コンピュータアニメーション特論 プログラミング演習資料
第2回 視点操作

九州工業大学 情報工学研究院　尾下 真樹

1　サンプルプログラム

視点操作のサンプルプログラム（view_sample.cpp）をもとに、3 通りの視点操作方法（Dolly、Scroll、Walk-through）を、2 通りの方法（媒介変数を使用する方法、変換行列を直接更新する方法）で実現するプログラムを作成する。キーボードの m キーを押すことで、3 通りの視点操作方法 ×2 通りの実現方法の計 6 通りのモードを順番に切り替えることができる。また、マウスの右ボタン・左ボタンを押しながらドラッグすることで、各視点操作モードに応じて視点を変更できる。

最初に、サンプルプログラムのソースコード全体を示す。その後、サンプルプログラムの主要な処理を説明する。

ソースコード 1: view_sample.cpp

```
1  //
2  //  コンピュータアニメーション特論
3  //  視点操作のサンプルプログラム
4  //
5
6  // GLUTヘッダファイルのインクルード
7  #include <GL/glut.h>
8
9
10 // グローバル変数
11
12 // ウィンドウのサイズ
13 int  win_width, win_height;
14
15 // 背景オブジェクトの位置
16 const int  num_trees = 100;
17 float  tree_pos[ num_trees ][2];
18
19 // マウスのドラッグのための変数
20 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）
21 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）
22 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
23
24
25 // 視点操作パラメタ
26 float  view_center_x;  // 注視点の位置
27 float  view_center_y;  // 注視点の位置
28 float  view_center_z;  // 注視点の位置
29 float  view_yaw;      // 視点の方位角
30 float  view_pitch;    // 視点の仰角
31 float  view_distance;  // 視点と注視点の距離
32
33
34 // 視点操作モード
35 enum  ViewControlModeEnum
```

```
36 {
37     VIEW_DOLLY_PARAM,       // Dollyモード（媒介変数）
38     VIEW_DOLLY_DIRECT,      // Dollyモード（直接更新）
39     VIEW_SCROLL_PARAM,      // Scrollモード（媒介変数）
40     VIEW_SCROLL_DIRECT,     // Scrollモード（直接更新）
41     VIEW_WALKTHROUGH_PARAM,   // Walkthroughモード（媒介変数）
42     VIEW_WALKTHROUGH_DIRECT,  // Walkthroughモード（直接更新）
43     NUM_VIEW_CONTROL_MODES  // 視点操作モードの種類数
44 };
45
46 // 視点操作モードの名前
47 const char *   mode_name[] = { "Dolly Mode (Parameter)", "Dolly Mode (Direct)", "Scroll
       Mode (Parameter)", "Scroll Mode (Direct)", "Walkthrough Mode (Parameter)", "
      Walkthrough Mode (Direct)" };
48
49 // 現在の視点操作モード
50 ViewControlModeEnum   mode = VIEW_DOLLY_PARAM;
51
52
53
54 //
55 //  視点操作のための処理
56 //
57
58
59 //
60 //  視点の初期化
61 //  （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）
62 //
63 void  InitView()
64 {
65     // 視点パラメタを初期化
66     if ( mode == VIEW_DOLLY_PARAM )
67     {
68         view_center_x = 0.0f;
69         view_center_y = 0.0f;
70         view_center_z = 0.0f;
71         view_yaw = -30.0f;
72         view_pitch = -30.0f;
73         view_distance = 15.0f;
74     }
75     if ( mode == VIEW_SCROLL_PARAM )
76     {
77         view_center_x = 0.0f;
78         view_center_y = 0.0f;
79         view_center_z = 0.0f;
80         view_yaw = 0.0f;
81         view_pitch = -30.0f;
82         view_distance = 15.0f;
83     }
84     if ( mode == VIEW_WALKTHROUGH_PARAM )
85     {
86         view_center_x = 0.0f;
87         view_center_y = 0.5f;
88         view_center_z = 0.0f;
89         view_yaw = 0.0f;
90         view_pitch = 0.0f;
91         view_distance = 0.0f;
92     }
93
94     // 変換行列を初期化
95     if ( mode == VIEW_DOLLY_DIRECT )
96     {
97         glMatrixMode( GL_MODELVIEW );
```

```
98          glLoadIdentity();
99          glTranslatef( 0.0, 0.0, -15.0 );
100         glRotatef( 30.0, 1.0, 0.0, 0.0 );
101         glRotatef( 30.0, 0.0, 1.0, 0.0 );
102
103         view_center_x = 0.0f;
104         view_center_y = 0.0f;
105         view_center_z = 0.0f;
106     }
107     if ( mode == VIEW_SCROLL_DIRECT )
108     {
109         glMatrixMode( GL_MODELVIEW );
110         glLoadIdentity();
111         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 //
134 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 //
153 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 //
337 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 //
368 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 //
407 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 //
447 void  DisplayCallback()
448 {
449     // 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）
450     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
451
452     // 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
453     UpdateViewMatrix();
454
455     // 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）
456     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
457     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
458
459     // 格子模様の床を描画
460     DrawFloor( 1.0f, 50, 50, 1.0, 1.0, 1.0, 0.8, 0.8, 0.8 );
461
462     // 背景の木を描画
463     int  i;
464     for ( i=0; i<num_trees; i++ )
465     {
466         glPushMatrix();
467         glTranslatef( tree_pos[ i ][ 0 ], 0.0f, tree_pos[ i ][ 1 ] );
468         RenderTree();
469         glPopMatrix();
470     }
471
472     // 注視点にオブジェクト（球）を描画
473     if ( ( mode != VIEW_WALKTHROUGH_PARAM ) && ( mode != VIEW_WALKTHROUGH_DIRECT ) )
474     {
```

```
475         glPushMatrix();
476         glTranslatef( view_center_x, view_center_y + 0.5f, view_center_z );
477         glColor3f( 1.0, 0.0, 0.0 );
478         glutSolidSphere( 0.5f, 24, 12 );
479         glPopMatrix();
480     }
481
482     // 文字情報（現在のモード名）を描画
483     DrawTextInformation();
484
485     // 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
486     glutSwapBuffers();
487 }
488
489
490 //
491 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
492 //
493 void  ReshapeCallback( int w, int h )
494 {
495     // 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）
496     glViewport(0, 0, w, h);
497
498     // カメラ座標系→スクリーン座標系への変換行列を設定
499     glMatrixMode( GL_PROJECTION );
500     glLoadIdentity();
501     gluPerspective( 45, (double)w/h, 1, 500 );
502
503     // ウィンドウのサイズを記録（テキスト描画処理のため）
504     win_width = w;
505     win_height = h;
506 }
507
508
509 //
510 // マウスクリック時に呼ばれるコールバック関数
511 //
512 void  MouseClickCallback( int button, int state, int mx, int my )
513 {
514     // 右ボタンが押されたらドラッグ開始
515     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
516         drag_mouse_r = 1;
517     // 右ボタンが離されたらドラッグ終了
518     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
519         drag_mouse_r = 0;
520
521     // 左ボタンが押されたらドラッグ開始
522     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
523         drag_mouse_l = 1;
524     // 左ボタンが離されたらドラッグ終了
525     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
526         drag_mouse_l = 0;
527
528     // 現在のマウス座標を記録
529     last_mouse_x = mx;
530     last_mouse_y = my;
531 }
532
533
534 //
535 // マウスドラッグ時に呼ばれるコールバック関数
536 //
537 void  MouseDragCallback( int mx, int my )
538 {
```

```
539     // マウスのドラッグ距離を計算
540     int  delta_mouse_right_x = 0, delta_mouse_right_y = 0, delta_mouse_left_x = 0,
            delta_mouse_left_y = 0;
541     if ( drag_mouse_r )
542     {
543         delta_mouse_right_x = mx - last_mouse_x;
544         delta_mouse_right_y = my - last_mouse_y;
545     }
546     if ( drag_mouse_l )
547     {
548         delta_mouse_left_x = mx - last_mouse_x;
549         delta_mouse_left_y = my - last_mouse_y;
550     }
551
552     // マウス操作に応じて視点パラメタ or 変換行列を更新
553     UpdateView( delta_mouse_right_x, delta_mouse_right_y, delta_mouse_left_x,
            delta_mouse_left_y );
554
555     // 今回のマウス座標を記録
556     last_mouse_x = mx;
557     last_mouse_y = my;
558
559     // 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 //
567 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 //
587 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 //
628 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 }
```

1.1 視点パラメタの定義

サンプルプログラム（view_sample.cpp）の 25~50 行で、グローバル変数として、視点パラメタ（媒介変数）を定義している。最初の 3 つの変数（view_center_x, view_center_y, view_center_z）は、画面の中央に来る注視点の位置を表す。次の 2 つの変数（view_yaw, view_pitch）は、視点の方位角・仰角を表す。最後の変数（view_distance）は、視点と注視点の距離を表す。

また、視点操作モードを表す列挙型と、各視点操作モードの名称を表示するための文字列、その列挙型を使った視点操作モードを表す変数を、グローバル変数として定義している。

1.2 コールバック関数

GLUT のコールバック関数として、サンプルプログラム（view_sample.cpp）の 444∼581 行で、5 つの関数を定義している。

画面描画時に呼ばれるコールバック関数（DisplayCallback 関数）では、1.4 節で説明する UpdateViewMatrix 関数を呼び出して、媒介変数を用いる場合の視野変換行列の設定を行っている。また、地面の描画、木の描画、注視点を表す赤い球の描画、文字情報の描画などを行っている。

マウスドラッグ時に呼ばれるコールバック関数（MouseDragCallback 関数）では、1.5 節で説明する UpdateView 関数を呼び出して、マウス操作に応じた視点パラメタ or 変換行列の更新の処理を行っている。

1.3 視点パラメタの初期化

サンプルプログラム（view_sample.cpp）の 59∼127 行で、視点バラメタ（1.1 節参照）の初期化を行う InitView 関数が記述されている。現在の視点操作モード（mode）に応じて、視点パラメタの初期値を設定している。

1.4 視点パラメタにもとづく視野変換行列の設定

サンプルプログラム（view_sample.cpp）の 130∼149 行で、媒介変数による視点操作のための、視点パラメタ（1.1 節参照）にもとづく視野変換行列の設定を行う UpdateViewMatrix 関数が記述されている。

1.5 マウス操作に応じた視点パラメタ or 変換行列の更新

サンプルプログラム（view_sample.cpp）の 149∼325 行で、マウス操作に応じた視点パラメタ or 変換行列の更新を行う UpdateView 関数が記述されている。本関数が、視点操作を実現するメインの関数となる。

引数として、以下のような、マウス操作の情報を引数として受け取る。

- delta_mouse_right_x 右ドラッグ中の左右のマウス移動量
- delta_mouse_right_y 右ドラッグ中の上下のマウス移動量
- delta_mouse_left_x 左ドラッグ中の左右のマウス移動量
- delta_mouse_left_y 左ドラッグ中の上下のマウス移動量

これらの引数がどのようにして計算されるかは、本関数を呼び出している、マウスドラッグ時に呼ばれるコールバック関数（MouseDragCallback 関数）の処理を参照する。

これらの入力にもとづいて、現在の視点操作モード（mode）に応じて、視点操作の処理を行う。媒介変数による視点操作では、視点パラメタ（1.1 節参照）の値を更新する。変換行列の直接更新による視点操作では、OpenGL に設定されている視野変換行列を変更する。

1.6 視点操作方法 1：Dolly（媒介変数）

横方向の右ボタンドラックに応じて視線を左右に回転するため、引数 delta_mouse_right_x の値に応じて、媒介変数 view_yaw の値を変化させる（適当な係数をかけた値を加算する）。このとき、媒介変数の値が一定の範囲（ 0 ～ 360 ） で連続して変化するように（下限と上限が連続するように）、制限を加える。

縦方向の右ボタンドラックに応じて視線を上下に回転するため、引数 delta_mouse_right_y の値に応じて、媒介変数 view_pitch の値を変化させる（適当な係数をかけた値を加算する）。このとき、媒介変数の値が一定の範囲（ -90 ～ 0 ） を超えないように、制限を加える。

縦方向の左ボタンドラックに応じて視点と注視点の距離を変更するため、引数 delta_mouse_left_y の値に応じて、媒介変数 view_distance の値を変化させる（適当な係数をかけた値を加算する）。このとき、媒介変数の値が一定の範囲（ 5.0 ） を超えないように、制限を加える。

ソースコード 2: 視点操作方法 1：Dolly（媒介変数）

```
void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
   // 視点パラメタを更新（Dollyモード・媒介変数）
   if ( mode == VIEW_DOLLY_PARAM )
   {
      // 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
      if ( delta_mouse_right_x != 0 )
      {
         view_yaw -= delta_mouse_right_x * 1.0;

         // パラメタの値が所定の範囲を超えないように修正
         if ( view_yaw < 0.0 )
            view_yaw += 360.0;
         else if ( view_yaw > 360.0 )
            view_yaw -= 360.0;
      }

      // 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
      if ( delta_mouse_right_y != 0 )
      {
         view_pitch -= delta_mouse_right_y * 1.0;

         // パラメタの値が所定の範囲を超えないように修正
         if ( view_pitch < -90.0 )
            view_pitch = -90.0;
         else if ( view_pitch > -2.0 )
            view_pitch = -2.0;
      }

      // 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
      if ( delta_mouse_left_y != 0 )
      {
         view_distance += delta_mouse_left_y * 0.2;

         // パラメタの値が所定の範囲を超えないように修正
         if ( view_distance < 5.0 )
            view_distance = 5.0;
      }
   }

   // 省略
}
```

1.7 視点操作方法 1：Dolly（直接更新）

横方向の右ボタンドラックに応じて視点を水平方向に回転するため、引数 delta_mouse_right_x の値に応じて、OpenGL に設定されている視野変換行列を変化させる。現在の視野変換行列に、右から、今回の回転をかけることで、視野変換行列を更新する。

また、縦方向の右ボタンドラックに応じて、視線を視点を上下方向に回転するため、引数 delta_mouse_right_y の値に応じて、OpenGL に設定されている視野変換行列を変化させる。まずは、現在の視野変換行列を取得して、ローカル変数（float m[16]）に記録し、その平行移動成分を変数 tx, ty, tz に設定しておく。現在の視野変換行列（m）の平行移動成分を 0 にした後、OpenGL に設定されている変換行列を初期化して、左から、取得した平行移動成分（tx, ty, tz)、今回の回転、平行移動成分を 0 にした元の視野変換行列の順番で変換行列をかけることで、視野変換行列を更新する。

ソースコード 3: 視点操作方法 1：Dolly（媒介変数）

```
void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
   // 省略

   // 変換行列を更新（Dollyモード・直接更新）
   if ( mode == VIEW_DOLLY_DIRECT )
   {
      // 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
      if ( delta_mouse_right_x != 0 )
      {
         // 視点の水平方向の回転量を計算
         float  delta_yaw = delta_mouse_right_x * 1.0;

         // 現在の変換行列の右側に、今回の回転変換をかける
         glMatrixMode( GL_MODELVIEW );
         glRotatef( delta_yaw, 0.0, 1.0, 0.0 );
      }

      // 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
      if ( delta_mouse_right_y != 0 )
      {
         // 視点の上下方向の回転量を計算
         float  delta_pitch = delta_mouse_right_y * 1.0;

         // 現在の変換行列を取得
         float  m[ 16 ];
         float  tx, ty, tz;
         glGetFloatv( GL_MODELVIEW_MATRIX, m );

         // 現在の変換行列の平行移動成分を記録
         tx = m[ 12 ];
         ty = m[ 13 ];
         tz = m[ 14 ];

         // 現在の変換行列の平行移動成分を0にする
         m[ 12 ] = 0.0f;
         m[ 13 ] = 0.0f;
         m[ 14 ] = 0.0f;

         // 変換行列を初期化
         glMatrixMode( GL_MODELVIEW );
         glLoadIdentity();

         // カメラの平行移動行列を設定
         glTranslatef( tx, ty, tz );
```

```
            // 右側に、今回の回転変換をかける
            glRotatef( delta_pitch, 1.0, 0.0, 0.0 );

            // 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
            glMultMatrixf( m );
        }
    }

    // 省略
}
```

2 レポート課題

レポート課題として、残りの 2 通りの視点操作方法（Scroll モード、Walkthrough モード）を、それぞれ 2 通りの方法（媒介変数を使用する方法、変換行列を直接更新する方法）で実現する、計 4 つの処理を作成する。

2.1 視点操作方法 2：Scroll（媒介変数）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

視点情報を表す媒介変数として、複数のグローバル変数が定義されている。

縦方向の右ボタンドラックに応じて視線を上下に回転するため、引数 空欄 A の値に応じて、媒介変数 空欄 B の値を変化させる（適当な係数をかけた値を加算する）。このとき、媒介変数の値が一定の範囲（ 空欄 C ～ 空欄 D ） を超えないように、制限を加える。

また、左ボタンドラックに応じて視点を前後左右に移動するため、左右方向は、引数 空欄 E の値に応じて、媒介変数 空欄 F の値を変化させ、前後方向は、引数 空欄 G の値に応じて、媒介変数 空欄 H の値を変化させる。

ソースコード 4: 視点操作方法 2：Scroll（媒介変数）

```
void UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
    // 省略

    // 視点パラメタを更新（Scrollモード・媒介変数）
    if ( mode == VIEW_SCROLL_PARAM )
    {
        // ※レポート課題（ここに自分が作成したプログラムを記述する）

        // 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
        if ( 空欄A != 0 )
        {
            空欄B += 空欄A * -1.0;

            // パラメタの値が所定の範囲を超えないように修正
            if ( 空欄B < 空欄C )
                空欄B = 空欄C;
            else if ( 空欄B > 空欄D )
                空欄B = 空欄D;
        }

        // 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）
        if ( ( 空欄E != 0 ) || ( 空欄G != 0 ) )
        {
            空欄F += 空欄E * 0.1;
```

```
            空欄H += 空欄G * 0.1;
        }

    }

    // 省略
}
```

2.2 視点操作方法3：Walkthrough（直接更新）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

横方向の右ボタンドラックに応じて視点を水平方向に回転するため、引数 空欄A の値に応じて、OpenGLに設定されている視野変換行列を変化させる。

まずは、現在の視野変換行列を取得し、ローカル変数（float m[16]）に記録しておく。プログラムコードは、
空欄B
のようになる。

その後、視野変換行列を初期化した後に、左から、
空欄C
の順番で変換行列をかけることで、視野変換行列を更新する。プログラムコードは、
空欄D
空欄E
のようになる。

また、左ボタンドラックに応じて視線を視点を前後左右に移動するため、引数 空欄F と 空欄G の値に応じて、OpenGLに設定されている視野変換行列を変化させる。

まずは、現在の視野変換行列を取得し、ローカル変数（float m[16]）に記録しておく。プログラムコードは、
空欄B
のようになる。

その後、視野変換行列を初期化した後に、左から、
空欄H
の順番で変換行列をかけることで、視野変換行列を更新する。プログラムコードは、
空欄I
空欄J
のようになる。

ソースコード 5: 視点操作方法3：Walkthrough（直接更新）

```
void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
    // 省略

    // 変換行列を更新（Walkthroughモード・直接更新）
    if ( mode == VIEW_WALKTHROUGH_DIRECT )
    {
        // ※レポート課題（ここに自分が作成したプログラムを記述する）

        // 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
        if ( 空欄A != 0 )
        {
            // 視点の水平方向の回転量を計算
            float  delta_yaw = 空欄A * 1.0;

            // 現在の変換行列（カメラの向き）を取得
```

```
            float  m[ 16 ];
            空欄B

            // 変換行列を初期化
            glMatrixMode( GL_MODELVIEW );
            glLoadIdentity();

            // 変換行列を更新
            空欄D
            空欄E
        }

        // 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）
        if ( ( 空欄F != 0 ) || ( 空欄G != 0 ) )
        {
            // 左右の移動量、前後の移動量を設定
            float  dx = 空欄F * 0.1f;;
            float  dz = 空欄G * 0.1f;;

            // 現在の変換行列（カメラの向き）を取得
            float  m[ 16 ];
            空欄B

            // 変換行列を初期化
            glMatrixMode( GL_MODELVIEW );
            glLoadIdentity();

            // 変換行列を更新
            空欄I
            空欄J
        }

    }

    // 省略
}
```

2.3 視点操作方法 2：Scroll（直接更新）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

縦方向の右ボタンドラックに応じて視線を上下に回転するため、引数 空欄 A の値に応じて、OpenGL に設定されている視野変換行列を変化させる。

視野変換行列を初期化した後に、左から、 空欄 B
の順番で変換行列をかけることで、視野変換行列を更新する。プログラムコードは、
空欄 C
空欄 D
空欄 E
空欄 F
のようになる。

また、縦方向の右ボタンドラックに応じて視線を視点を前後左右に移動するため、引数 空欄 G と 空欄 H の値に応じて、OpenGL に設定されている視野変換行列を変化させる。

現在の視野変換行列の 空欄 I から、今回の平行移動の変換行列をかけることで、視野変換行列を更新する。プログラムコードは、
空欄 J

のようになる。

ソースコード 6: 視点操作方法 2：Scroll（直接更新）

```
void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
   // 省略

   // 視点パラメタを更新（Scrollモード・直接更新）
   if ( mode == VIEW_SCROLL_DIRECT )
   {
      // ※レポート課題（ここに自分が作成したプログラムを記述する）

      // 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
      if ( 空欄A != 0 )
      {
         // 視点の上下方向の回転量を計算
         float  delta_pitch = 空欄A * 1.0;

         // 現在の変換行列（カメラの向き）を取得
         float  m[ 16 ];
         glGetFloatv( GL_MODELVIEW_MATRIX, m );

         // 変換行列を初期化
         glMatrixMode( GL_MODELVIEW );
         glLoadIdentity();

         // 変換行列を更新
         空欄C
         空欄D
         空欄E
         空欄F
      }

      // 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）
      if ( ( 空欄G != 0 ) || ( 空欄H != 0 ) )
      {
         // 左右の移動量、前後の移動量を設定
         float  dx = 空欄G * 0.1f;
         float  dz = 空欄H * 0.1f;

         // 変換行列を更新
         空欄J

         // 変換行列とは別に、注視点の位置を表すパラメタも更新する
         //（注視点の位置にオブジェクトを描画するため）
         view_center_x += dx;
         view_center_z += dz;
      }
   }

   // 省略
}
```

2.4 視点操作方法 3：Walkthrough（媒介変数）

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

縦方向の右ボタンドラックに応じて視点を水平方向に回転するため、引数 空欄A の値に応じて、媒介変数 空欄B の値を変化させる（適当な係数をかけた値を加算する）。

このとき、媒介変数の値が一定の範囲（例えば、0 ～ 空欄C など）を超えないように、制限を加える。水平方向の向きが連続するように、値が 0 よりも小さい値になったときには 空欄D を加算し、値が 空欄C よりも大きい値になったときには 空欄E を加算する。

また、左ボタンドラックに応じて視線を視点を前後左右に移動するため、引数 空欄F と 空欄G の値に応じて、媒介変数 空欄H と 空欄I の値を変化させる。

前後方向の移動は、空欄F の値に応じて、変換行列から取得した前後方向のベクトルに応じて移動させる。プログラムコードは、

空欄H += 空欄J

空欄I += 空欄K

のようになる。

左右方向の移動は、空欄G の値に応じて、変換行列から取得した前後方向のベクトルに応じて移動させる。プログラムコードは、

空欄H += 空欄L

空欄I += 空欄M

のようになる。

ソースコード 7: 視点操作方法 3：Walkthrough（媒介変数）

```
void  UpdateView( int delta_mouse_right_x, int delta_mouse_right_y, int
    delta_mouse_left_x, int delta_mouse_left_y )
{
   // 省略

   // 視点パラメタを更新（Walkthroughモード・媒介変数）
   if ( mode == VIEW_WALKTHROUGH_PARAM )
   {
      // ※レポート課題（ここに自分が作成したプログラムを記述する）

      // 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
      if ( 空欄A != 0 )
      {
         空欄B -= 空欄A * 1.0;

         // パラメタの値が所定の範囲を超えないように修正
         if ( 空欄B < 0.0 )
            空欄B += 空欄D;
         else if ( 空欄B > 空欄C )
            空欄B += 空欄E;
      }

      // 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）
      if ( ( 空欄F != 0 ) || ( 空欄G != 0 ) )
      {
         // 左右の移動量、前後の移動量を計算
         float  dz, dx;
         dz = 空欄F * 0.1;
         dx = 空欄G * 0.1;

         // 現在の変換行列（カメラの向き）を取得
         float  m[ 16 ];
         glGetFloatv( GL_MODELVIEW_MATRIX, m );

         // ワールド座標系でのカメラの移動量を計算（前後方向）
         空欄H += 空欄J
```

```
            空欄I += 空欄K

            // ワールド座標系でのカメラの移動量を計算（左右方向）
            空欄H += 空欄L
            空欄I += 空欄M
        }
    }

    // 省略
}
```