```
//
//  コンピュータグラフィックス特論II
//  視点操作のサンプルプログラム
//

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


// グローバル変数

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

// 背景オブジェクトの位置
const int  num_trees = 100;
float  tree_pos[ num_trees ][2];

// マウスのドラッグのための変数
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  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  last_mouse_x, last_mouse_y; // 最後に記録されたマウスカーソルの座標


// 視点操作パラメタ
float  view_center_x;  // 注視点の位置
float  view_center_y;  // 注視点の位置
float  view_center_z;  // 注視点の位置
float  view_yaw;       // 視点の方位角
float  view_pitch;   // 視点の仰角
float  view_distance;  // 視点と注視点の距離


// 視点操作モード
enum  ViewControlModeEnum
{
    VIEW_DOLLY_PARAM,         // Dollyモード (媒介変数)
    VIEW_DOLLY_DIRECT,       // Dollyモード (直接更新)
    VIEW_SCROLL_PARAM,       // Scrollモード (媒介変数)
    VIEW_SCROLL_DIRECT,     // Scrollモード (直接更新)
    VIEW_WALKTHROUGH_PARAM,   // Walkthroughモード (媒介変数)
    VIEW_WALKTHROUGH_DIRECT,  // Walkthroughモード (直接更新)
    NUM_VIEW_CONTROL_MODES  // 視点操作モードの種類数
};

// 視点操作モードの名前
const char *  mode_name[] = { "Dolly Mode (Parameter)", "Dolly Mode (Direct)", "Scroll Mode (Parameter)", "Scroll Mode (Direct)", "Walkthrough Mod
e (Parameter)", "Walkthrough Mode (Direct)" };

// 現在の視点操作モード
ViewControlModeEnum  mode = VIEW_DOLLY_PARAM;



//
//  視点操作のための処理
//


//
//  視点の初期化
//  (最初の初期化時と視点モードが切り替えられたときに呼ばれる)
//
void  InitView()
{
    // 視点パラメタを初期化
    if ( mode == VIEW_DOLLY_PARAM )
    {
        view_center_x = 0.0f;
        view_center_y = 0.0f;
        view_center_z = 0.0f;
        view_yaw = -30.0f;
        view_pitch = -30.0f;
        view_distance = 15.0f;
    }
    if ( mode == VIEW_SCROLL_PARAM )
    {
        view_center_x = 0.0f;
        view_center_y = 0.0f;
        view_center_z = 0.0f;
        view_yaw = 0.0f;
        view_pitch = -30.0f;
        view_distance = 15.0f;
    }
    if ( mode == VIEW_WALKTHROUGH_PARAM )
    {
        view_center_x = 0.0f;
        view_center_y = 0.5f;
        view_center_z = 0.0f;
        view_yaw = 0.0f;
        view_pitch = 0.0f;
        view_distance = 0.0f;
    }

    // 変換行列を初期化
    if ( mode == VIEW_DOLLY_DIRECT )
    {
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
        glTranslatef( 0.0, 0.0, -15.0 );
        glRotatef( 30.0, 1.0, 0.0, 0.0 );
        glRotatef( 30.0, 0.0, 1.0, 0.0 );

        view_center_x = 0.0f;
        view_center_y = 0.0f;
        view_center_z = 0.0f;
    }
    if ( mode == VIEW_SCROLL_DIRECT )
    {
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity();
        glTranslatef( 0.0, 0.0, -15.0 );
```

```
112         glRotatef( 30.0, 1.0, 0.0, 0.0 );
113         glRotatef( 0.0, 0.0, 1.0, 0.0 );
114
115         view_center_x = 0.0f;
116         view_center_y = 0.0f;
117         view_center_z = 0.0f;
118     }
119     if ( mode == VIEW_WALKTHROUGH_DIRECT )
120     {
121         glMatrixMode( GL_MODELVIEW );
122         glLoadIdentity();
123         glTranslatef( 0.0, -0.5, 0.0 );
124         glRotatef( 0.0, 1.0, 0.0, 0.0 );
125         glRotatef( 0.0, 0.0, 1.0, 0.0 );
126     }
127 }
128
129
130 //
131 //  視点パラメタに応じて変換行列（カメラ座標系からワールド座標系への変換行列）を更新
132 //  (画面描画時のコールバック関数 DisplayCallback() から呼ばれる)
133 //
 H3 void  UpdateViewMatrix()
135 {
136     // 視点パラメタを使った操作時のみ変換行列を更新
137     if ( ( mode == VIEW_DOLLY_PARAM ) || ( mode == VIEW_SCROLL_PARAM ) || ( mode == VIEW_WALKTHROUGH_PARAM ) )
138     {
139         glMatrixMode( GL_MODELVIEW );
140         glLoadIdentity();
141         glTranslatef( 0.0,  0.0, - view_distance );
142         glRotatef( - view_pitch, 1.0, 0.0, 0.0 );
143         glRotatef( - view_yaw, 0.0, 1.0, 0.0 );
144         glTranslatef( - view_center_x, - view_center_y, - view_center_z );
145     }
146 }
147
148
149 //
150 //  マウス操作に応じて視点パラメタ or 変換行列を更新
151 //  (マウスドラッグ時のコールバック関数 MouseDragCallback() から呼ばれる)
152 //
 H3 void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int delta_mouse_left_x, int delta_mouse_left_y )
154 {
155     // 視点パラメタを更新 (Dollyモード・媒介変数)
156     if ( mode == VIEW_DOLLY_PARAM )
157     {
158         // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
159         if ( delta_mouse_right_x != 0 )
160         {
161             view_yaw -= delta_mouse_right_x * 1.0;
162
163             // パラメタの値が所定の範囲を超えないように修正
164             if ( view_yaw < 0.0 )
165                 view_yaw += 360.0;
166             else if ( view_yaw > 360.0 )
167                 view_yaw -= 360.0;
168         }
169
170         // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
171         if ( delta_mouse_right_y != 0 )
172         {
173             view_pitch -= delta_mouse_right_y * 1.0;
174
175             // パラメタの値が所定の範囲を超えないように修正
176             if ( view_pitch < -90.0 )
177                 view_pitch = -90.0;
178             else if ( view_pitch > -2.0 )
179                 view_pitch = -2.0;
180         }
181
182         // 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
183         if ( delta_mouse_left_y != 0 )
184         {
185             view_distance += delta_mouse_left_y * 0.2;
186
187             // パラメタの値が所定の範囲を超えないように修正
188             if ( view_distance < 5.0 )
189                 view_distance = 5.0;
190         }
191     }
192
193     // 視点パラメタを更新 (Scrollモード・媒介変数)
194     if ( mode == VIEW_SCROLL_PARAM )
195     {
196         // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
197         if ( delta_mouse_right_y != 0 )
198         {
199             // ※レポート課題
200         }
201
202         // 左ボタンドラッグに応じて、視点を前後左右に移動 (ワールド座標系を基準とした前後左右)
203         if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
204         {
205             // ※レポート課題
206         }
207     }
208
209     // 視点パラメタを更新 (Walkthroughモード・媒介変数)
210     if ( mode == VIEW_WALKTHROUGH_PARAM )
211     {
212         // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
213         if ( delta_mouse_right_x != 0 )
214         {
215             // ※レポート課題
216         }
217
218         // 左ボタンドラッグに応じて、視点を前後左右に移動 (カメラの向きを基準とした前後左右)
219         if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
220         {
221             // ※レポート課題
222         }
223     }
```

```
224
225     // 変換行列を更新 (Dollyモード・直接更新)
226     if ( mode == VIEW_DOLLY_DIRECT )
227     {
228         // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
229         if ( delta_mouse_right_x != 0 )
230         {
231             // 視点の水平方向の回転量を計算
232             float  delta_yaw = delta_mouse_right_x * 1.0;
233
234             // 現在の変換行列の右側に、今回の回転変換をかける
235             glMatrixMode( GL_MODELVIEW );
236             glRotatef( delta_yaw, 0.0, 1.0, 0.0 );
237         }
238
239         // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
240         if ( delta_mouse_right_y != 0 )
241         {
242             // 視点の上下方向の回転量を計算
243             float  delta_pitch = delta_mouse_right_y * 1.0;
244
245             // 現在の変換行列を取得
246             float  m[ 16 ];
247             float  tx, ty, tz;
248             glGetFloatv( GL_MODELVIEW_MATRIX, m );
249
250             // 現在の変換行列の平行移動成分を記録
251             tx = m[ 12 ];
252             ty = m[ 13 ];
253             tz = m[ 14 ];
254
255             // 現在の変換行列の平行移動成分を０にする
256             m[ 12 ] = 0.0f;
257             m[ 13 ] = 0.0f;
258             m[ 14 ] = 0.0f;
259
260             // 変換行列を初期化
261             glMatrixMode( GL_MODELVIEW );
262             glLoadIdentity();
263
264             // カメラの平行移動行列を設定
265             glTranslatef( tx, ty, tz );
266
267             // 右側に、今回の回転変換をかける
268             glRotatef( delta_pitch, 1.0, 0.0, 0.0 );
269
270             // さらに、右側に、もとの変換行列から平行移動成分をとり除いたものをかける
271             glMultMatrixf( m );
272         }
273
274         // 縦方向の左ボタンドラッグに応じて、視点と注視点の距離を変更
275         if ( delta_mouse_left_y )
276         {
277             // 視点と注視点の距離の変化量を計算
278             float  delta_dist = delta_mouse_left_y * 1.0;
279
280             // 現在の変換行列（カメラの向き）を取得
281             float  m[ 16 ];
282             glGetFloatv( GL_MODELVIEW_MATRIX, m );
283
284             // 変換行列を初期化して、カメラ移動分の平行移動行列を設定
285             glMatrixMode( GL_MODELVIEW );
286             glLoadIdentity();
287             glTranslatef( 0.0,  0.0, - delta_dist );
288
289             // 右からこれまでの変換行列をかける
290             glMultMatrixf( m );
291         }
292     }
293
294     // 視点パラメタを更新 (Scrollモード・直接更新)
295     if ( mode == VIEW_SCROLL_DIRECT )
296     {
297         // 縦方向の右ボタンドラッグに応じて、視点を上下方向に回転
298         if ( delta_mouse_right_y != 0 )
299         {
300             // ※レポート課題
301         }
302
303         // 左ボタンドラッグに応じて、視点を前後左右に移動 (ワールド座標系を基準として前後左右に移動)
304         if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
305         {
306             // ※レポート課題
307         }
308     }
309
310     // 変換行列を更新 (Walkthroughモード・直接更新)
311     if ( mode == VIEW_WALKTHROUGH_DIRECT )
312     {
313         // 横方向の右ボタンドラッグに応じて、視点を水平方向に回転
314         if ( delta_mouse_right_x != 0 )
315         {
316             // ※レポート課題
317         }
318
319         // 左ボタンドラッグに応じて、視点を前後左右に移動 (カメラの向きを基準として前後左右に移動)
320         if ( ( delta_mouse_left_x != 0 ) || ( delta_mouse_left_y != 0 ) )
321         {
322             // ※レポート課題
323         }
324     }
325 }
326
327
328
329 //
330 //  以下、プログラムのメイン処理
331 //
332
333
334 //
335 //  木を描画
```

```
336 //
 H3 void  RenderTree()
338 {
339     static GLUquadricObj *  quad_obj = NULL;
340     if ( quad_obj == NULL )
341         quad_obj = gluNewQuadric();
342 
343     glPushMatrix();
344         glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
345         glColor3f( 0.8, 0.7, 0.0 );
346         gluCylinder( quad_obj, 0.25f, 0.25f, 1.0f, 16, 1 );
347     glPopMatrix();
348 
349     glPushMatrix();
350         glTranslatef( 0.0f, 0.5f, 0.0f );
351         glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
352         glColor3f( 0.3, 0.7, 0.3 );
353         gluCylinder( quad_obj, 0.5f, 0.0f, 1.0f, 16, 1 );
354     glPopMatrix();
355 
356     glPushMatrix();
357         glTranslatef( 0.0f, 1.0f, 0.0f );
358         glRotatef( -90.0f, 1.0f, 0.0f, 0.0f );
359         glColor3f( 0.3, 0.7, 0.3 );
360         gluCylinder( quad_obj, 0.5f, 0.0f, 1.0f, 16, 1 );
361     glPopMatrix();
362 }
363 
364 
365 //
366 //  格子模様の床を描画
367 //
 H3 void  DrawFloor( int tile_size, int num_x, int num_z, float r0, float g0, float b0, float r1, float g1, float b1 )
369 {
370     int  x, z;
371     float  ox, oz;
372 
373     glBegin( GL_QUADS );
374     glNormal3d( 0.0, 1.0, 0.0 );
375 
376     ox = - ( num_x * tile_size ) / 2;
377     for ( x=0; x<num_x; x++ )
378     {
379         oz = - ( num_z * tile_size ) / 2;
380         for ( z=0; z<num_z; z++ )
381         {
382             if ( ( (x + z) % 2 ) == 0 )
383                 glColor3f( r0, g0, b0 );
384             else
385                 glColor3f( r1, g1, b1 );
386 
387             glTexCoord2d( 0.0f, 0.0f );
388             glVertex3d( ox, 0.0, oz );
389             glTexCoord2d( 0.0f, 1.0f );
390             glVertex3d( ox, 0.0, oz + tile_size );
391             glTexCoord2d( 1.0f, 1.0f );
392             glVertex3d( ox + tile_size, 0.0, oz + tile_size );
393             glTexCoord2d( 1.0f, 0.0f );
394             glVertex3d( ox + tile_size, 0.0, oz );
395 
396             oz += tile_size;
397         }
398         ox += tile_size;
399     }
400     glEnd();
401 }
402 
403 
404 //
405 //  文字情報（現在のモード名）を描画
406 //
 H3 void  DrawTextInformation()
408 {
409     // 表示するメッセージ
410     int   i;
411     const char *  message = mode_name[ mode ];
412 
413     // 射影行列を初期化（初期化の前に現在の行列を退避）
414     glMatrixMode( GL_PROJECTION );
415     glPushMatrix();
416     glLoadIdentity();
417     gluOrtho2D( 0.0, win_width, win_height, 0.0 );
418 
419     // モデルビュー行列を初期化（初期化の前に現在の行列を退避）
420     glMatrixMode( GL_MODELVIEW );
421     glPushMatrix();
422     glLoadIdentity();
423 
424     // Ｚバッファ・ライティングはオフにする
425     glDisable( GL_DEPTH_TEST );
426     glDisable( GL_LIGHTING );
427 
428     // メッセージの描画
429     glColor3f( 1.0, 0.0, 0.0 );
430     glRasterPos2i( 16, 16 + 18 );
431     for ( i=0; message[i]!='¥0'; i++ )
432         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
433 
434     // 設定を全て復元
435     glEnable( GL_DEPTH_TEST );
436     glEnable( GL_LIGHTING );
437     glMatrixMode( GL_PROJECTION );
438     glPopMatrix();
439     glMatrixMode( GL_MODELVIEW );
440     glPopMatrix();
441 }
442 
443 
444 //
445 //  画面描画時に呼ばれるコールバック関数
446 //
 H3 void  DisplayCallback()
```

```
{
    // 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 );

    // 視点パラメタに応じて変換行列（カメラ座標系からワールド座標系への変換行列）を更新
    UpdateViewMatrix();

    // 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 );

    // 格子模様の床を描画
    DrawFloor( 1.0f, 50, 50, 1.0, 1.0, 1.0, 0.8, 0.8, 0.8 );

    // 背景の木を描画
    int  i;
    for ( i=0; i<num_trees; i++ )
    {
        glPushMatrix();
        glTranslatef( tree_pos[ i ][ 0 ], 0.0f, tree_pos[ i ][ 1 ] );
        RenderTree();
        glPopMatrix();
    }

    // 注視点にオブジェクト（球）を描画
    if ( ( mode != VIEW_WALKTHROUGH_PARAM ) && ( mode != VIEW_WALKTHROUGH_DIRECT ) )
    {
        glPushMatrix();
        glTranslatef( view_center_x, view_center_y + 0.5f, view_center_z );
        glColor3f( 1.0, 0.0, 0.0 );
        glutSolidSphere( 0.5f, 24, 12 );
        glPopMatrix();
    }

    // 文字情報（現在のモード名）を描画
    DrawTextInformation();

    // 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  ReshapeCallback( 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();
    gluPerspective( 45, (double)w/h, 1, 500 );

    // ウィンドウのサイズを記録（テキスト描画処理のため）
    win_width = w;
    win_height = h;
}


//
// マウスクリック時に呼ばれるコールバック関数
//
void  MouseClickCallback( int button, int state, int mx, int my )
{
    // 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;

    // 左ボタンが押されたらドラッグ開始
    if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
        drag_mouse_l = 1;
    // 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 ) )
        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  MouseDragCallback( int mx, int my )
{
    // マウスのドラッグ距離を計算
    int  delta_mouse_right_x = 0, delta_mouse_right_y = 0, delta_mouse_left_x = 0, delta_mouse_left_y = 0;
    if ( drag_mouse_r )
    {
        delta_mouse_right_x = mx - last_mouse_x;
        delta_mouse_right_y = my - last_mouse_y;
    }
    if ( drag_mouse_l )
    {
        delta_mouse_left_x = mx - last_mouse_x;
        delta_mouse_left_y = my - last_mouse_y;
    }

    // マウス操作に応じて視点パラメタ or 変換行列を更新
    UpdateView( delta_mouse_right_x, delta_mouse_right_y, delta_mouse_left_x, delta_mouse_left_y );

    // 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（この後で再描画のコールバック関数が呼ばれる）
```

```
560     glutPostRedisplay();
561 }
562 
563 
564 //
565 //  キーボードのキーが押されたときに呼ばれるコールバック関数
566 //
H3  void  KeyboardCallback( unsigned char key, int mx, int my )
568 {
569     // Mキーで視点操作モードを順番に切り替え
570     if ( key == 'm' )
571     {
572         // 次の視点操作モードに切り替え
573         mode = (ViewControlModeEnum)( ( mode + 1 ) % NUM_VIEW_CONTROL_MODES );
574 
575         // 視点の初期化
576         InitView();
577     }
578 
579     // 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）
580     glutPostRedisplay();
581 }
582 
583 
584 //
585 //  環境初期化関数
586 //
H3  void  InitEnvironment()
588 {
589     // 光源を作成する
590     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
591     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
592     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
593     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
594     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
595     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
596     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
597     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
598     glEnable( GL_LIGHT0 );
599 
600     // 光源計算を有効にする
601     glEnable( GL_LIGHTING );
602 
603     // 物体の色情報を有効にする
604     glEnable( GL_COLOR_MATERIAL );
605 
606     // Ｚテストを有効にする
607     glEnable( GL_DEPTH_TEST );
608 
609     // 背面除去を有効にする
610     glCullFace( GL_BACK );
611     glEnable( GL_CULL_FACE );
612 
613     // 背景色を設定
614     glClearColor( 0.5, 0.5, 0.8, 0.0 );
615 
616     // 背景に配置するツリーの位置をランダムに初期化
617     for ( int i=0; i<num_trees; i++ )
618     {
619         for ( int j=0; j<2; j++ )
620             tree_pos[ i ][ j ] = ( rand() % 1000 - 500 ) * 0.04f;
621     }
622 }
623 
624 
625 //
626 //  メイン関数（プログラムはここから開始）
627 //
H3  int  main( int argc, char ** argv )
629 {
630     // GLUTの初期化
631     glutInit( &argc, argv );
632     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA );
633     glutInitWindowSize( 640, 640 );
634     glutInitWindowPosition( 0, 0 );
635     glutCreateWindow("View Control");
636 
637     // コールバック関数の登録
638     glutDisplayFunc( DisplayCallback );
639     glutReshapeFunc( ReshapeCallback );
640     glutMouseFunc( MouseClickCallback );
641     glutMotionFunc( MouseDragCallback );
642     glutKeyboardFunc( KeyboardCallback );
643 
644     // 環境初期化
645     InitEnvironment();
646 
647     // 視点の初期化
648     InitView();
649 
650     // GLUTのメインループに処理を移す
651     glutMainLoop();
652     return 0;
653 }
654 
655 
```