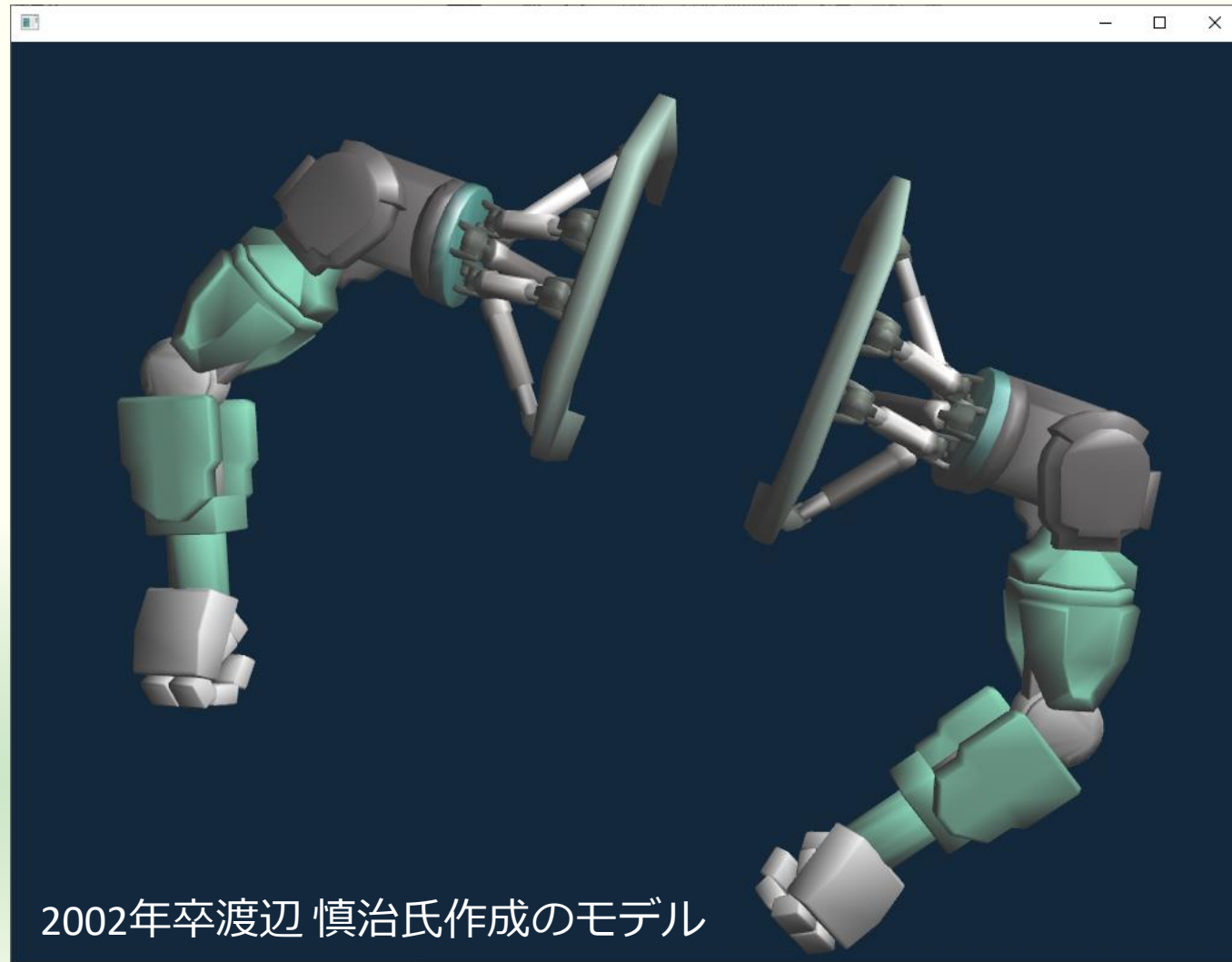




メディアプログラミング演習

第7回

ユーザインタフェースを持つアプリの作成



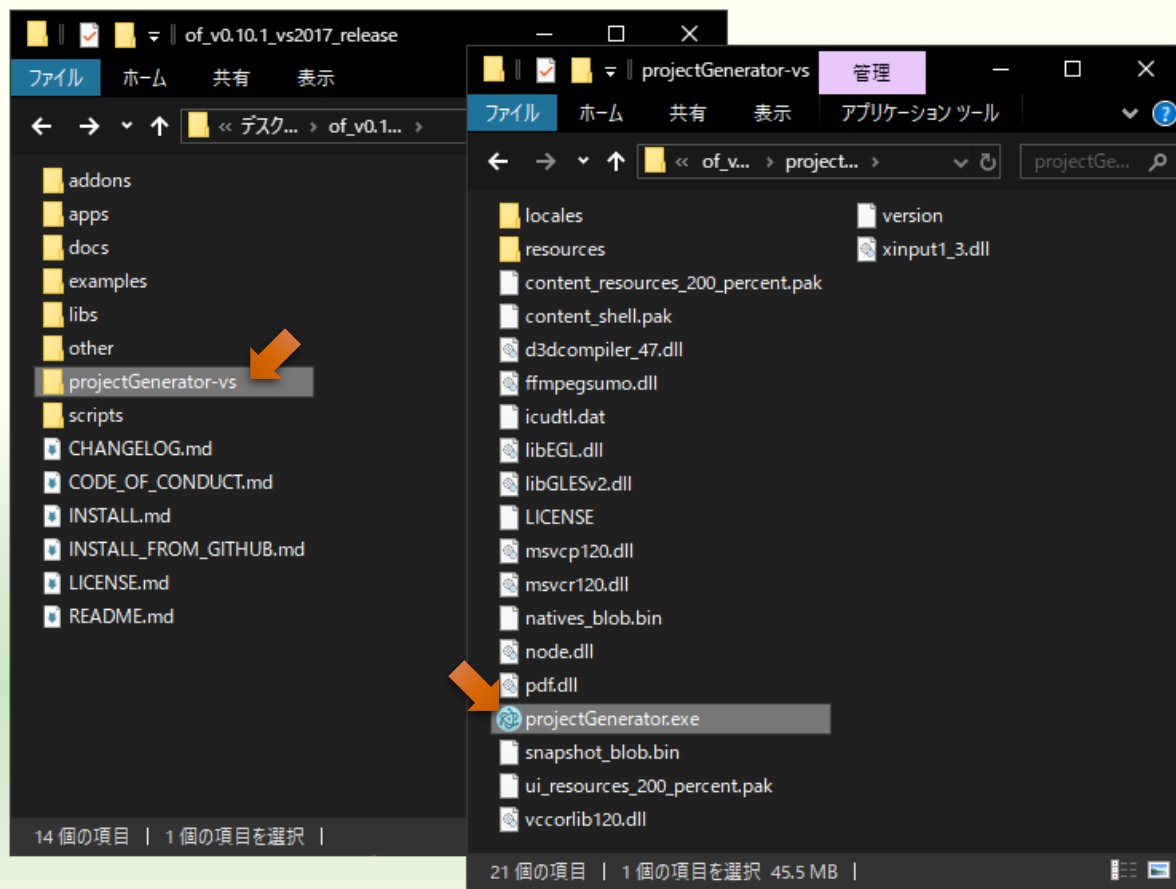


準備

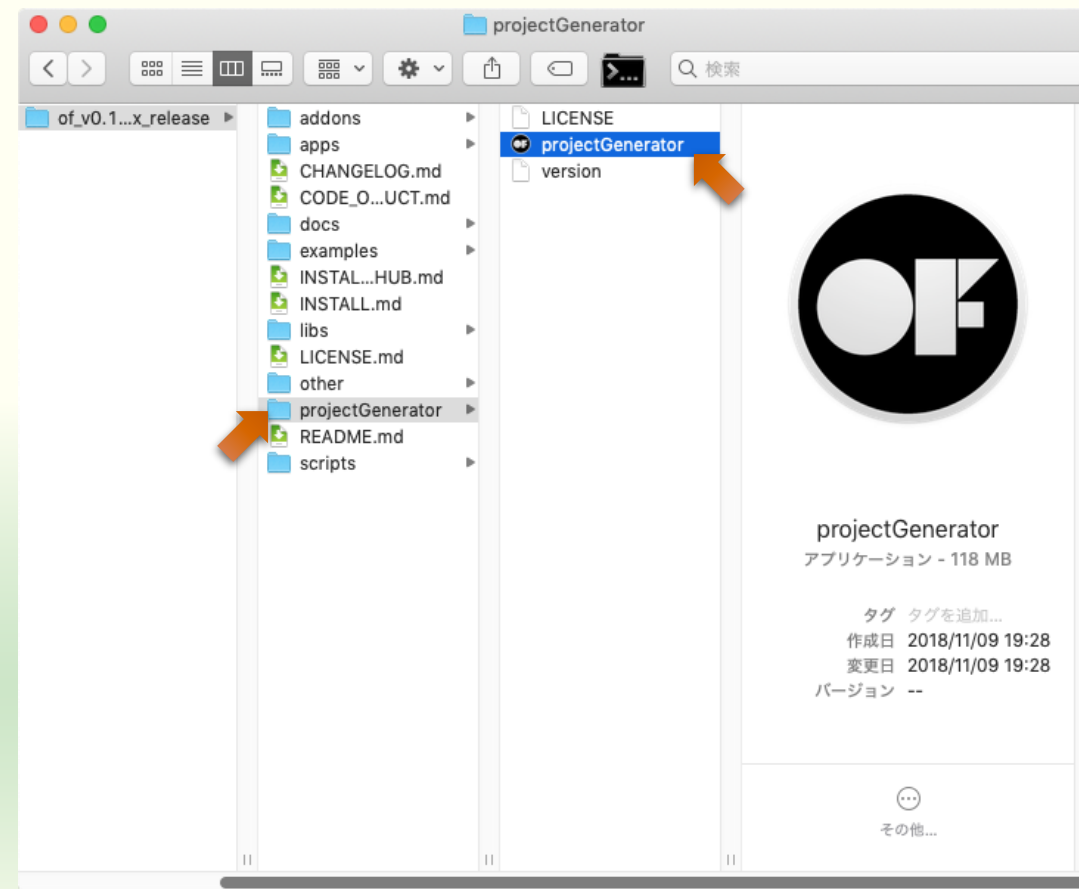
プロジェクトの作成

projectGenerator を起動する

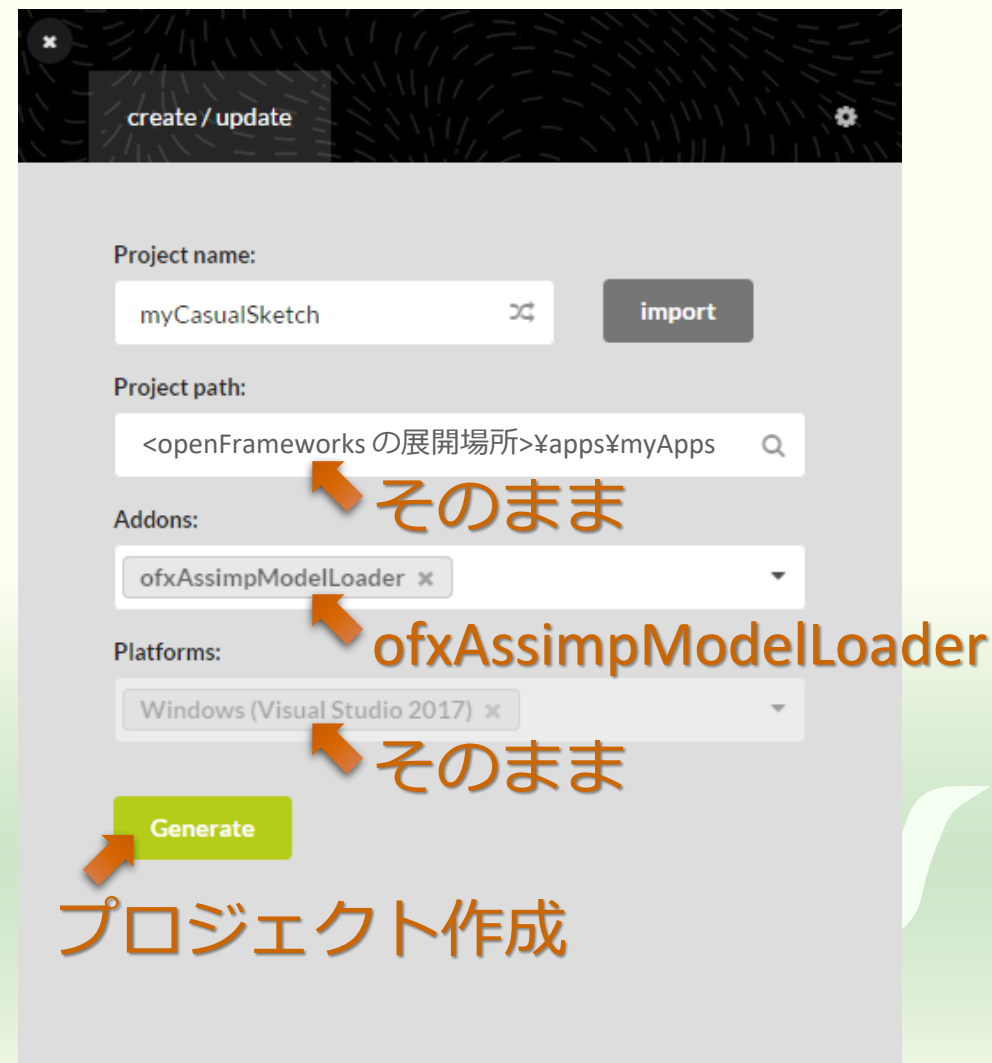
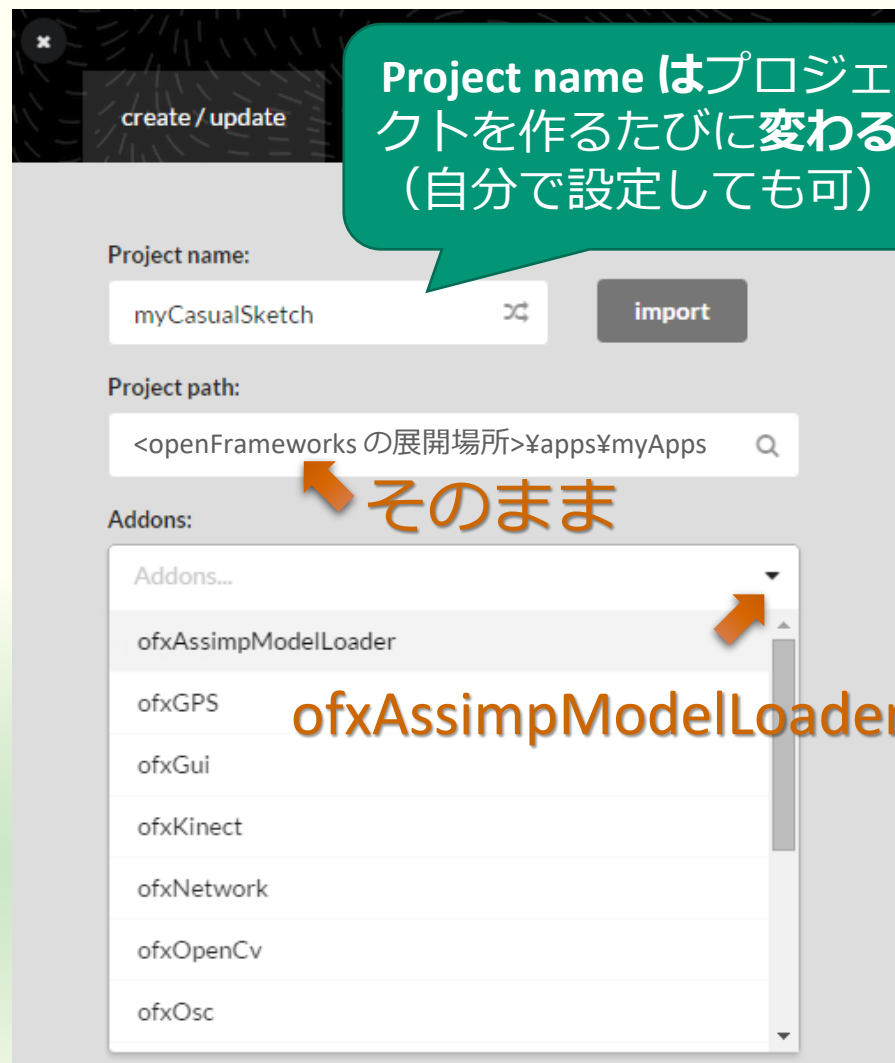
windows 版のパッケージ



macOS 版のパッケージ



空のプロジェクトの作成



ofxAssimpModelLoader アドオンについて

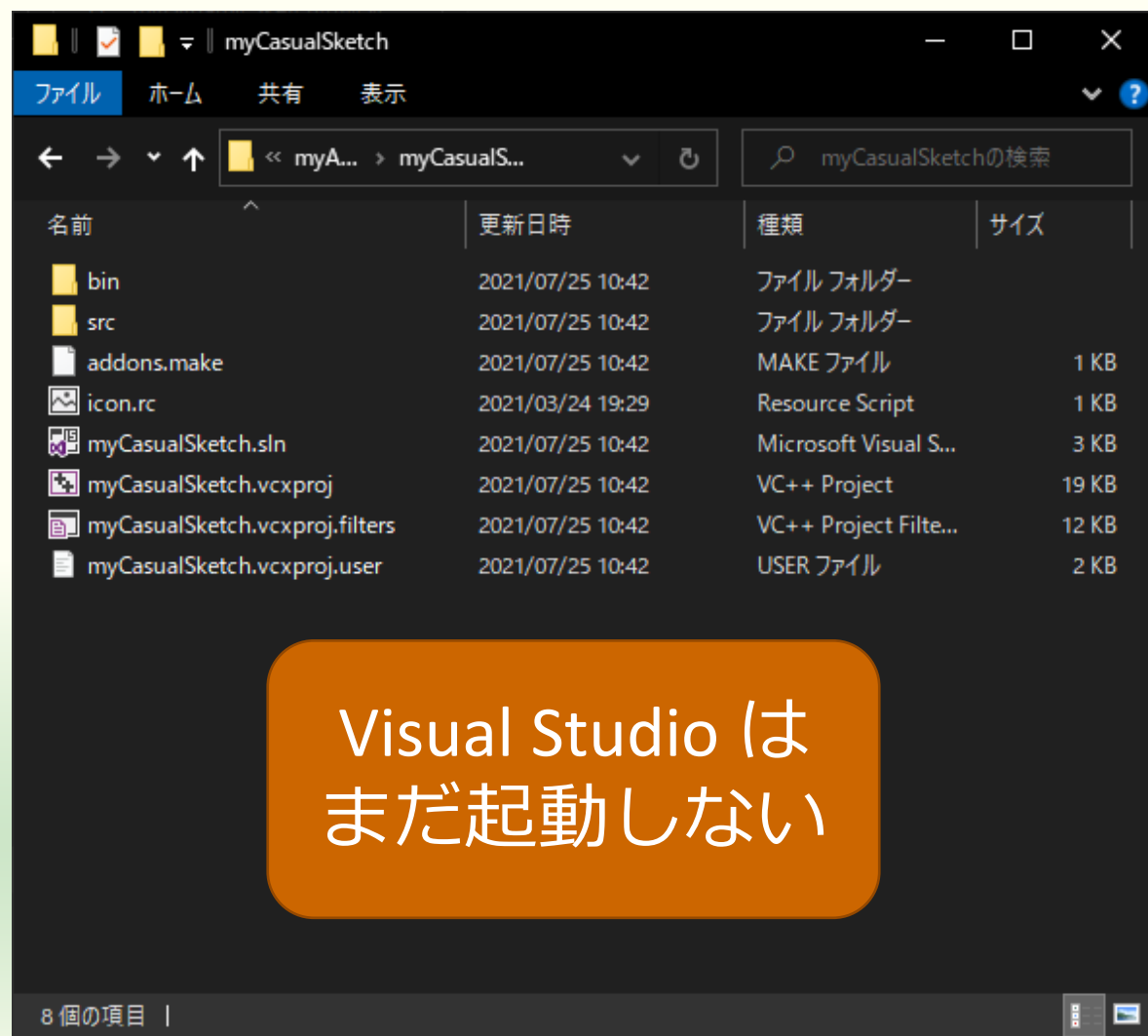
- 3D モデルやアニメーションの読み込みと表示を行う
 - Open Asset Import Library" (assimp) <https://www.assimp.org/>
- マニュアル
 - <https://openframeworks.cc/documentation/ofxAssimpModelLoader/ofxAssimpModelLoader/>
 - 現時点で 3DS, ASE, DXF, HMP, MD2, MD3, MD5, MDC, MDL, NFF, PLY, STL, X, LWO, OBJ, SMD, Collada, Ogre XML, partly LWS の読み込みに対応



プロジェクトの作成成功



クリックして開く



Visual Studio は
まだ起動しない

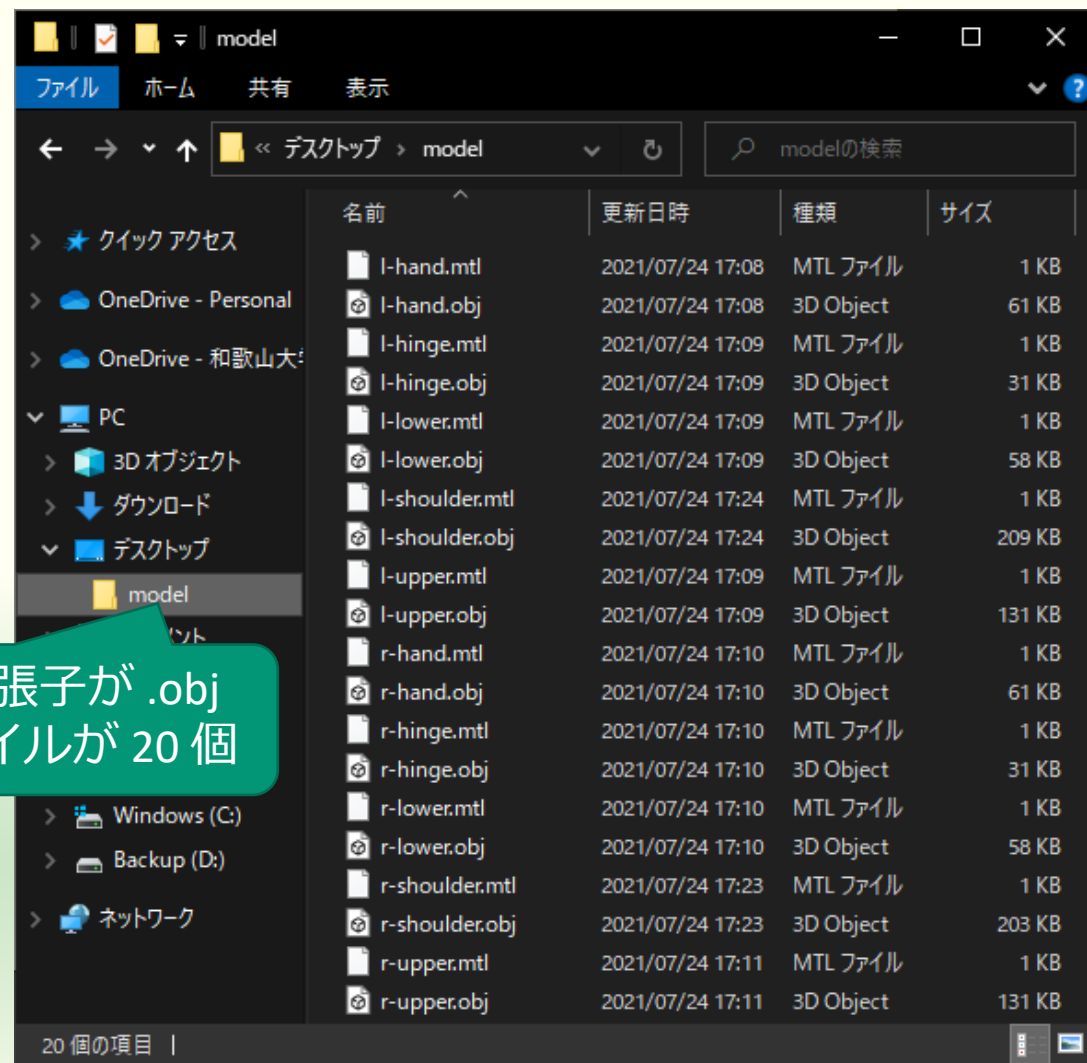
model.zip のダウンロードと展開

[第7回] ユーザインタフェース

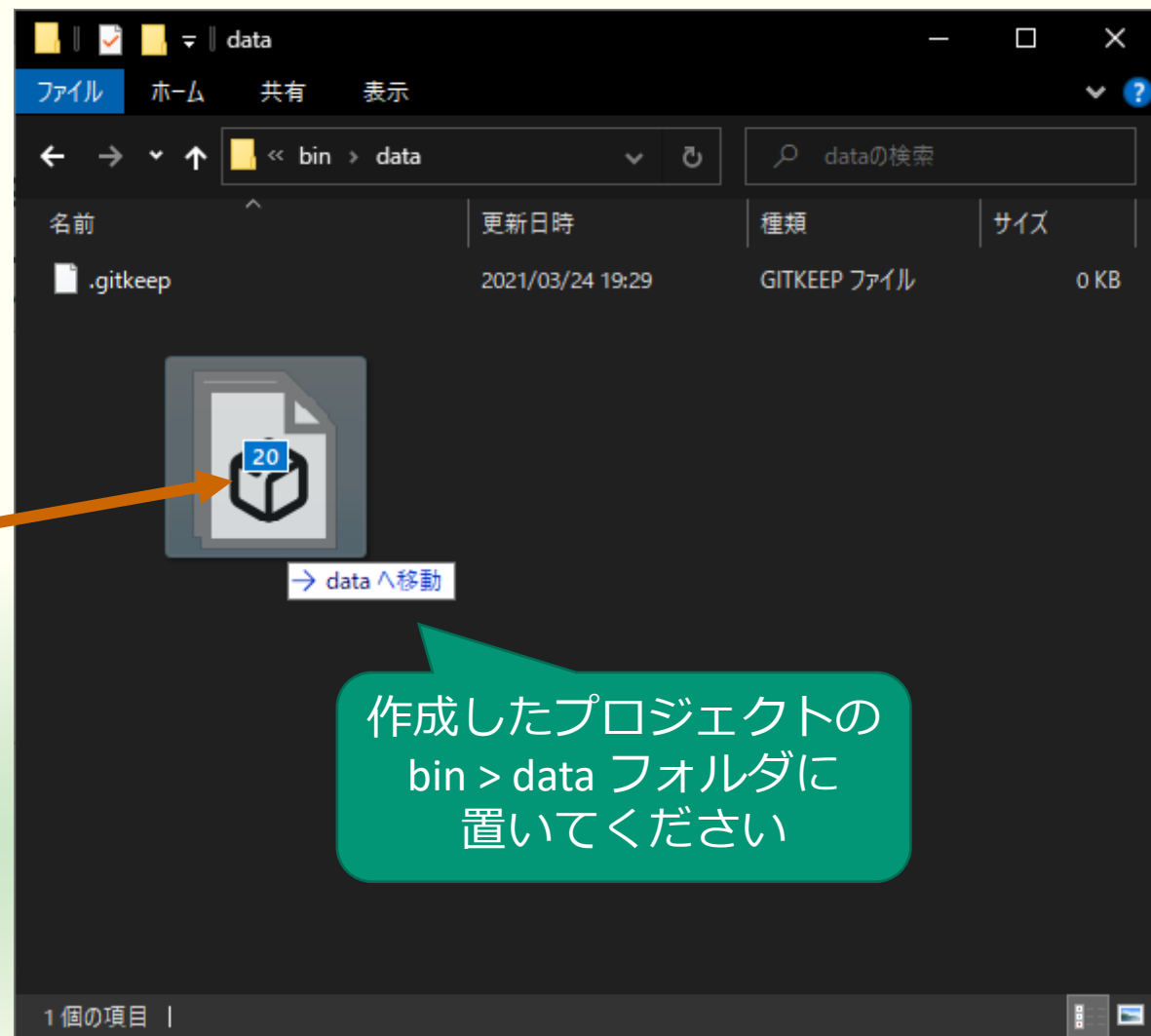
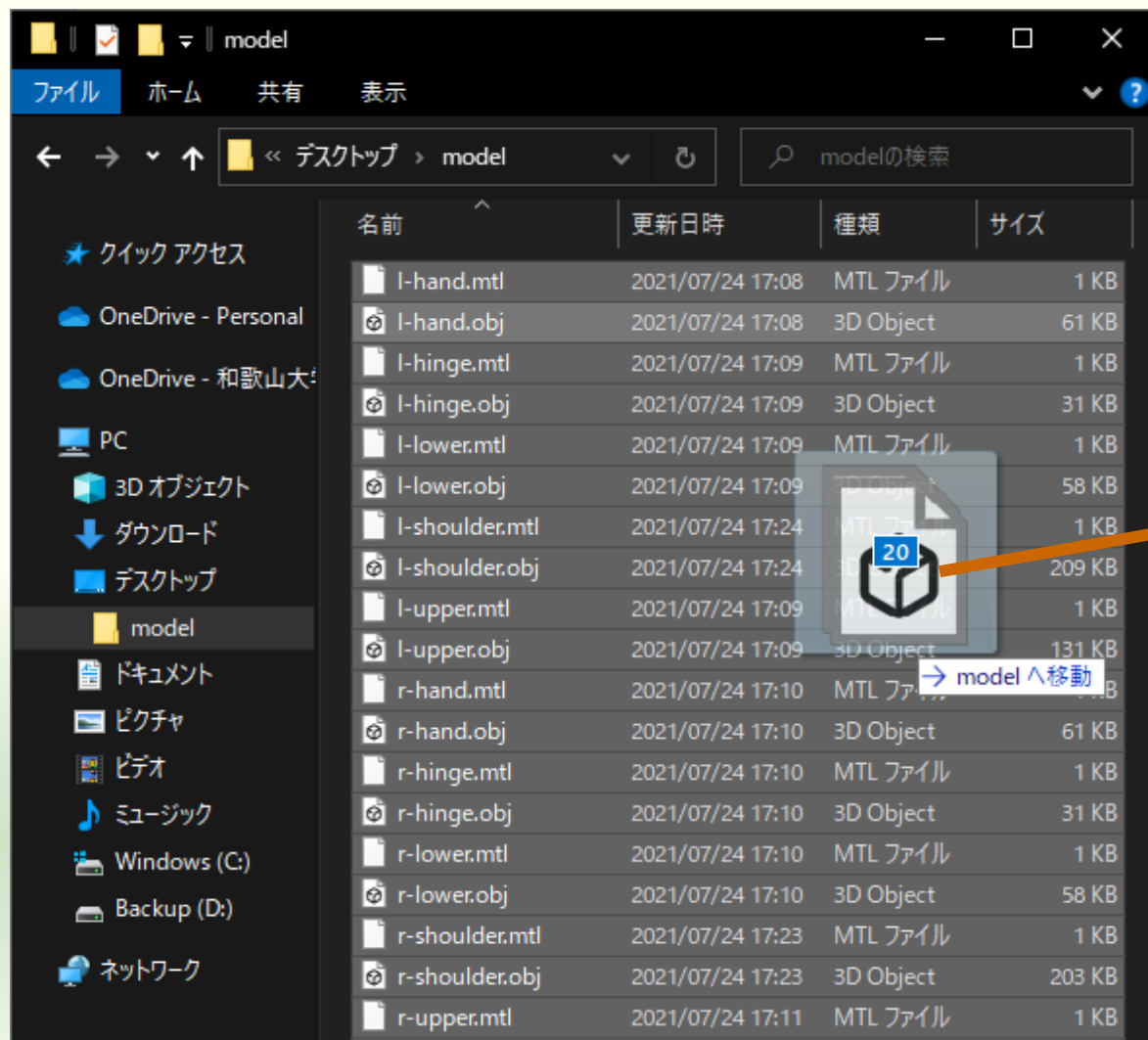
- 第7回資料
- model.zip
- ofxImGui-1.75.zip
- 第7回課題
- 何か一言

Moodle から [model.zip](#) をダウンロードしてください

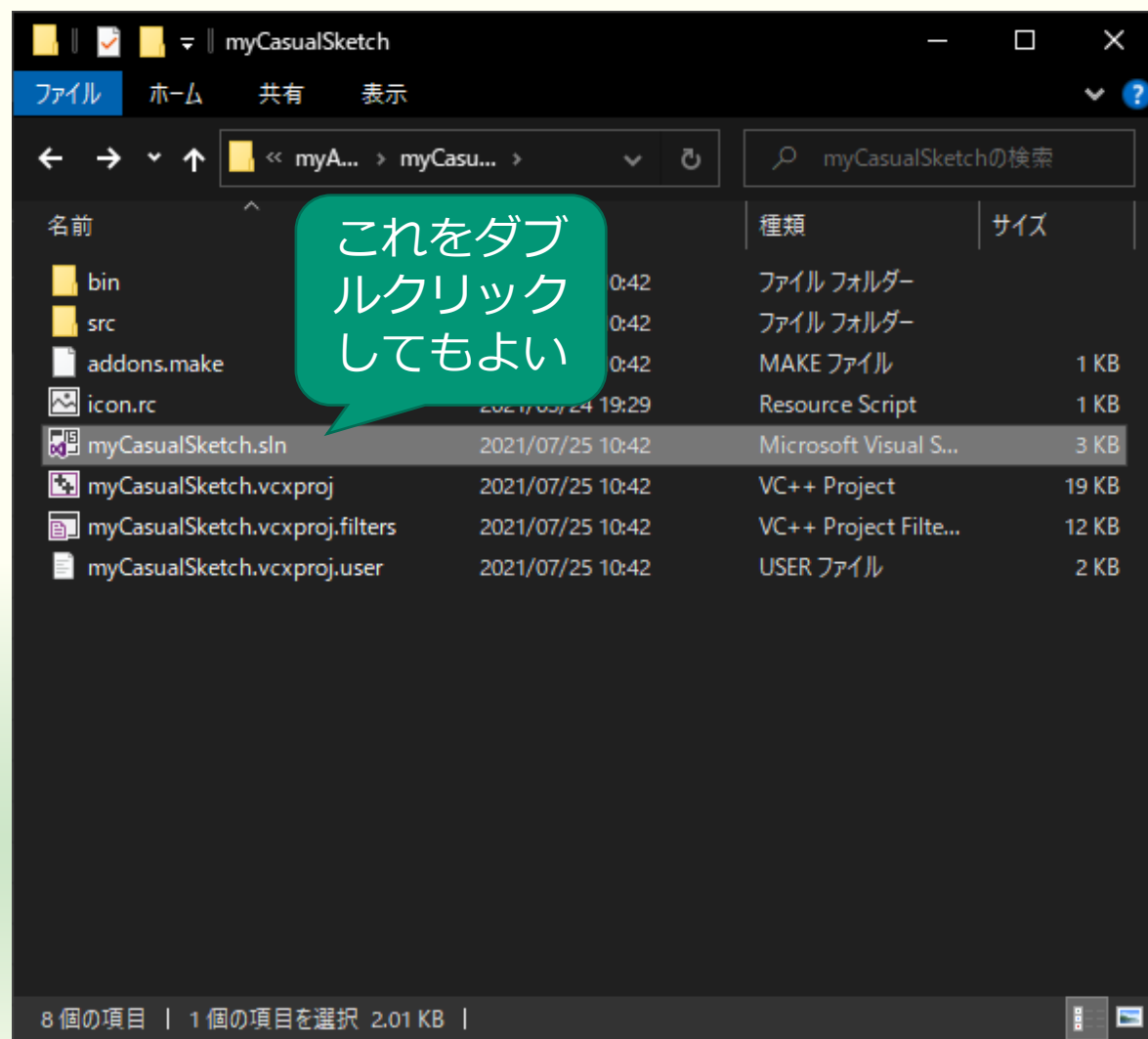
展開すると拡張子が .obj と .mtl のファイルが 20 個



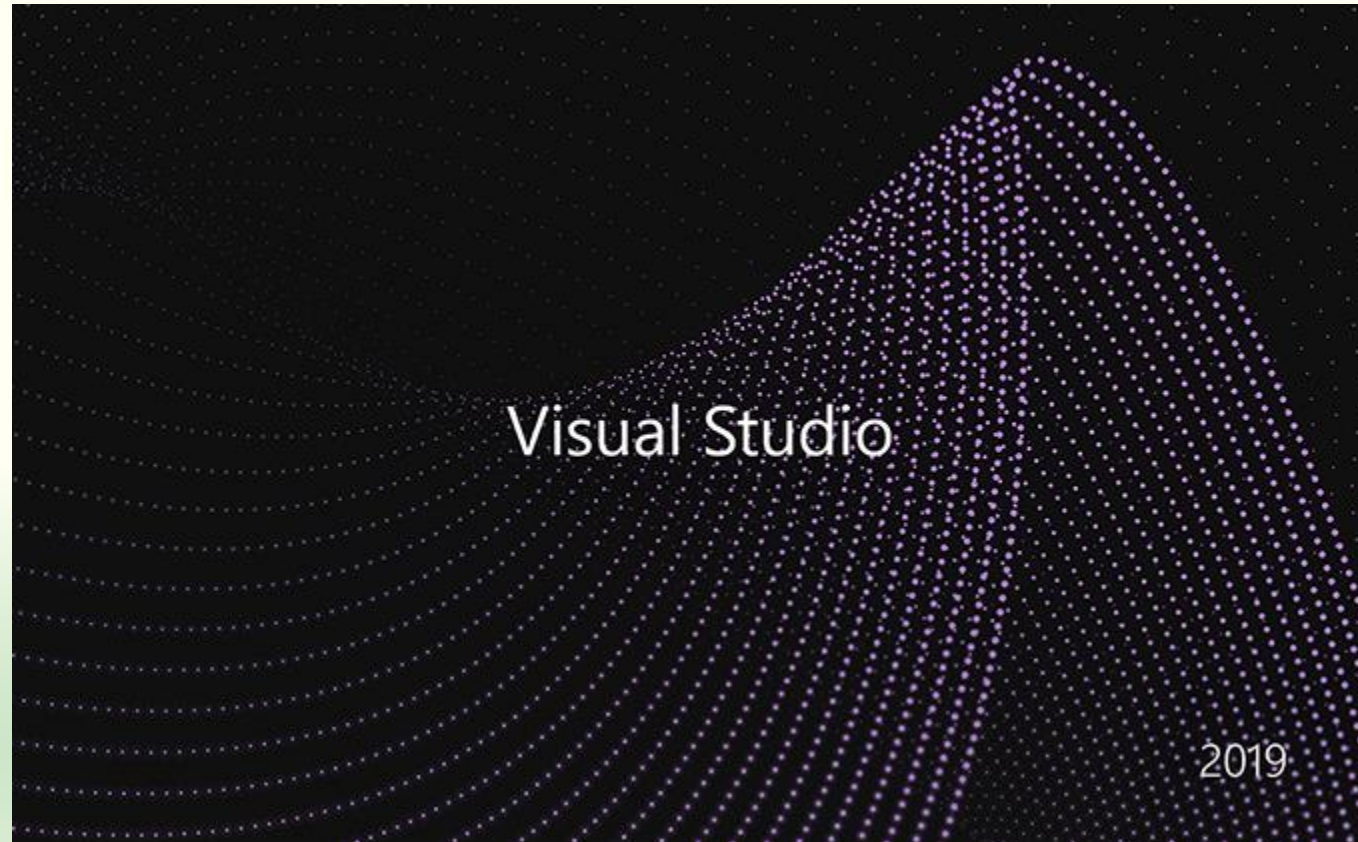
bin > data にファイルを配置



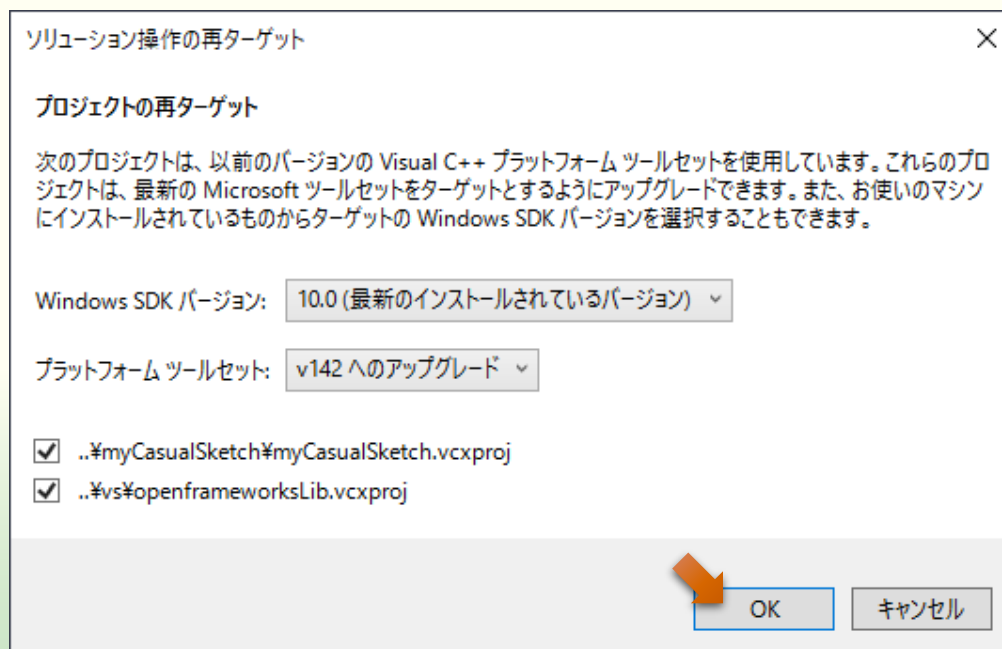
ソリューションファイルを開く



Visual Studio が起動する

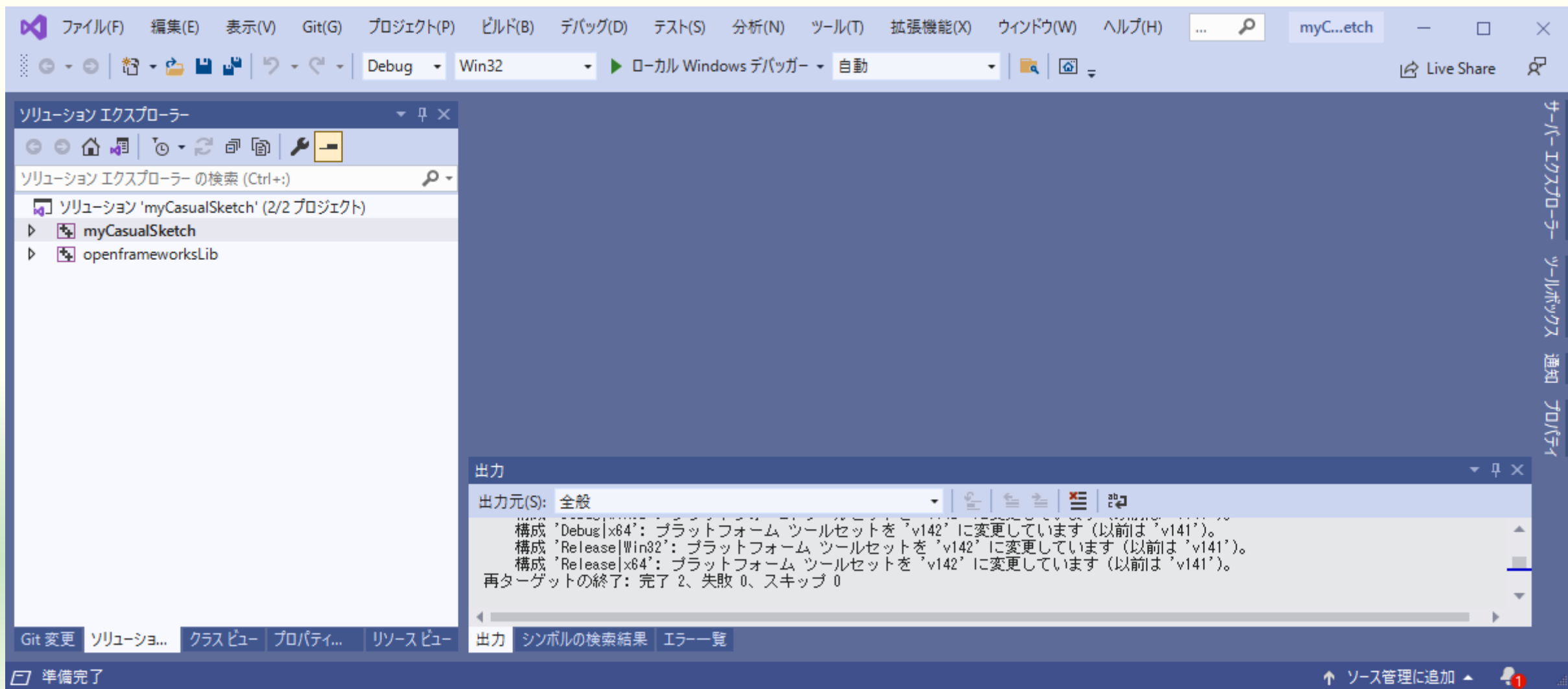


ソリューションの再ターゲット



Visual Studio は頻繁に更新しているので皆さんがお使いの Visual Studio SDK のバージョンと合わない場合がある

Visual Studio 起動

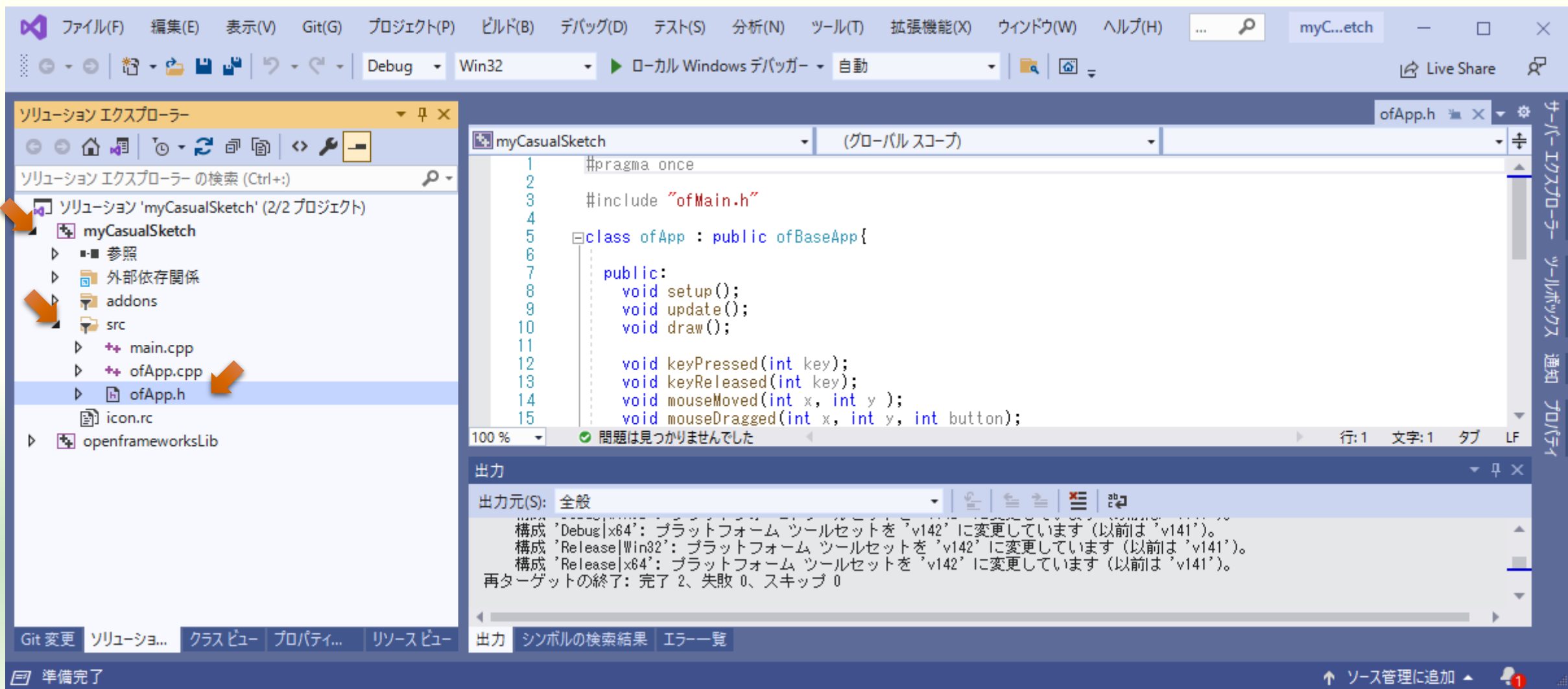




3D モデルの読み込みと表示

ofxAssimpModelLoader を使う

ofApp.h を開く



カメラ, ライト, モデルのメンバ変数を追加

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"

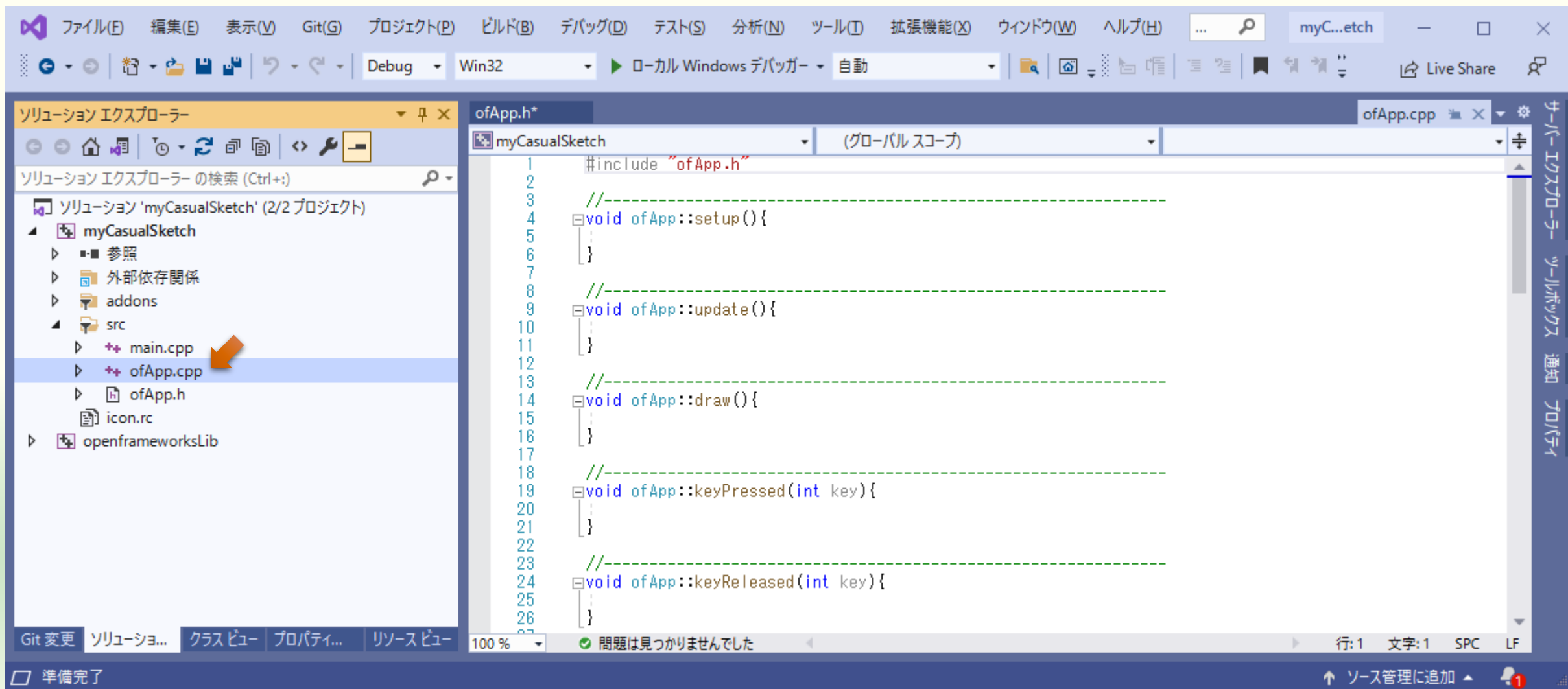
class ofApp : public ofBaseApp{
    ofEasyCam camera;
    ofLight light;
    ofxAssimpModelLoader model;

public:
    void setup();
    void update();
    void draw();
```

(以下略)

- 3D 表示に必要なメンバを追加する
 - ofCamera クラスのメンバ変数を追加する
 - ofEasyCam クラスはマウス操作が可能
 - ofLight クラスのメンバ変数を追加する
- 3D モデルの読み込みと表示を行う
 - ofxAssimpModelLoader クラスのメンバ変数を追加する

ofApp.cpp を開く



3D 表示の設定と 3D モデルファイルの読み込み

```
#include "ofApp.h"

//-----
void ofApp::setup(){
    ofEnableDepthTest();
    light.enable();
    light.setPosition(300.0f, 400.0f, 500.0f);
    model.loadModel("l-hand.obj");
}
```

(以下略)

- ofEnableDepthTest();
 - 隠面消去処理を有効にする
- light.enable();
 - light による陰影付けを有効にする
- model.loadModel("l-hand.obj");
 - プロジェクトのフォルダの bin の data の中の l-hand.obj という 3D モデルファイルを model に読み込む
 - “.obj” は Wavefront OBJ 形式の 3D モデルファイルで Blender 等で作成できる

カメラを有効にして 3D モデルを表示する

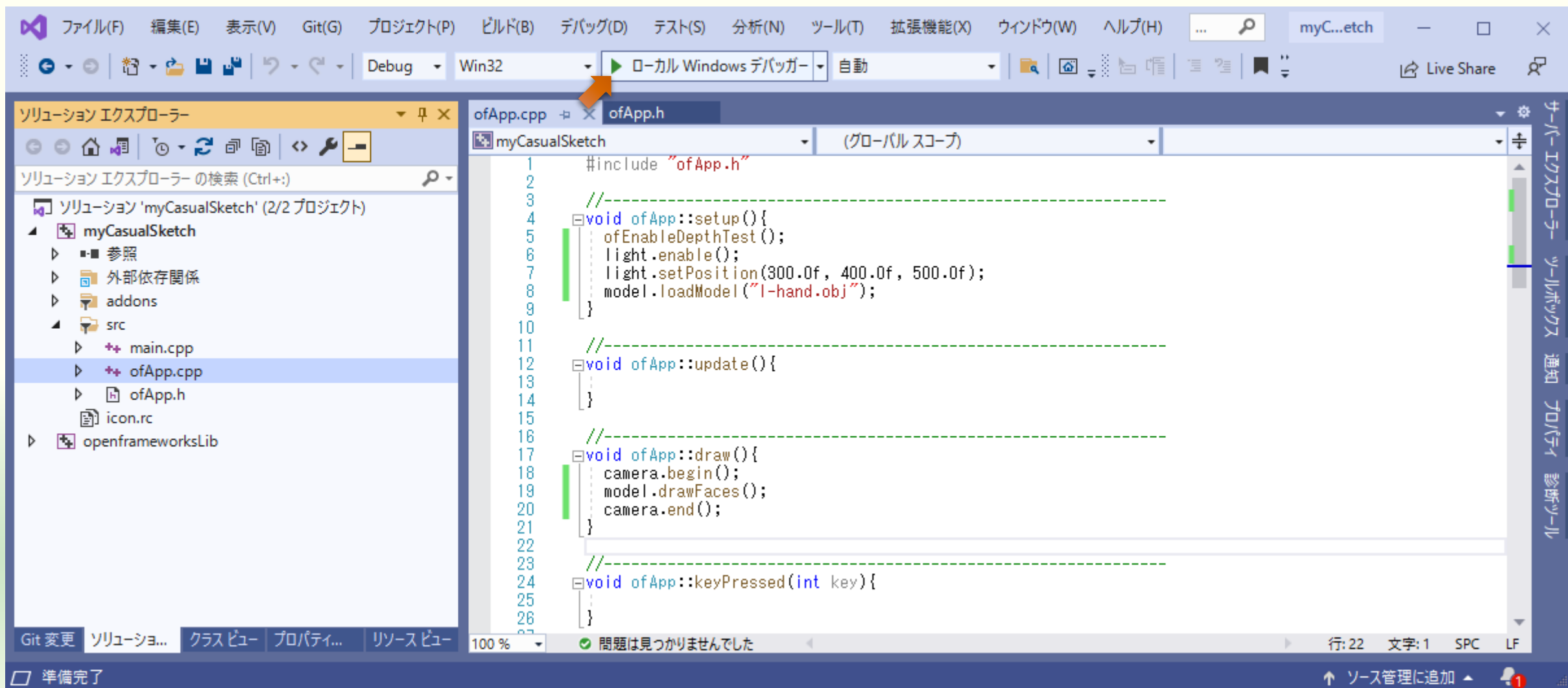
```
//-----  
void ofApp::draw(){  
    camera.begin();  
    model.drawFaces();  
    camera.end();  
}
```

(以下略)

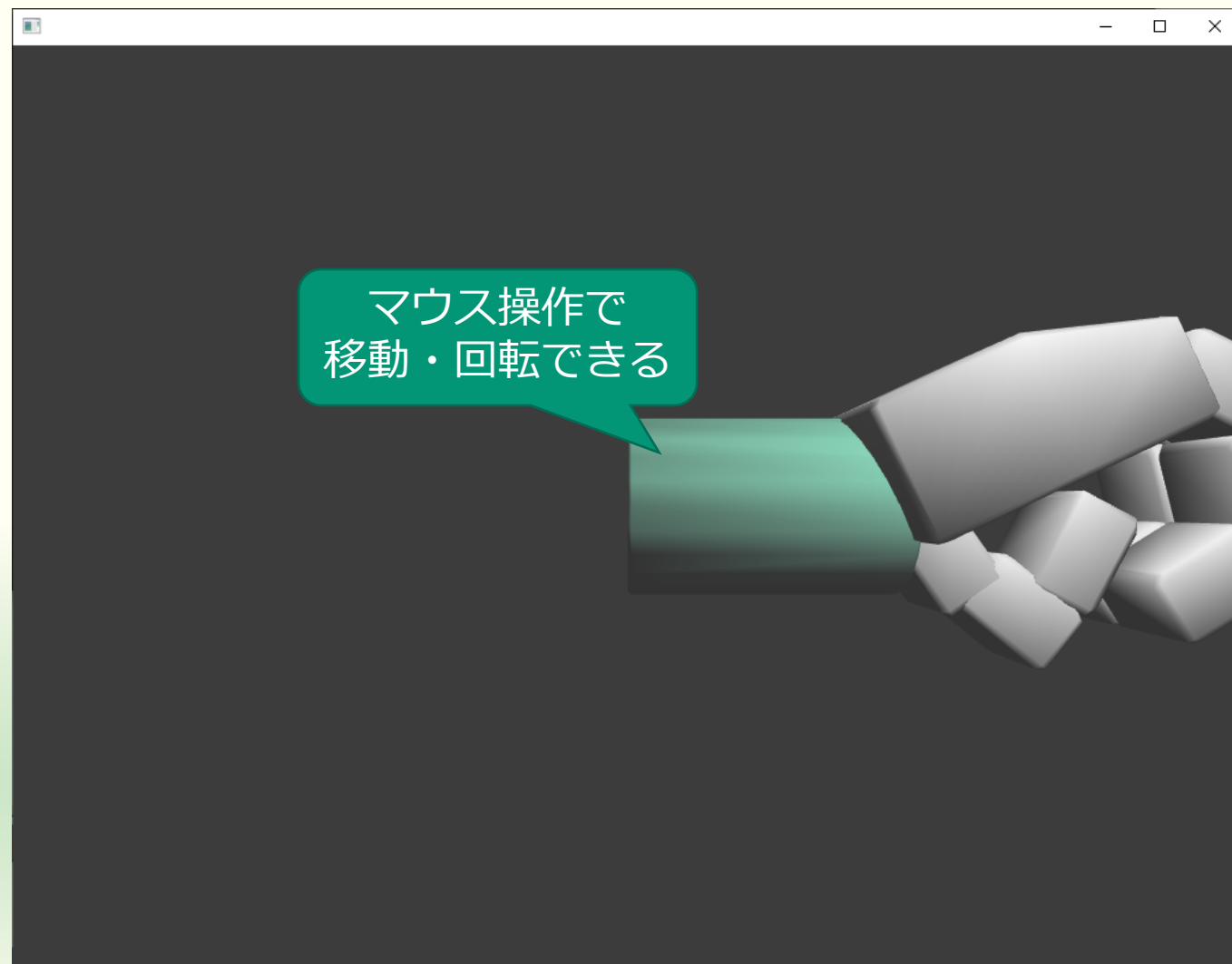
- camera.begin() と camera.end() の間は 3 次元空間の表示になる
- 第 3 回参照



ビルドと実行



実行結果



3D モデルの大きさを正規化しないようにする

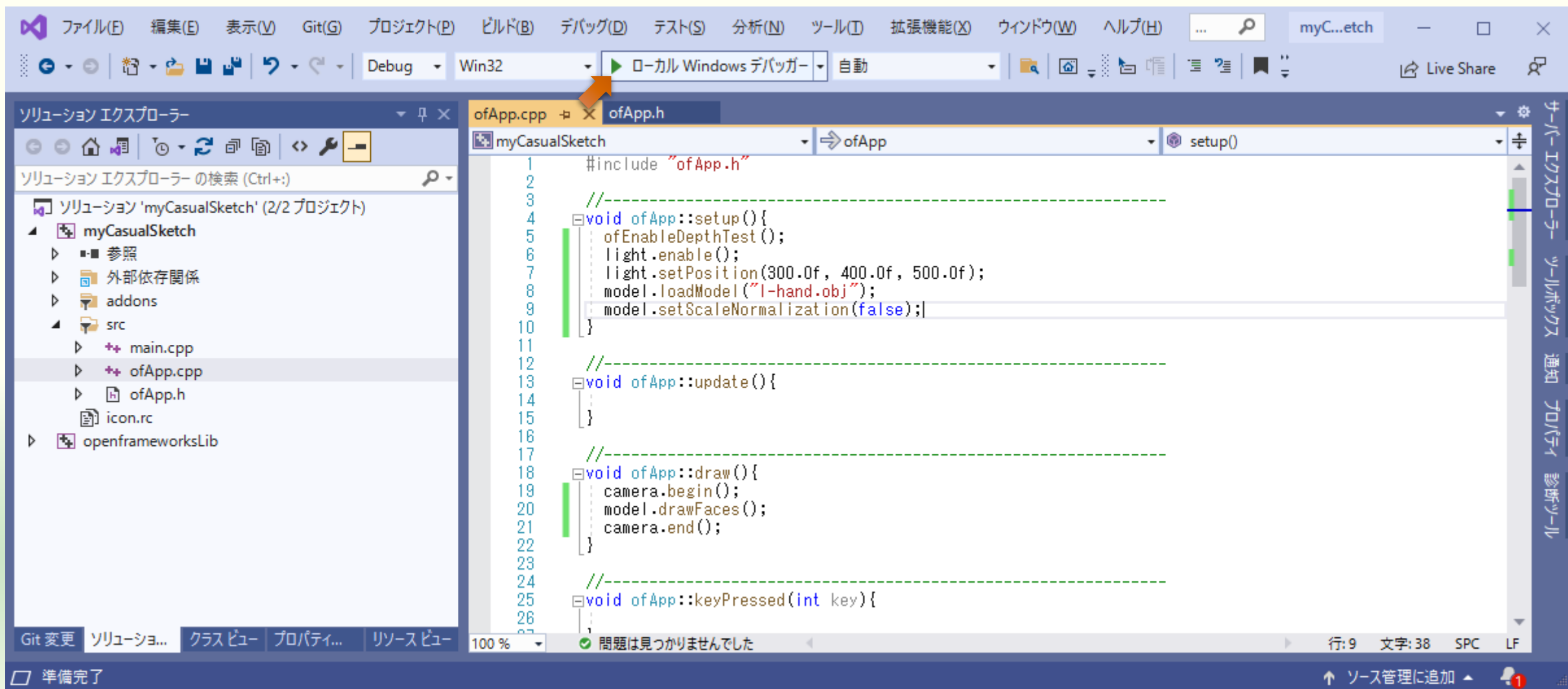
```
#include "ofApp.h"

//-----
void ofApp::setup(){
    ofEnableDepthTest();
    light.enable();
    light.setPosition(300.0f, 400.0f, 500.0f);
    model.loadModel("l-hand.obj");
    model.setScaleNormalization(false);
}
```

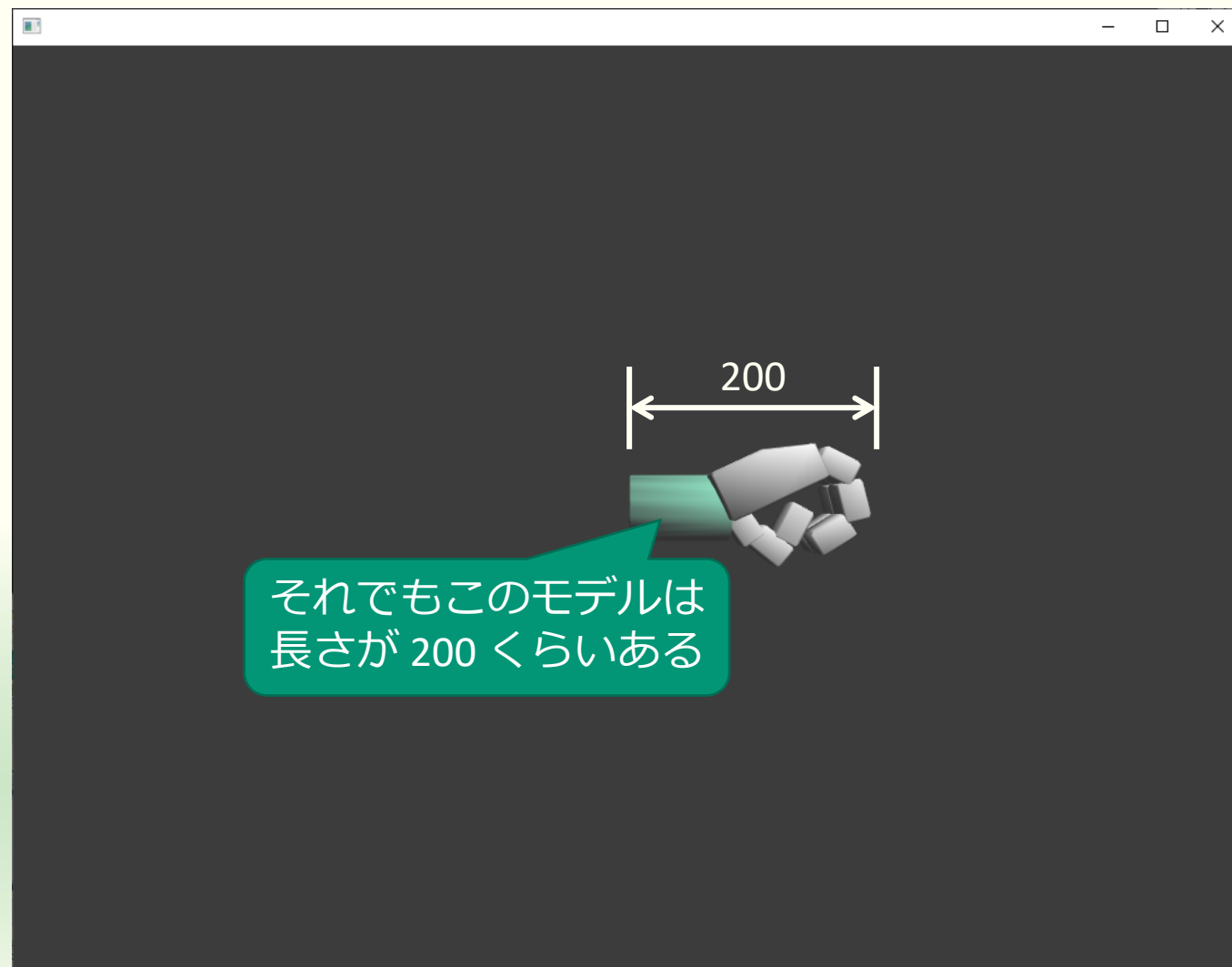
(以下略)

- `model.setScaleNormalization(false)`
 - 引数が `false` なら `model` に読み込んだ 3D モデルのサイズを**正規化しない**
 - openFrameworks の 3D モデルのサイズはウィンドウサイズを基準にしている(数100～数1000)
 - しかし一般的な 3D モデルのサイズはこれに準じていない
 - そこで openFrameworks はデフォルトで 3D モデルのサイズを正規化する
 - そうすると複数の 3D モデルを配置する際に位置の基準がサイズと一致しないため位置合わせが難しくなる

ビルドと実行



実行結果





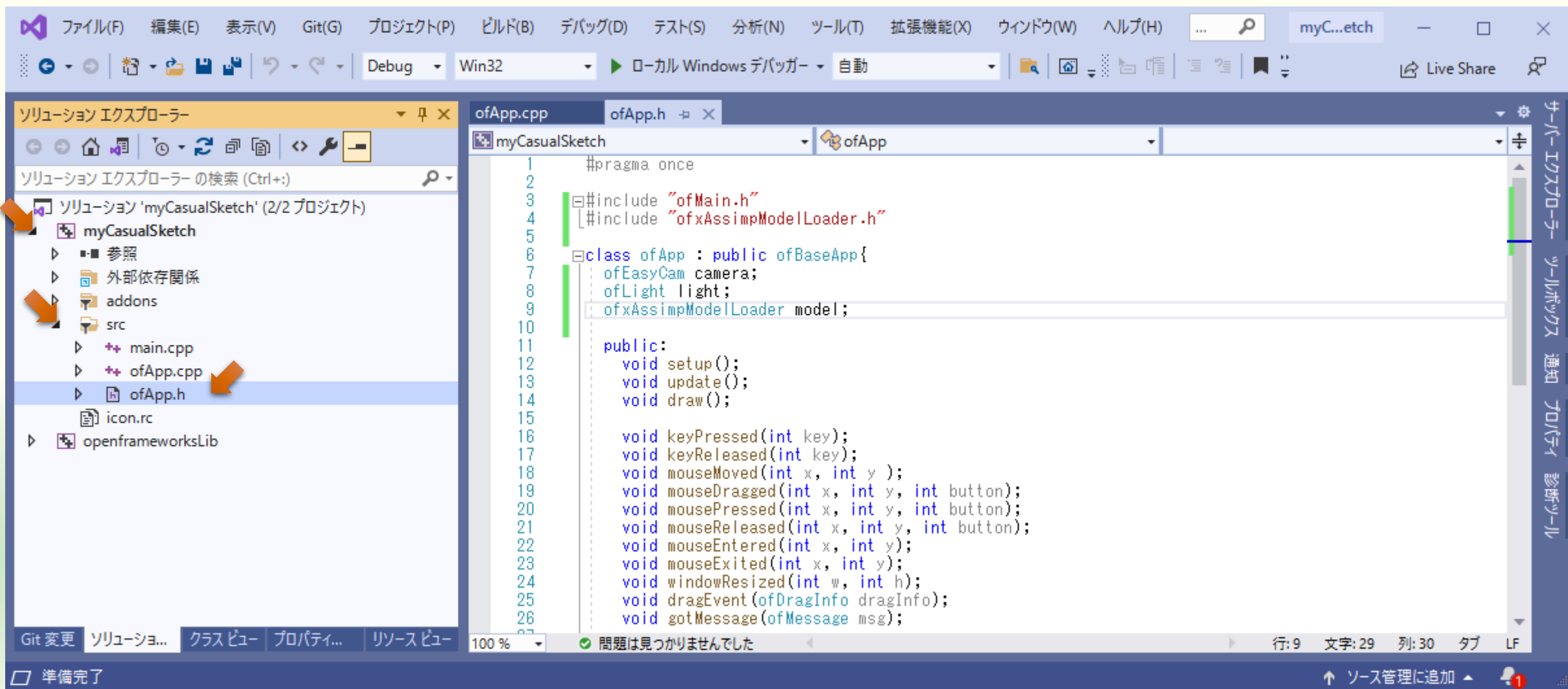
階層化できるクラスを作る

ofNode クラスから派生する

ofxAssimpModelLoader を階層化可能にする

- ofxAssimpModelLoader は ofNode の派生クラスではない
 - 階層化できない
 - setParent() メソッドを持たない
 - setPosition(), setScale(), setRotation() メソッドによる設定は対象の 3D モデルのみに対して行われる
- ofNode を継承したクラスを自分で定義する
 - of3dPrimitive は ofNode の派生クラス
 - ofBoxPrimitive, ofSpherePrimitive などは of3dPrimitive の派生クラス
 - ofNode クラスから setParent() メソッドが**継承**される
 - 第 3 回参照

ofApp.h を開く



ofNode を継承して Model クラスを定義する

```
#pragma once
```

```
#include "ofMain.h"
```

```
#include "ofxAssimpModelLoader.h"
```

```
class Model : public ofNode{
```

ofNode を継承する

```
public:
```

```
    ofxAssimpModelLoader model;  
};
```

```
class ofApp : public ofBaseApp{  
    ofEasyCam camera;  
    ofLight light;  
    ofxAssimpModelLoader model;  
    Model hand;  
public:
```

削除

```
public:
```

(以下略)

■ class Model : public ofNode

- ofNode クラスを public 継承して派生クラスの Model を定義する
 - 継承すると基底クラスである ofNode の機能（メソッド等）が使える

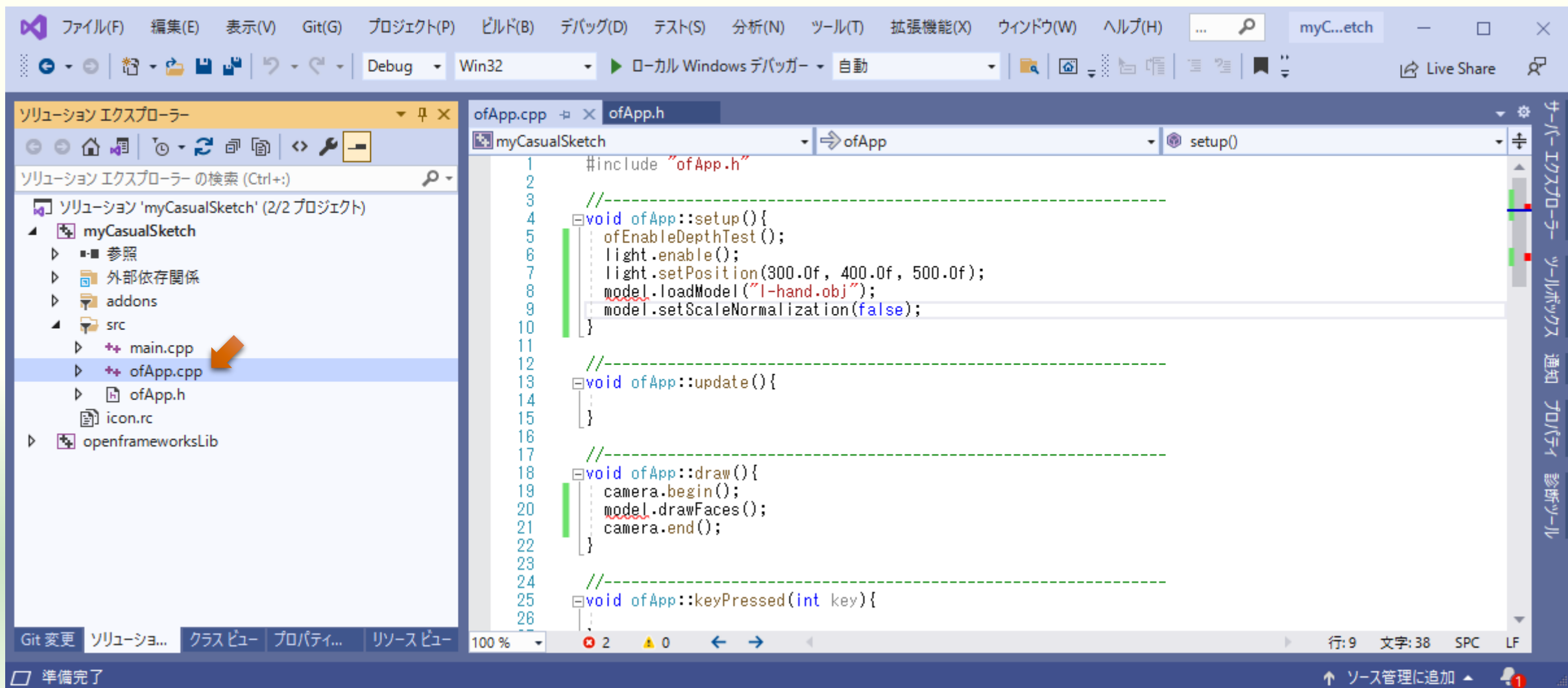
■ ofxAssimpModelLoader model;

- ofxAssimpModelLoader 型の変数 model を派生クラス Model のメンバにする

■ Model hand;

- model の代わりに自分で定義した Model 型の変数 hand を ofApp クラスのメンバにする

ofApp.cpp を開く



ofNode から派生した Model クラスを使う

```
#include "ofApp.h"

//-----
void ofApp::setup(){
    ofEnableDepthTest();
    light.enable();
    light.setPosition(300.0f, 400.0f, 500.0f);
    hand.model.loadModel("l-hand.obj");
    hand.model.setScaleNormalization(false);
}

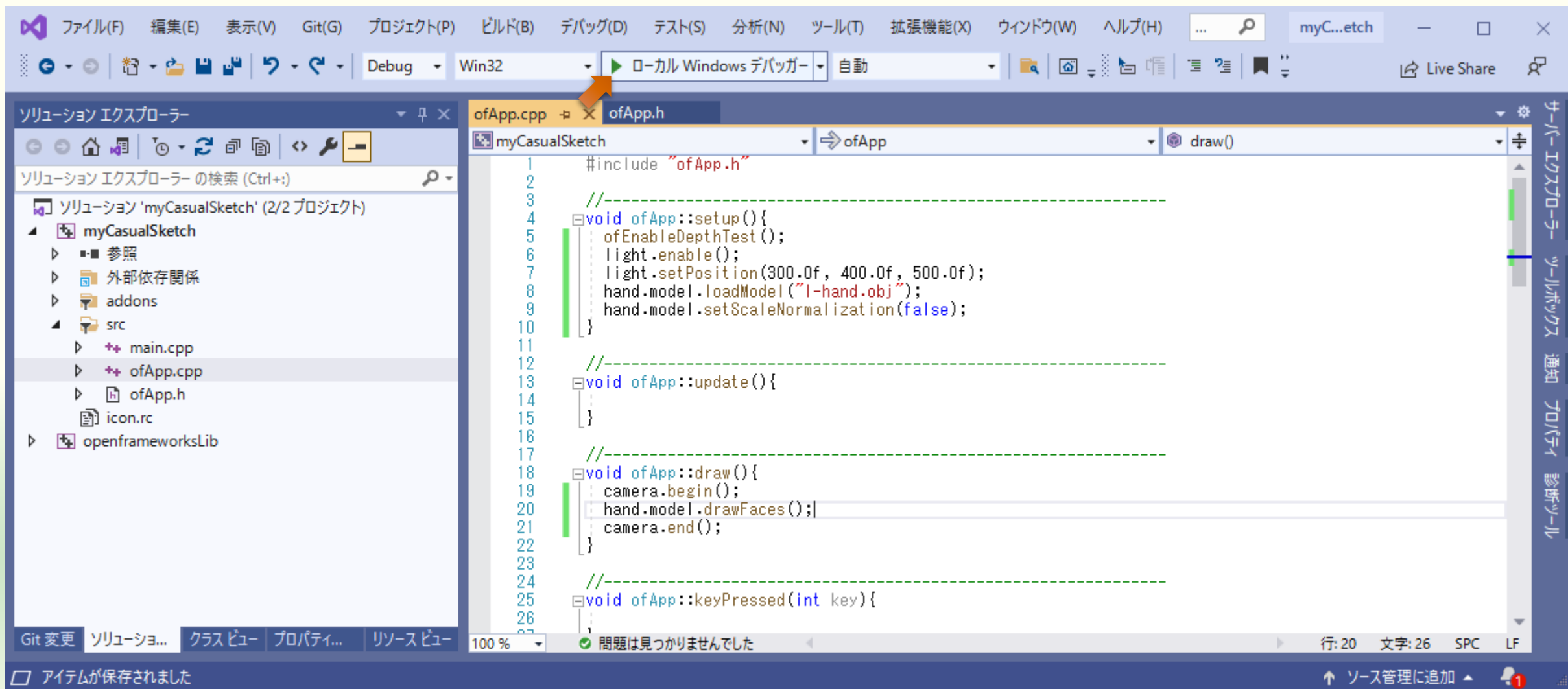
(途中略)

//-----
void ofApp::draw(){
    camera.begin();
    hand.model.drawFaces();
    camera.end();
}
```

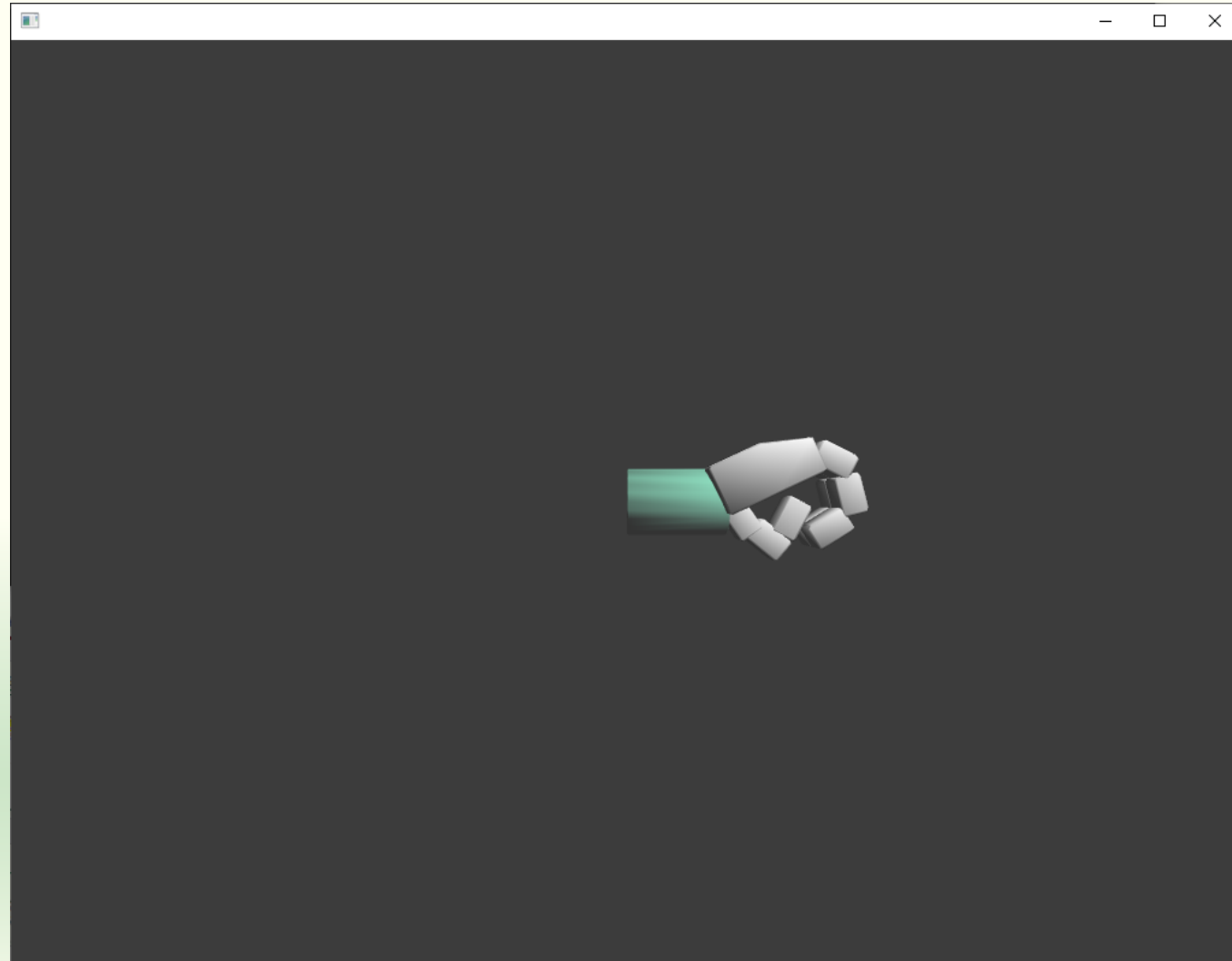
- hand は Model クラスのインスタンス（オブジェクト）
- model は Model クラスのメンバで ofAssimpModelLoader 型
 - hand.model.loadModel("l-hand.obj");
 - hand.model.setScaleNormalization(false);
 - hand.model.drawFaces();



ビルドと実行



実行結果は変わらない



Model クラスにインスタンス lower を追加する

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"

class Model : public ofNode{
public:
    ofxAssimpModelLoader model;
};

class ofApp : public ofBaseApp{
    ofEasyCam camera;
    ofLight light;
    Model hand, lower;

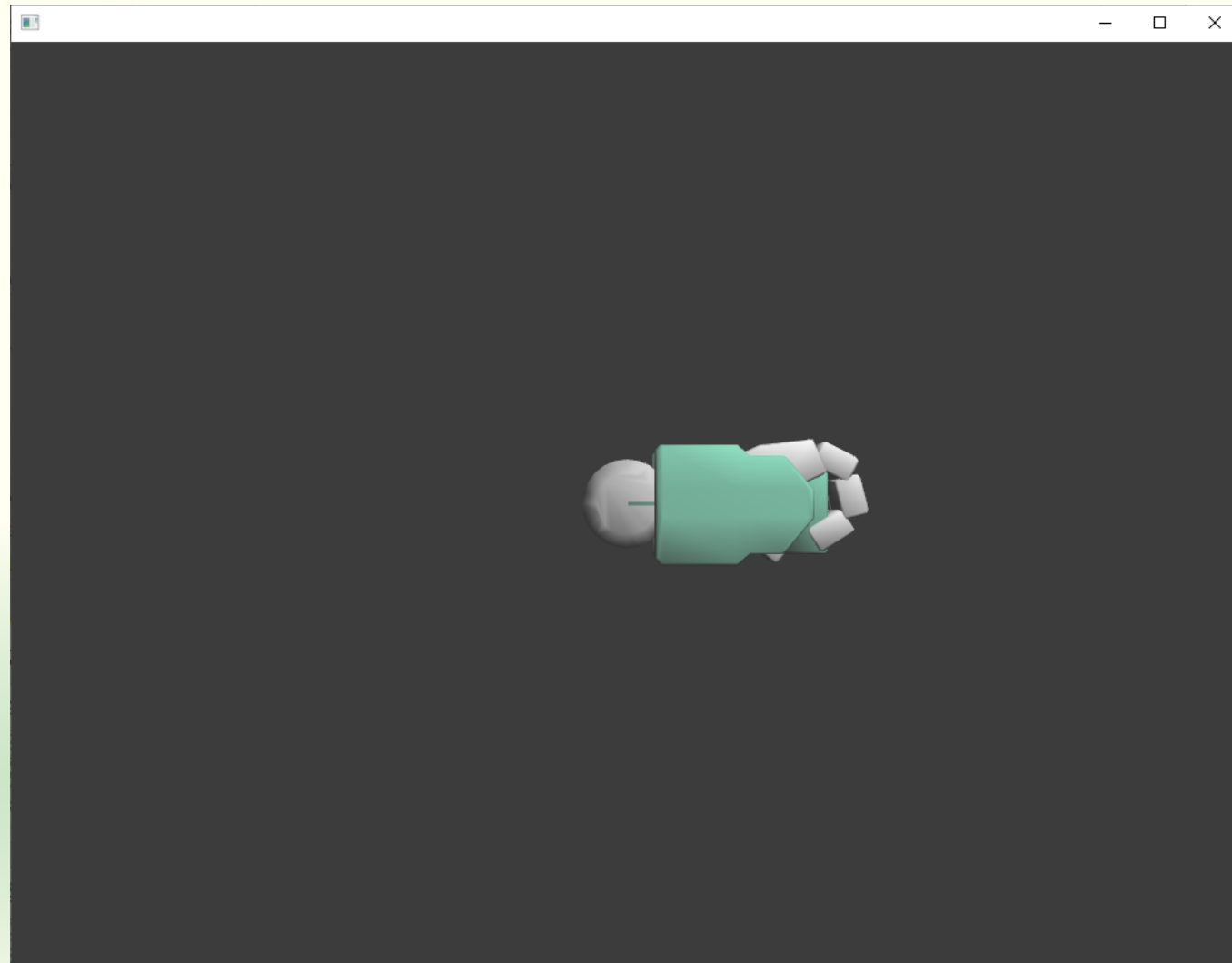
public:
    (以下略)
```

```
//-----
void ofApp::setup(){
    ofEnableDepthTest();
    light.enable();
    light.setPosition(300.0f, 400.0f, 500.0f);
    lower.model.loadModel("l-lower.obj");
    lower.model.setScaleNormalization(false);
    hand.model.loadModel("l-hand.obj");
    hand.model.setScaleNormalization(false);
}

(途中略)

//-----
void ofApp::draw(){
    camera.begin();
    lower.model.drawFaces();
    hand.model.drawFaces();
    camera.end();
}
```

ビルドして実行すると2つのモデルが重なる



hand の位置を設定する

```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    lower.model.loadModel("l-lower.obj");  
    lower.model.setScaleNormalization(false);  
    hand.model.loadModel("l-hand.obj");  
    hand.model.setScaleNormalization(false);  
    hand.setPosition(150.0f, 0.0f, 0.0f);  
}
```

(以下略)

- ofNode を継承したクラス Model のインスタンス hand に位置を設定する
 - hand に設定した位置は親からの相対位置になる
 - 現時点ではまだ hand に親子関係（階層構造）は設定していない
 - hand.model.setPosition(...) のように model に直接位置を設定できるが、model は ofAssimpModelLoader クラスのインスタンスなので、階層構造を持つことができない

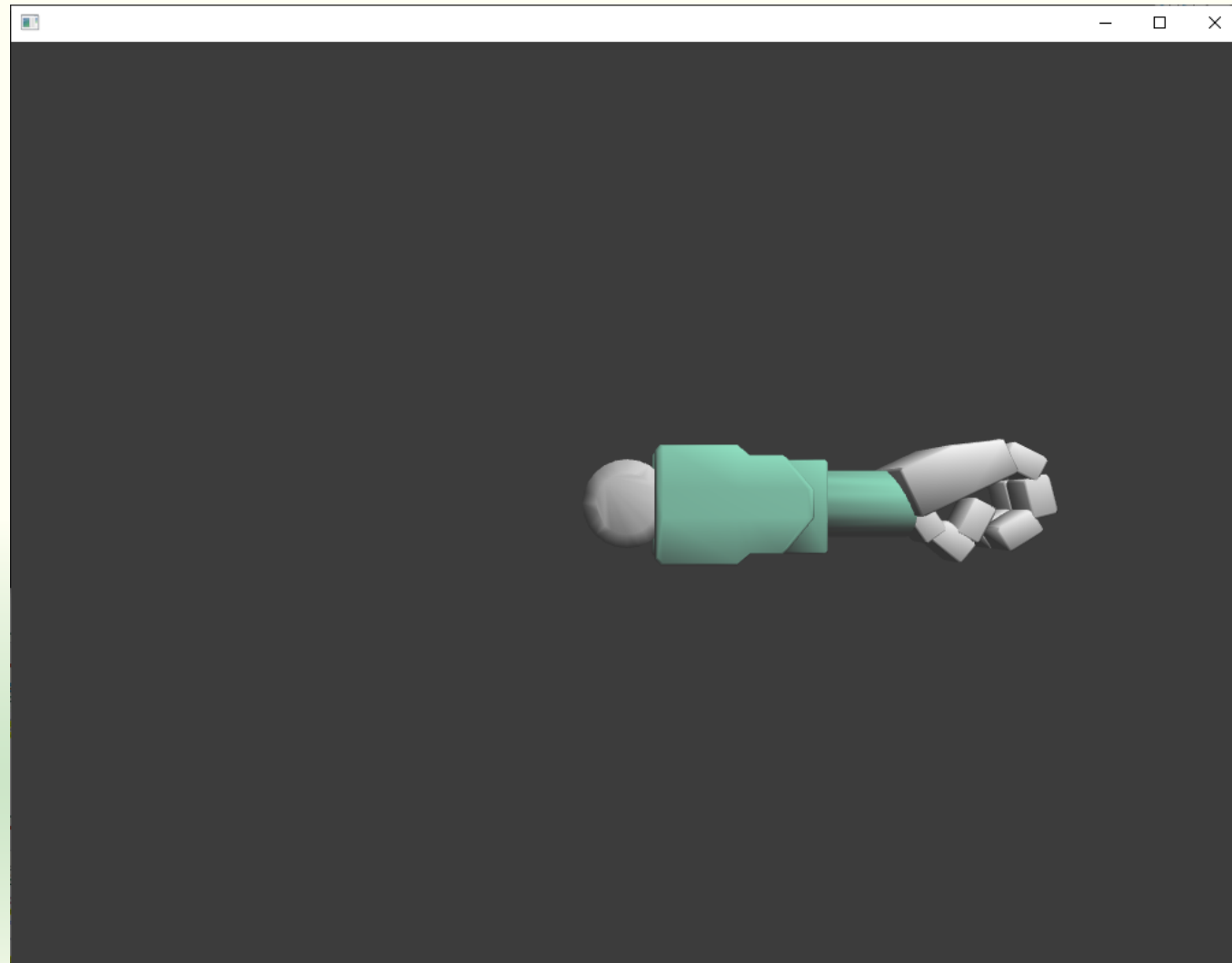
hand に設定した座標変換を使って表示する

```
//-----  
void ofApp::draw(){  
    camera.begin();  
    lower.model.drawFaces();  
    hand.transformGL();  
    hand.model.drawFaces();  
    hand.restoreTransformGL();  
    camera.end();  
}
```

(以下略)

- transformGL() メソッド
 - 現在の座標変換を保存した後、それに ofNode クラスの階層構造にもとづいた座標変換を合成する
- restoreTransformGL() メソッド
 - transformGL() メソッドで保存した変換行列を復帰する
- この2つの間で図形の表示処理を行えば ofNode クラスに設定した階層構造にもとづいて座標変換を行って図形が表示される

ビルドして実行すると hand が移動している



lower の位置を設定して図形を表示する

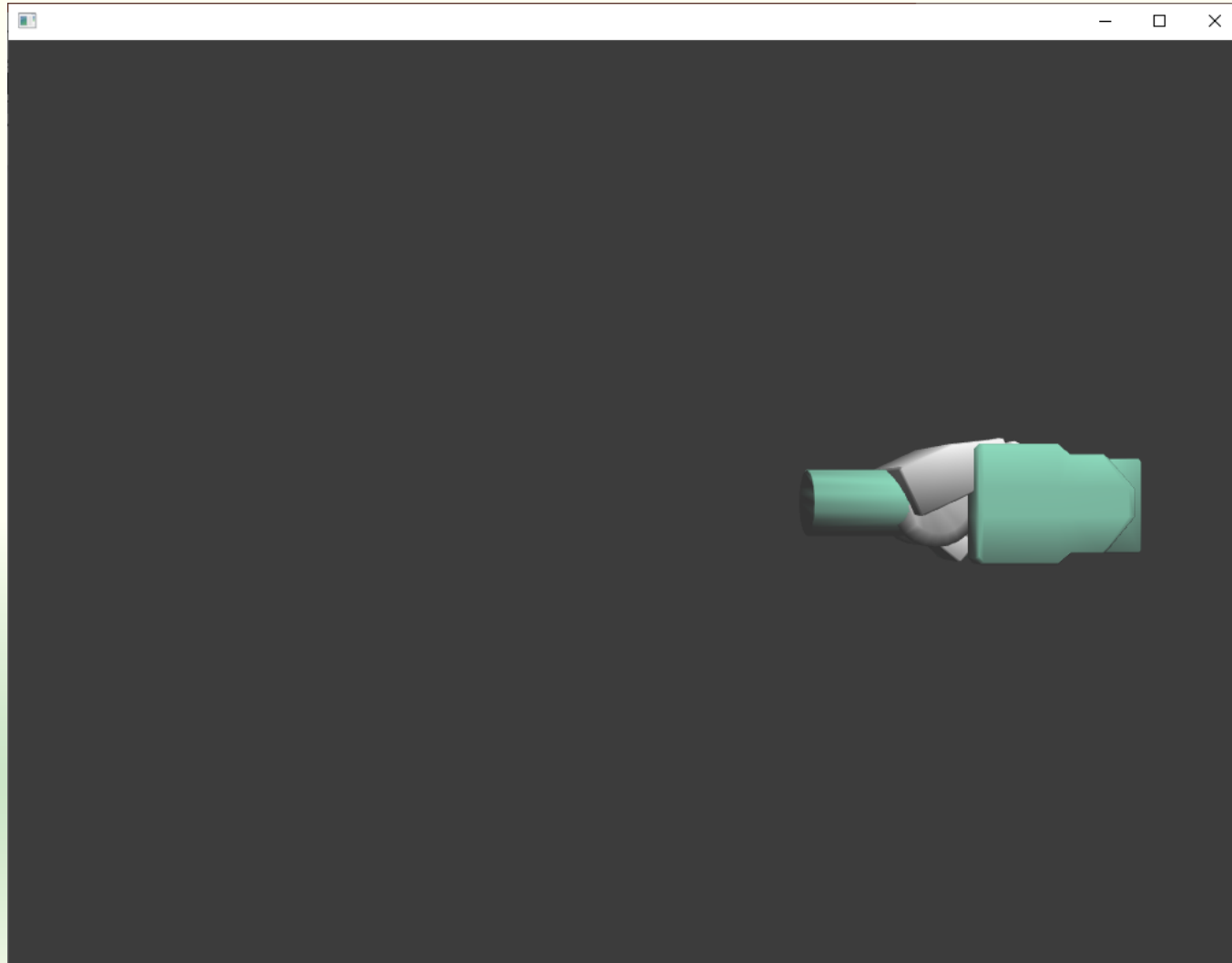
```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    lower.model.loadModel("l-lower.obj");  
    lower.model.setScaleNormalization(false);  
    lower.setPosition(250.0f, 0.0f, 0.0f);  
    hand.model.loadModel("l-hand.obj");  
    hand.model.setScaleNormalization(false);  
    hand.setPosition(150.0f, 0.0f, 0.0f);  
}
```

(以下略)

```
//-----  
void ofApp::draw(){  
    camera.begin();  
    lower.transformGL();  
    lower.model.drawFaces();  
    lower.restoreTransformGL();  
    hand.transformGL();  
    hand.model.drawFaces();  
    hand.restoreTransformGL();  
    camera.end();  
}
```

(以下略)

ビルドして実行すると lower だけ移動している



hand の親を lower にする

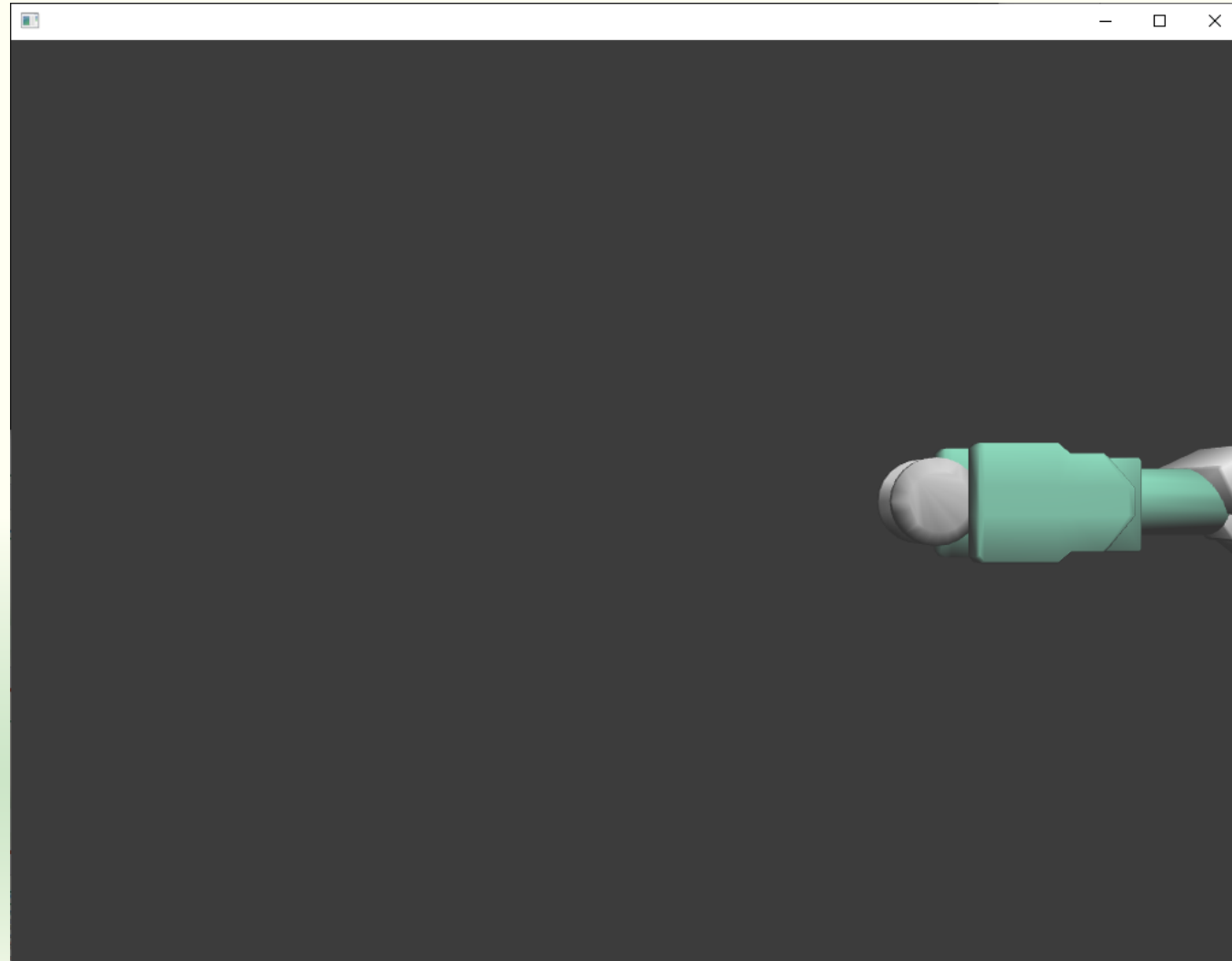
```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    lower.model.loadModel("l-lower.obj");  
    lower.model.setScaleNormalization(false);  
    lower.setPosition(250.0f, 0.0f, 0.0f);  
    hand.model.loadModel("l-hand.obj");  
    hand.model.setScaleNormalization(false);  
    hand.setPosition(150.0f, 0.0f, 0.0f);  
    hand.setParent(lower);  
}
```

(以下略)

- hand.setParent(lower);
 - hand の親を lower に設定する
 - setParent() は ofNode から継承されたメソッド



hand が lower の先に移動している





メソッドを追加する

インスタンスに共通する処理はメソッド化する

Model クラスに図形表示のメソッドを追加する

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"

class Model : public ofNode{

public:
    ofxAssimpModelLoader model;

    void drawFaces(){
        transformGL();
        model.drawFaces();
        restoreTransformGL();
    }
};
```

drawFaces() という
メソッド名に特別
な意味はないので
他の名前でもよい

- この例では drawFaces() という名前で定義している
 - drawFaces() 内にある transformGL() や restoreTransformGL() は適用するオブジェクトを指定していない
 - これらはこの drawFaces() を適用するオブジェクトに対して適用される
 - transformGL() や restoreTransformGL() は ofNode から継承された Model クラスのメソッド
 - model も drawFaces() を適用するオブジェクトのメンバが対象になる
 - model は Model クラスのメンバ
 - model に適用している drawFaces() は ofAssimpModelLoader クラスのメソッド

定義した drawFaces() メソッドを使う

```
//-----  
void ofApp::draw(){  
    camera.begin();  
    lower.transformGL();  
    lower.model.drawFaces();  
    lower.restoreTransformGL();  
    lower.drawFaces();  
    hand.transformGL();  
    hand.model.drawFaces();  
    hand.restoreTransformGL();  
    hand.drawFaces();  
    camera.end();  
}
```

削除

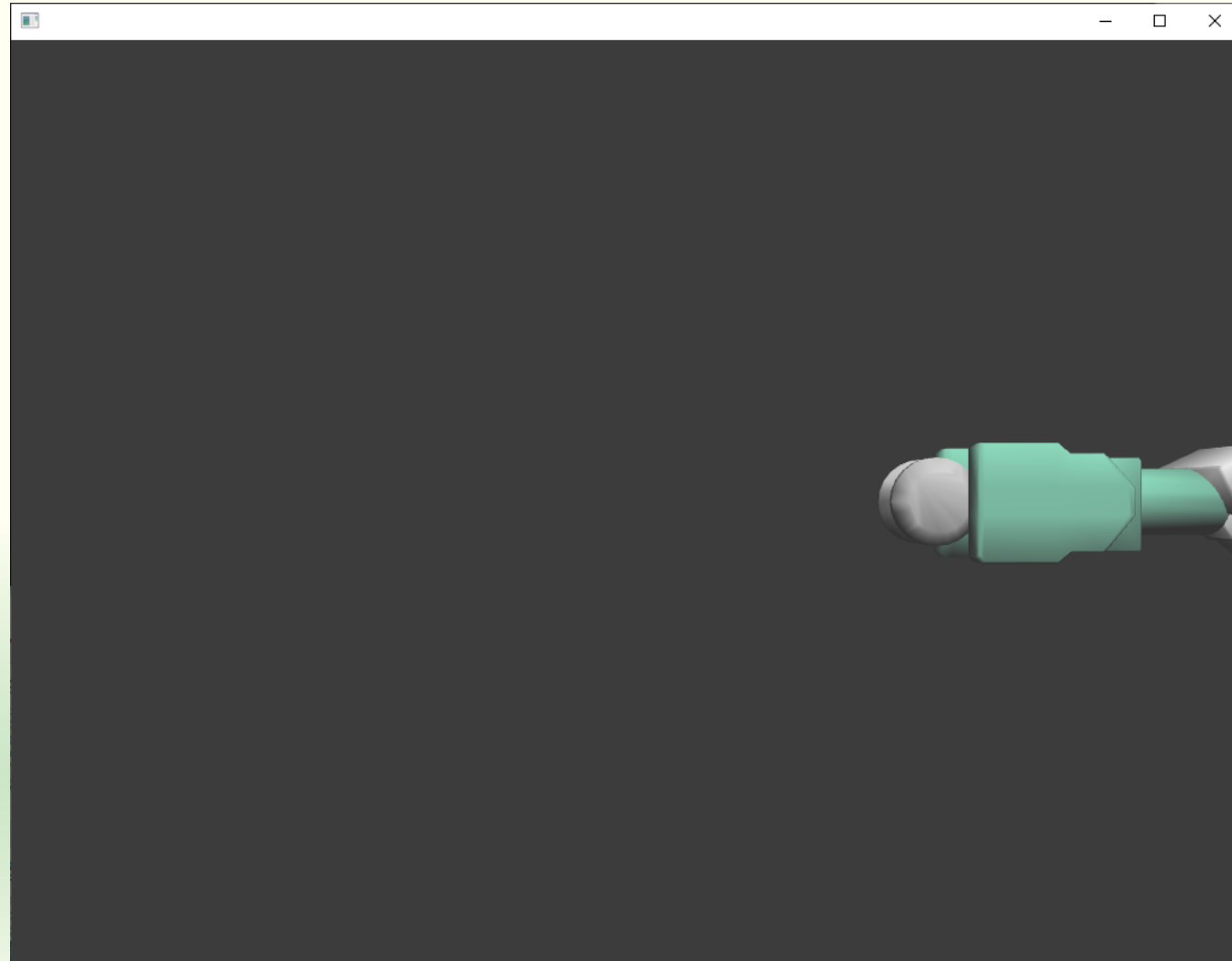
削除

(以下略)

- 定義した drawFaces() メソッドを使う
 - transformGL(), restoreTransformGL() および model.drawFaces() メソッドは draw() メソッドから呼び出している



ビルドして実行すると結果は変わらない





コンストラクタを定義する

オブジェクト（インスタンス）の生成時に実行する処理

Model の vector である parts をメンバにする

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"

class Model : public ofNode{

public:
    ofxAssimpModelLoader model;
};

class ofApp : public ofBaseApp{
    ofEasyCam camera;
    ofLight light;
    Model hand, lower;
    vector<Model> parts;

public:

    (以下略)
```

- parts は Model クラスの vector



parts にオブジェクトを追加する

```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    parts.emplace_back();  
    parts.back().model.loadModel("l-lower.obj");  
    parts.back().model.setScaleNormalization(false);  
    parts.back().setPosition(250.0f, 0.0f, 0.0f);  
    parts.emplace_back();  
    parts.back().model.loadModel("l-hand.obj");  
    parts.back().model.setScaleNormalization(false);  
    parts.back().setPosition(150.0f, 0.0f, 0.0f);  
    parts.back().setParent(parts[0]);  
}
```

(以下略)

- parts.emplace_back();
 - parts の最後に空のオブジェクトを追加する
- parts.back()
 - parts の最後のオブジェクト
- parts[0]
 - parts の最初のオブジェクト
 - parts.front() と同じ
- 第 3 回参照



vector に格納されている全ての図形を表示する

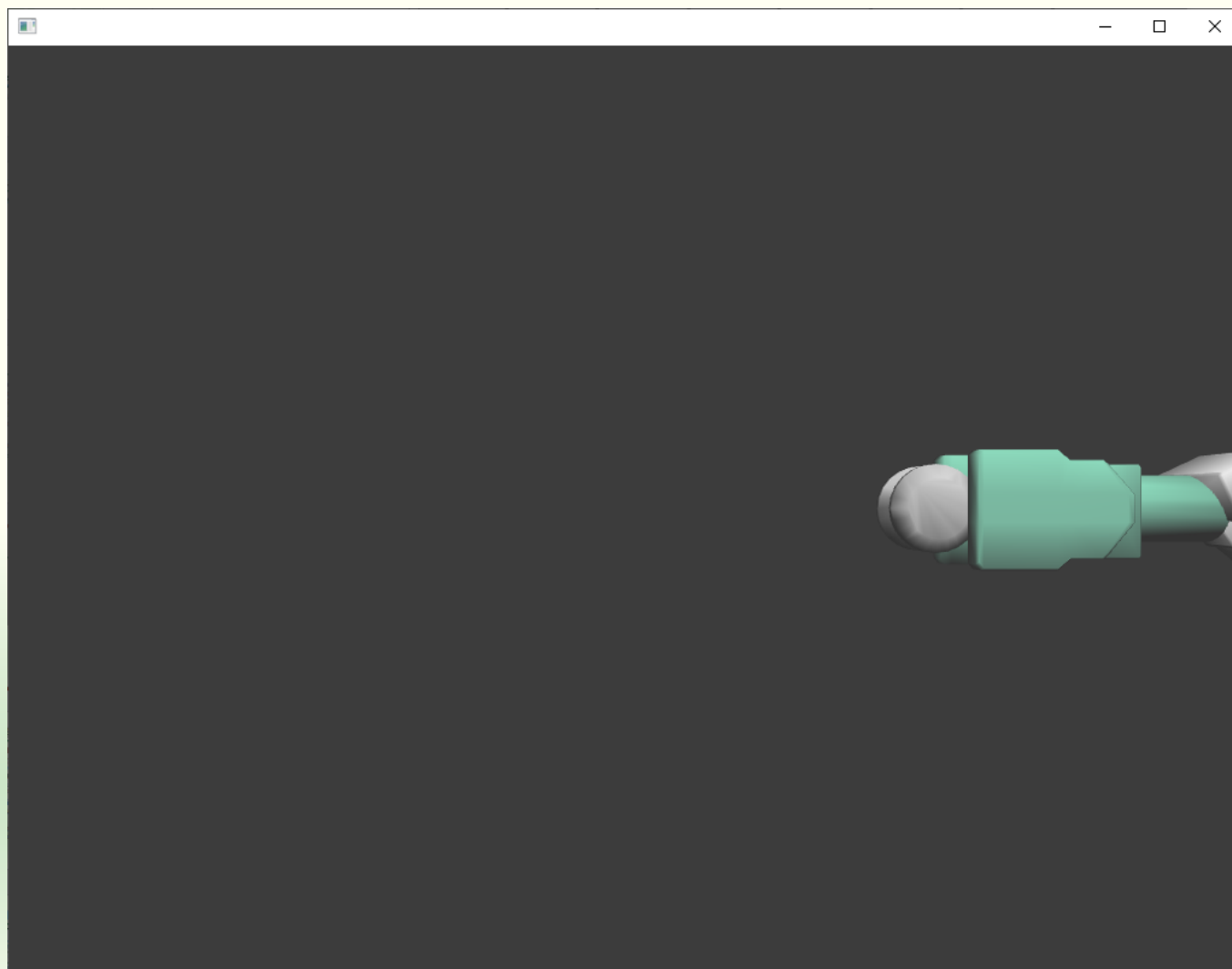
```
//-----  
void ofApp::draw(){  
    camera.begin();  
    for (auto& part : parts){  
        part.drawFaces();  
    }  
    camera.end();  
}
```

(以下略)

- for (auto& part : parts){
 ...
}
- parts の一つ一つの要素を part に入れて {...} を実行する
- これを parts のすべての要素について繰り返す
- 第3回参照



ビルドして実行するとやっぱり変わらない



コンストラクタを定義する

```
#pragma once
```

```
#include "ofMain.h"
```

```
#include "ofxAssimpModelLoader.h"
```

```
class Model : public ofNode{  
    ofxAssimpModelLoader model;
```

public を指定していないので private メンバになる

```
public:
```

```
    Model(const char* file){  
        model.loadModel(file);  
        model.setScaleNormalization(false);  
    }
```

```
    void drawFaces(){  
        (途中略)  
    }
```

(以下略)

- コンストラクタはクラスと同じ名前 (Model) のメンバ関数
 - オブジェクト（インスタンス）を生成するときに呼び出される
 - 値を返さない（戻り値はない）
- コンストラクタの引数 file で読み込むファイル名を受け取る
 - そのファイル名を使って 3D モデルファイルを読み込む
- メンバ変数の model はコンストラクタからしか操作しないので private にできる

3D モデルを `emplace_back` の引数に指定する

```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    parts.emplace_back("l-lower.obj");  
parts.back().model.loadModel("l-lower.obj");  
parts.back().model.setScaleNormalization(false);  
    parts.back().setPosition(250.0f, 0.0f, 0.0f);  
    parts.emplace_back("l-hand.obj");  
parts.back().model.loadModel("l-hand.obj");  
parts.back().model.setScaleNormalization(false);  
    parts.back().setPosition(150.0f, 0.0f, 0.0f);  
    parts.back().setParent(parts[0]);  
}
```

削除

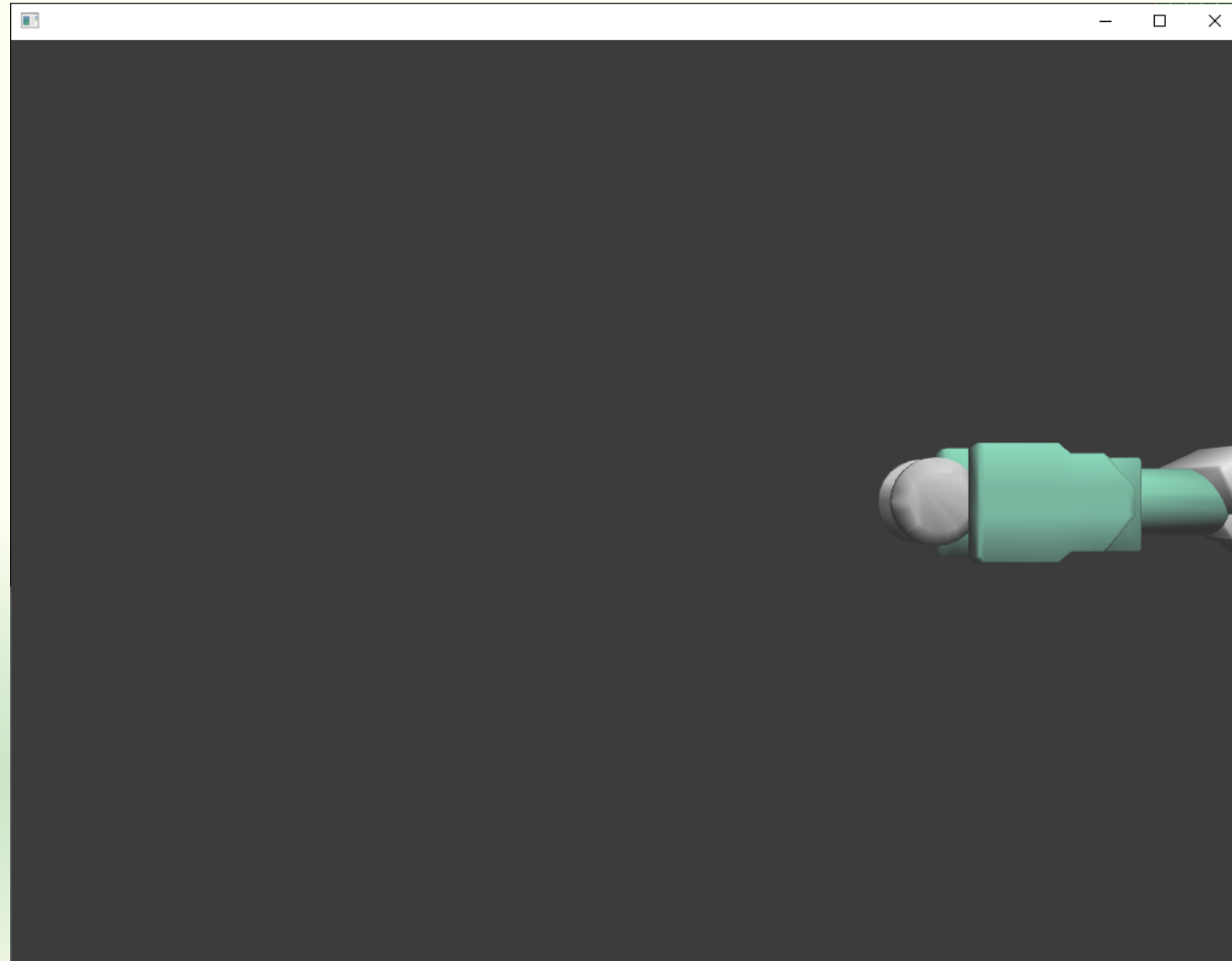
削除

(以下略)

- `emplace_back()` はオブジェクトを一つ生成して `vector` の最後に追加する
 - その際にコンストラクタが実行される
 - `emplace_back()` の引数がコンストラクタの引数に与えられる
- 第3回参照



これも結果にコミットしない



位置もコンストラクタの引数で指定する

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"

class Model : public ofNode{
    ofxAssimpModelLoader model;

public:

    Model(const char* file, float x, float y, float z){
        model.loadModel(file);
        model.setScaleNormalization(false);
        setPosition(x, y, z);
    }

    void drawFaces(){
        (途中略)
    }

    (以下略)
```

- コンストラクタの引数でモデルの位置 x, y, z を受け取る
- setPosition() は ofNode から継承した Model クラスのメソッド
 - 生成したオブジェクト（インスタンス）に対して適用される



emplace_back() の引数に位置を追加する

```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    parts.emplace_back("1-lower.obj",  
        250.0f, 0.0f, 0.0f);  
    parts.back().setPosition(250.0f, 0.0f, 0.0f);  
    parts.emplace_back("1-hand.obj",  
        150.0f, 0.0f, 0.0f);  
    parts.back().setPosition(150.0f, 0.0f, 0.0f);  
    parts.back().setParent(parts[0]);  
}
```

削除

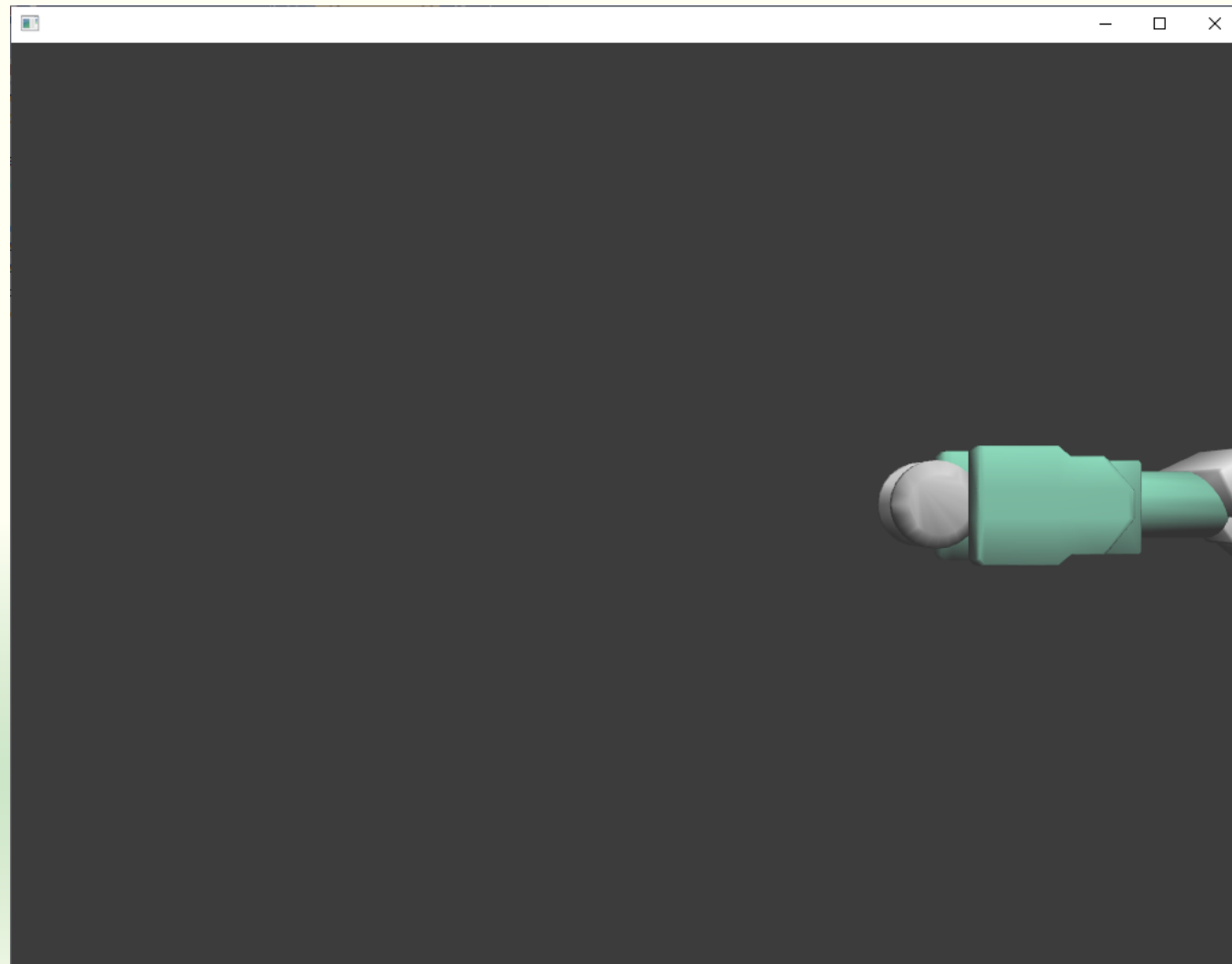
削除

(以下略)

emplace_back() でコンマ“,” の
後で改行しているのは、ここ
に書くには1行が長いから

- 読み込んだ 3D モデルの位置を
emplace_back() の引数に指定する
- Model クラスのコンストラクタは
引数に指定したファイル名の 3D
モデルファイルを読み込む
 - ofxAssimpModelLoader クラスの
loadModel() メソッド
- 読み込んだ 3D モデルを表示する
位置を引数で指定された位置に設
定する
 - ofNode クラスの setPosition() メソッド

まだ結果にコミットしない



別のオブジェクトを追加する

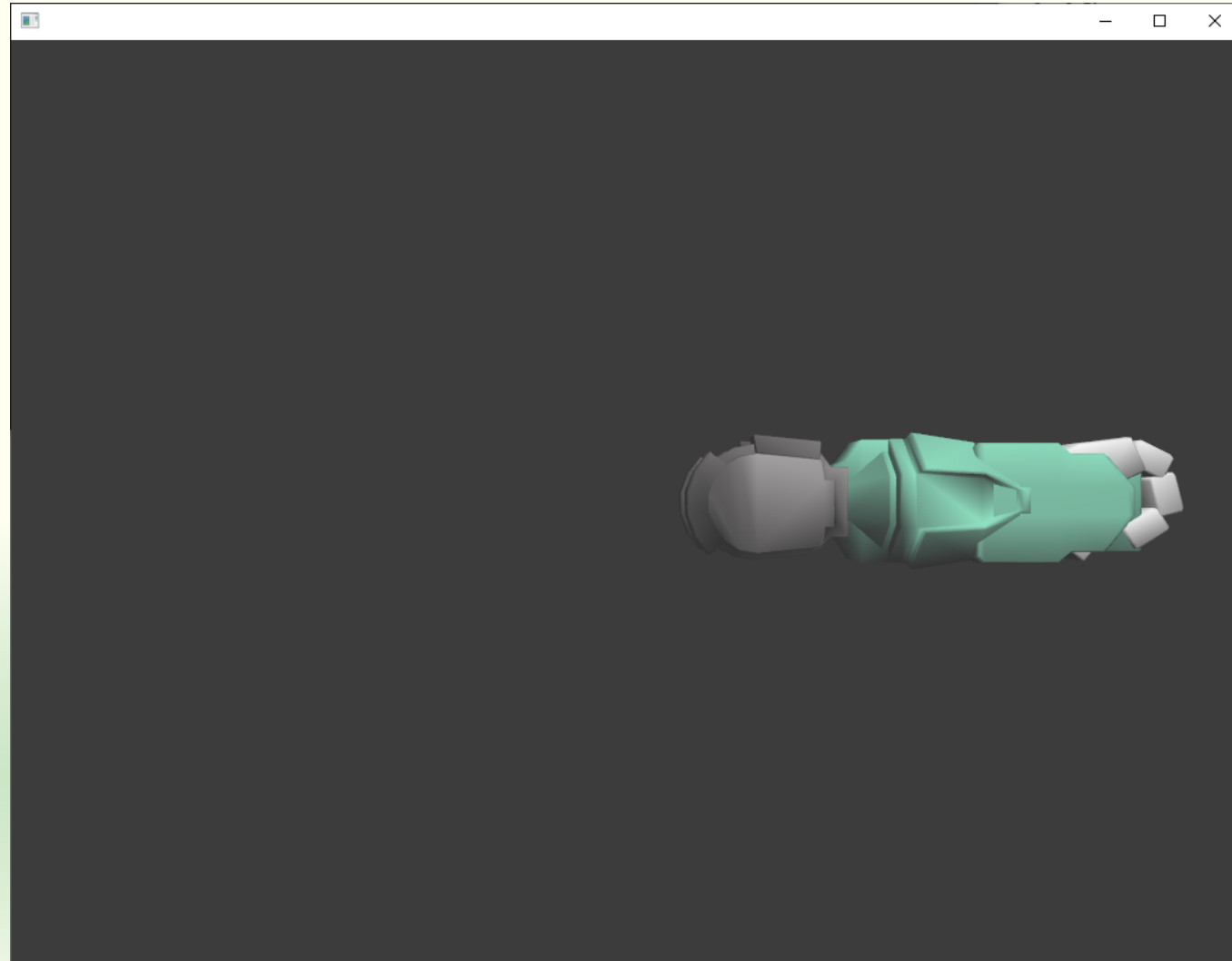
```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    parts.emplace_back("l-upper.obj",  
        100.0f, 0.0f, 0.0f);  
    parts.emplace_back("l-lower.obj",  
        250.0f, 0.0f, 0.0f);  
    parts.emplace_back("l-hand.obj",  
        150.0f, 0.0f, 0.0f);  
    parts.back().setParent(parts[0]);  
}
```

(以下略)

- "l-upper.obj" という 3D モデル
ファイルを読み込んで追加する
- モデルの位置は (100, 0, 0) とする



ビルドして実行すると重なって表示される



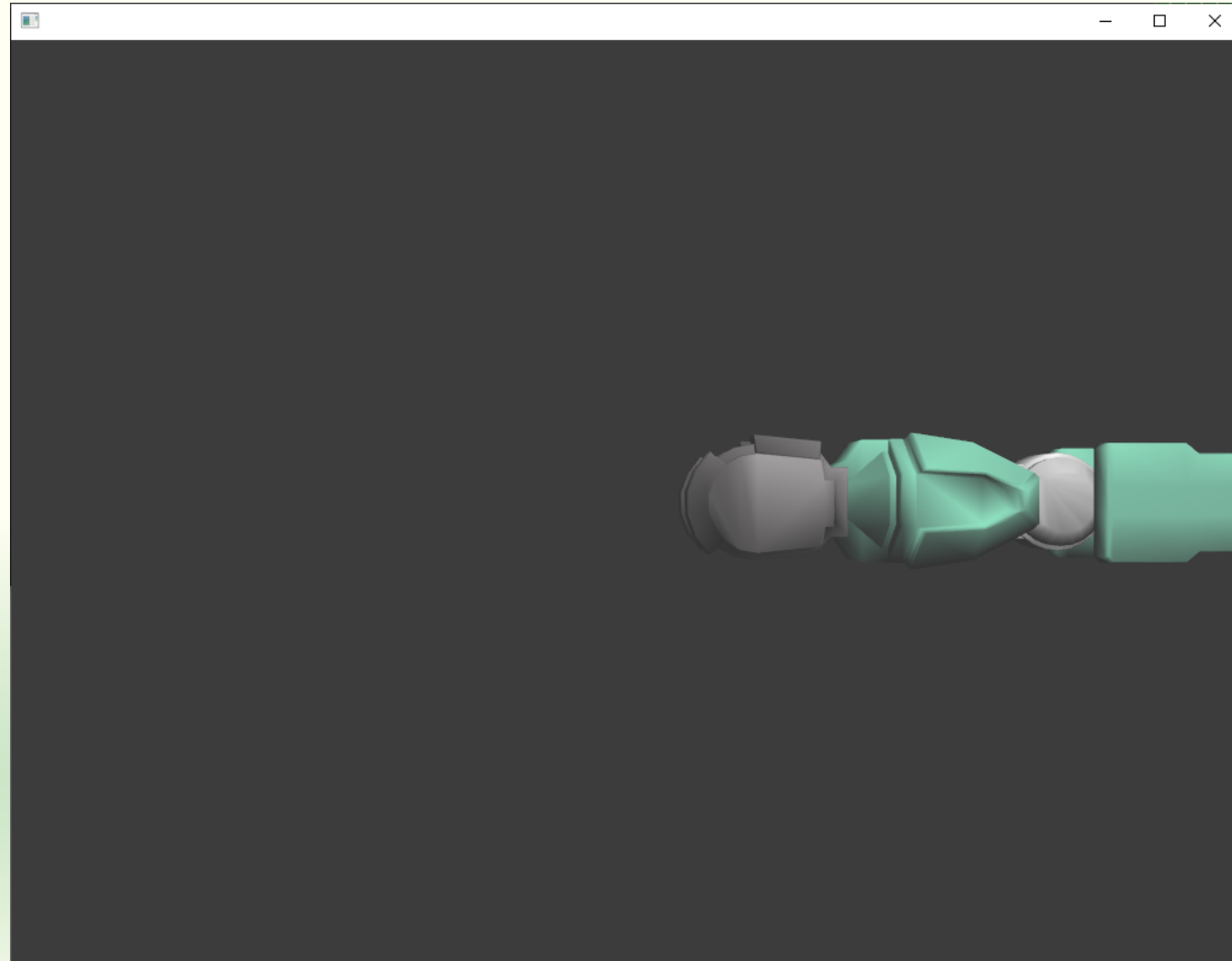
階層構造を修正する

```
//-----  
void ofApp::setup(){  
    ofEnableDepthTest();  
    light.enable();  
    light.setPosition(300.0f, 400.0f, 500.0f);  
    parts.emplace_back("1-upper.obj",  
        100.0f, 0.0f, 0.0f);  
    parts.emplace_back("1-lower.obj",  
        250.0f, 0.0f, 0.0f);  
    parts.emplace_back("1-hand.obj",  
        150.0f, 0.0f, 0.0f);  
    parts[1].setParent(parts[0]);  
    parts[2].setParent(parts[1]);  
}
```

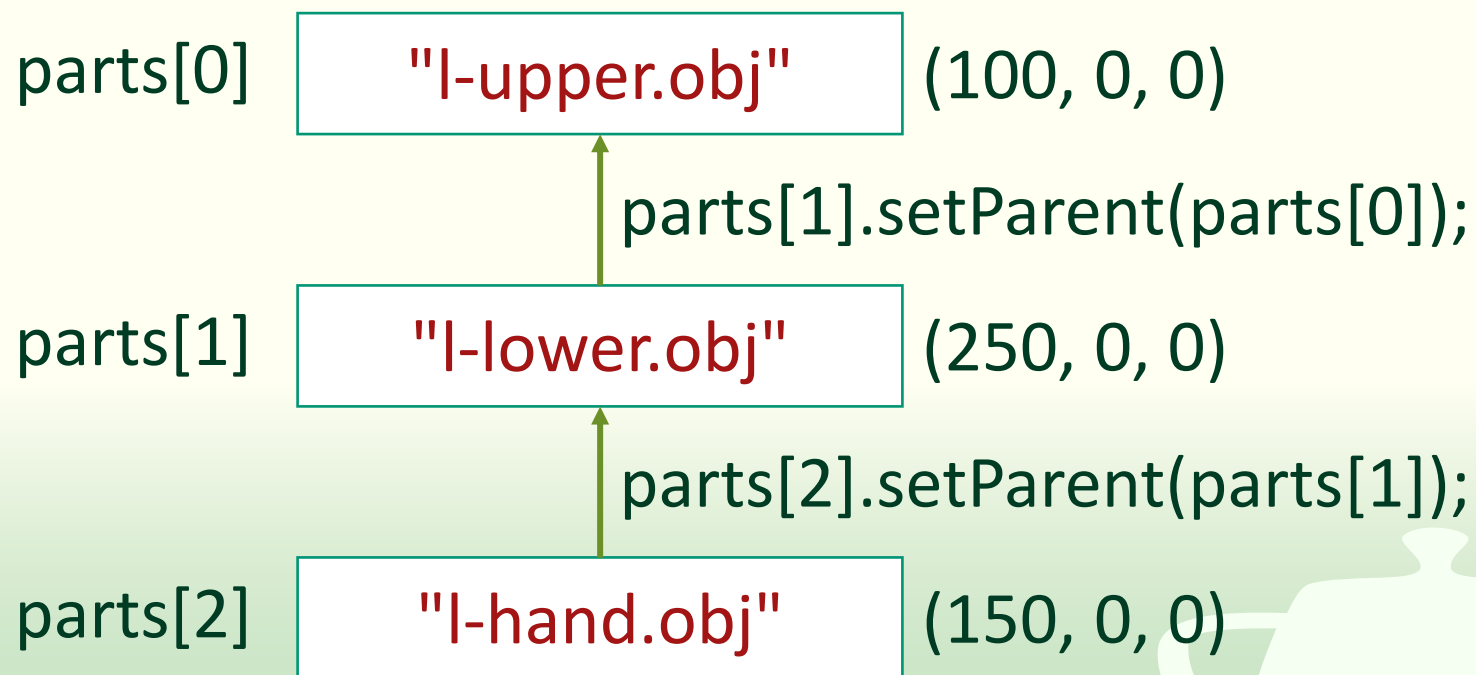
(以下略)

- 親子関係を変更する
 - 2つ目のパーツ (parts[1]) の親を 1つ目のパーツ (parts[0]) にする
 - 3つ目のパーツ (parts[2]) の親を 2つ目のパーツ (parts[1]) にする
- 階層構造の設定は vector への要素の追加が完了した後に行う
 - vector は要素を追加する際にメモリが足りないとメモリを確保し直す場合があり、その時にオブジェクトの格納場所が移動してしまう

ビルドして実行するとひと続きになる



3D モデルの階層構造





課題 7 - 1

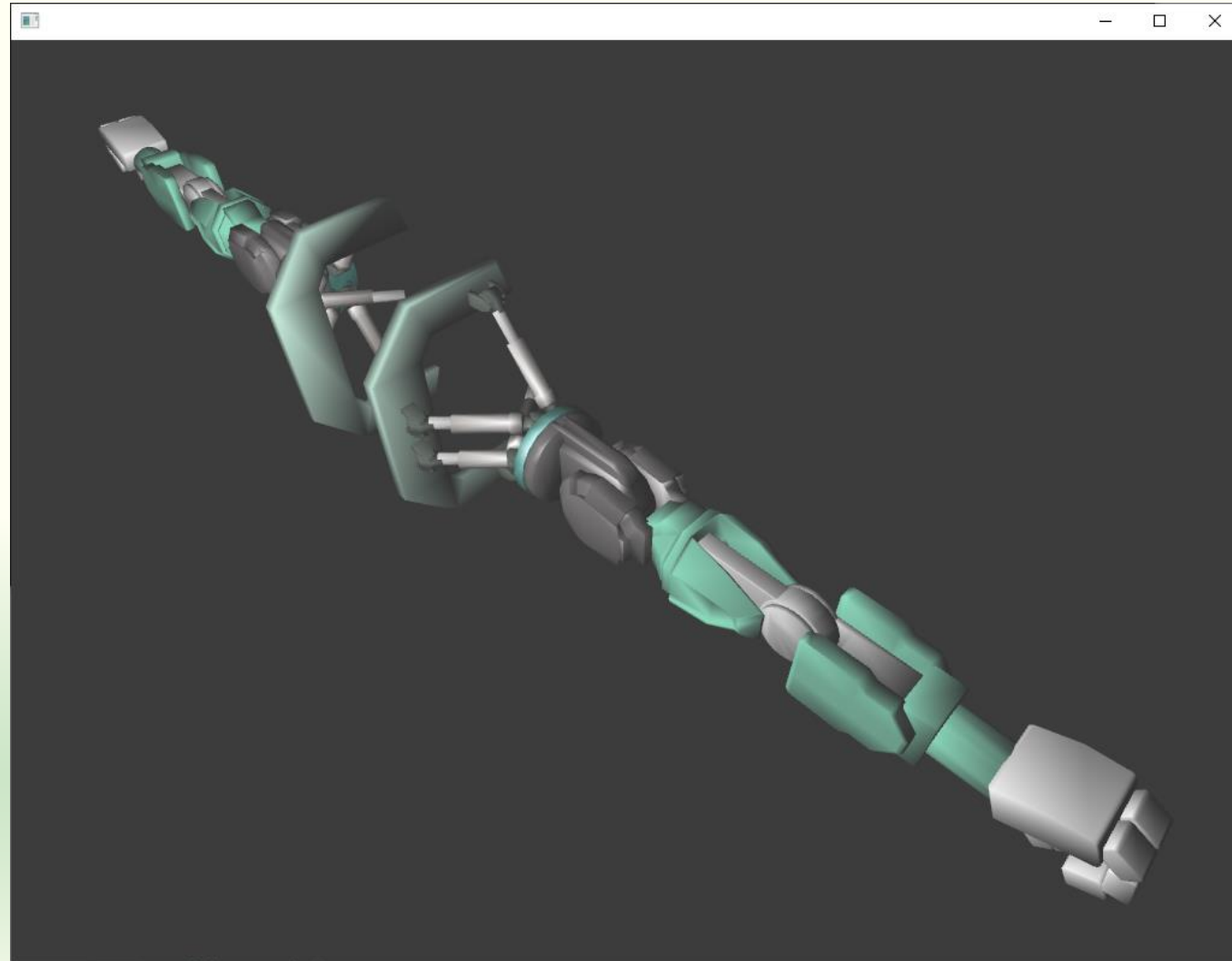
ロボットの両腕を完成させる

ロボットの両腕を完成させなさい

- bin > data フォルダに置いた 3D モデルファイルを読み込み、右の表に従って階層構造を与えて配置しなさい
 - l-upper.obj, l-lower.obj, l-hand.obj の読み込みと配置は完了している

ファイル名	相対位置	親
l-shoulder.obj	(100, 0, 0)	(指定しない)
l-hinge.obj	(170, 0, 0)	l-shoulder.obj
l-upper.obj	(100, 0, 0)	l-hinge.obj
l-lower.obj	(250, 0, 0)	l-upper.obj
l-hand.obj	(100, 0, 0)	l-lower.obj
r-shoulder.obj	(-100, 0, 0)	(指定しない)
r-hinge.obj	(-170, 0, 0)	r-shoulder.obj
r-upper.obj	(-100, 0, 0)	r-hinge.obj
r-lower.obj	(-250, 0, 0)	r-upper.obj
r-hand.obj	(-100, 0, 0)	r-lower.obj

結果の例



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **7-1.png** というファイル名で保存し、Moodle の第 7 回課題にアップロードしてください
- マウスを使ってウィンドウ内に両腕が全部収まるようにしてください





課題 7 - 2

GUI で操作する

ロボットの腕のすべての関節の角度を操作する ユーザインタフェースを追加しなさい

- パーツの角度を変更するには rotate メソッドを使う

// 0 番目のパーツの角度を Y 軸中心に現状から 1 度回転する
`parts[0].rotateDeg(1.0f, 0.0f, 1.0f, 0.0f);`

- https://openframeworks.cc/documentation/3d/ofNode/#show_rotateDeg

- ‘+’ キーか ‘-’ キーかで回転方向を変えられると良い
- パーツの番号 (0~9) もキー操作 (‘0’~‘9’) で切り替えられると良い
 - これは static 変数に保存しておく必要があると思う



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **7-2.png** というファイル名で保存し、Moodle の第 7 回課題にアップロードしてください
 - GUI を操作してロボットの腕の姿勢を初期状態から変えてください
 - マウスを使ってウィンドウ内に両腕が全部収まるようにしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第 7 回課題にアップロードしてください






ユーザインタフェース

ofxImGui の使い方

ofxImGui を入手する

[第7回] ユーザーインターフェース

 第7回資料

 model.zip

 ofxImGui-1.75.zip

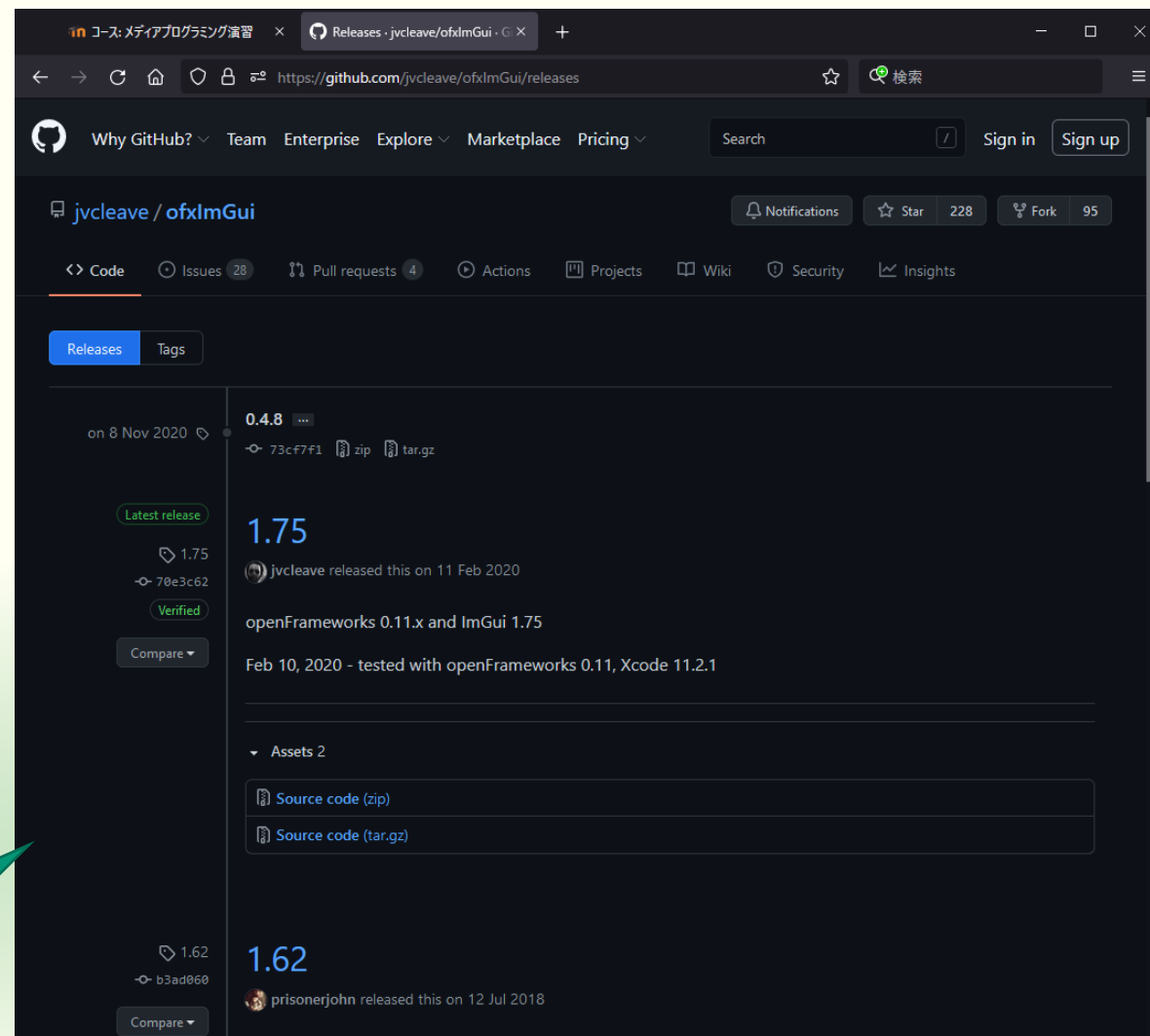
 第7回課題

 何か一言

Moodle から [ofxImGui-1.75.zip](https://github.com/jvcleave/ofxImGui/releases) をダウンロードしてください

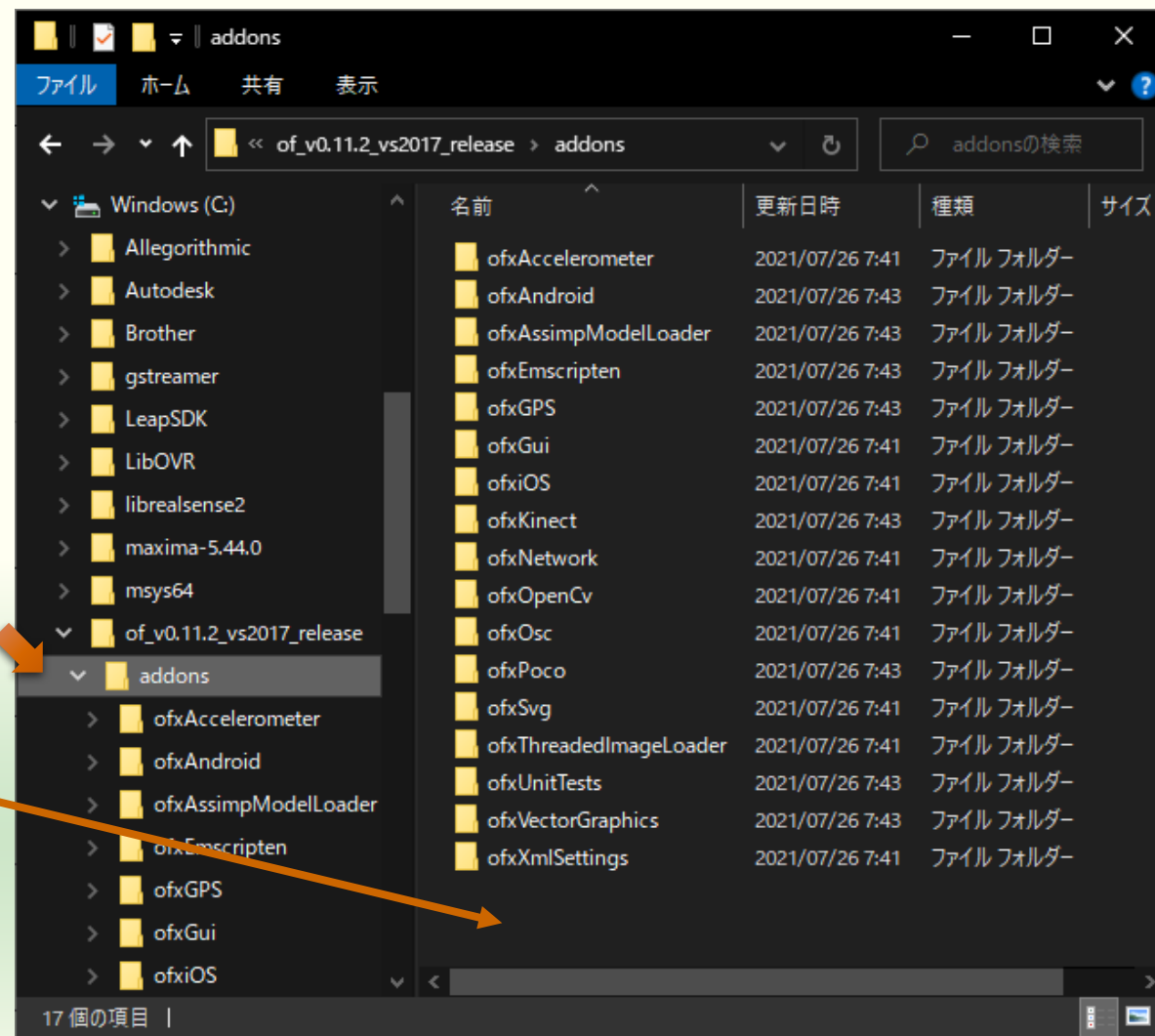
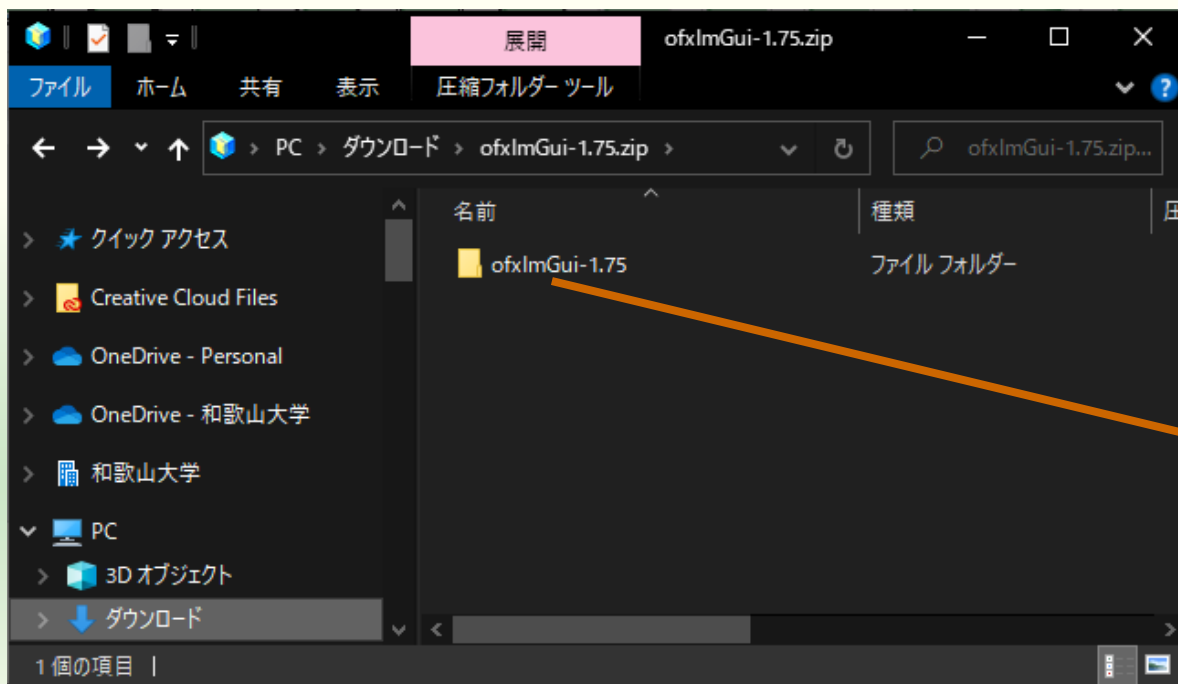
正式なものは GitHub にあります

<https://github.com/jvcleave/ofxImGui/releases>

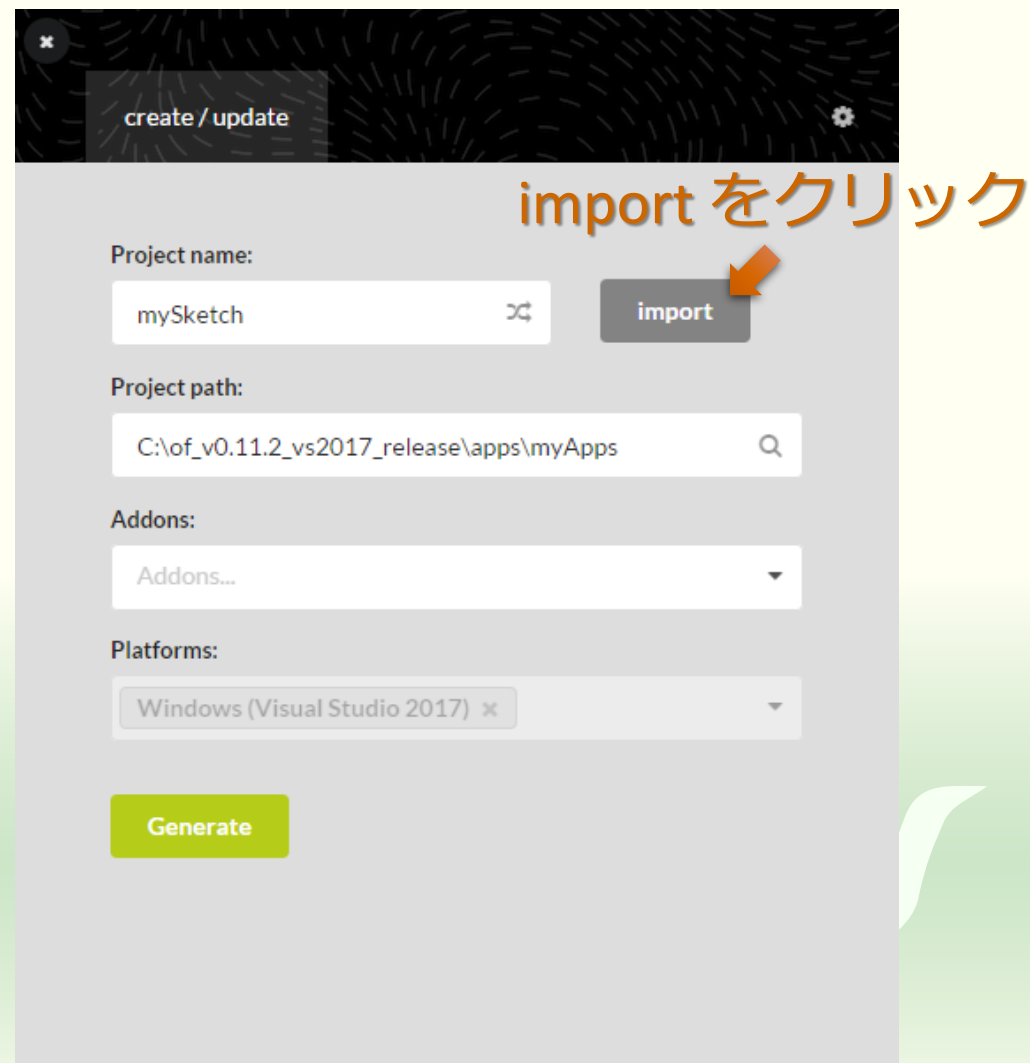
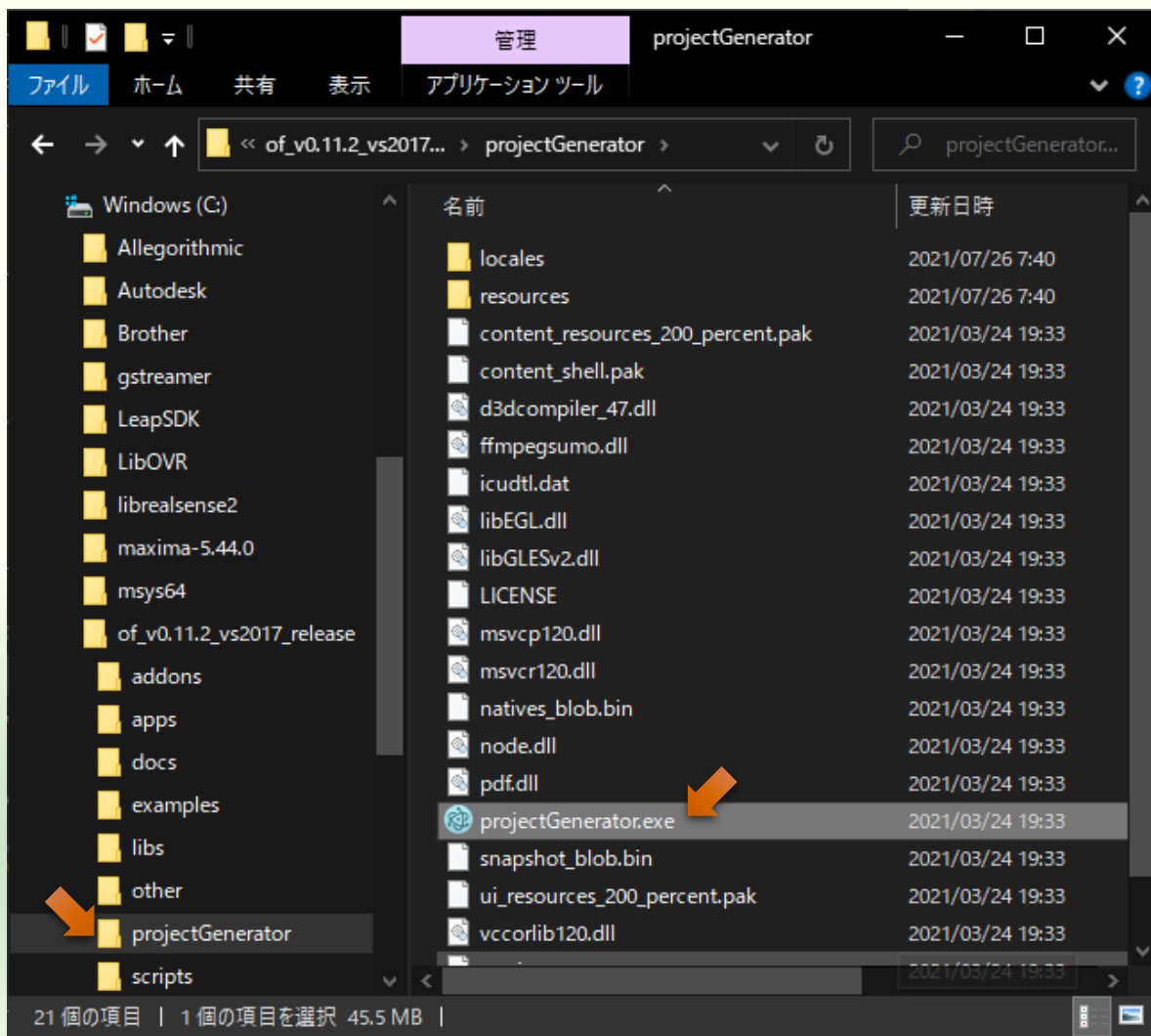


addons に追加する

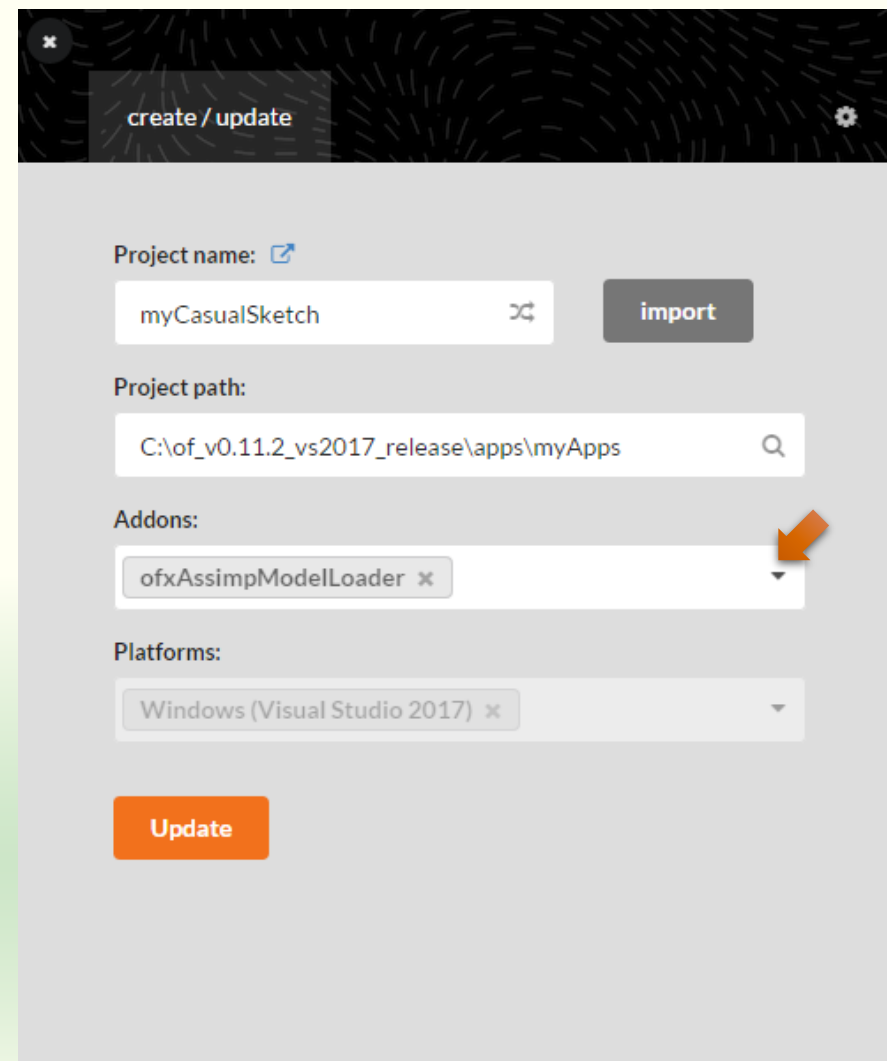
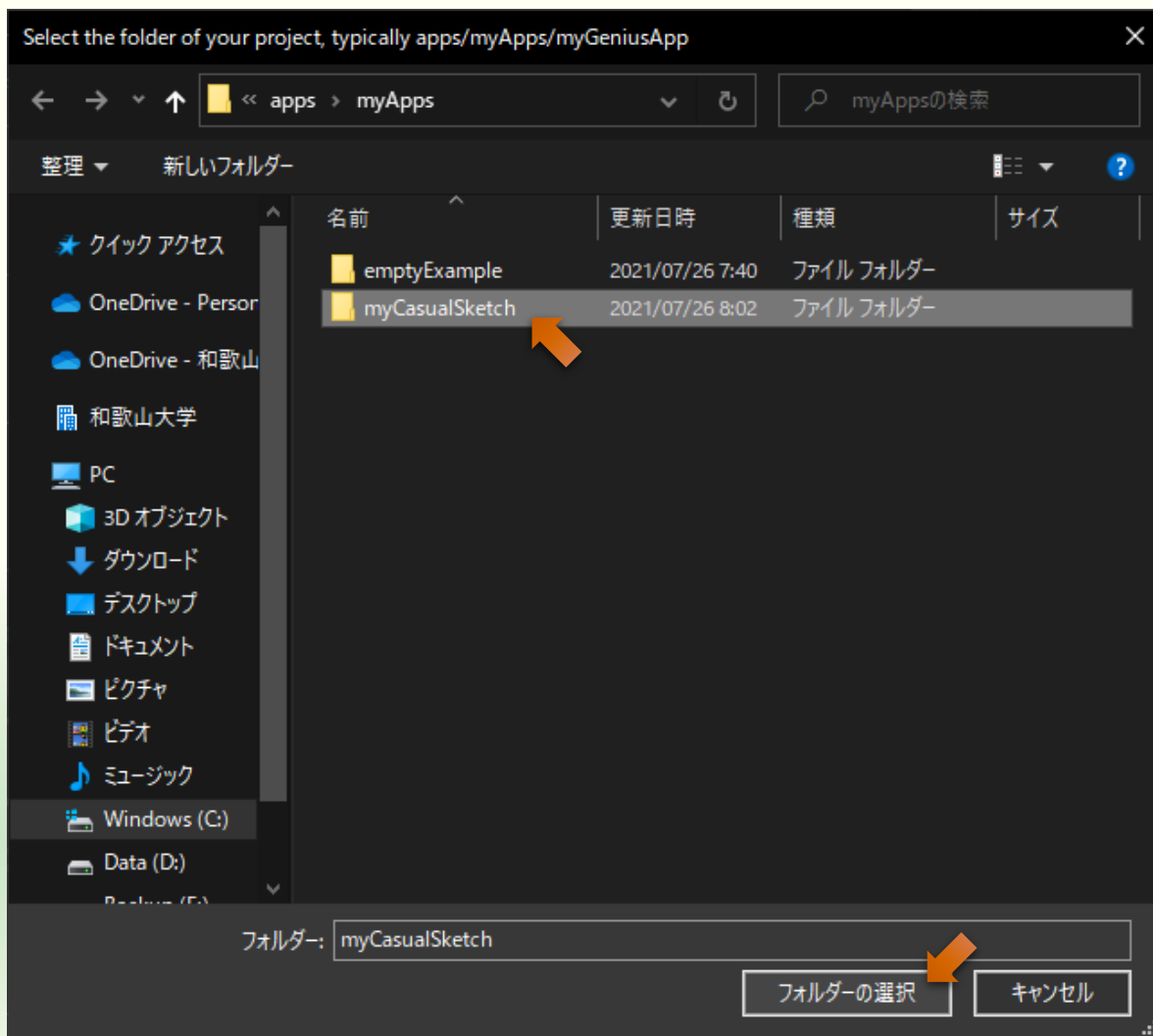
- ofxImGui-1.75.zip を展開した中のフォルダを openFrameworks を展開したフォルダの中の addons の中に移す



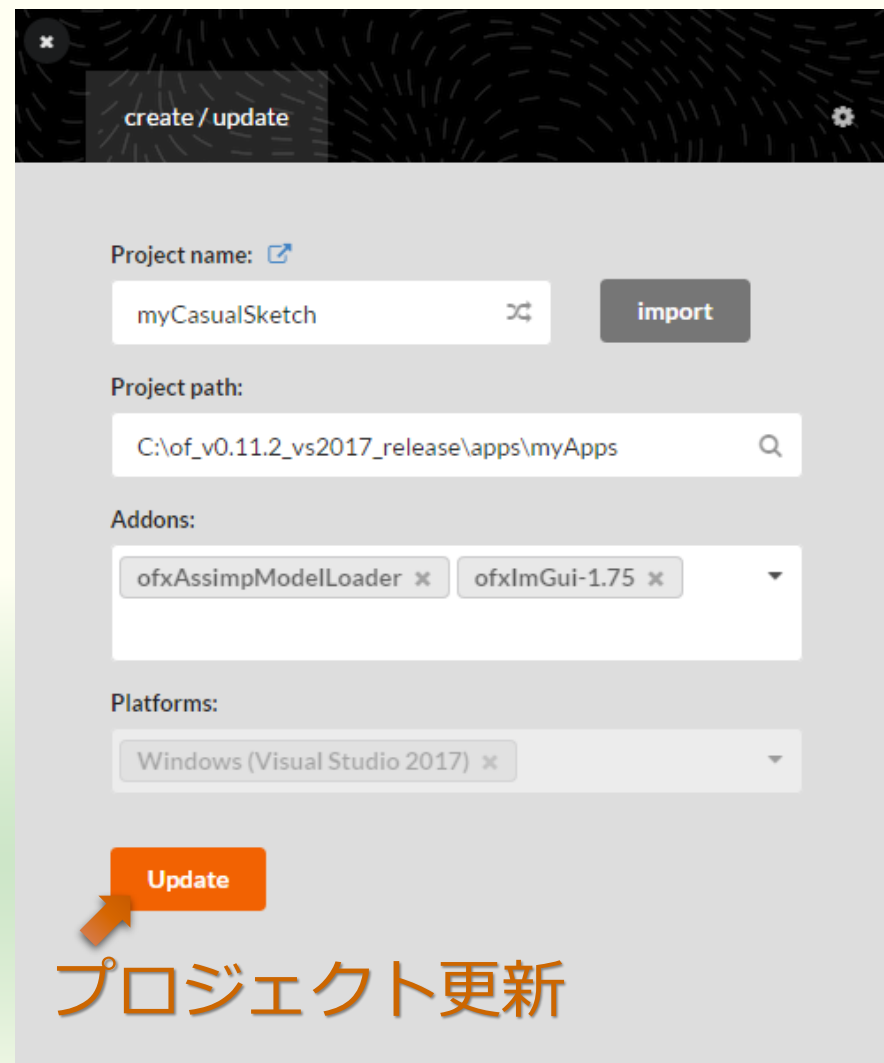
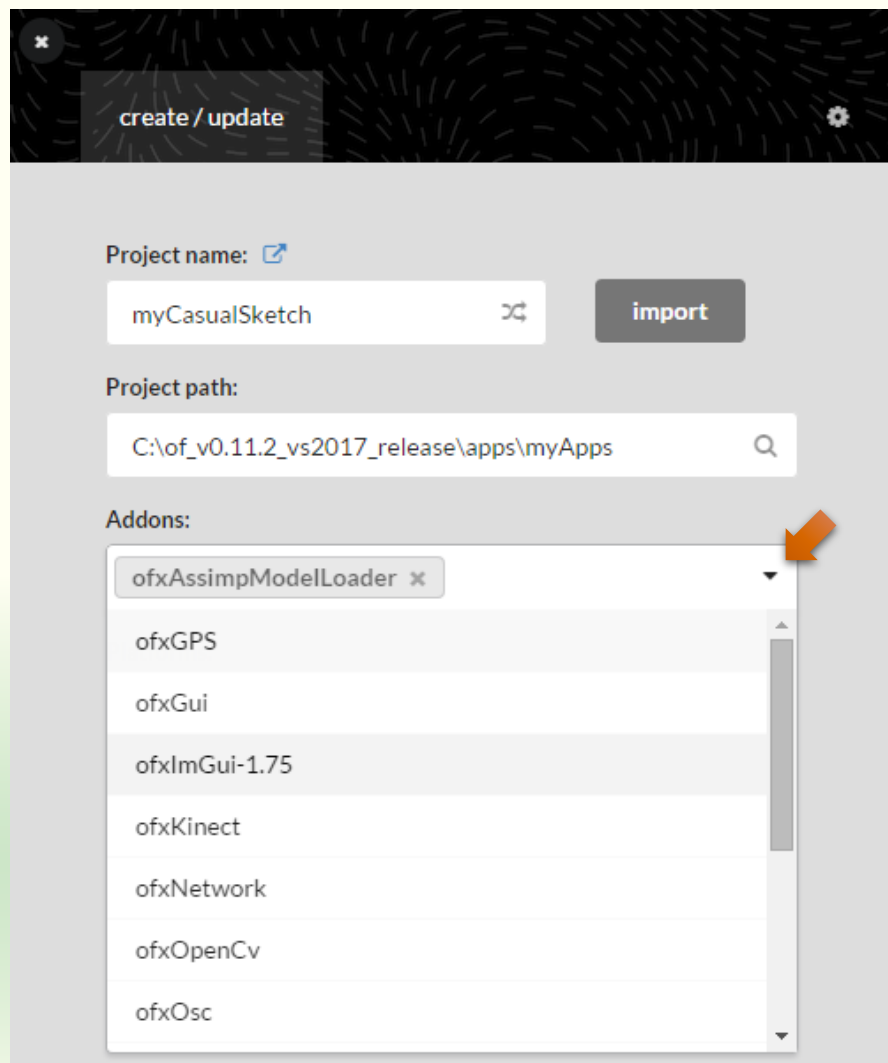
projectGenerator を起動する



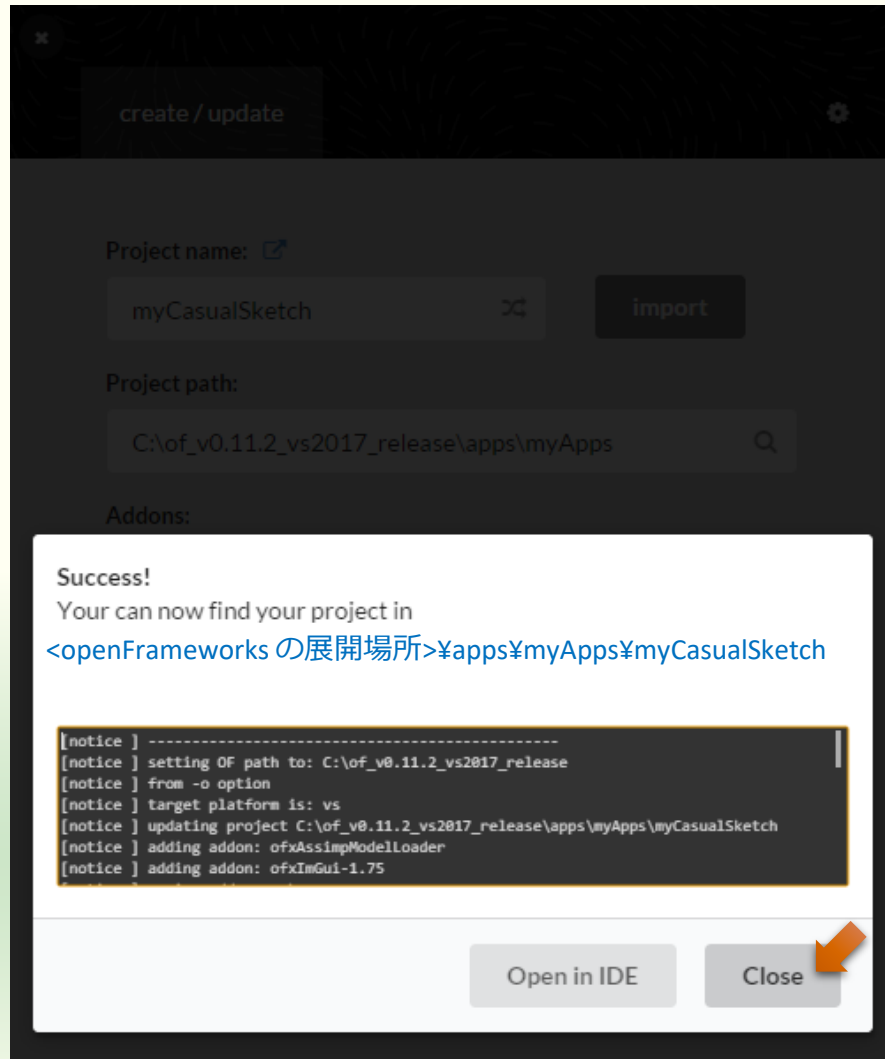
このプロジェクトのフォルダを選択する



ofxImGui-1.75 を addon として追加



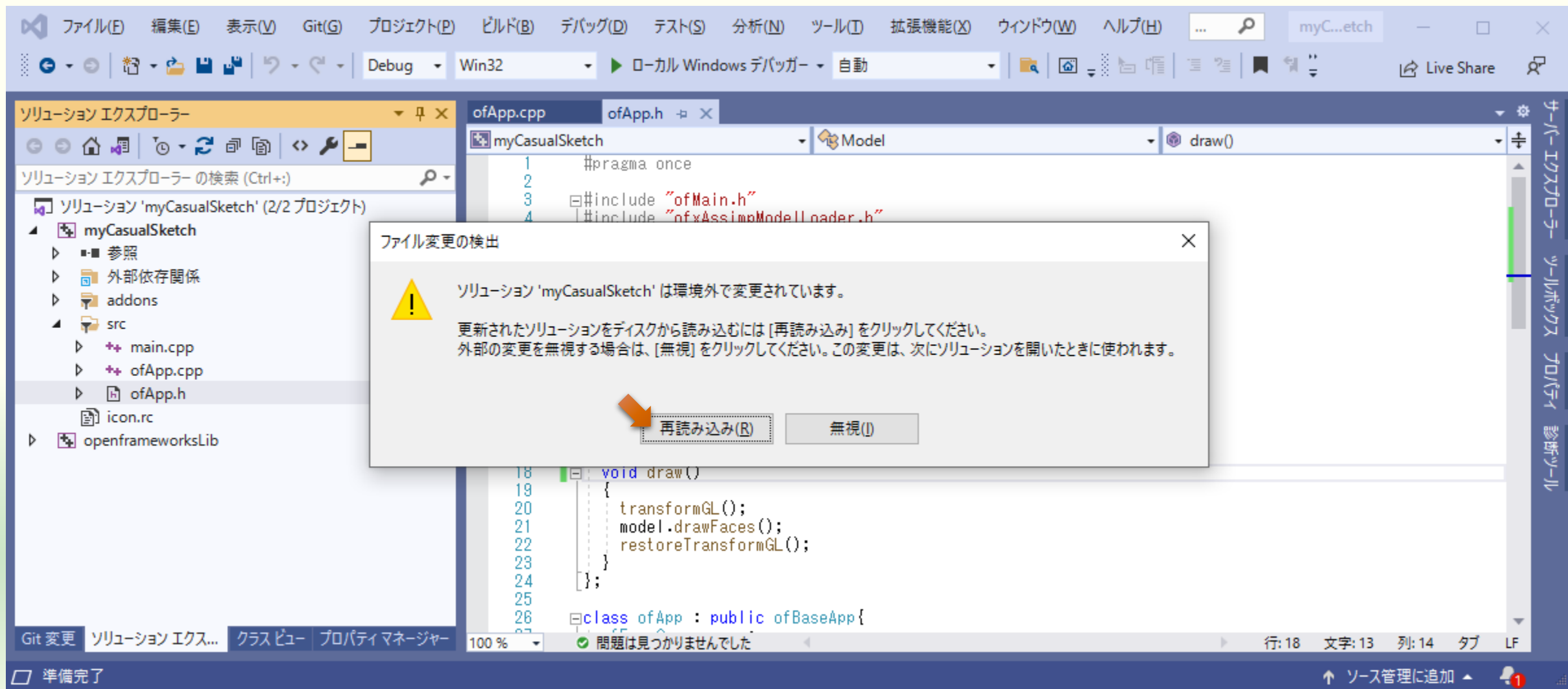
プロジェクト更新成功



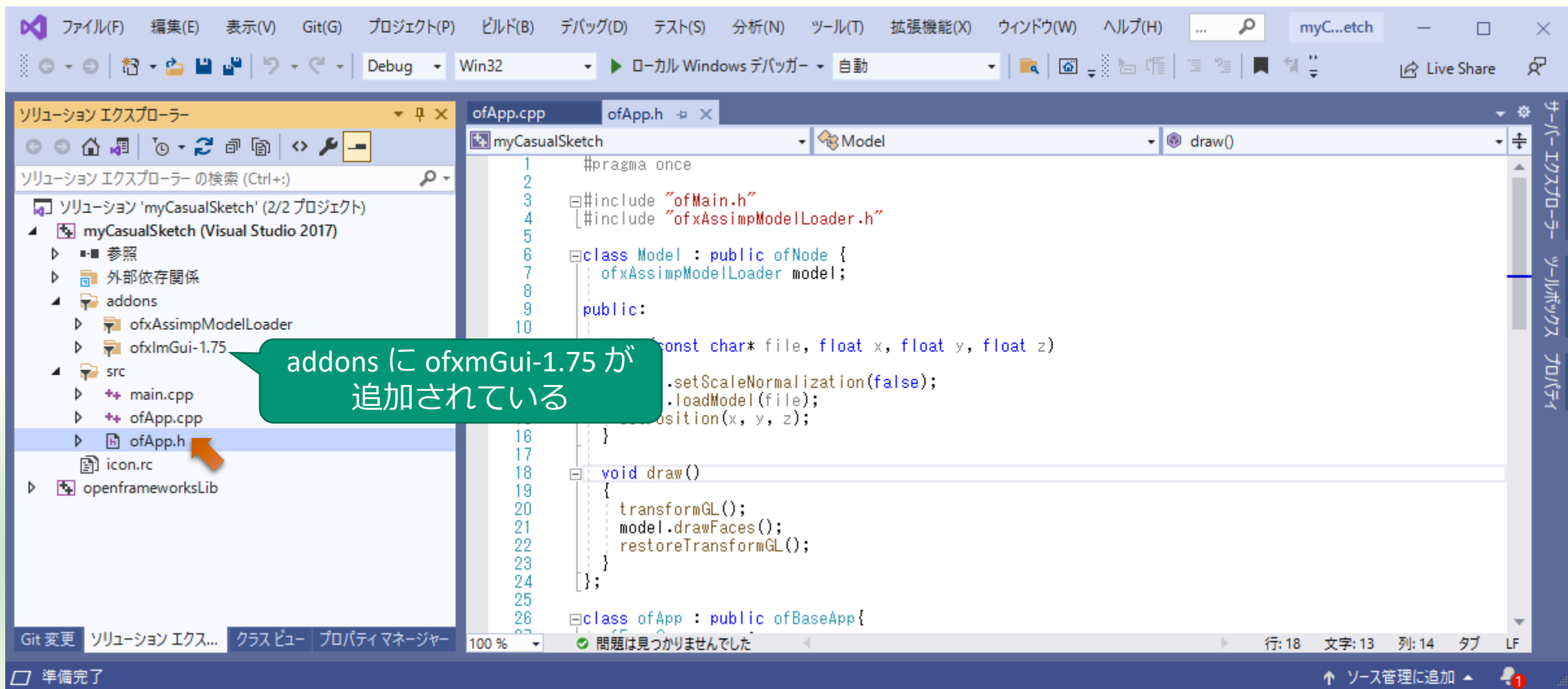
- 既に Visual Studio でプロジェクトを開いているときは Close
 - Visual Studio に戻るとこのスライドの次のページに示す警告が出る
- 現在 Visual Studio でプロジェクトを開いていなければ Open in IDE で構わない



プロジェクトを開いているときは再読み込み



ofApp.h を開く



ofxImGui.h を #include して ofxImGui::Gui のインスタンスをメンバーに追加

```
#pragma once

#include "ofMain.h"
#include "ofxAssimpModelLoader.h"
#include "ofxImGui.h"

class Model : public ofNode{
    (途中略)
};

class ofApp : public ofBaseApp{
    ofCamera camera;
    ofLight light;
    vector<Model> parts;
    ofxImGui::Gui gui;

public:
    (以下略)
```

- ofApp.h で ofxImGui.h を #include する
- ofApp クラスに ofxImGui::Gui クラスの変数 gui をメンバに追加する
- camera オブジェクトのクラスを ofEasyCam から ofCamera に変更する
 - GUI の操作で視点位置が動いてしまわないようにするため

ofxImGui を初期化する

```
#include "ofApp.h"

//-----
void ofApp::setup(){
    ofEnableDepthTest();
    light.enable();
    light.setPosition(300.0f, 400.0f, 500.0f);
    gui.setup();
    (以下略)
```

- ofApp.cpp の setup() で gui.setup() を実行して gui を初期化する



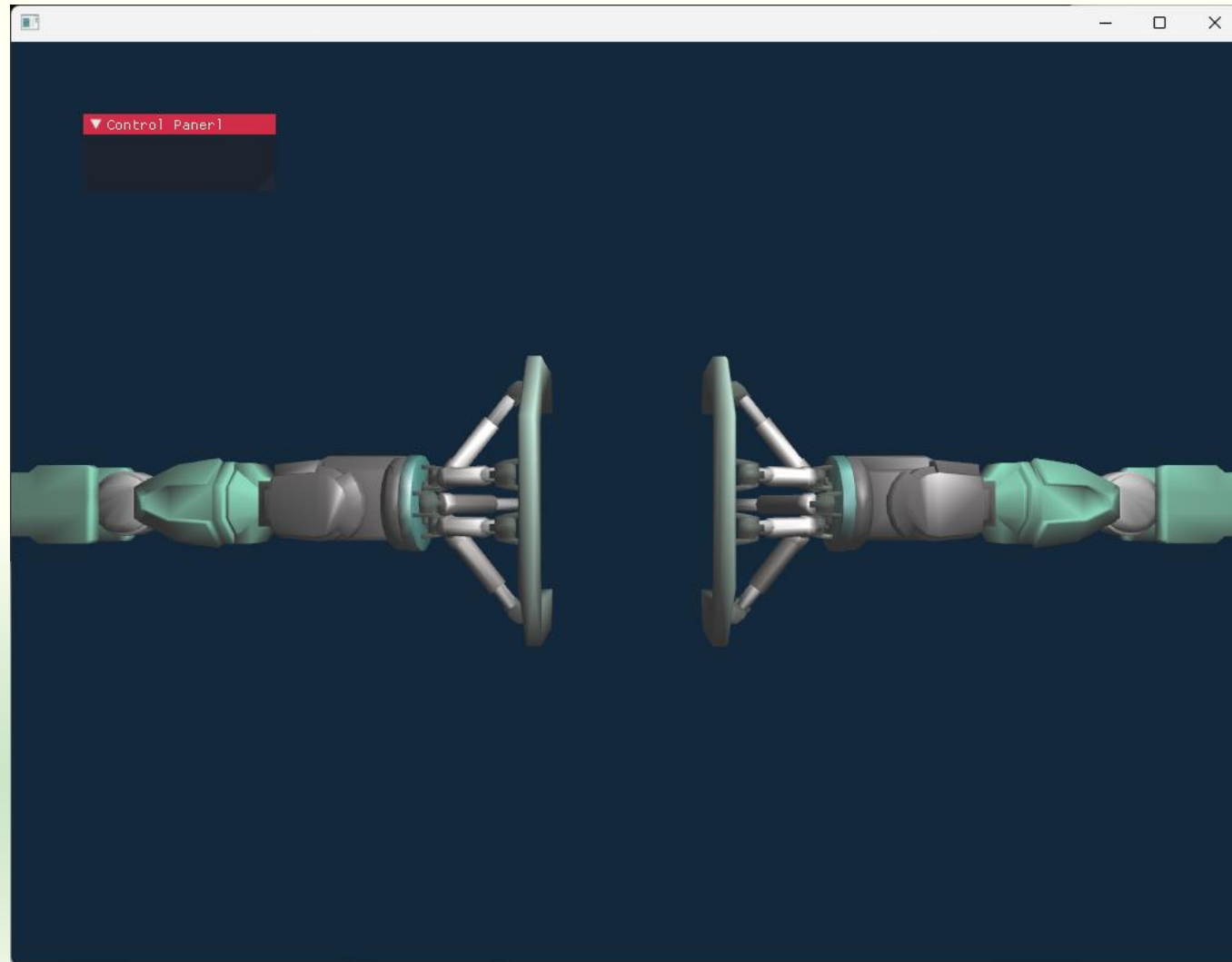
GUI のウィンドウを開く

```
//-----  
Void ofApp::draw(){  
  ofBackground(20, 40, 60, 0);  
  ofDisableLighting();  
  gui.begin();  
  ImGui::Begin("Control Panerl");  
  ImGui::End();  
  gui.end();  
  ofEnableLighting();  
  camera.begin();  
  camera.setPosition(0.0f, 0.0f, 1000.0f);  
  for (auto& part : parts){  
    part.draw();  
  }  
  camera.end();  
}
```

↑
camera のクラスを ofEasyCam
から ofCamera に切り替えた
のでカメラの位置を設定する

- ofDisableLighting();
 - GUI は 2 次元なので陰影付け（ライティング）はオフにする
- gui.begin();～gui.end();
 - この間に GUI の設定を書く
- ImGui::Begin("Control Panerl");～ImGui::End();
 - この間に GUI ウィンドウの設定を書く
 - 引数 ("Control Panerl") はウィンドウのタイトル

GUI のウィンドウが表示される



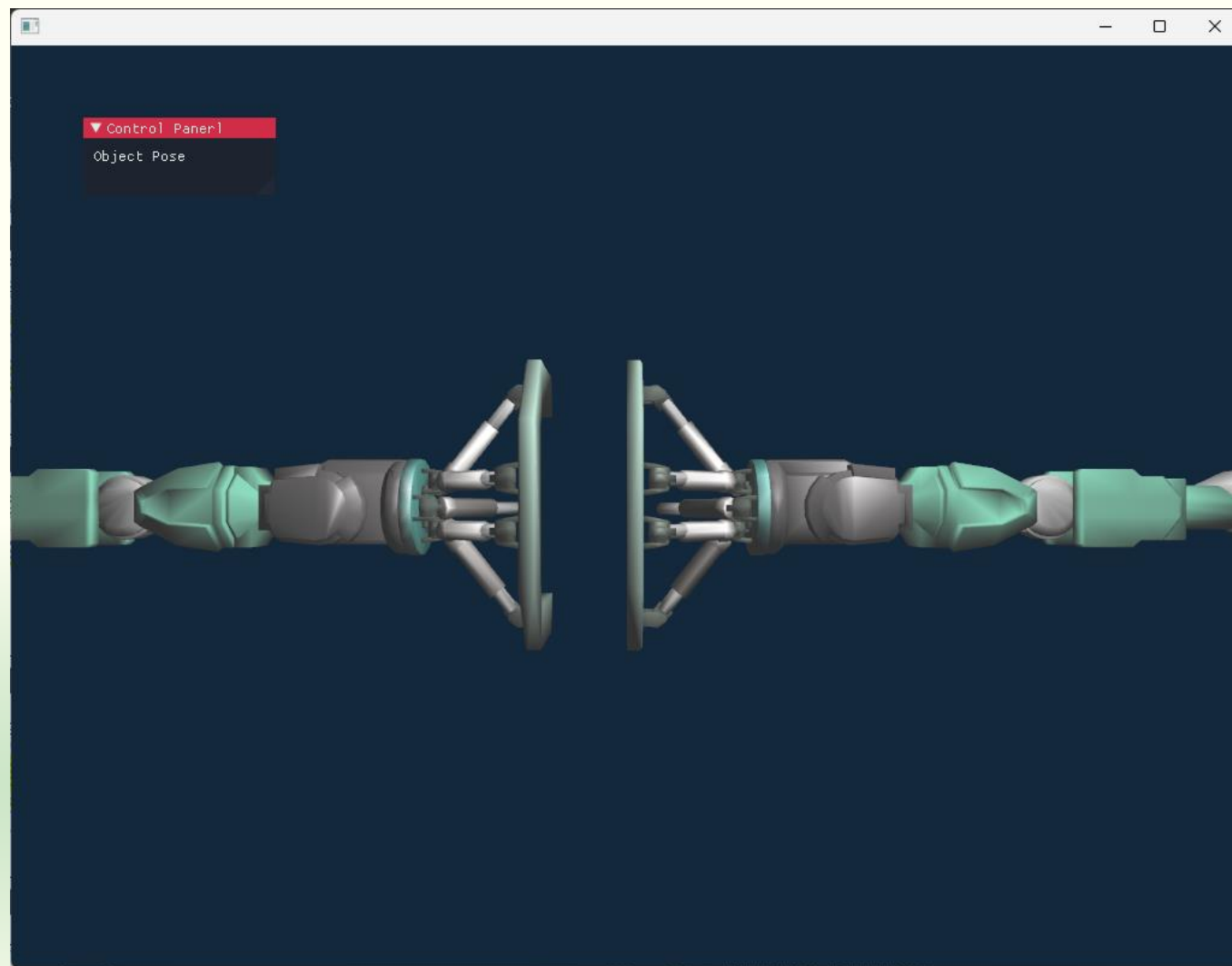
GUI に文字を表示する

```
//-----  
Void ofApp::draw(){  
  ofBackground(20, 40, 60, 0);  
  ofDisableLighting();  
  gui.begin();  
  ImGui::Begin("Control Panerl");  
  ImGui::Text("Object Pose");  
  ImGui::End();  
  gui.end();  
  ofEnableLighting();  
  camera.begin();  
  camera.setPosition(0.0f, 0.0f, 1000.0f);  
  for (auto& part : parts){  
    part.draw();  
  }  
  camera.end();  
}
```

- `ImGui::Text("Object Pose");`
 - GUI のウィンドウに文字を表示する
 - 日本語も表示できるが日本語を表示できるフォントに切り替える必要がある
 - "Object Pose" は表示する文字列



GUI のウィンドウに文字が表示される



図形の位置のメンバを追加する

```
class ofApp : public ofAppBaseApp{  
    ofCamera camera;  
    ofLight light;  
    vector<Model> parts;  
    ofxImGui::Gui gui;  
    glm::vec3 position;  
  
    public:  
        void setup();  
        void update();  
        void draw();  
}
```

(以下略)

- ofApp.h で ofApp クラスに glm::vec3 型のメンバ変数 position を追加する

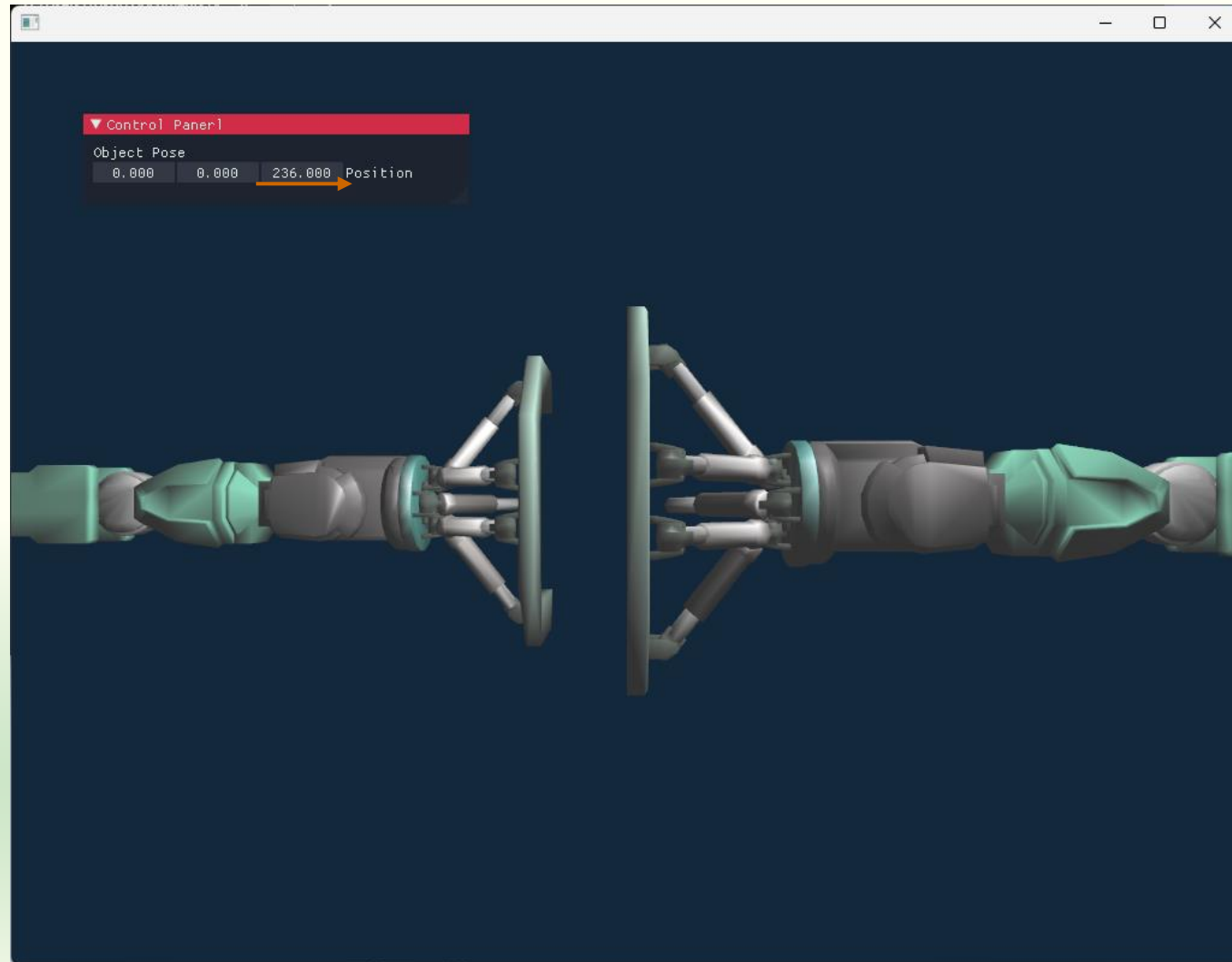


図形の位置を GUI で操作する

```
//-----  
void ofApp::draw(){  
    ofBackground(20, 40, 60, 0);  
    ofDisableLighting();  
    gui.begin();  
    ImGui::Begin("Control Panerl");  
    ImGui::Text("Object Pose");  
    ImGui::DragFloat3("Position",  
        reinterpret_cast<float*>(&position));  
    ImGui::End();  
    gui.end();  
    ofEnableLighting();  
    camera.begin();  
    camera.setPosition(0.0f, 0.0f, 1000.0f);  
    parts[0].setPosition(position);  
    for (auto& part : parts){  
        part.draw();  
    }  
    camera.end();  
}
```

- `ImGui::DragFloat3("Position", reinterpret_cast<float*>(&position));`
- `ImGui::DragFloat3()` は 3 つの要素を持つ `float` 型の配列の内容をマウスのドラッグで変更する
- `position` は `glm::vec3` 型だがメモリ上のデータの並びは 3 つの要素を持つ `float` 型の配列と同じなので `reinterpret_cast<float*>()` を使ってそれに見せかけている
- `position` には `setup()` で 0 を入れる

数字の上をドラッグすると右側だけ動く



図形の角度のメンバを追加する

```
class ofApp : public ofAppBaseApp{
    ofCamera camera;
    ofLight light;
    vector<Model> parts;
    ofxImGui::Gui gui;
    glm::vec3 position, orientation;
public:
    void setup();
    void update();
    void draw();
```

(以下略)

- ofApp.h で ofApp クラスに
glm::vec3 型のメンバ変数
orientation を追加する



スライダーで角度を設定する

```
//-----  
void ofApp::draw(){  
    ofBackground(20, 40, 60, 0);  
    ofDisableLighting();  
    gui.begin();  
    ImGui::Begin("Control Panel");  
    ImGui::Text("Object Pose");  
    ImGui::DragFloat3("Position",  
        reinterpret_cast<float*>(&position));  
    ImGui::SliderFloat("Yaw", &orientation.y,  
        -180.0f, 180.0f);  
    ImGui::SliderFloat("Pitch", &orientation.z,  
        -180.0f, 180.0f);  
    ImGui::SliderFloat("Roll", &orientation.x,  
        -180.0f, 180.0f);  
    ImGui::End();  
    gui.end();  
    ofEnableLighting();  
    camera.begin();  
    camera.setPosition(0.0f, 0.0f, 1000.0f);  
    parts[0].setPosition(position);  
    parts[0].setOrientation(orientation);  
}
```

- `ImGui::SliderFloat("Yaw", &orientation.y, -180.0f, 180.0f);`
 - `orientation.y` に -180~180 の範囲でスライダで値を設定する





課題 7 – 3

両方とも GUI で操作する

右側だけでなく左側も GUI で動くようにする

- これまでのプログラムでは左側の腕しか動きません
- これを右側の腕も GUI で動かせるように修正してください



課題のアップロード

- 作成したプログラムの実行結果のスクリーンショットを撮って **7-3.png** というファイル名で保存し、Moodle の第 7 回課題にアップロードしてください
 - GUI を操作してロボットの腕の姿勢を初期状態から変えてください
 - マウスを使ってウィンドウ内に両腕が全部収まるようにしてください
- ソースプログラム **ofApp.h** と **ofApp.cpp** を Moodle の第 7 回課題にアップロードしてください
 - 課題 7 - 2 に GUI を追加しただけなので上書きしてかまいません

