グラフィックス演習（第３週目）

日付：

学生番号　　氏名

● グループの学生のリスト（参加者のみ）

	ファーストネーム	特記すべき事項
1.本人		
2		
3		
4		
5		
6		

● 本日の課題の評価

項目	評価	1 Excellent	2 Good	3 Fair	4 Poor
複数ファイルの開発の手法の理解		makefile を自由に作成する事ができる。	makefile を読み取り、そのプログラムの作成の手順が読み取れる。	make の必要性は理解できた。	make の必要性が理解できない。
C言語によるプログラミングの理解		適切なデータ構造やアルゴリズムを駆使したプログラム開発が可能である。	リストなどのデータ構造を利用したプログラムを作成できる。	仕様書が指定されれば、プログラムを記述できる。	構造化プログラミングの実現方法が分からない。
OpenGL による三次元グラフィックス		OpenGL を使ったユーザーインターフェースをもつソフトウェアを記述できる。	OpenGL を使って３次元物体を記述できる。	OpenGL を使ってプログラミングが可能である。	OpenGL とは何かを知らない。
グループでの活動		グループで活動することにより、自分だけでは洗い出せなかった問題を明確化できた。	グループで問題点を出し合い、個別の活動に活かせた。	自分自身が分からない点をグループ員から明らかにすることができた。	グループでの意見交換はあまり活発ではなかった。
今後の課題の現状調査					
最終成果物の理解		分子グラフィックスを自由に作成できる	OpenGL の理解を通して、分子グラフィックスの	目標とするプログラムが理解できた。	何を作成するべきかが理解できていない。

● 先週からのグループ演習についてのアンケートに答えて下さい。

1．gitBucketを用いた、情報交換・ソースコードの共有はできましたか？（いずれかに○）

順調であった。　　若干滞った　　問題が生じた　　利用できなかった。

（自由記述：問題点等を記述して下さい）

2．プログラムの開発環境は順調でしたか？（いずれかに○）

主な開発環境（複数回答可）：

開発環境は適切であった。　　まずますの開発環境であった。　　開発環境が利用できなかった。

（自由記述：問題点・改善点、要望等がありましたら記述して下さい。）

3．今週開発予定のプログラムについてお聞きします。

予定している開発環境（複数回答可、同上でもよい）

4．本日のグループ型の演習についての意見を、下記の語句を選択した後、下記に述べて下さい。

学習する上で有効　　有効性を感じない　　未だよく分からない

（自由記述）

その他、本グラフィックス演習に関する意見・質問、要望があれば記述して下さい。

ミニテスト（個人）：三週目

学生番号　　氏名

　次のキーワードに関して、今回の課題に関連して、各ＡＰＩの使用上のポイントをグループで話し合った後、最後に自分自身で今日の振り返りを行います。

○　コールバック関数
（キーボードが押されたとき）

（マウスのボタンがおされたとき）

（マウスによるドラッグ）

（アニメーションの実施方法）

○　ポリゴンの記述方法

○　作成予定の３ＤＣＧアニメーションの概略を下記に図示せよ。あくまで予定で良い。

演習問題: glut_sample5 の作成

glut_sample.c, glut_sample2.c, glut_sample3.c, glut_sample4.c を例しながら、 以下の機能をもつ glut_sample5 を作成しなさい。

1. gitBucket 上に、個人用のレポジトリ(glut-sample5)を作成せよ。
    1. まず、レポジトリ(glut-sample5)を作成せよ。
    2. 次に、クローンによりレポジトリをローカルに作成し、Moodle 上からダウンロードしてきた openGLBasic.tar.gz の内部にある glut_sample～glut_sample4 及び makefile を全て個人用のレポジトリに格納せよ。
    3. まず、これらのソースコードの登録(add/commit/push)を実施せよ。

【メモ】グループの中で、リーダーを中心にこの作業手順に関して確認しておこう。

2. 下記の機能の進捗状況報告（要件定義表（下記））を作成せよ。出来る限り、要件定義（機能）は分けた方がよい。

機能番号	機能（機能の略記）	ファイル名	関数名	予定時間	作成時間	完成度	備考（何をしたかを略記）
1	Makefile の作成	makefile	-	0:10	0:15	100%	glut_sample5 への対応
2	透視投影の実現	glut_sample5.c	reshape	0:10	0:10	100%	平行投影からの変更
3	animation モード：変数宣言	myGUI.h/glut_sample5.c	-	0:15	0:05	100%	外部変数の宣言 movable/velocity

3. Makefile の修正
    1. glut_sample4.c のファイルを glut_sample5.c としてコピ- せよ。
    2. glut_sample5.c から glut_sample5 が作成されるように、Makefile を修正しなさい。
       ※ この段階では、glut_sample5 は glut_sample4 と同じ動き方をする。
    3. その後、glut_my_mouse.c, glut_my_key.c, glut_sample5.c 等を変更し、下記の仕様を満たす glut_sample5 を作成せよ。
       ※その結果として、glut_sample4 に関しても、mouse/key の機能が変更されるものとする。

4. 画像表示のモードは次のものを満たすように変更する
    1. 透視投影モード
       ※ glut_sample/glut_sample2/glut_sample3 と同じ
       ※ glut_sample4 は平行投影モードなので注意が必要

5. 回転し続けるアニメーションの仕様変更
    1. 現在の仕様確認
       theta += 0.3*turn;
         0.3: 回転速度
         turn: 回転の向き
    2. 仕様変更
       theta += movable*turn*velocity;
         movable: 0 or 1
           1: 回転、0:停止　（キーボード s により制御、交互に変化）
         turn: -1 or +1
           +1/-1 （キーボード r により制御、交互に変化）
         velocity: 浮動小数点(float 型)
           浮動小数点として、正負も許可する。マウスにより、速度を制御

6. キーボードによる GUI の設計
    1. キーボードコールバック関数が受ける情報を fprintf を使って画面(console)に出せるようにする。
       押されたキー、押された時のマウスの位置が引数としたコールバック関数が呼ばれる。
    2. q または Q を押すことによって、プログラムが終了(exit(EXIT_SUCCESS)を呼び出す)する。
    3. r を押すことによって、回転している右回転、左回転が切り替わる。
       ※ glut_sample 等の左ボタンクリックと同じ機能である。マウスの機能の変更により、左ボタンクリックでは変更できなくなる。
       ※ このとき、回転速度は変化せず、向きだけが変化する。３で指定した turn の+1/-1 を変更する。
       ※ if を利用しないで+1/-1 を切り替える方法を考えてみよう。（発展課題）
    4. s を押すと、回転と停止を交互に繰り返す(トグルボタン)。
       ※ ３で指定した movable の 0/1 を変更する。
       ※ if を利用しないで、0/1 を切り替える方法を考えてみよう。（発展課題）

7. マウスによる回転の制御
    1. マウスコールバック関数が受ける情報を fprintf を使って画面(console)にだせるようにする。
       押されたマウスボタンの種類、そのときの状態、その時のマウスの位置が引数としたコールバック関数が呼び出される。
    2. マウスの左ボタンにより、回転速度(velocity)を変更する。
       ※ 元々の左ボタンの機能（回転の向き）はなくなる。
    3. 左ボタンを押した最初の位置ら現在のマウスの位置までの x 座標の変化量に比例して、
       回転速度(velocity)を変化させる。
       ※ 比例係数は各自、適切な量を設定せよ。(0.01～0.1 を想定している)
       ※ 押したところより、右側にずれた時には、velocity を正。
       ※ 押したところより、左側にずれた時には、velocity を負。
       ※ 回転するかしないかは、キーボード s によって制御される movable 変数により決定する。
       ※ 回転していれば、回転するモードとする。
       ※ 停止していた場合には、movable 変数を 1 として、回転する。
       ※ ただし、マウスを押した直後は、変化量が０のため、velocity=0 となり、停止したようにみえる。
       ※ その後、左右にマウスを動かすと回転を始める。
    4. 右ボタンの機能は、glut_sample3/glut_sample4 と同じ機能をもつ。
    5. SHIFT+左ドラッグの機能は、glut_sample3 と同じ機能（roll の機能）を実装する。（発展課題）
       ※ glut_sample4 では、機能を失っているので、glut_sample3 を参考にする事。
    6. 中ボタンを押して、上下に動かすと ZoomIn/Out を行うことができる。(発展課題)
       ※ glut_sample4 では、機能を失っているので、glut_sample3 を参考にする事。

8. ポリゴン画像の生成
    1. 適当なポリゴンを一枚以上加え,適当な絵を作成する。
    2. 現在のポリゴン（四角錐３つ）は、消しても良い。
    3. アニメーション機能は残し、idle 状態でも動き続ける画像とせよ。

提出物

1. 適切な画像のスナップショットをとって、課題提出（スナップショット）に提出して下さい。
2. ソースコードを、全て一つのファイルにアーカイブして、課題提出（ソースコード）に提出して下さい。
3. 要件定義書を作成して、課題提出（要件定義書）に提出して下さい。

資料1：Makefile

```
.SUFFIXES: .c .o .a .exe

LIBS=-lGL -lGLU -lglut -lm
#LIBS=-framework GLUT -framework OpenGL -lm
OBJS=glut_my_key.c glut_my_mouse.c

.c.o:
	gcc -g -Wall -c $*.c -o $*.o
	#gcc -g -Wall -DWIN32 -c $*.c -o $*.o -Wno-deprecated

#gcc -g -Wall -DWIN32 -c $*.c -o $*.o

.o.exe:
	gcc -g -Wall -o $*.exe $*.o  -lglut32 -lglu32 -lopengl32 -lm

all: glut_sample glut_sample2 glut_sample3 glut_sample4
win: glut_sample.exe glut_sample2.exe glut_sample3.exe glut_sample4.exe

glut_sample: glut_sample.o
	gcc -g -Wall -o $@ $@.o $(LIBS)

glut_sample2: glut_sample2.o
	gcc -g -Wall -o $@ $@.o $(LIBS)

glut_sample3: glut_sample3.o
	gcc -g -Wall -o $@ $@.o $(LIBS)

glut_sample4: glut_sample4.o libmyGUI.a
	gcc -g -Wall -o $@ $@.o $(LIBS) -L./ -lmyGUI

libmyGUI.a: $(OBJS:.c=.o)
	ar r $@ $(OBJS:.c=.o)

clean:
	rm -rf *.o  *.a *.exe glut_sample glut_sample2 glut_sample3 glut_sample4

include dependency

depend: $(SOURCE)
	gcc -M *.c > dependency
```

Makefileの動きを理解し、どう加えればよいかを下記に整理しておきましょう。

資料２：glut_sample.c

```
#include <stdio.h>
#include <stdlib.h>

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

#include <math.h>

// グローバル変数
// 同一ファイル内部の関数間で変数を持ち合う
static int    turn = 1;
static float  theta = 0.0;

//
//  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 )
{
        float  angle;

        // 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（モデル座標系）
        //（カメラが (0.0, 1.0, 5.0) の位置にあるとする）
    glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();               // 単位行列とする
        glTranslatef( 0.0, -1.0, -5.0 ); // モデルを移動させる行列操作

        // 地面を描画
        glBegin( GL_POLYGON ); // 多角形の描画開始：法線ベクトルに対して右回り
                glColor3f( 0.5, 0.8, 0.5 );  // 色の設定
                glNormal3f( 0.0, 1.0, 0.0 ); // 面の法線ベクトルの設定
                glVertex3f( 3.0, 0.0, 3.0 ); // 面の頂点の設定
                glVertex3f( 3.0, 0.0,-3.0 );
                glVertex3f(-3.0, 0.0,-3.0 );
                glVertex3f(-3.0, 0.0, 3.0 );
        glEnd();

        // 物体の回転角度を計算（360 度表現→ラジアンに変換）
        angle = M_PI * theta / 180.0;

        // 変換行列を設定（物体のモデル座標系）
        //（物体が (0.0, 1.0, 0.0) の位置にあり、Y 軸を中心に回転しているとする）
        // 移動
        glTranslatef( 0.0, 1.0, 0.0 );
        // 回転角、回転軸
        glRotatef( angle, 0.0f, 1.0f, 0.0f );

        // 物体（２枚のポリゴン）を描画：１枚の面の表側と裏側の設定
        glBegin( GL_TRIANGLES );
                glColor3f( 0.0, 0.0, 1.0 );  // 色の設定
                glNormal3f( 0.0, 0.0, 1.0 ); // 面の法線ベクトルの設定
                glVertex3f(-1.0, 1.0, 0.0 ); // 面の頂点の設定
```

```
                glVertex3f( 0.0,-1.0, 0.0 );
                glVertex3f( 1.0, 0.5, 0.0 );

                glColor3f( 1.0, 0.0, 0.0 );  // 色の設定
                glNormal3f( 0.0, 0.0,-1.0 ); // 面の法線ベクトルの設定
                glVertex3f(-1.0, 1.0, 0.0 ); // 面の頂点の設定
                glVertex3f( 1.0, 0.5, 0.0 );
                glVertex3f( 0.0,-1.0, 0.0 );
        glEnd();

        // 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();
}
//
//  ウィンドウサイズ変更時に呼ばれるコールバック関数
//
void  reshape( int w, int h )
{
        // ウィンドウ内の描画を行う範囲を設定（ここではウィンドウ全体に描画）
    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();
        // 視界角, アスペクト比, 視点からの距離(手前), 視点からの距離(奥)
        // 視錐台の設定
    gluPerspective(45, (double)w/h, 1, 500);
}
//
// 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 )
{
        // 左ボタンがクリックされたら回転方向を正にする
        if(( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN )) {
                turn = 1.0;
        // 右ボタンがクリックされたら回転方向を負にする
        } else if( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) ) {
                turn = -1.0;
        }

       // button state mx, my; mx, my はマウスがクリックされた位置
#undef DEBUG
#ifdef DEBUG
        fprintf(stderr, "");
#endif
}
//
//  アイドル時に呼ばれるコールバック関数
//  (なにもしないとき)
//
void  idle( void )
{
        // 物体を回転
        theta += 10 * turn;

        //  再描画の指示を出す（ この後で再描画のコールバック関数が呼ばれる： この場合は、
display()関数 ）
        glutPostRedisplay();
}
```

```
//
//  環境初期化関数
//
void  init_environment( void )
{
        // 光源を作成する
        float  light0_position[] = { 1.0, 1.0, 1.0, 1.0 }; // 光源位置
        float  light0_diffuse[]  = { 0.8, 0.8, 0.8, 1.0 }; // 拡散光
        float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 }; // 鏡面反射
        float  light0_ambient[]  = { 0.1, 0.1, 0.1, 1.0 }; // 環境光
        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 );                          // ライト 0: 点灯

        // 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 );
}

//
// キーボードが押されたときに呼び出されるコールバック関数
//
void
key(unsigned char key, int x, int y)
{

#undef DEBUG
#ifdef DEBUG
        fprintf(stderr, "What key ? ¥n");
#endif
        switch(key) {
                case 'q':
                        exit(EXIT_FAILURE);
                        break;
        }
}

//
//  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 | GLUT_DEPTH ); //ディスプレイモード
の設定: ダブルバッファ、RBGA(Alpha 値)
    glutInitWindowSize(400, 400 );  // ウィンドウの大きさ
```

```
    glutInitWindowPosition(100, 100 ); // ウィンドウの位置
    glutCreateWindow("New GLUT program"); // ウィンドウの生成

// コールバック関数の登録：イベントが生じたときに呼び出される関数を登録（引数は決まっている）
// 途中で変更も可能（例えば、マウスの利用の仕方が変更になる）
    glutDisplayFunc( display ); // ディスプレイの再描画
    glutReshapeFunc( reshape ); // ウィンドウの大きさが変更されたとき
    glutMouseFunc( mouse ); // マウスのボタンが押されたとき
    glutIdleFunc( idle ); // イベントが何もないとき
    glutKeyboardFunc( key ); // キーボードが押されたとき

// 環境初期化
    init_environment();

// GLUT のメインループに処理を移す
// イベントが生じたときに上記の関数を呼び出す
    glutMainLoop();
    return 0;
}
```

ポイント：下記の二つの点について、そのポイントをまとめて下さい。

① コールバック関数

② ポリゴンの生成

③　アニメーションを `idle` コールバック関数および `displey` コールバック関数によってどのように実現しているかを整理しておこう。

④　ｚバッファー、背面除去、照明の設定がどのようになっているかを確認しておこう。

⑤　投影方法に関して確認しておこう。

資料３：glut_sample2.c

```
//
//  角すいを描画ためのルーチン
//    三角形を組み合わせることで角錐を表現する
//
void  renderPyramid1()
{
        glBegin( GL_TRIANGLES );
                // +Z 方向の面
                glNormal3f( 0.0, 0.0, 1.0 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f(-1.0,-0.8, 1.0 );
                glVertex3f( 1.0,-0.8, 1.0 );

                // -Z 方向の面
                glNormal3f( 0.0, 0.0,-1.0 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f( 1.0,-0.8,-1.0 );
                glVertex3f(-1.0,-0.8,-1.0 );

                // +X 方向の面
                glNormal3f( 1.0, 0.0, 0.0 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f( 1.0,-0.8, 1.0 );
                glVertex3f( 1.0,-0.8,-1.0 );

                // -X 方向の面
                glNormal3f(-1.0, 0.0, 0.0 );
                glVertex3f( 0.0, 1.0, 0.0 );
                glVertex3f(-1.0,-0.8,-1.0 );
                glVertex3f(-1.0,-0.8, 1.0 );

                // 底面 1
                glNormal3f( 0.0,-1.0, 0.0 );
                glVertex3f( 1.0,-0.8, 1.0 );
                glVertex3f(-1.0,-0.8, 1.0 );
                glVertex3f( 1.0,-0.8,-1.0 );

                // 底面 1
                glNormal3f( 0.0,-1.0, 0.0 );
                glVertex3f(-1.0,-0.8,-1.0 );
                glVertex3f( 1.0,-0.8,-1.0 );
                glVertex3f(-1.0,-0.8, 1.0 );
        glEnd();
}
//
//  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）
        glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        // 変換行列を設定（モデル座標系→カメラ座標系）
        //（カメラが (0.0, -2.0, -12.0) の位置にあるとする）
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
        glTranslatef( 0.0, -2.0, -12.0 );

        // 地面を描画
```

```
        glBegin( GL_POLYGON );
                glColor3f( 0.5, 0.8, 0.5 );
                glNormal3f( 0.0, 1.0, 0.0 );
                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();

        //
        // glPushMatrix/glPopMatrix
        //   Push:ポリゴンを描きだす座標変換行列をスタックにプッシュする
        //   Pop: スタックから取り出す
        // 中央の角すいを描画
        glPushMatrix();
                glTranslatef( 0.0, 1.0, 0.0 );
                glRotatef( theta, 0.0f, 1.0f, 0.0f );
                glColor3f( 1.0, 0.0, 0.0 );
                renderPyramid1();
        glPopMatrix(); // glTranslate/glRotate を実施しなかった状態に戻す

        // 右の角すいを描画
        glPushMatrix();
                glTranslatef( 3.0, 1.0, 0.0 );
                glRotatef( theta, 1.0f, 0.0f, 0.0f );
                glColor3f( 0.0, 1.0, 0.0 );
                renderPyramid1();
        glPopMatrix();

        // 左の角すいを描画
        glPushMatrix();
                glTranslatef(-3.0, 1.0, 0.0 );
                glRotatef( theta, 0.0f, 0.0f, 1.0f );
                glColor3f( 0.0, 0.0, 1.0 );
                renderPyramid1();
        glPopMatrix();

        // 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();
}
```

グループ課題：ポイント： Push/Pop を用いた座標表現
※　何が行われているかを、下記に説明してみて下さい。もし、この Push/Pop を全て取り払うとどのような動きを生じると思われますか。

資料４：glut_sample3.c

```
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("GLUT sample program");

        // 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;
}

// 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 )
{
        // 右ボタンがクリックされたらオブジェクトの回転方向を反転する
        if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
                turn *= -1.0;

        // 右ボタンが押されたらドラッグ開始
        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
        else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
                drag_mouse_r = 0;

        // 中ボタンが押されたらドラッグ開始
        if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
                drag_mouse_m = 1;
        // 中ボタンが離されたらドラッグ終了
        else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
                drag_mouse_m = 0;

        // シフトキーを押しながら、左ボタンが押されたらドラッグ開始
        if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN )
                && (glutGetModifiers()&GLUT_ACTIVE_SHIFT ))
                drag_mouse_l = 1;
        // シフトキーを押しながら、左ボタンが離されたらドラッグ終了
        else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP )
                && (glutGetModifiers()&GLUT_ACTIVE_SHIFT ) )
                drag_mouse_l = 0;
        // 現在のマウス座標を記録
        last_mouse_x = mx;
        last_mouse_y = my;
}
//
```

```
// マウスドラッグ時に呼ばれるコールバック関数
//
void  motion( int mx, int my )
{
        // 右ボタンのドラッグ中は視点を回転する
        if ( drag_mouse_r )
        {
                // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転

                // マウスの横移動に応じてＹ軸を中心に回転
                camera_yaw -= ( mx - last_mouse_x ) * 1.0;
                if ( camera_yaw < 0.0 )
                        camera_yaw += 360.0;
                else if ( camera_yaw > 360.0 )
                        camera_yaw -= 360.0;

                // 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 > 90.0 )
                        camera_pitch = 90.0;

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

        // シフトキーを押しながら、左ボタンドラッグ
        if ( drag_mouse_l )
        {
                // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転

                // マウスの横移動に応じてＹ軸を中心に回転
                camera_roll -= ( mx - last_mouse_x ) * 1.0;
                if ( camera_roll < 0.0 )
                        camera_roll += 360.0;
                else if ( camera_roll > 360.0 )
                        camera_roll -= 360.0;

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

        // 中ボタン
        if ( drag_mouse_m )
        {
                // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転

                // マウスの横移動に応じてＺ軸を中心に回転
                camera_distance -= ( my - last_mouse_y ) * 0.05;

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

ドラッグをどのように実現しているかを整理しておきましょう。

資料 5：glut_sample4

>>>>>--- myGUI.h ---<<<<<

```
#ifndef MY_GUI_H
#define MY_GUI_H


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


// 物体の回転のための変数
extern int    turn;
extern float  theta;

// カメラの回転のための変数
extern float  camera_yaw; // Ｙ軸を中心とする回転角度
extern float  camera_pitch; // Ｘ軸を中心とする回転角度
extern float  camera_roll; // Ｘ軸を中心とする回転角度
extern float  camera_distance; // 中心からカメラの距離

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

// ウィンドウのサイズ
extern int    win_width;
extern int    win_height;

// glut_my_mouse.c
extern void  motion( int mx, int my );
extern void  mouse( int button, int state, int mx, int my );

// glut_my_key.c
extern void key(unsigned char c, int x, int y);

#endif /* MY_GUI_H */
```

>>>>>--- glut_sample4.c ---<<<<<

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

// 自分たちが定義した OpenGL のための関数の呼び出し
#include "myGUI.h"

// 物体の回転のための変数
int    turn = 1;
float  theta = 0.0;
```

```
// カメラの回転のための変数
float   camera_yaw     = -45.0; // Y軸を中心とする回転角度
float   camera_pitch   = -30.0; // X軸を中心とする回転角度
float   camera_roll    =   0.0; // X軸を中心とする回転角度
float   camera_distance =  15.0; // 中心からカメラの距離

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

// ウィンドウのサイズ
int     win_width, win_height;

>>>>>--- glut_my_key.c ---

#include <stdio.h>
#include <stdlib.h>
#include "myGUI.h"
//
// Keyboard
//
void key(unsigned char c, int x, int y)
{
        switch(c) {
                case 'q': {
                        exit(EXIT_FAILURE);
                        break;
                }
        }
}



                        >>>>>--- glut_my_mouse.c ---<<<<<

#include "myGUI.h"

//
// 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 )
{
    // 右ボタンがクリックされたらオブジェクトの回転方向を反転する
    if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
        turn *= -1.0;

    // 右ボタンが押されたらドラッグ開始
    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
    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;
}

//
// マウスドラッグ時に呼ばれるコールバック関数
//
void  motion( int mx, int my )
{
    // 右ボタンのドラッグ中は視点を回転する
    if ( drag_mouse_r )
    {
        // 前回のマウス座標と今回のマウス座標の差に応じて視点を回転

        // マウスの横移動に応じてＹ軸を中心に回転
        camera_yaw -= ( mx - last_mouse_x ) * 1.0;
        if ( camera_yaw < 0.0 )
            camera_yaw += 360.0;
        else if ( camera_yaw > 360.0 )
            camera_yaw -= 360.0;

        // 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 > 90.0 )
            camera_pitch = 90.0;

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

●複数ファイルにわかれているものが Makefile を通してどのようにリンクされるか。

●複数ファイルの間で変数のやり取りを行う方法について気にしておきましょう。