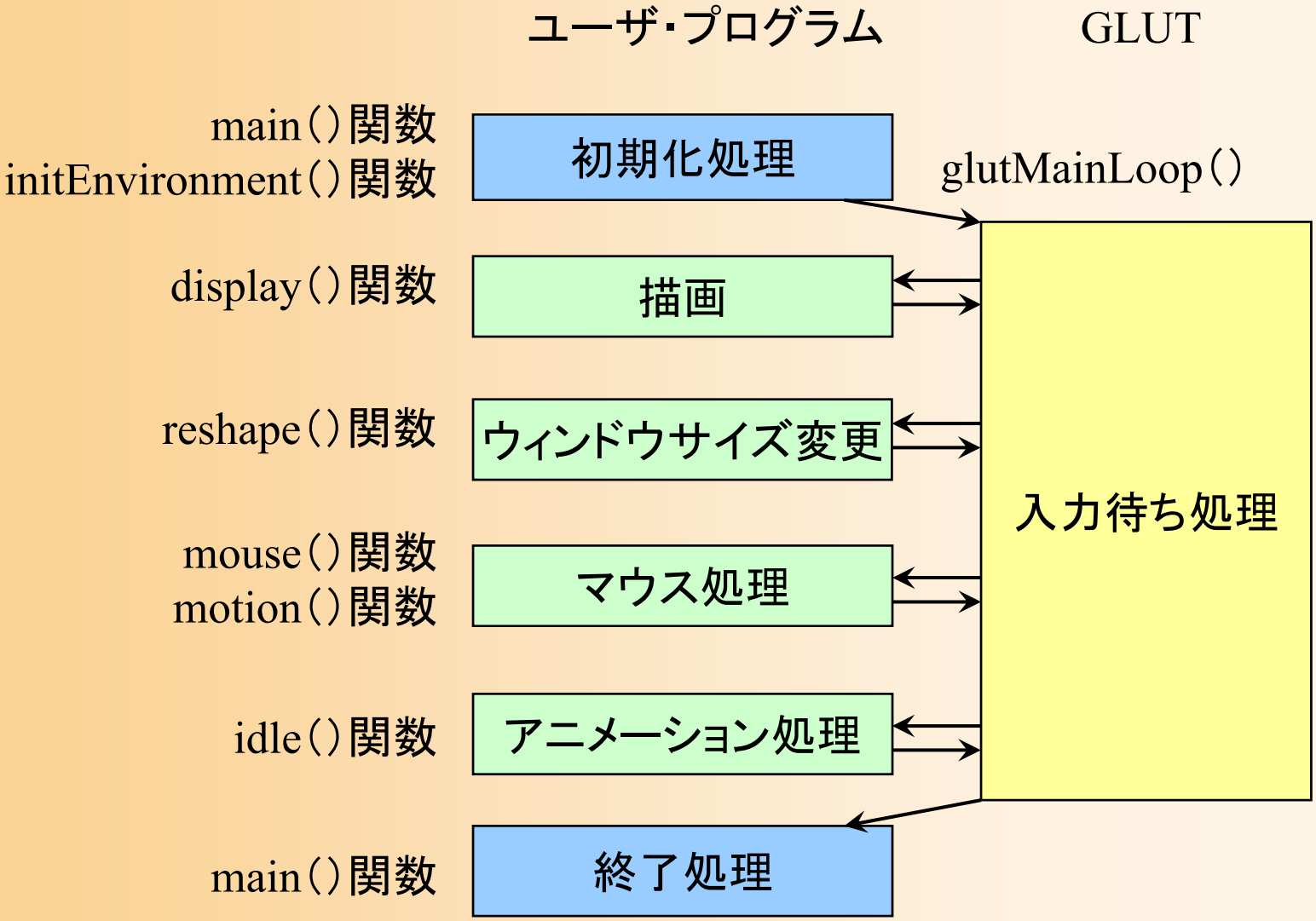
- 同じ機能で、微妙に違う名前の関数がある
  - 例：glVertex3f(x, y, z), glVertex3d(x, y, z)
    - f は引数が float 型であることを表す
    - d は引数が double 型であることを表す
      - C言語なので、関数のオーバーロード（同じ名前で引数が異なる関数）はサポートしていない
    - 必要に応じて使い分ける

サンプルプログラム

- グローバル変数の定義
- コールバック関数
  - display()
  - reshape()
  - mouse()
  - motion()
  - idle()
- 開始・初期化関数
  - initEnvironment()
  - main()



サンプルプログラムの構成

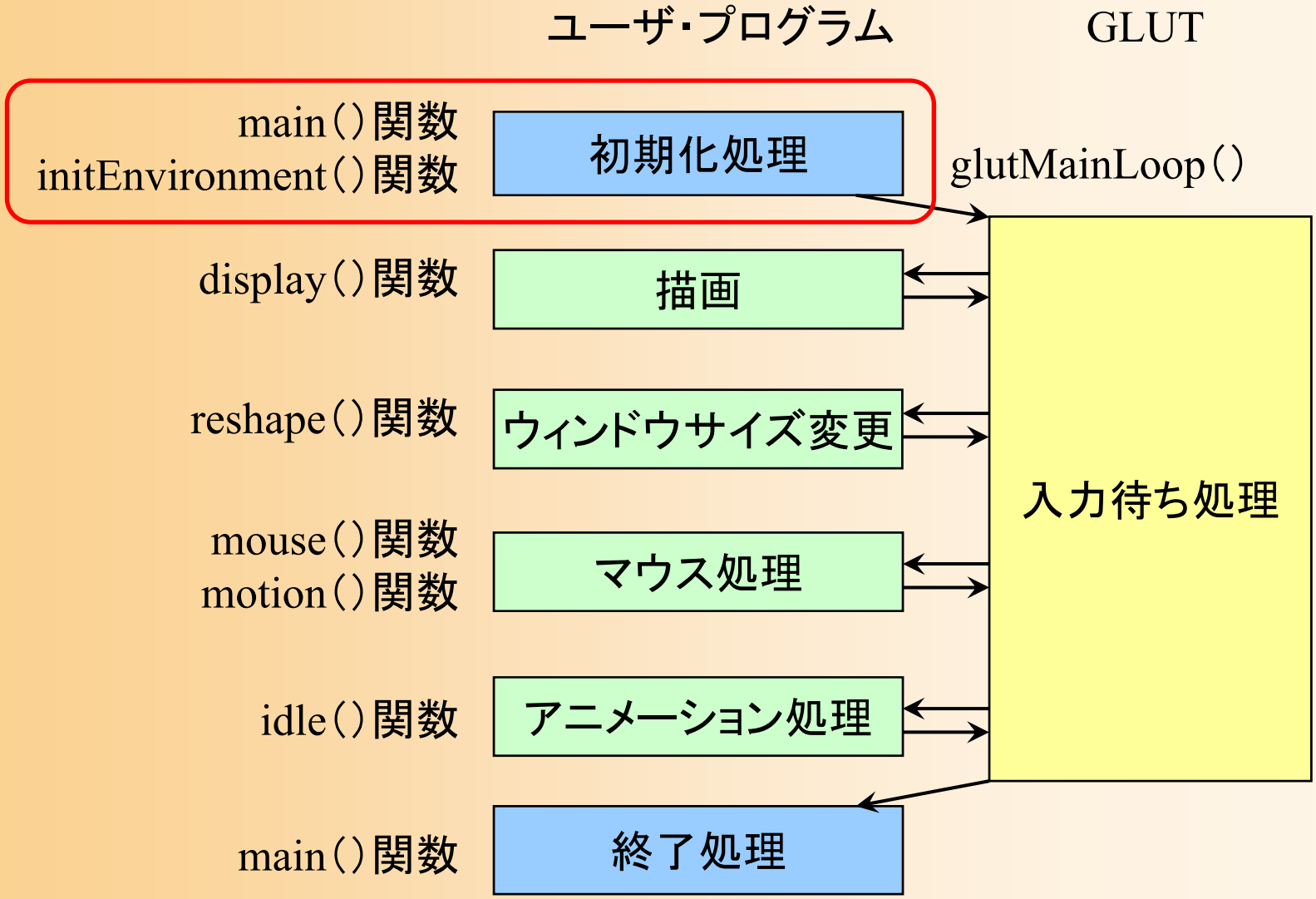


グローバル変数の定義

- グローバル変数
  - 全ての関数からアクセスできる変数
  - ここでは視点操作に関する変数を定義
    - 詳細は後で説明

```
// 視点操作のための変数
float  camera_pitch = -30.0;  // X軸を軸とするカメラの回転角度

// マウスのドラッグのための変数
int  drag_mouse_r = 0;  // 右ボタンをドラッグ中かどうかのフラグ
                                    (0:非ドラッグ中,1:ドラッグ中)
int  last_mouse_x;      // 最後に記録されたマウスカーソルのX座標
int  last_mouse_y;      // 最後に記録されたマウスカーソルのY座標
```

サンプルプログラムの構成



開始・初期化処理

- main関数
  - GLUTの初期化（メイン関数）
  - コールバック関数の設定
  - initEnvironment関数の呼び出し
  - GLUTのメインループの開始
- initEnvironment関数
  - レンダリングの設定
  - 光源の設定

GLUTの初期化

```
int  main( int argc, char ** argv )
{
        // GLUTの初期化
        glutInit( &argc, argv );
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
        glutInitWindowSize( 320, 320 );
        glutInitWindowPosition( 0, 0 );
        glutCreateWindow("OpenGL & GLUT sample program");

        ......
}
```

コールバック関数の設定

```
int  main( int argc, char ** argv )
{
        ......
        // コールバック関数の登録
        glutDisplayFunc( display );
        glutReshapeFunc( reshape );
        glutMouseFunc( mouse );
        glutMotionFunc( motion );
        glutIdleFunc( idle );

        // 環境初期化
        initEnvironment();

        // GLUTのメインループに処理を移す
        glutMainLoop();
        return 0;
}
```

光源の設定

- シェーディングのための光源情報の設定
  - 1つの点光源を設定（詳しい内容は後日説明）

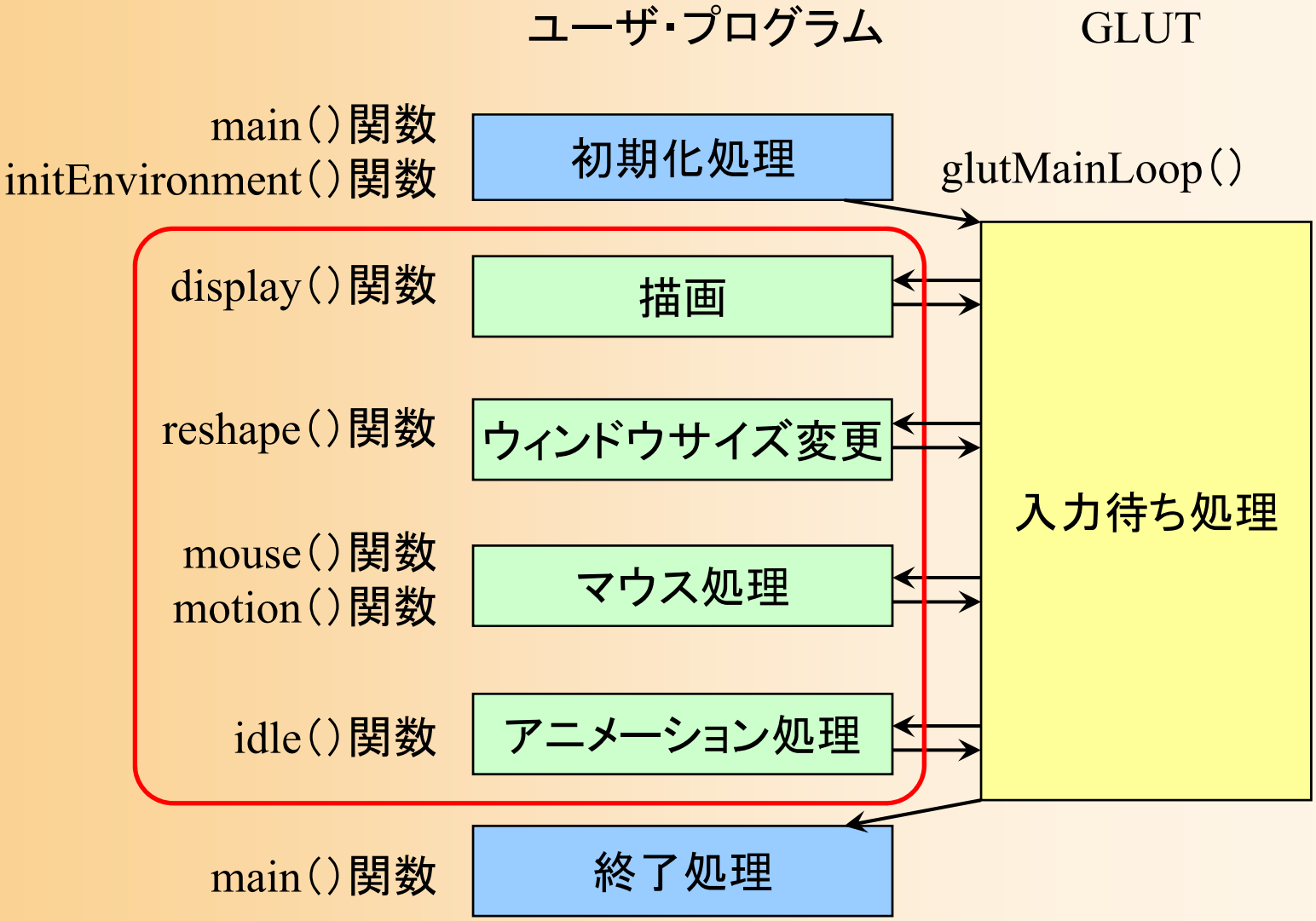
```
void  initEnvironment( void )
{
        float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
        float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
        float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };

        glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
        glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
        glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
        glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );

        glEnable( GL_LIGHT0 );
        glEnalbe( GL_LIGHTING );
        ……
}
```

レンダリングの設定

- Zバッファ法によるレンダリングの各種設定
  - 標準的な描画機能を設定（詳しい内容は後日説明）

```
void  initEnvironment( void )
{
        ......
        // 光源計算を有効にする
        glEnable( GL_LIGHTING );

        // 物体の色情報を有効にする
        glEnable( GL_COLOR_MATERIAL );

        // Zテストを有効にする
        glEnable( GL_DEPTH_TEST );

        // 背面除去を有効にする
        glCullFace( GL_BACK );
        glEnable( GL_CULL_FACE );

        // 背景色を設定
        glClearColor( 0.5, 0.5, 0.8, 0.0 );
}
```

サンプルプログラムの構成



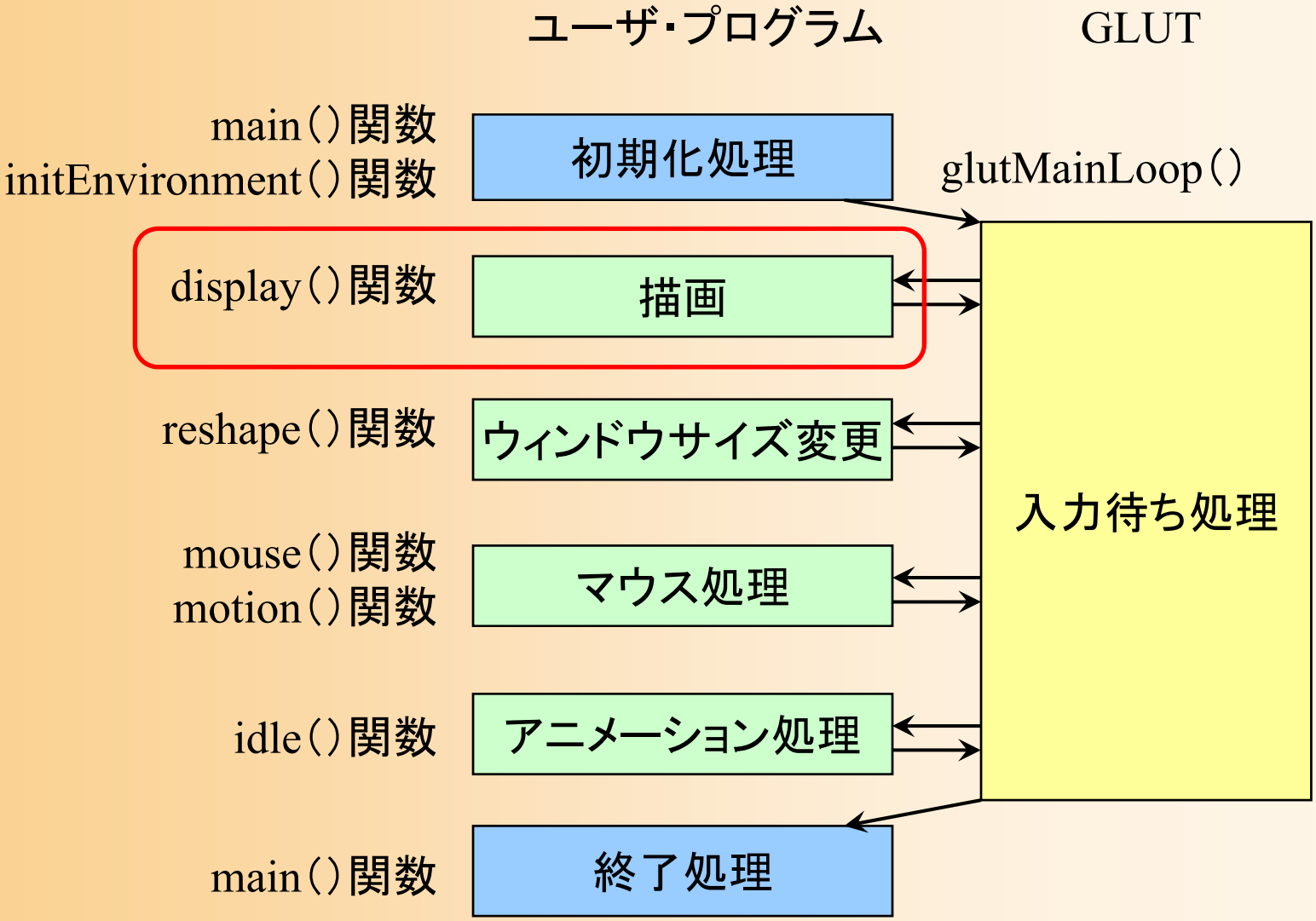
コールバック関数(1)

- 描画コールバック関数 display()
  - 画面の再描画が必要な時に呼ばれる
  - 本プログラムでは、変換行列の設定、地面と1枚のポリゴンの描画、を行っている
- ウィンドウサイズ変更コールバック関数 reshape()
  - ウィンドウサイズ変更時に呼ばれる
  - 本プログラムでは、視界の設定、ビューポート変換の設定、を行っている

コールバック関数(2)

- マウスクリック・コールバック関数 mouse()
  - マウスのボタンが押されたとき、離されたときに呼ばれる
  - 本プログラムでは、右ボタンの押下状態を記録
- マウスドラッグ・コールバック関数 motion()
  - マウスがウィンドウ上でドラッグされたときに呼ばれる
  - 本プログラムでは、右ドラッグされたときに、視点の回転角度を変更
- アイドル・コールバック関数 idle()
  - 処理が空いた時に定期的に呼ばれる
  - 本プログラムでは、現在は何の処理も行っていない

サンプルプログラムの構成



描画関数の流れ

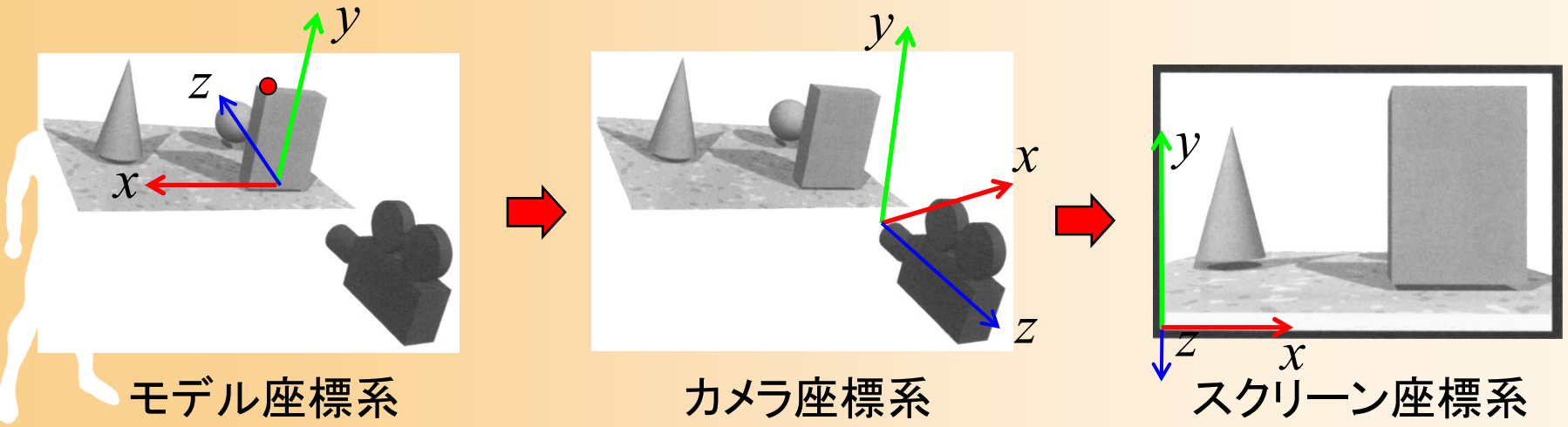
```
//
//  ウィンドウ再描画時に呼ばれるコールバック関数
//
void  display( void )
{
        // 画面をクリア（ピクセルデータとZバッファの両方をクリア）
        // 変換行列を設定（ワールド座標系→カメラ座標系）
        // 光源位置を設定（モデルビュー行列の変更にあわせて再設定）
        // 地面を描画
        // 変換行列を設定（物体のモデル座標系→カメラ座標系）
        // 物体（１枚のポリゴン）を描画
        // バックバッファに描画した画面をフロントバッファに表示
}
```

描画関数(1/4)

```
void  display( void )
{
        // 画面をクリア(ピクセルデータとZバッファの両方をクリア)
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        // 変換行列を設定(ワールド座標系→カメラ座標系)
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
        glTranslatef( 0.0,  0.0, - 15.0 );
        glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );

        // 光源位置を設定(モデルビュー行列の変更にあわせて再設定)
        float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
        glLightfv( GL_LIGHT0, GL_POSITION, light0_position );

        ......
}
```

座標変換(復習)

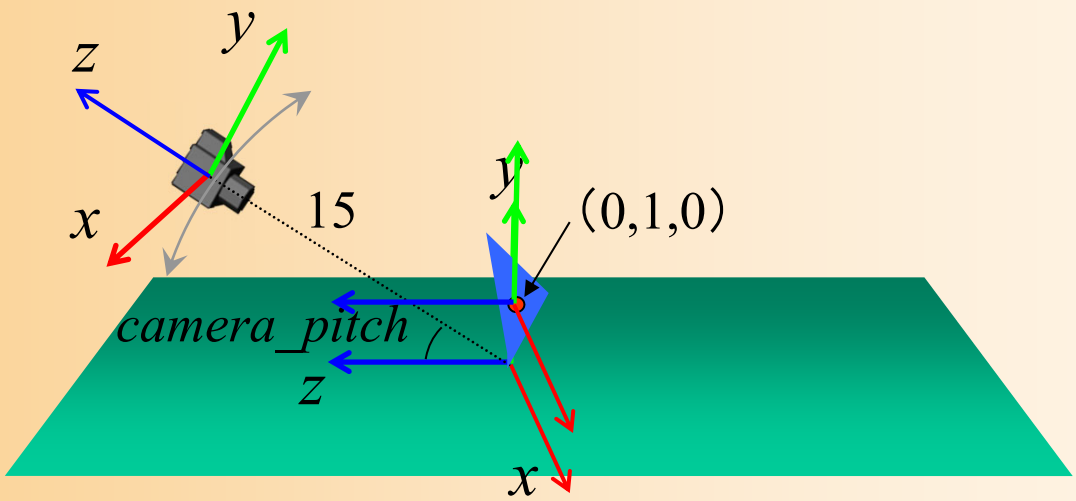
- 座標変換(Transformation)
  - 行列演算を用いて、ある座標系から、別の座標系に、頂点座標やベクトルを変換する技術
    - カメラから見た画面を描画するためには、モデルの頂点座標をカメラ座標系(最終的にはスクリーン座標系)に変換する必要がある



モデル座標系　　カメラ座標系　　スクリーン座標系

変換行列の設定

- サンプルプログラムでのカメラ位置の設定



- 以下の変換行列により表せる（詳細は後日説明）

ポリゴンを基準とする座標系での頂点座標

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-camera\_pitch) & -\sin(-camera\_pitch) & 0 \\ 0 & \sin(-camera\_pitch) & \cos(-camera\_pitch) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

カメラから見た頂点座標（描画に使う頂点座標）

変換行列の設定

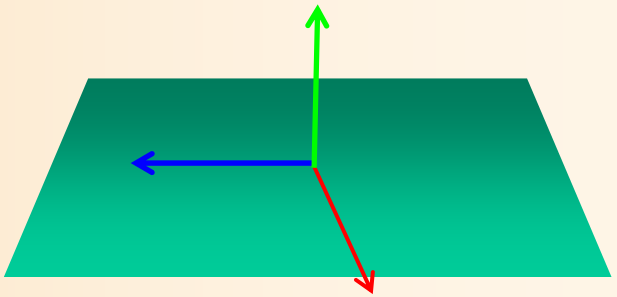
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-camera\_pitch) & -\sin(-camera\_pitch) & 0 \\ 0 & \sin(-camera\_pitch) & \cos(-camera\_pitch) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

```
// 変換行列を設定（ワールド座標系→カメラ座標系）
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( 0.0,  0.0, - 15.0 );
glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );

// 地面を描画
・・・・・・

// 変換行列を設定（物体のモデル座標系→カメラ座標系）
glTranslatef( 0.0, 1.0, 0.0 );

// 物体（１枚のポリゴン）を描画
・・・・・・
```

描画関数(2/4)

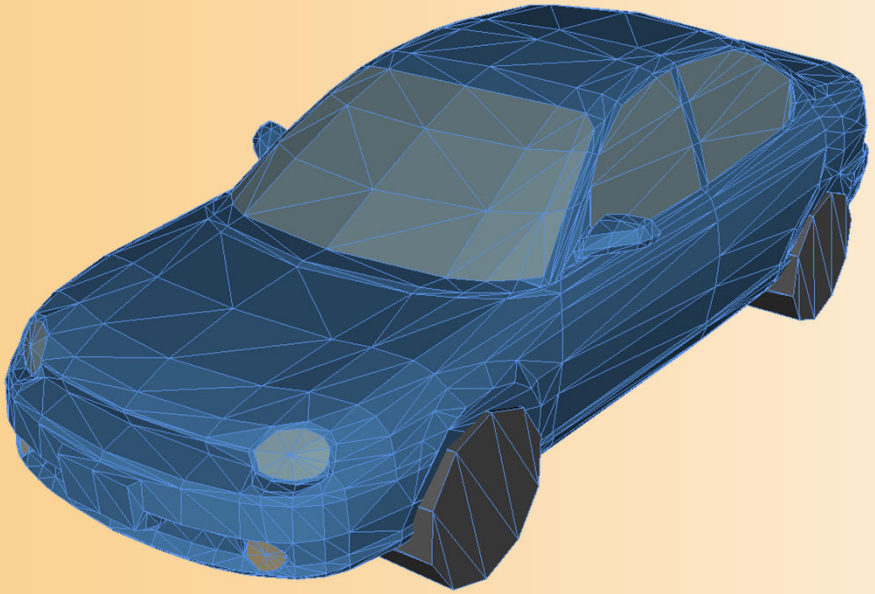
- １枚の四角形として地面を描画
  - 各頂点の頂点座標、法線、色を指定して描画
  - 真上(0,1,0)を向き、水平方向の長さ10の四角形

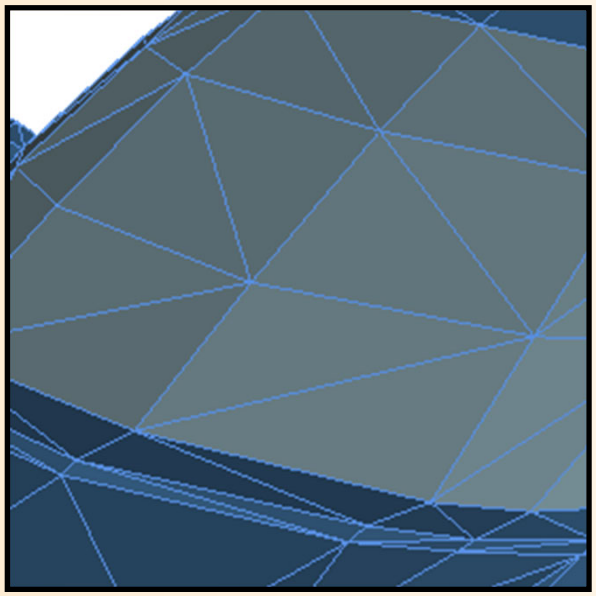
```
// 地面を描画
glBegin( GL_POLYGON );
        glNormal3f( 0.0, 1.0, 0.0 );
        glColor3f( 0.5, 0.8, 0.5 );

        glVertex3f( 5.0, 0.0, 5.0 );
        glVertex3f( 5.0, 0.0,-5.0 );
        glVertex3f(-5.0, 0.0,-5.0 );
        glVertex3f(-5.0, 0.0, 5.0 );
glEnd();
```



ポリゴンモデル（復習）

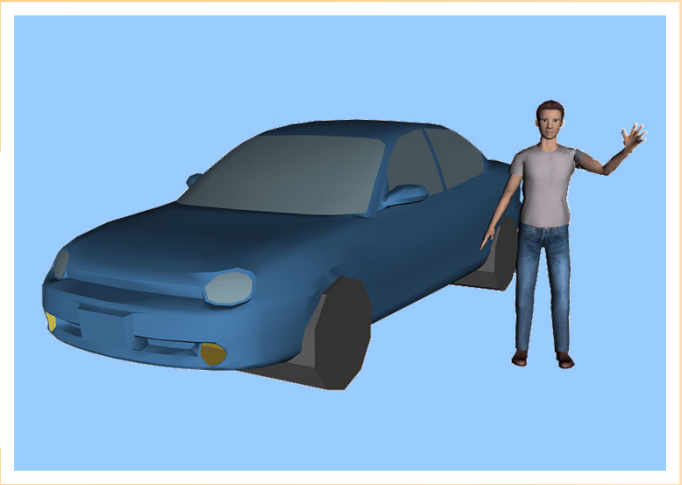
- 物体の表面の形状を、多角形（ポリゴン）の集まりによって表現する方法
  - 最も一般的なモデリング技術
  - 本講義の演習でも、ポリゴンモデルを扱う

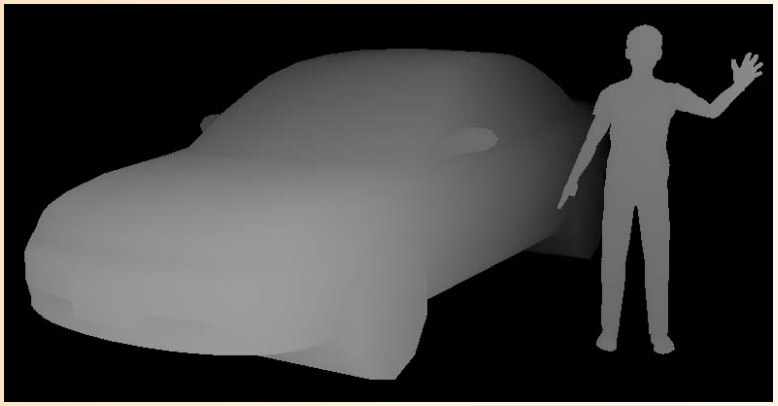




Zバッファ法（復習）

- Zバッファ法
  - 描画を行う画像とは別に、画像の各ピクセルの奥行き情報を持つ Zバッファ を使用する
  - コンピュータゲームなどのリアルタイム描画で、最も一般的な方法（本講義の演習でも使用）



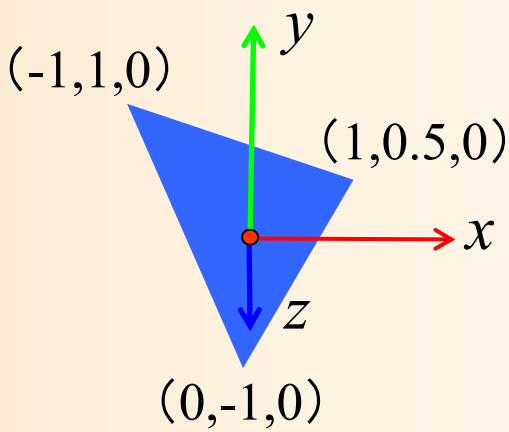


Zバッファの値（手前にあるほど明るく描画）

描画関数(3/4)

- 同じく、1枚の三角形を描画
  - 各頂点の頂点座標、法線、色を指定して描画
  - ポリゴンを基準とする座標系(モデル座標系)で頂点位置・法線を指定

```
glBegin( GL_TRIANGLES );
        glColor3f( 0.0, 0.0, 1.0 );
        glNormal3f( 0.0, 0.0, 1.0 );
        glVertex3f(-1.0, 1.0, 0.0 );
        glVertex3f( 0.0,-1.0, 0.0 );
        glVertex3f( 1.0, 0.5, 0.0 );
glEnd();
```



参考：複雑なポリゴンモデルの描画

- プログラムに直接頂点座標等を記述するのではなく、以下のように、配列を使ってデータを管理するのが一般的（詳しくは後日説明）

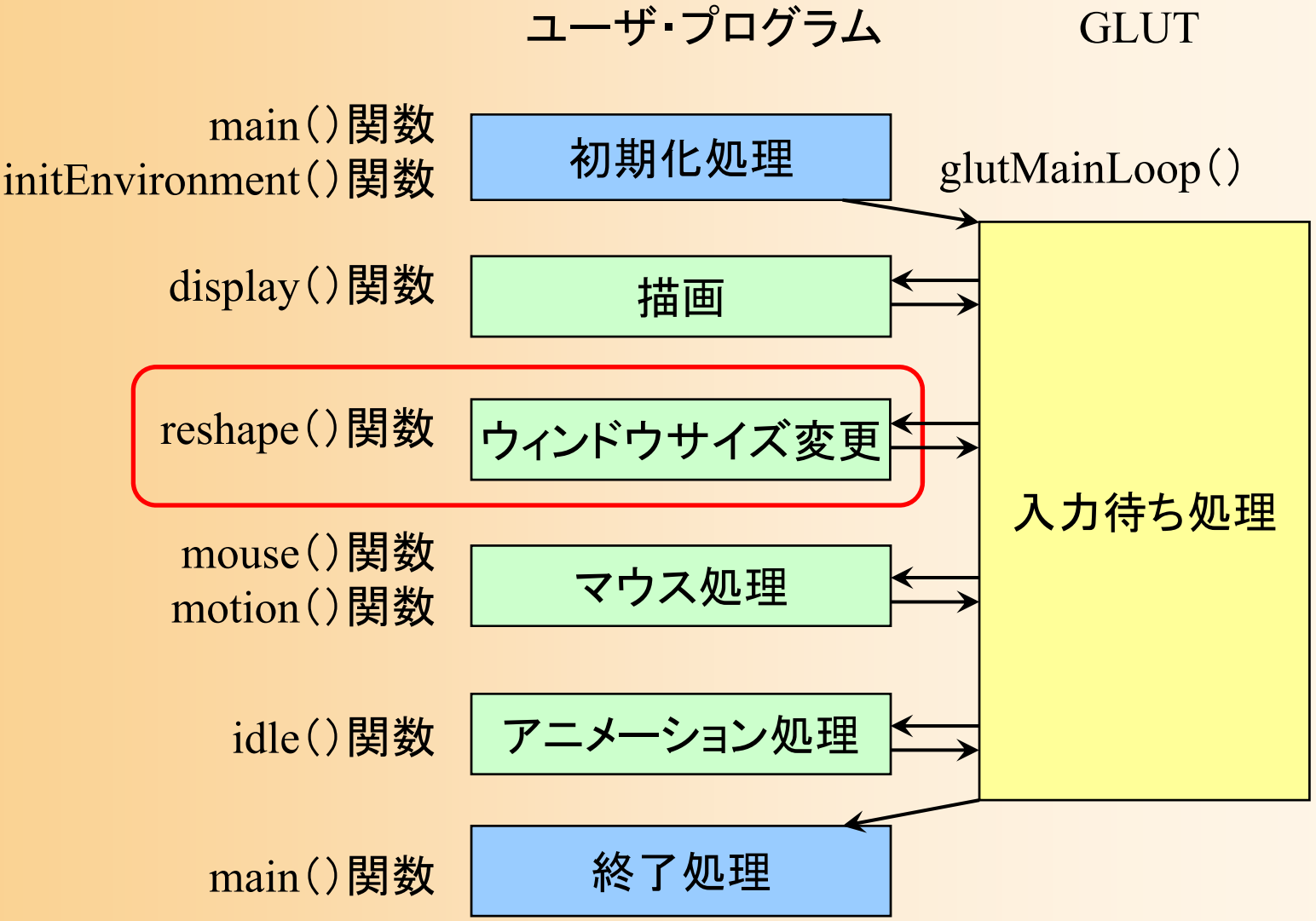
```
const int  num_pyramid_vertices = 5;    // 頂点数
const int  num_pyramid_triangles = 6;   // 三角面数

// 角すいの頂点座標の配列
float  pyramid_vertices[ num_pyramid_vertices ][ 3 ] = {
          { 0.0, 1.0, 0.0 }, { 1.0,-0.8, 1.0 }, { 1.0,-0.8,-1.0 },
          {-1.0,-0.8, 1.0 }, {-1.0,-0.8,-1.0 }
};

// 三角面インデックス（各三角面を構成する頂点の頂点番号）の配列
int  pyramid_tri_index[ num_pyramid_triangles ][ 3 ] = {
          { 0,3,1 }, { 0,2,4 }, { 0,1,2 }, { 0,4,3 }, { 1,3,2 }, { 4,2,3 }
};
```

描画関数(4/4)

- 描画完了
  - 描画途中の画面が表示されることを避けるために、描画は裏画面(バックバッファ)に行い、描画が完了したところで、表画面(フロントバッファ)に表示する

```
        ……

        // バックバッファに描画した画面をフロントバッファに表示
        glutSwapBuffers();
}
```

サンプルプログラムの構成



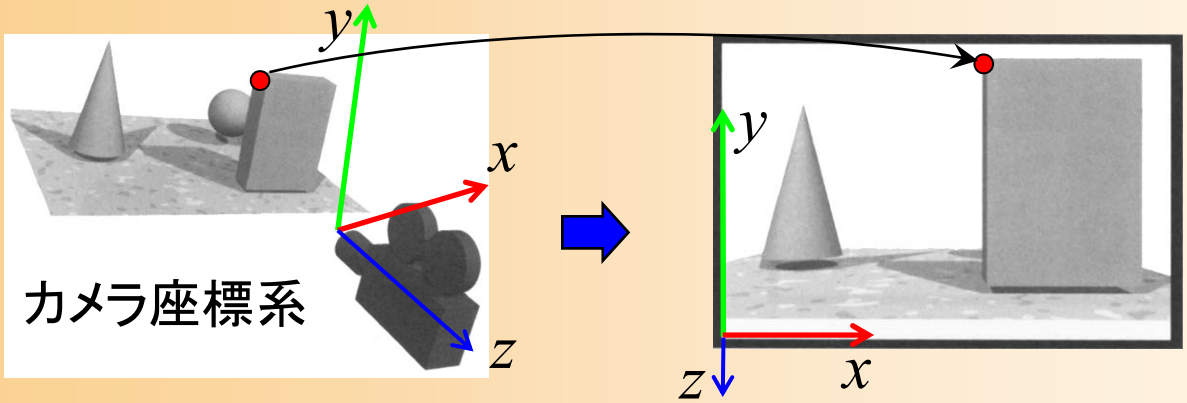
ウィンドウサイズ変更時の処理

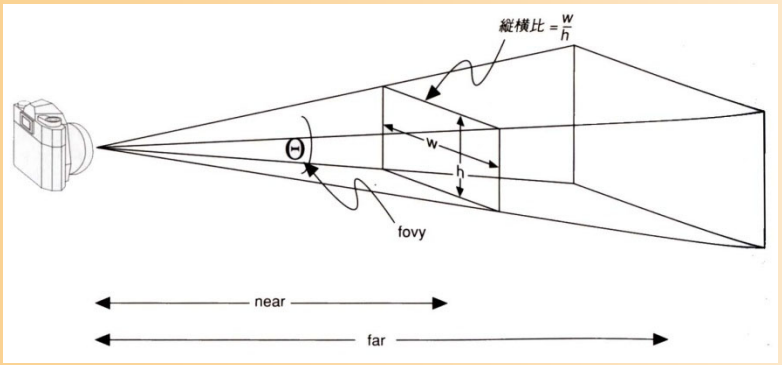
- 画面全体に描画を行うよう設定
- 射影変換行列の設定（視野角を45度とする）
  - 通常は、この設定のままで、変更は必要ない

```
void  reshape( int w, int h )
{
        // ウィンドウ内の描画を行う範囲を設定
        //（ウィンドウ全体に描画するよう設定）
        glViewport(0, 0, w, h);

        // カメラ座標系→スクリーン座標系への変換行列を設定
        glMatrixMode( GL_PROJECTION );
        glLoadIdentity();
        gluPerspective( 45, (double)w/h, 1, 500 );
}
```

参考：射影変換の設定

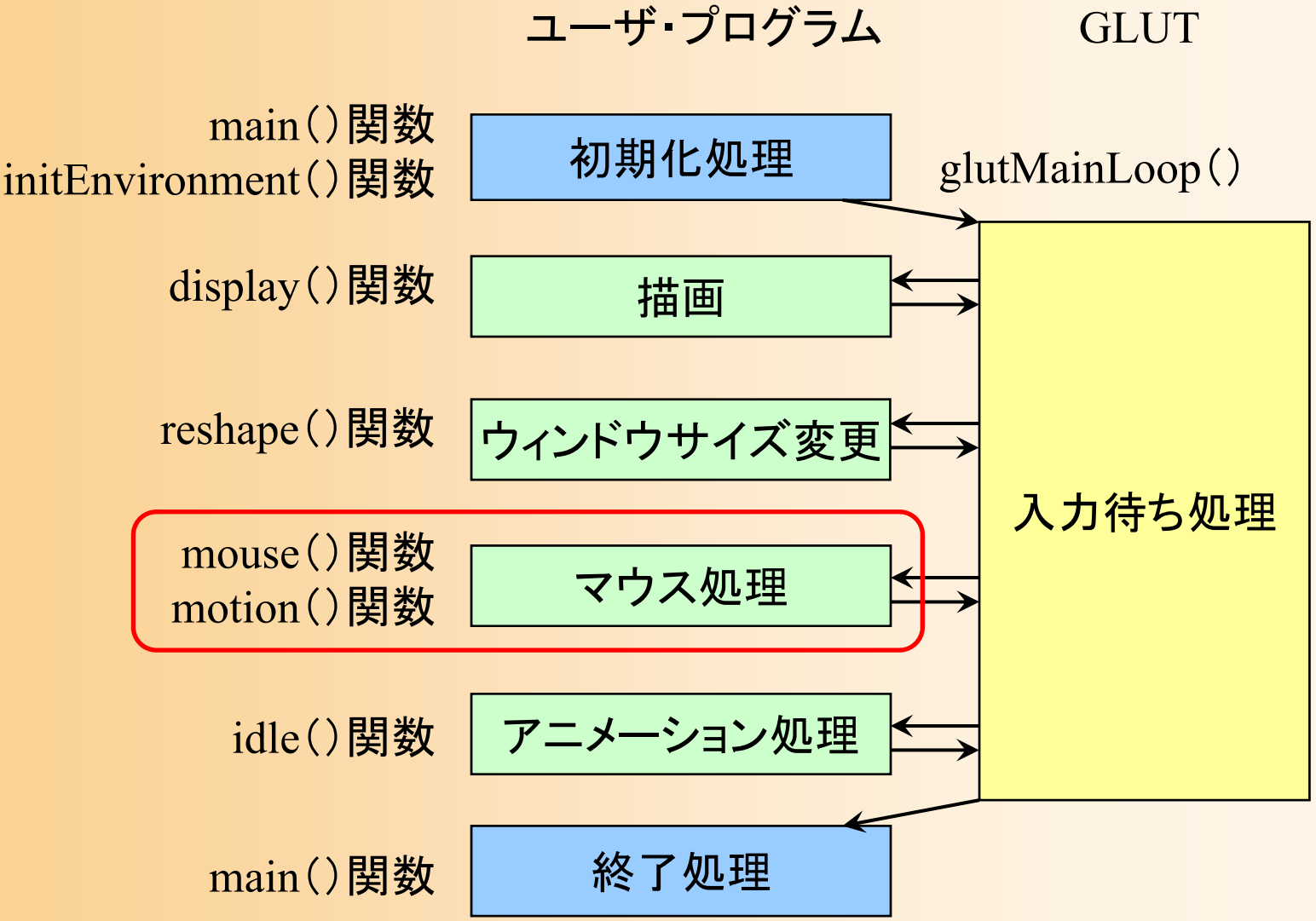
- カメラ座標系からスクリーン座標系への座標変換（射影変換）の設定（詳細は後日）





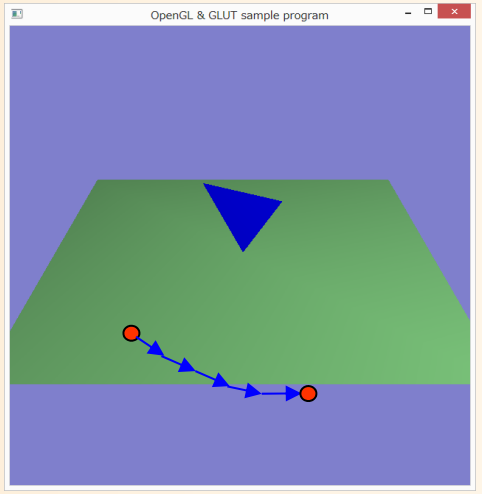
遠くにあるものほど小さく描画されるような変換（透視射影変換）を適用

サンプルプログラムの構成



マウス操作時の処理

- マウス操作のコールバック関数
  - ボタン処理（mouse（）関数）
    - マウスのボタンが、押されたとき、または、離されたときに呼ばれる
  - ドラッグ処理（motion（）関数）
    - マウスのボタンが押された状態で、マウスが動かされたとき（ドラッグ時）に定期的に呼ばれる
    - ボタンが押されない状態で、マウスが動かされたときに呼ばれる関数もある（今回は使用しない）



視点操作のための変数

- グローバル変数として定義
  - カメラの回転角度
  - マウスに関する変数
    - ドラッグフラグ、前回のXY座標

```
// 視点操作のための変数
float  camera_pitch = -30.0;  // X軸を軸とするカメラの回転角度

// マウスのドラッグのための変数
int  drag_mouse_r = 0;  // 右ボタンをドラッグ中かどうかのフラグ
                                (0:非ドラッグ中,1:ドラッグ中)
int  last_mouse_x;      // 最後に記録されたマウスカーソルのX座標
int  last_mouse_y;      // 最後に記録されたマウスカーソルのY座標
```

マウス操作の処理

- マウスクリック時に呼ばれる関数（mouse）
  - ボタンの状態を記録
  - マウス座標を記録
- マウスドラッグ時に呼ばれる関数（mouse）
  - マウス座標の変化量を計算
  - 視点の回転量（camera_pitch）を計算
  - マウス座標を記録
  - 画面の再描画

マウス操作時の処理(クリック処理関数)

- 右ボタンがクリックされたことを記録
  - 変数 drag_mouse_r に状態を格納

```
// マウスクリック時に呼ばれるコールバック関数
void  mouse( int button, int state, int mx, int my )
{
        // 右ボタンが押されたらドラッグ開始のフラグを設定
        if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
                drag_mouse_r = 1;
        // 右ボタンが離されたらドラッグ終了のフラグを設定
        else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
                drag_mouse_r = 0;

        // 現在のマウス座標を記録
        last_mouse_x = mx;
        last_mouse_y = my;
}
```

マウス操作時の処理(ドラッグ処理関数1)

- ドラッグされた距離に応じて視点を変更
  - 視点の方位角 camera_pitch を変化
    - 前回と今回のマウス座標の差から計算

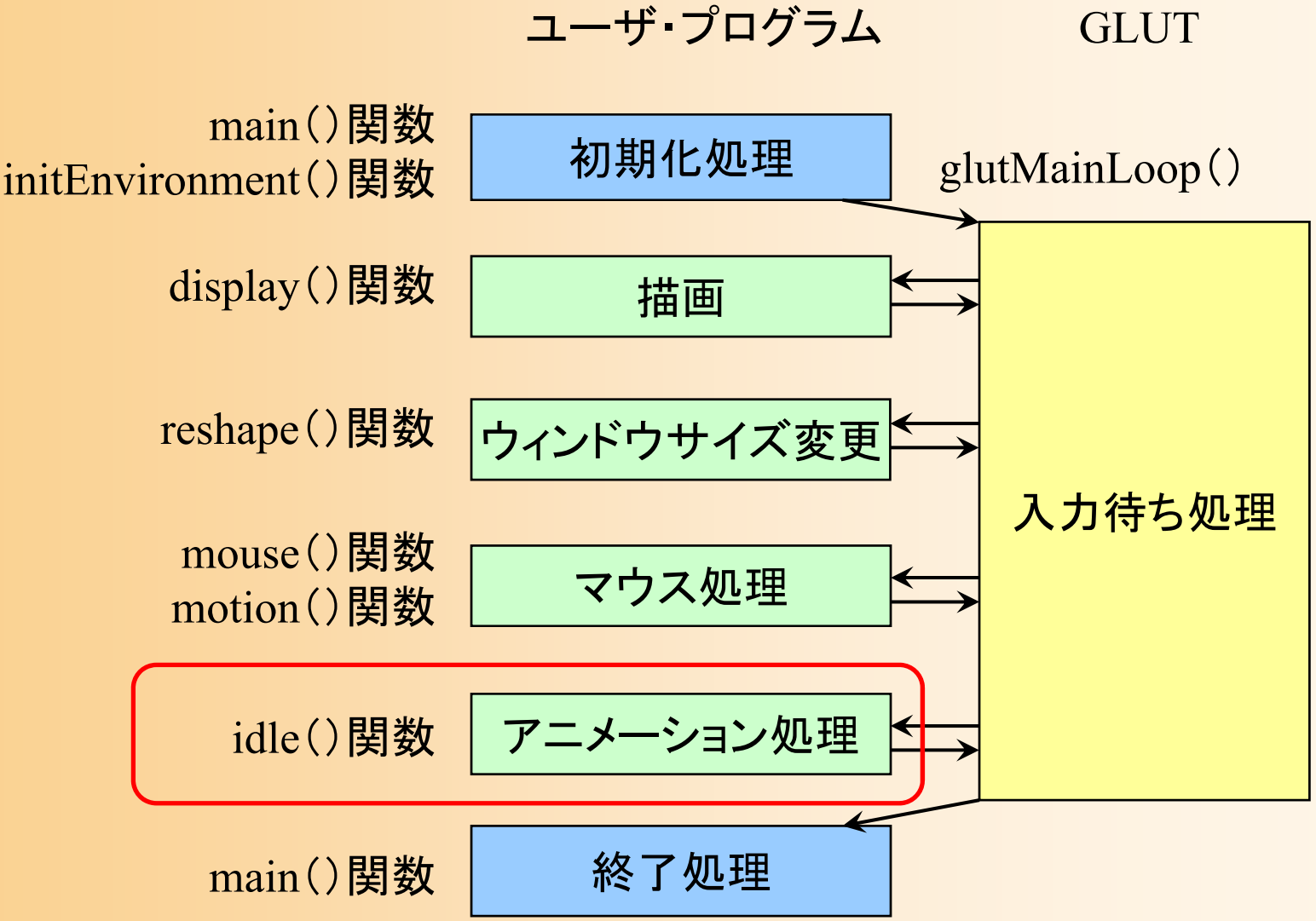
```
void  motion( int mx, int my )
{
        // 右ボタンのドラッグ中であれば、
        // マウスの移動量に応じて視点を回転する
        if ( drag_mouse_r == 1 )
        {
                // マウスの縦移動に応じてX軸を中心に回転
                camera_pitch -= ( my - last_mouse_y ) * 1.0;
                if ( camera_pitch < -90.0 )
                        camera_pitch = -90.0;
                else if ( camera_pitch > 0.0 )
                        camera_pitch = 0.0;
        }
        ......
```

マウス操作時の処理(ドラッグ処理関数2)

- 再描画の指示を行う
  - 視点の方位角 camera_pitch の変化に応じて、画面を再描画するため

```
        // 今回のマウス座標を記録
        last_mouse_x = mx;
        last_mouse_y = my;

        // 再描画の指示を出す
        glutPostRedisplay();
}
```

サンプルプログラムの構成



アイドル時の処理

- 描画やマウス入力を処理する必要がないときに定期的に呼ばれる関数
  - 物体の位置・向きを少しずつ変化させるといった、アニメーションを実現するために利用できる
  - サンプルプログラムでは、現在は何も処理を行っていない（今後処理を追加する）

```
void  idle( void )
{
        // 現在は、何も処理を行なわない
}
```

プログラムのコンパイルと実行

演習環境

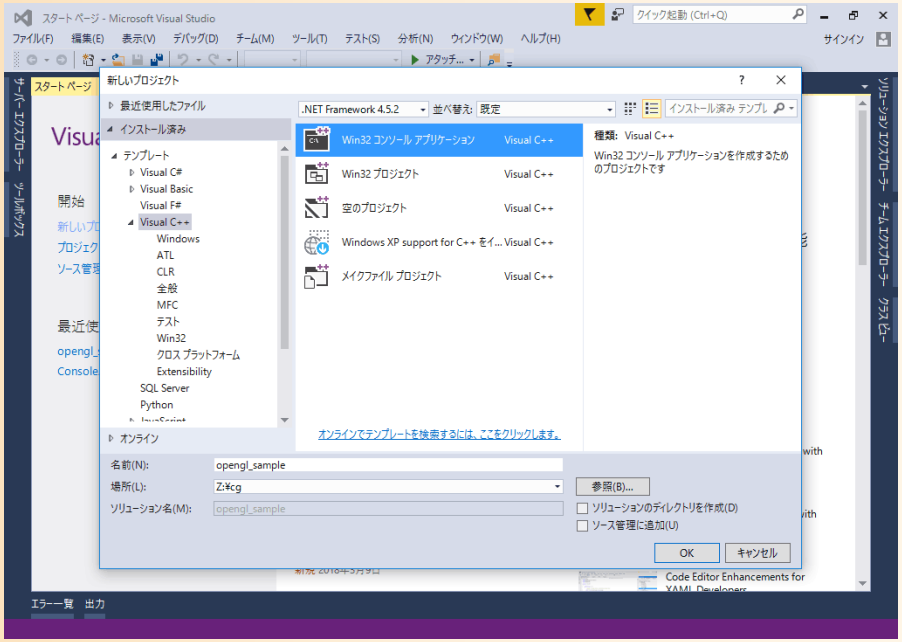
- Windows ＋ Visual Studio
  - 統合開発環境
- ソリューション ＋ プロジェクト
  - ソースファイルをコンパイルするために必要
  - プロジェクトはプログラムを構成するソースファイルや設定を定義
  - リューションは複数のプロジェクトをまとめるもの
    - 一つのソースファイルのみで構成される単純なプログラムをコンパイルする場合でも、必ずソリューションやプロジェクトを作成する必要がある

サンプルプログラム

- Moodleに置かれているサンプルプログラムをダウンロードして使用
- opengl_sample.cpp
  - 拡張子は cpp（C++ソースファイル）
  - 文字コード UTF 8、改行コード CR＋LF
    - Linux等の別の環境でも演習を行いたい場合は、文字コード・改行コードの変換が必要

コンパイル方法

- Moodleのコンパイル方法の資料を参照
  - ソリューション＋プロジェクトの作成
  - プロジェクトにソースファイルを追加
  - コンパイルと実行
  - デバッグ
- ホームディレクトリの適当な場所にプロジェクトを作成・保存する

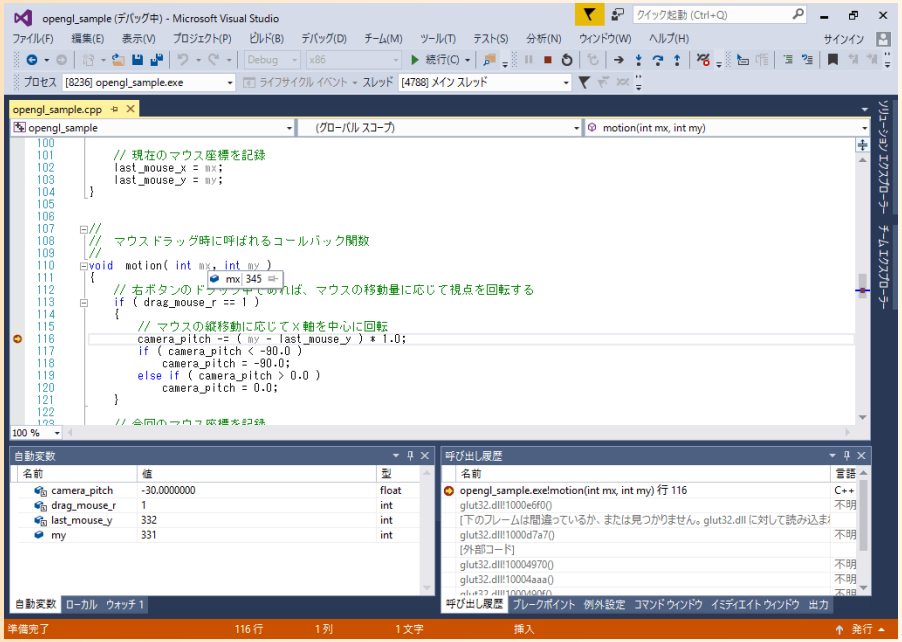


コンパイルエラーが出たとき

- ソースファイルにエラーがあるときには、エラーの原因や、エラーのある行が表示されるので、それらの情報をもとに、プログラムを修正する
  - Javaのプログラムでコンパイルエラーが出たときと同じ要領
  - エラーメッセージをよく読むことが重要
- サンプルプログラムは、何も修正しなければ、コンパイルエラーは出ないはず

デバッガ

- Visual Studio のデバッガの機能を活用することで、効率的にデバッグができる
  - プログラムの中断、ステップ実行
  - 変数の値を確認



参考：Linux環境でのコンパイル方法

- ターミナルから、gcc（GNU C コンパイラ）を、以下のようなコマンドで実行し、コンパイル

```
gcc opengl_sample.cpp -L/usr/X11R6/lib -lglut
-lGLU -lGL -lXmu -lm -o opengl_sample
```

  - opengl_sample.c ・・・ 入力のソースファイル名
  - opengl_sample ・・・ 出力の実行形式ファイル名

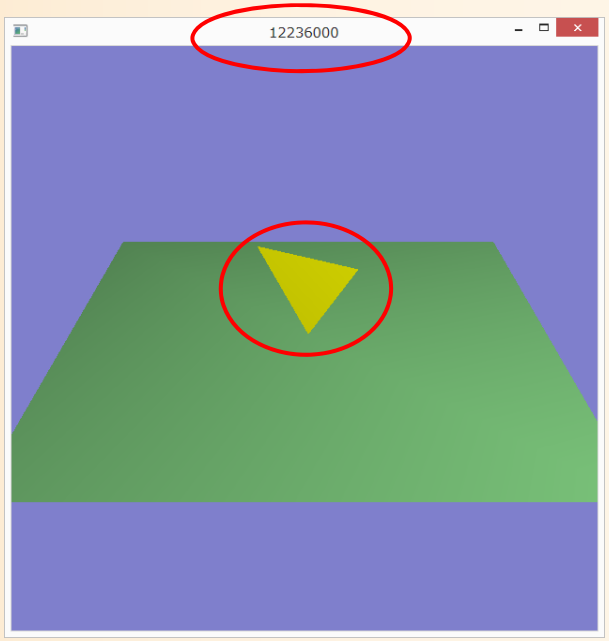
- 実行方法（出力ファイル名をターミナルから入力）

```
opengl_sample
```

演習課題

演習内容

1. コンパイル・実行ができることを確認する
2. プログラムを以下の通り変更する
   - ウィンドウのタイトルに、自分の学生番号が表示されるようにする
   - 三角形の色を、青から黄に変更する
3. 修正が終わったら、Moodleからプログラムを提出



プログラムの修正(1)

- ウィンドウのタイトルに、自分の学生番号が表示されるように変更する

```
//
//  メイン関数(プログラムはここから開始)
//
int  main( int argc, char ** argv )
{
        // GLUTの初期化
        glutInit( &argc, argv );
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA );
        glutInitWindowSize( 320, 320 );
        glutInitWindowPosition( 0, 0 );
        glutCreateWindow( "OpenGL & GLUT sample program" );

        ......                          ↓
                                  “14236000"(自分の学生番号)
}
```

プログラムの修正(2)

- 三角形の色を青から黄に変更する

```
//
//  ウィンドウ再描画時に呼ばれるコールバック関数
//
void  display( void )
{
        ......

        // 物体(1枚のポリゴン)を描画
        glBegin( GL_TRIANGLES );
                glColor3f( 0.0, 0.0, 1.0 );  → RGBの各成分を 0.0～1.0 の範囲で指定
                glNormal3f( 0.0, 0.0, 1.0 );
                glVertex3f(-1.0, 1.0, 0.0 );
                glVertex3f( 0.0,-1.0, 0.0 );
                glVertex3f( 1.0, 0.5, 0.0 );
        glEnd();

        ......
}
```

プログラムの提出

- 作成したプログラムを「学生番号.cpp」のファイル名で、Moodleから提出
  - 本講義時間中に終らせて、提出
  - 終らなかった人は、下記の締め切りまでに提出
- 締め切り: 6月25日（火）18:00
  - 演習問題と同様、成績に加える
    - 未完成のプログラムが提出されていれば、減点
  - 今回の内容で、演習のやり方をきちんとおさえておかないと、次回以降の演習が全くできないため、必ず課題を行って提出すること

まとめ

- 前回の復習
- 演習環境
  - OpenGLとGLUTの概要
- サンプルプログラムの解説
  - サンプルプログラムの概要
  - 変換行列の設定、描画、光源情報の設定、等
- プログラミング演習
  - コンパイルと実行、演習課題

次回予告

- モデリング
  - オブジェクトの形状表現
  - オブジェクトの作成方法

オブジェクトの作成方法

オブジェクトの形状表現

オブジェクト

表面の素材の表現

動きのデータの生成

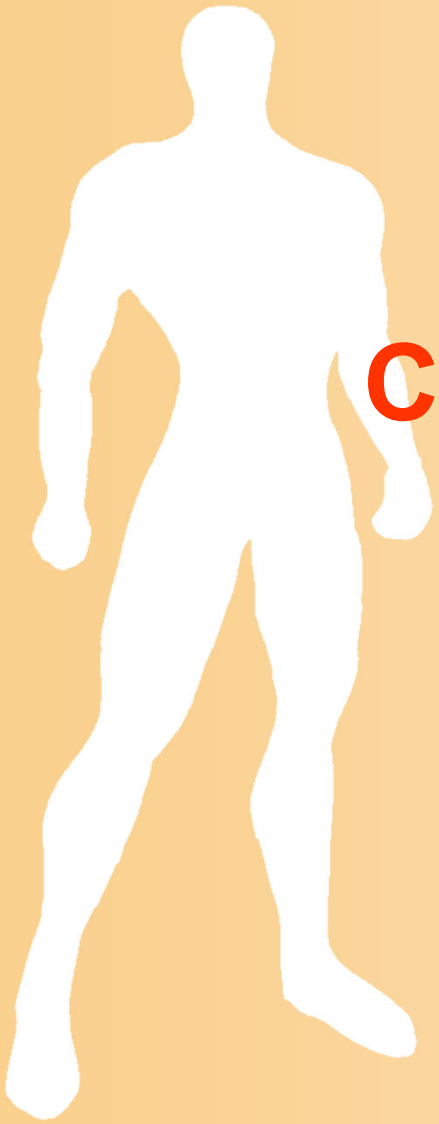
光源

光の効果の表現

カメラから見える画像を計算

画像処理

以下、補足資料

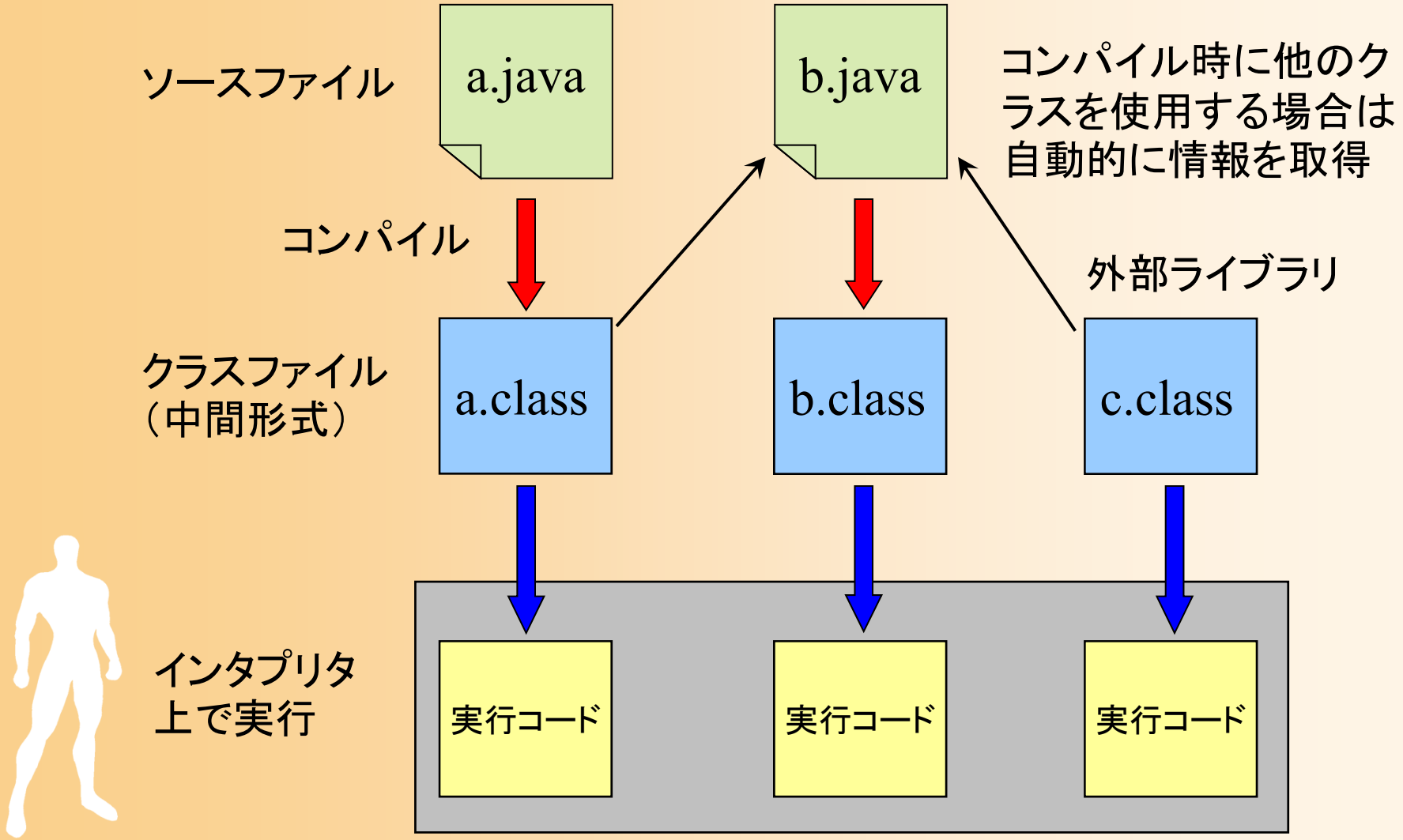


C言語とJavaの違い

Javaのコンパイルと実行の仕組み

- Java
  - それぞれのソースファイルをコンパイルして中間形式のクラスファイルを生成
    - コンパイル時に必要な他のクラスの情報は、他のクラスのクラスファイルを直接読みにいく
  - 実行時は、インタプリタがクラスファイルを読み込んで、実行形式（CPUが理解する命令群）に変換しながら実行
  - 必要なライブラリは、実行時に読み込まれる

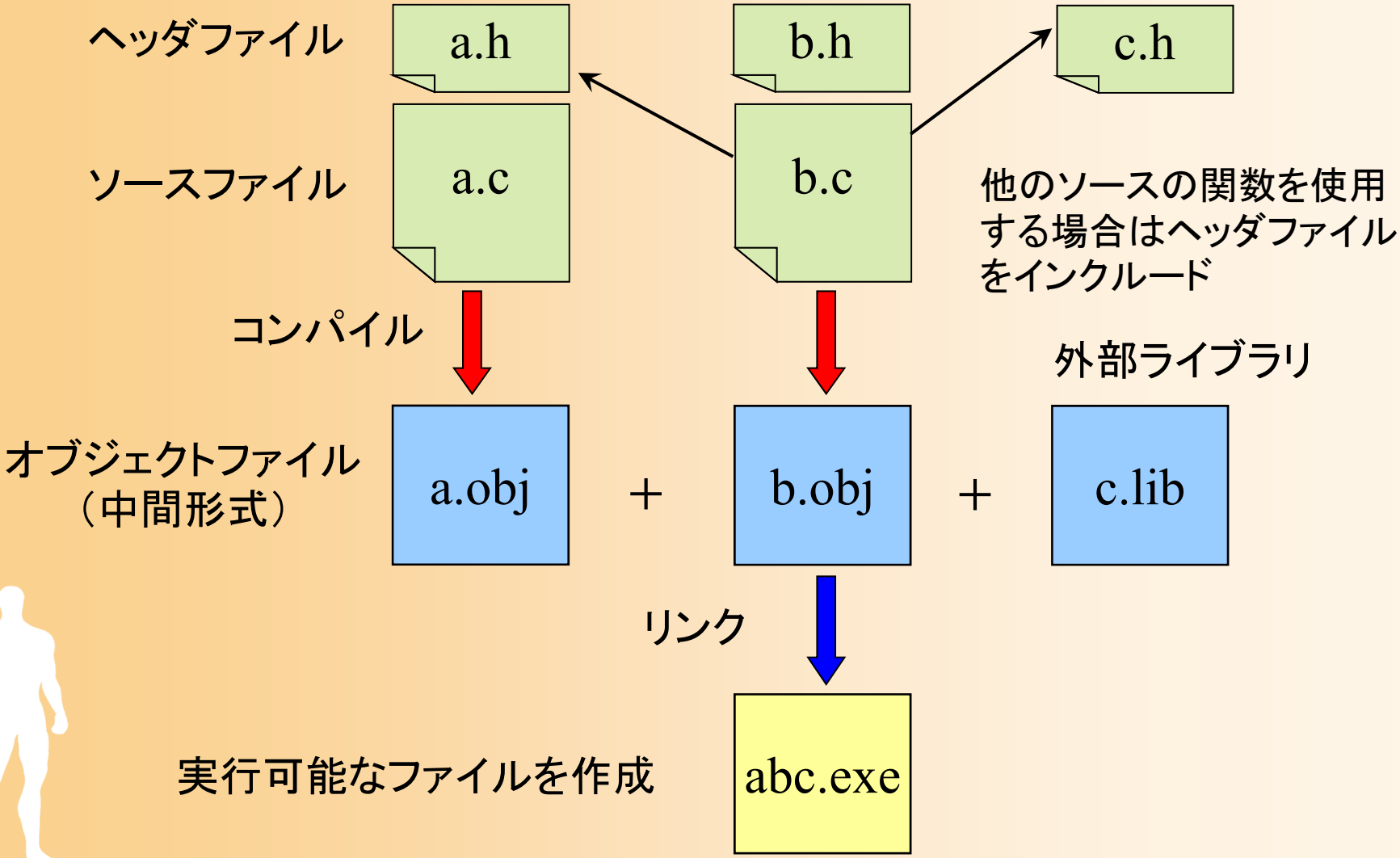
Javaのコンパイルと実行の仕組み



Cのコンパイルと実行の仕組み

- C言語（C++）
  - それぞれのソースファイルをコンパイルして中間ファイル（オブジェクトファイル）を生成
  - 全てのオブジェクトファイルと必要なライブラリをリンクして実行形式のプログラムを生成
  - プログラムは直接実行可能
  - （ CやC++にも実行時に動的にライブラリを読み込む方法がある－ダイナミックリンクライブラリ ）

Cのコンパイルと実行の仕組み



比較

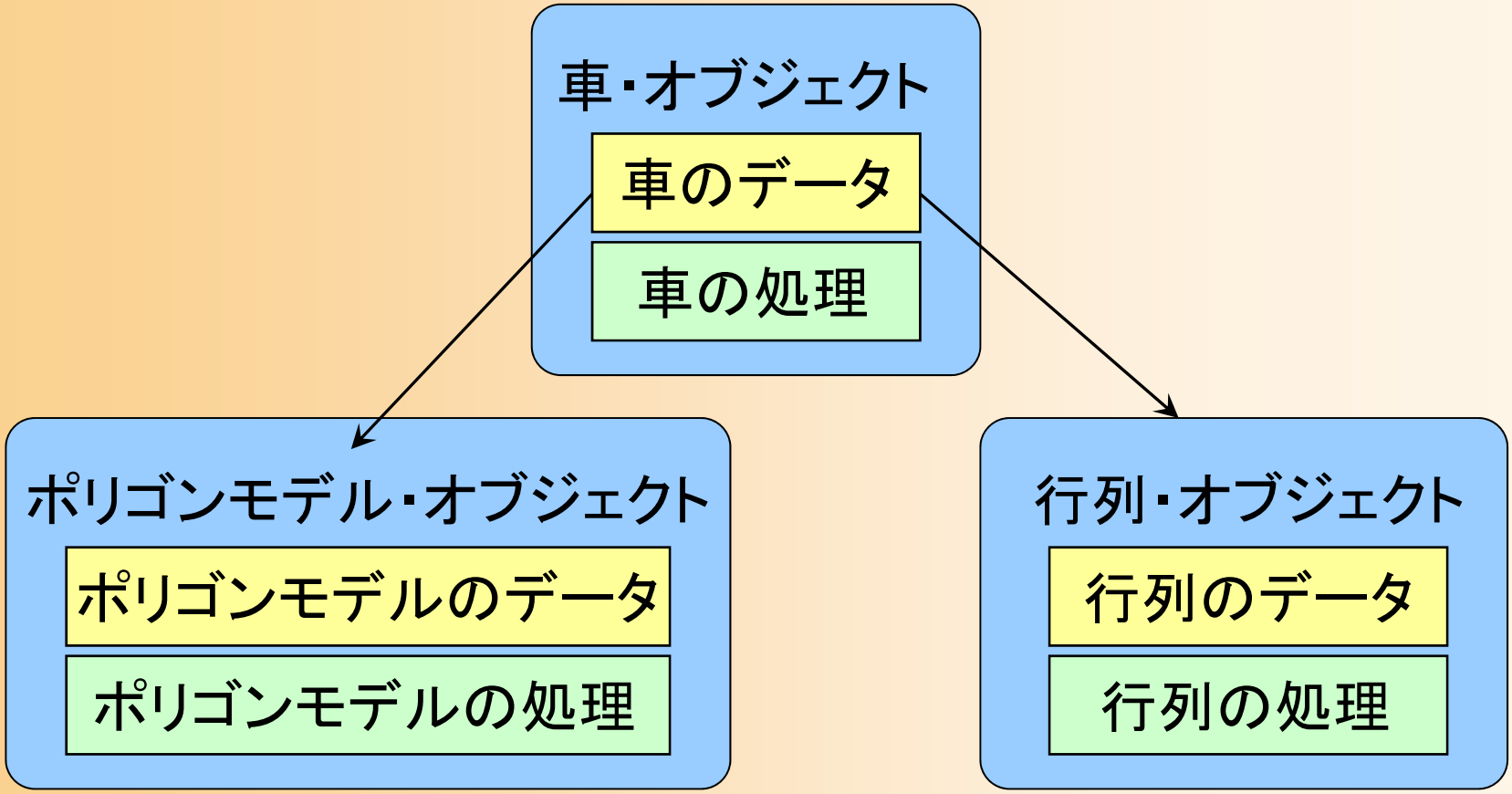
- Java
  - 中間形式なのでどの環境でもインタプリタさえあれば動く
  - 変換しながら実行するので遅い（特に起動に時間がかかる）
  - 新しい言語なので仕様が洗練されている
  - 標準ライブラリの機能が充実している
- C言語（C++）
  - 違う環境で動かすためには再コンパイルが必要
  - よりコンピュータに近い言語

言語思想の違い

- オブジェクト指向言語
  - データと処理をまとめたものをクラスとして定義
  - オブジェクト同士がデータを交換し合うことで大きな処理を実現
  - 複雑なデータをモデル化するのに適している
- 構造化言語
  - データと処理を分離し、データを処理する独立した関数を定義
  - 細かい機能を組み合わせ大きな機能を実現

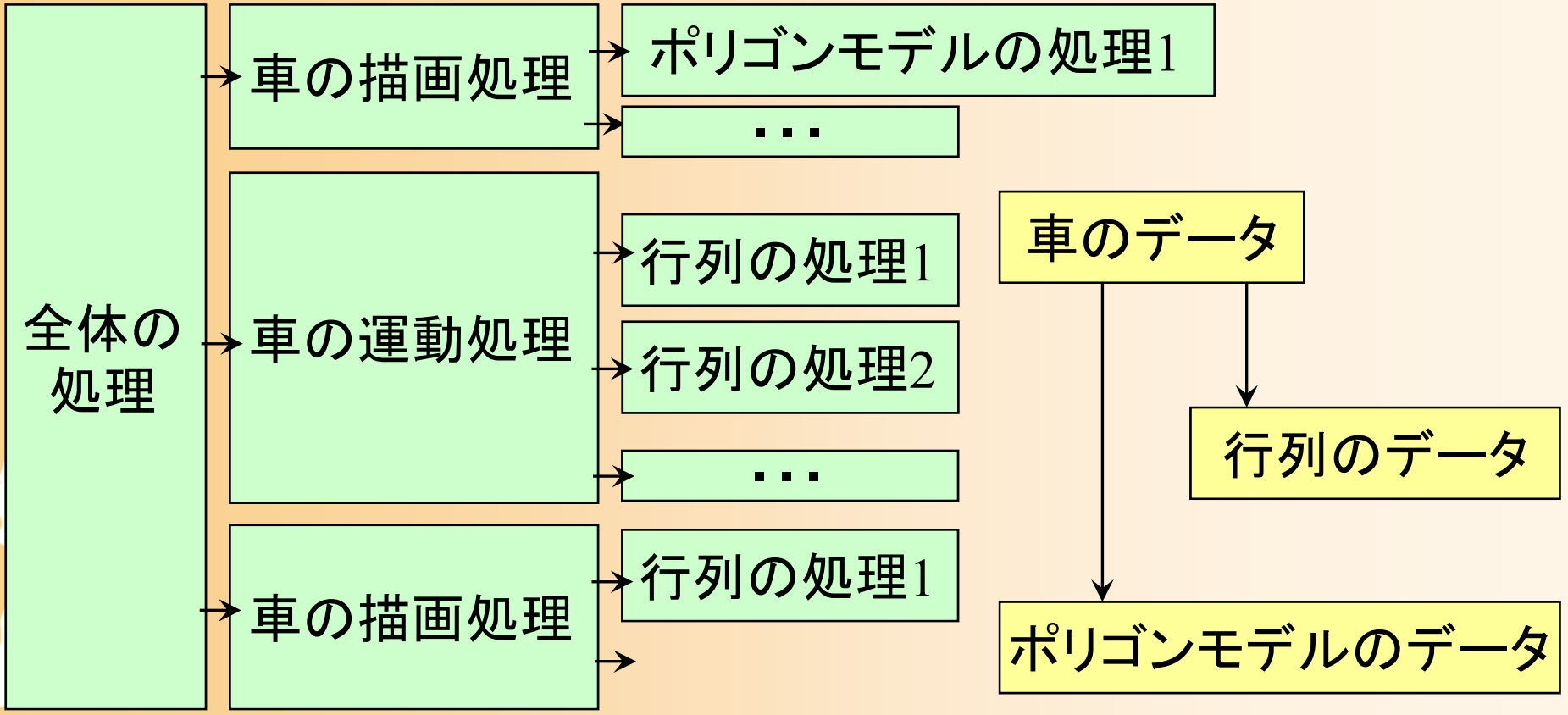
オブジェクト指向

- オブジェクト指向による設計の例



構造化言語

- 構造化言語による設計の例



オブジェクト指向の機能

- カプセル化
  - オブジェクトの変数をメソッドを通じてしか操作できないようにすることで安全性を保つ
- 継承
  - クラスを継承して新たな変数やメソッドを追加
- 多態（ポリモーフィズム）
  - 同じ名前のメソッドをそれぞれの派生クラスで実装することで、同じインターフェースで異なる振る舞いを行わせることができる

使うときの違い

- 構造化言語
  - それぞれの関数を理解すれば使える
  - 機能ごとの再利用がやりやすい
  - データと処理を分離できる
- オブジェクト指向言語
  - オブジェクトやクラスライブラリの位置づけを理解しないと使うのが難しい
  - オブジェクト単位で再利用することになる
  - データと処理を統合できる

使い分け

- 両者の使い分けの注意
  - 作ろうとするプログラムに合った言語を使うことが望ましい
  - JavaやC++を使ったからといって必ずしもオブジェクト指向になるわけではないことに注意！
    - C言語を使ってもオブジェクト指向風の設計はできる
    - オブジェクト指向の方が使いこなすのが難しい
    - オブジェクト指向と構造化設計の混在も可能
  - 良いプログラムを書くためには、自分がどういう方針で設計するのか意識しておく必要がある

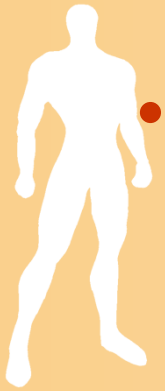
C言語

C言語

- Cプログラムの構造
  - 関数の集まりによって構成される
- 文法の簡単な説明
  - 変数の型、文字列型、構造体、ポインタ、スコープ、制御構文、メモリ確保
- C言語で多くの人がつまづくところ
  - ポインタ、ポインタ配列、分割コンパイル、多次元配列
    - 今回の演習では不要なので、本講義では説明しない

変数の型

- 基本型・演算子はJavaと同じ
- 基本型
  - 整数　int (32ビット)
  - 実数　float (32ビット), double (64ビット)
  - 文字　char (8ビット)
  - 真偽　bool (8ビット)
- 演算子
  - +, *, /, ==, !=, ||, &&, !, ^, |, &, >>, <<, a?b:c



変数の型

- C言語には文字列型（String）がない！
  - 文字型の配列を文字列として使用
  - 配列のゼロ（¥0）が入っていればそこまでを一つの文字列とする
    - C言語の標準ライブラリの文字列処理関数は、このお約束を前提として文字型の配列を操作する
  - 配列の長さを自動的に変えることができないので、必要に応じて確保しなおす必要がある
  - C++には string型がある

char name[8] = “OpenGL”;

O	p	e	n	G	L	¥0	

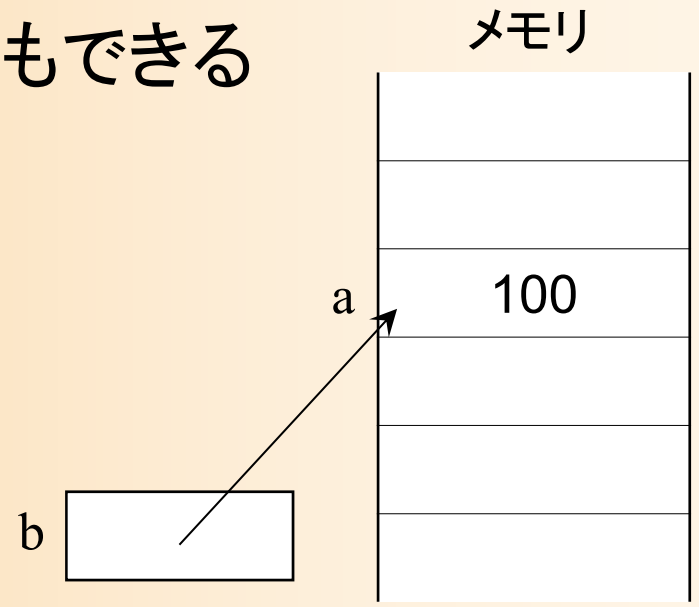
変数の型

- 構造体（struct）
  - クラスのように複数の変数をセットにして扱える
  - ただしメソッドは定義できない
- 型の別名（typedef）
  - 型に別名をつけられる
  - 例：typedef　unsigned int　GLuint;
  - OpenGLの関数の引数は全部独自の別名で定義されている（GLint, GLfloat）など
    - コンパイラによって int のサイズが変わったりする時のため

ポインタ

- 変数ではなく、変数のアドレスを指す変数
  - 他のデータへの参照などに使われる
  - Javaの参照と似ている
    - C++にはポインタとは別に参照もある
  - 参照と異なり足し算などもできる

```
int    a = 100;
int  * b; // ポインタ変数の宣言
b = & a;  // アドレス演算子
*b = 200; // ポインタの指す
             先に入力
```



変数のスコープ

- Java
  - ローカル（関数内）
  - クラスの属性
  - クラスの属性（静的）
    - 公開（public）、非公開（private）
- C言語
  - ローカル（関数内）
  - グローバル
    - 公開、非公開（static）


```
class  Test
{
   public int   num;
   private static int  g;

   public int put( int i )
   {
       float  f;
```

```
static int   num;

int put( int i )
{
   float  f;
}
```

制御構文

- 制御構文はJavaと同じ
  - if
  - for
  - while
  - do～while
  - switch
  - break

変数の確保と開放

- 配列などの確保と開放
  - new(), delete() を使う
  - （昔は malloc(), free() という関数を使用）
- C言語では確保した変数はかならず自分で開放する必要がある
  - そのまま残しておくとメモリリークの原因となる
  - Javaでは参照にnullを代入しておけば、どのオブジェクトからも参照されていないオブジェクトは、ガベージ・コレクタが自動的に削除

プリプロセッサ

- C/C++では、コンパイルの前にプリプロセッサによってソースが処理される
  - ソースに対してテキスト処理（文字列の置き換えなど）を行う
  - # で始まるのがプリプロセッサのためのコマンド
    - #include<～>・・・他のファイルを読み込む、主にヘッダファイルの読み込みに使用
    - #define A b ・・・ Aをbで置き換える