```
//
//  コンピュータグラフィックス特論Ⅱ
//  キーフレームアニメーション サンプルプログラム
//


#ifdef  WIN32
    #include <Windows.h>
#endif
#include <string.h>
#include <vector>

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

// vecmathヘッダファイルのインクルード
#include <vecmath.h>
#include "vecmath_gl.h"

// 複数オブジェクトの位置・向きをマウスで操作するためのクラス
#include "ObjectLayout.h"

// 幾何形状オブジェクト、及び、読み込み・描画関数
#include "Obj.h"


// π
#define  M_PI  3.14159268


//
//  カメラ・GLUTの入力処理に関するグローバル変数
//

// カメラの回転のための変数
static float   camera_yaw = 15.0f; // 30.0;     // Ｙ軸を中心とする回転角度
static float   camera_pitch = -20.0f; // -30.0;   // Ｘ軸を中心とする回転角度
static float   camera_distance = 5.0f; // 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
static 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)
static 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)
static 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)
static 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
static int     win_width, win_height;


//
//  オブジェクトの配置・表示に関するグローバル変数
//

// 複数オブジェクトの位置・向きをマウスで操作するためのモジュール
ObjectLayout *  layout = NULL;

// 表示用の幾何形状オブジェクト
Obj *  object;
Vector3f  object_size;

// 点光源の位置 (影の投影方向)
Vector3f  light_pos( 0.0f, 10.0f, 0.0f );

// 影の色
Color4f   shadow_color( 0.2f, 0.2f, 0.2f, 0.5f );



//
//  位置・向きの補間に関するグローバル変数
//

// 位置補間方法を表す列挙型
enum  PositionInterpolationEnum
{
    PI_LINEAR,
    PI_HERMIT,
    PI_BEZIER,
    PI_BSPLINE,
    NUM_PI_METHOD
};

// 向き補間方法を表す列挙型
enum  OrientationInterpolationEnum
{
    OI_NONE,
    OI_EULAR,
    OI_QUAT,
    NUM_OI_METHOD
};

// 位置補間方法の名前を表す文字列 (表示用)
const char *  pi_name[] = {
    "Linear", "Hermit", "Bezier", "B-Spline" };

// 向き補間方法の名前を表す文字列 (表示用)
const char *  oi_name[] = {
    "None", "Eular", "Quat" };

// 使用する位置・向き補間方法
PositionInterpolationEnum      pos_method = PI_LINEAR;
OrientationInterpolationEnum   ori_method = OI_EULAR;


//
//  キーフレーム情報に関するグローバル変数
//

```

```
109 // キーフレーム情報
110 struct Keyframe
111 {
112     float     time; // 時刻
113     Point3f   pos;  // 位置
114     Matrix3f  ori;  // 向き
115 };
116 
117 // 設定されている全キーフレーム情報 (可変長配列)
118 vector< Keyframe >   keyframes;
119 
120 
121 //
122 //  アニメーション関連のグローバル変数
123 //
124 
125 // アニメーション中かどうかを表すフラグ
126 bool   on_animation = false;
127 
128 // 全フレーム描画モード・軌道描画モード
129 bool   on_draw_frames = false;
130 bool   on_draw_trajectory = true;
131 
132 // アニメーションの再生時間
133 float  animation_time = 0.0f;
134 
135 // アニメーション中のオブジェクトの位置・向きを表す変換行列
136 float  model_mat[ 16 ];
137 
138 
139 
140 //
141 //  キーフレームアニメーションのための処理
142 //
143 
144 
145 //
146 //  オブジェクト配置にもとづいて全キーフレーム情報を更新
147 //
 H3 void  UpdateKeyframes()
149 {
150     Keyframe  key;
151 
152     // キーフレーム数を設定
153     int  num_keyframes = layout->GetNumObjects();
154     keyframes.resize( num_keyframes );
155 
156     // 各キーフレームの情報を設定
157     for ( int i=0; i<num_keyframes; i++ )
158     {
159         // i番目のキーフレームの時刻を i秒とする
160         key.time = (float) i;
161 
162         // 位置・向きを設定
163         key.pos = layout->GetPosition( i );
164         key.ori = layout->GetOrientation( i );
165 
166         // キーフレームの情報を設定
167         keyframes[ i ] = key;
168     }
169 }
170 
171 
172 //
173 //  回転行列からオイラー角への変換 (yaw → pitch → roll の順の場合) (vecmathの行列を引数とする)
174 //
175 void  ConvMatToEular( const Matrix3f & m, float & yaw, float & pitch, float & roll )
176 {
177     Vector3f  y_axis, z_axis;
178     m.getColumn( 1, &y_axis );
179     m.getColumn( 2, &z_axis );
180 
181     yaw = atan2( z_axis.x, z_axis.z );
182 
183     float  cos_yaw = cos( yaw );
184     pitch = atan2( -z_axis.y, sqrt( z_axis.x * z_axis.x + z_axis.z * z_axis.z ) );
185 
186     float  sin_yaw = sin( yaw );
187     float  cos_pitch = cos( pitch );
188     roll = atan2( cos_pitch * ( sin_yaw * y_axis.z - cos_yaw * y_axis.x ), y_axis.y );
189 }
190 
191 
192 //
193 //  回転行列からオイラー角への変換 (yaw → pitch → roll の順の場合) (配列表現の行列を引数とする)
194 //
195 void  ConvMatToEular( const float m[9], float & yaw, float & pitch, float & roll )
196 {
197     struct Vector3f
198     {
199         float  x, y, z;
200     };
201     Vector3f  y_axis, z_axis;
202     z_axis.x = m[2];
203     z_axis.y = m[5];
204     z_axis.z = m[8];
205     y_axis.x = m[1];
206     y_axis.y = m[4];
207     y_axis.z = m[7];
208 
209     yaw = atan2( z_axis.x, z_axis.z );
210 
211     float  cos_yaw = cos( yaw );
212     pitch = atan2( cos_yaw * z_axis.y, fabs( z_axis.z ) );
213 
214     float  sin_yaw = sin( yaw );
215     float  cos_pitch = cos( pitch );
216     roll = atan2( cos_pitch * ( sin_yaw * y_axis.z - cos_yaw * y_axis.x ), y_axis.y );
```

```
217 }
218
219
220 //
221 //  物体の位置・向きを更新
222 //
 H3 void  UpdateModelMat( float time, float mat[ 16 ] )
224 {
225     if ( keyframes.size() == 0 )
226         return;
227
228     // 指定時刻に対応する区間の番号と区間内での正規化時間 (0.0～1.0)
229     int  seg_no = -1;
230     float  t = 0.0f;
231
232     // 指定時刻に対応する区間の番号を取得
233     for ( int i=0; i<keyframes.size()-1; i++ )
234     {
235         // 指定時刻が i番目の区間に対応するかを判定
236         if ( ( time >= keyframes[ i ].time ) && ( time <= keyframes[ i+1 ].time ) )
237         {
238             seg_no = i;
239
240             // 区間内での正規化時間を計算
241             t = ( time - keyframes[ i ].time ) / ( keyframes[ i+1 ].time  - keyframes[ i ].time );
242
243             break;
244         }
245     }
246     if ( seg_no == -1 )
247     {
248         // 最初のキーフレームより前の時刻が指定されたら、最初の区間の開始時刻を使用
249         if ( time < keyframes[ 0 ].time )
250         {
251             seg_no = 0;
252             t = 0.0f;
253         }
254         // 最後のキーフレームより後の時刻が指定されたら、最後の区間の終了時刻を使用
255         else
256         {
257             seg_no = keyframes.size() - 2;
258             t = 1.0f;
259         }
260     }
261
262
263     // 指定時刻におけるオブジェクトの位置・向き
264     Vector3f  p;
265     Matrix3f  o;
266
267     // 位置を線形補間により計算
268     if ( pos_method == PI_LINEAR )
269     {
270         // 区間の両端点の位置を取得
271         const Point3f &  p0 = keyframes[ seg_no ].pos;
272         const Point3f &  p1 = keyframes[ seg_no + 1 ].pos;
273
274         // 両端点を線形に補間
275         p.scaleAdd( t, p1 - p0, p0 );
276
277         // 両端点を線形に補間 (下記の書き方でも可)
278 //      p = t * ( p1 - p0 ) + p0;
279     }
280
281     // エルミート補間
282     else if ( pos_method == PI_HERMIT )
283     {
284         // 区間の両端点の位置を取得
285         const Point3f &  p0 = keyframes[ seg_no ].pos;
286         const Point3f &  p1 = keyframes[ seg_no + 1 ].pos;
287
288         // 区間の両端点の傾きを取得
289         Vector3f  v0, v1;
290         const Matrix3f &  o0 = keyframes[ seg_no ].ori;
291         const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;
292         o0.getColumn( 2, &v0 );
293         o1.getColumn( 2, &v1 );
294         v0.negate();
295         v1.negate();
296
297         // Hermite関数の値を計算
298         // 各自実装 (式・プログラムは講義資料を参照)
299
300         // ※レポート課題
301
302     }
303
304     // Bezierによる補間
305     else if ( pos_method == PI_BEZIER )
306     {
307         // 指定時刻に対応するBezier補間の区間の番号と区間内での正規化時間 (0.0～1.0)
308         int  bezier_seg_no = -1;
309         float  s = 0.0f;
310
311         // Bezier補間の区間番号を計算
312         // 連続する４つのキーフレーム（３区間）をまとめて１つの区間として扱う
313         // 区間数×３＋１個のキーフレームが必要
314         bezier_seg_no = ( seg_no == 0 ) ? 0 : (int) floor( seg_no / 3 );
315
316         // 最後の区間でキーフレーム数が４つに足りない場合は、前の区間の最後の時刻を使用
317         if ( ( bezier_seg_no + 1 ) * 3 + 1 > keyframes.size() )
318         {
319             bezier_seg_no --;
320             s = 1.0f;
321         }
322         // 区間内での正規化時間 (0.0～1.0) を計算
323         else
324         {
```

```
325             s = ( time - keyframes[ bezier_seg_no * 3 ].time ) / ( keyframes[ bezier_seg_no * 3 + 3 ].time - keyframes[ bezier_seg_no * 3 ].time );
326         }
327
328         // 一つも区間が存在しない場合（キーフレーム数が３個以下の場合）は、最初のキーフレームの位置を出力
329         if ( keyframes.size() < 4 )
330         {
331             p = keyframes[ 0 ].pos;
332         }
333         // Bezier補間を計算
334         else
335         {
336             // Bezier補間の区間の４つの制御点（両端点と、中間の２つの点）の位置を取得
337             const Point3f &  p0 = keyframes[ bezier_seg_no * 3 ].pos;
338             const Point3f &  p1 = keyframes[ bezier_seg_no * 3 + 1 ].pos;
339             const Point3f &  p2 = keyframes[ bezier_seg_no * 3 + 2 ].pos;
340             const Point3f &  p3 = keyframes[ bezier_seg_no * 3 + 3 ].pos;
341
342             // Bezier関数の値を計算
343             // 各自実装（式は講義資料を参照）
344
345             // ※レポート課題
346
347         }
348     }
349
350     // B-Splineによる補間
351     else if ( pos_method == PI_BSPLINE )
352     {
353         // 区間の両端点と、さらにその隣の点（もしあれば）の位置を取得
354         int  k0, k1, k2, k3;
355         k0 = ( seg_no > 0 ) ? ( seg_no - 1 ) : seg_no;
356         k1 = seg_no;
357         k2 = seg_no + 1;
358         k3 = ( seg_no + 2 == keyframes.size() ) ? ( seg_no + 1 ) : ( seg_no + 2);
359         const Point3f &  p0 = keyframes[ k0 ].pos;
360         const Point3f &  p1 = keyframes[ k1 ].pos;
361         const Point3f &  p2 = keyframes[ k2 ].pos;
362         const Point3f &  p3 = keyframes[ k3 ].pos;
363
364         // B-Sprine 関数の値を計算
365         // 各自実装（式は講義資料を参照）
366
367         // ※レポート課題
368
369     }
370
371
372     // 向きの補間なし
373     if ( ori_method == OI_NONE )
374     {
375         o = layout->GetOrientation( seg_no );
376     }
377
378     // 向きをオイラー角で補間
379     else if ( ori_method == OI_EULAR )
380     {
381         // 区間の両端点の向きを取得
382         const Matrix3f &  o0 = keyframes[ seg_no ].ori;
383         const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;
384
385         // オイラー角に変換
386         float  y0, p0, r0;
387         float  y1, p1, r1;
388         ConvMatToEular( o0, y0, p0, r0 );
389         ConvMatToEular( o1, y1, p1, r1 );
390
391         // 各回転角度を線形補間
392         float  y, p, r;
393         if ( y0 < y1 - M_PI )
394             y0 += 2.0f * M_PI;
395         else if ( y0 > y1 + M_PI )
396             y0 -= 2.0f * M_PI;
397         y = ( y1 - y0 ) * t + y0;
398         p = ( p1 - p0 ) * t + p0;
399         r = ( r1 - r0 ) * t + r0;
400
401         // 行列に変換
402         Matrix3f  rot;
403         o.rotY( y );
404         rot.rotX( p );
405         o.mul( o, rot );
406         rot.rotZ( r );
407         o.mul( o, rot );
408     }
409
410     // 向きを四元数と球面線形補間により計算
411     else if ( ori_method == OI_QUAT )
412     {
413         // 区間の両端点の向きを取得
414         const Matrix3f &  o0 = keyframes[ seg_no ].ori;
415         const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;
416
417         // 四元数を使って球面線形補間を計算
418         // 各自実装（式は講義資料を参照）
419
420         // ※レポート課題
421
422     }
423
424     // オブジェクトの位置・向きを表す変換行列を配列にコピー
425     Matrix4f  f;
426     f.set( o, p, 1.0f );
427     f.transpose();
428     memcpy( mat, &f.m00, sizeof( float ) * 16 );
429 }
430
431
```

```
432
433 //
434 //  以下、プログラムのメイン処理
435 //
436
437
438 //
439 //  あらかじめ定義されたオブジェクト配置を設定
440 //
 H3 void  SetupScene( int no )
442 {
443     if ( !layout )
444         return;
445
446     Matrix3f  ori, rot;
447
448     if ( no == 1 )
449     {
450         layout->DeleteAllObjects();
451         layout->AddObject();
452         layout->SetObjectSize( 0, object_size );
453         layout->SetObjectPos( 0, Point3f( -1.0f, 0.5f, -1.5f ) );
454         ori.rotY( M_PI * 0.6f );
455         layout->SetObjectOri( 0, ori );
456
457         layout->AddObject();
458         layout->SetObjectSize( 1, object_size );
459         layout->SetObjectPos( 1, Point3f( 0.0f, 1.0f, 0.0f ) );
460         ori.rotY( M_PI * 1.2f );
461         rot.rotX( M_PI / 3.0f );
462         ori.mul( ori, rot );
463         rot.rotZ( M_PI / 4.0f );
464         ori.mul( ori, rot );
465         layout->SetObjectOri( 1, ori );
466
467         layout->AddObject();
468         layout->SetObjectSize( 2, object_size );
469         layout->SetObjectPos( 2, Point3f( -2.0f, 0.5f, 1.0f ) );
470         ori.rotY( M_PI );
471         rot.rotX( - M_PI / 6.0f );
472         ori.mul( ori, rot );
473         layout->SetObjectOri( 2, ori );
474
475         layout->AddObject();
476         layout->SetObjectSize( 3, object_size );
477         layout->SetObjectPos( 3, Point3f( 0.0f, 0.4f, 1.5f ) );
478         ori.rotY( M_PI * 1.2f );
479         layout->SetObjectOri( 3, ori );
480
481         layout->AddObject();
482         layout->SetObjectSize( 4, object_size );
483         layout->SetObjectPos( 4, Point3f( 1.0f, 0.35f, 1.75f ) );
484         ori.rotY( M_PI * -0.4f );
485         rot.rotX( M_PI / 12.0f );
486         ori.mul( ori, rot );
487         layout->SetObjectOri( 4, ori );
488
489         layout->AddObject();
490         layout->SetObjectSize( 5, object_size );
491         layout->SetObjectPos( 5, Point3f( 1.5f, 0.6f, 0.0f ) );
492         ori.rotY( 0.0f );
493         layout->SetObjectOri( 5, ori );
494
495         layout->AddObject();
496         layout->SetObjectSize( 6, object_size );
497         layout->SetObjectPos( 6, Point3f( 1.0f, 1.0f, -1.0f ) );
498         ori.rotY( M_PI / 3 );
499         layout->SetObjectOri( 6, ori );
500
501         camera_yaw = 15.0f;
502         camera_pitch = -20.0f;
503         camera_distance = 6.0f;
504     }
505     else if ( no == 2 )
506     {
507         layout->DeleteAllObjects();
508
509         layout->AddObject();
510         layout->SetObjectSize( 0, object_size );
511         layout->SetObjectPos( 0, Point3f(-1.0f, 0.5f, 0.0f ) );
512         ori.rotY( M_PI * 0.6f );
513         layout->SetObjectOri( 0, ori );
514
515         layout->AddObject();
516         layout->SetObjectSize( 1, object_size );
517         layout->SetObjectPos( 1, Point3f( 1.0f, 0.5f, 0.0f ) );
518         ori.rotY( M_PI * 1.2f );
519         rot.rotX( M_PI / 3.0f );
520         ori.mul( ori, rot );
521         rot.rotZ( M_PI / 4.0f );
522         ori.mul( ori, rot );
523         layout->SetObjectOri( 1, ori );
524
525         camera_yaw = 0.0f;
526         camera_pitch = -20.0f;
527         camera_distance = 5.0f;
528     }
529
530     // オブジェクト配置にもとづいて全キーフレーム情報を更新
531     UpdateKeyframes();
532 }
533
534
535 //
536 //  格子模様の床を描画
537 //
 H3 void  DrawFloor( int tile_size, int num_x, int num_z, float r0, float g0, float b0, float r1, float g1, float b1 )
539 {
```

```
540     int  x, z;
541     float  ox, oz;
542 
543     glBegin( GL_QUADS );
544     glNormal3d( 0.0, 1.0, 0.0 );
545 
546     ox = - ( num_x * tile_size ) / 2;
547     for ( x=0; x<num_x; x++ )
548     {
549         oz = - ( num_z * tile_size ) / 2;
550         for ( z=0; z<num_z; z++ )
551         {
552             if ( ( (x + z) % 2 ) == 0 )
553                 glColor3f( r0, g0, b0 );
554             else
555                 glColor3f( r1, g1, b1 );
556 
557             glTexCoord2d( 0.0f, 0.0f );
558             glVertex3d( ox, 0.0, oz );
559             glTexCoord2d( 0.0f, 1.0f );
560             glVertex3d( ox, 0.0, oz + tile_size );
561             glTexCoord2d( 1.0f, 1.0f );
562             glVertex3d( ox + tile_size, 0.0, oz + tile_size );
563             glTexCoord2d( 1.0f, 0.0f );
564             glVertex3d( ox + tile_size, 0.0, oz );
565 
566             oz += tile_size;
567         }
568         ox += tile_size;
569     }
570     glEnd();
571 }
572 
573 
574 //
575 //  幾何形状モデル (Obj形状) の影の描画
576 //
 H3 void  RenderShadow( Obj * obj, const float matrix[ 16 ] )
578 {
579     RenderObjShadow( obj, matrix, light_pos.x, light_pos.y, light_pos.z, shadow_color.x, shadow_color.y, shadow_color.z, shadow_color.w );
580 }
581 
582 void  RenderShadow( Obj * obj, Matrix4f & mat )
583 {
584     Matrix4f  frame( mat );
585     frame.transpose();
586     RenderObjShadow( obj, &frame.m00, light_pos.x, light_pos.y, light_pos.z, shadow_color.x, shadow_color.y, shadow_color.z, shadow_color.w
 );
587 }
588 
589 
590 //
591 //  テキストを描画
592 //
 H3 void  DrawTextInformation( int line_no, const char * message )
594 {
595     if ( message == NULL )
596         return;
597 
598     // 射影行列を初期化 (初期化の前に現在の行列を退避)
599     glMatrixMode( GL_PROJECTION );
600     glPushMatrix();
601     glLoadIdentity();
602     gluOrtho2D( 0.0, win_width, win_height, 0.0 );
603 
604     // 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)
605     glMatrixMode( GL_MODELVIEW );
606     glPushMatrix();
607     glLoadIdentity();
608 
609     // Ｚバッファ・ライティングはオフにする
610     glDisable( GL_DEPTH_TEST );
611     glDisable( GL_LIGHTING );
612 
613     // メッセージの描画
614     glColor3f( 1.0, 0.0, 0.0 );
615     glRasterPos2i( 16, 40 + 24 * line_no );
616     for ( int i=0; message[i]!='¥0'; i++ )
617         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
618 
619     // 設定を全て復元
620     glEnable( GL_DEPTH_TEST );
621     glEnable( GL_LIGHTING );
622     glMatrixMode( GL_PROJECTION );
623     glPopMatrix();
624     glMatrixMode( GL_MODELVIEW );
625     glPopMatrix();
626 }
627 
628 
629 //
630 //  画面描画時に呼ばれるコールバック関数
631 //
 H3 void  DisplayCallback( void )
633 {
634     // 画面をクリア
635     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
636 
637     // 変換行列を設定 (ワールド座標系→カメラ座標系)
638     glMatrixMode( GL_MODELVIEW );
639     glLoadIdentity();
640     glTranslatef( 0.0,  0.0, - camera_distance );
641     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
642     glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
643     glTranslatef( 0.5,  0.0, 0.0 ); // ワールド座標系での注視点 (適当な位置を設定)
644 
645     // 光源の位置を更新
646     float  light0_position[] = { 0.0, 10.0, 0.0, 1.0 };
```

```
647     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
648
649     // 格子模様の床を描画
650     DrawFloor( 1.0f, 50, 50, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );
651
652     // オブジェクトの軌道を描画
653     if ( on_draw_trajectory )
654     {
655         // オブジェクトの軌道を描画
656         float  mat[ 16 ];
657         glDisable( GL_LIGHTING );
658         glLineWidth( 2.0 );
659         glColor3f( 1.0f, 0.0f, 0.0f );
660         glBegin( GL_LINE_STRIP );
661         for ( float t=0.0f; t<=layout->GetNumObjects()-1.0f+0.001f; t+=0.1f )
662         {
663             UpdateModelMat( t, mat );
664             glVertex3f( mat[12], mat[13], mat[14] );
665         }
666         glEnd();
667         glEnable( GL_LIGHTING );
668
669         // キーフレームのオブジェクトを描画
670         if ( on_animation )
671         {
672             for ( int i=0; i<layout->GetNumObjects(); i++ )
673             {
674                 // オブジェクトを描画
675                 glPushMatrix();
676                     glMultMatrixf( layout->GetFrame( i ) );
677                     RenderObj( object );
678                 glPopMatrix();
679
680                 // オブジェクトの影を描画
681                 RenderShadow( object, layout->GetFrame( i ) );
682             }
683         }
684     }
685     // 全フレームのオブジェクトを描画
686     else if ( on_draw_frames )
687     {
688         float  mat[ 16 ];
689         for ( float t=0.0f; t<=layout->GetNumObjects()-1.0f+0.001f; t+=0.2f )
690         {
691             UpdateModelMat( t, mat );
692
693             // オブジェクトを描画
694             glPushMatrix();
695                 glMultMatrixf( mat );
696                 RenderObj( object );
697             glPopMatrix();
698
699             // オブジェクトの影を描画
700             RenderShadow( object, mat );
701         }
702     }
703
704     // アニメーション中のオブジェクトを描画
705     if ( on_animation && !on_draw_frames )
706     {
707         // アニメーション中のオブジェクトを描画
708         glPushMatrix();
709             glMultMatrixf( model_mat );
710             RenderObj( object );
711         glPopMatrix();
712
713         // オブジェクトの影を描画
714         RenderShadow( object, model_mat );
715     }
716     // 編集モード中の描画
717     else
718     {
719         // 各オブジェクトを描画
720         for ( int i=0; i<layout->GetNumObjects(); i++ )
721         {
722             glPushMatrix();
723                 glMultMatrixf( layout->GetFrame( i ) );
724                 RenderObj( object );
725             glPopMatrix();
726
727             // オブジェクトの影を描画
728             RenderShadow( object, layout->GetFrame( i ) );
729         }
730
731         // 操作用の情報を描画
732         layout->Render();
733     }
734
735     // 現在の描画モードを表示
736     if ( on_animation )
737         DrawTextInformation( 0, "Animation Mode" );
738     else
739         DrawTextInformation( 0, "Layout Mode" );
740
741     // 現在の補間モードを表示
742     if ( on_draw_frames || on_draw_trajectory || on_animation )
743     {
744         char  message[ 64 ] = "";
745         sprintf( message, "Position: %s, Orientation: %s", pi_name[ pos_method ], oi_name[ ori_method ] );
746         DrawTextInformation( 1, message );
747     }
748     // 現在の操作モードを表示
749     else
750     {
751         DrawTextInformation( 1, layout->GetOperationMode() );
752     }
753
754     // 現在の時刻を表示
```

```
755     if ( on_animation && !on_draw_frames )
756     {
757         char  message[ 64 ] = "";
758         sprintf( message, "Time: %2.2f", animation_time );
759         DrawTextInformation( 2, message );
760     }
761 
762     // 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
763     glutSwapBuffers();
764 }
765 
766 
767 //
768 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
769 //
 H3 void  ReshapeCallback( int w, int h )
771 {
772     // 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）
773     glViewport(0, 0, w, h);
774 
775     // カメラ座標系→スクリーン座標系への変換行列を設定
776     glMatrixMode( GL_PROJECTION );
777     glLoadIdentity();
778     gluPerspective( 45, (double)w/h, 1, 500 );
779 
780     // ウィンドウのサイズを記録（テキスト描画処理のため）
781     win_width = w;
782     win_height = h;
783 }
784 
785 
786 //
787 //  マウスクリック時に呼ばれるコールバック関数
788 //
 H3 void  MouseClickCallback( int button, int state, int mx, int my )
790 {
791     // 左ボタンが押されたらドラッグ開始
792     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
793         drag_mouse_l = 1;
794     // 左ボタンが離されたらドラッグ終了
795     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
796         drag_mouse_l = 0;
797 
798     // 右ボタンが押されたらドラッグ開始
799     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
800         drag_mouse_r = 1;
801     // 右ボタンが離されたらドラッグ終了
802     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
803         drag_mouse_r = 0;
804 
805     // 中ボタンが押されたらドラッグ開始
806     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
807         drag_mouse_m = 1;
808     // 中ボタンが離されたらドラッグ終了
809     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
810         drag_mouse_m = 0;
811 
812     // シーン配置機能に左クリックを通知
813     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
814         layout->OnMouseDown( mx, my );
815     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
816         layout->OnMouseUp( mx, my );
817 
818     // 再描画
819     glutPostRedisplay();
820 
821     // 現在のマウス座標を記録
822     last_mouse_x = mx;
823     last_mouse_y = my;
824 }
825 
826 
827 //
828 // マウス移動時に呼ばれるコールバック関数
829 //
 H3 void  MouseMotionCallback( int mx, int my )
831 {
832     // シーン配置機能にマウス移動を通知
833     if ( layout )
834         layout->OnMoveMouse( mx, my );
835 
836     // 再描画
837     glutPostRedisplay();
838 }
839 
840 
841 //
842 //  マウスドラッグ時に呼ばれるコールバック関数
843 //
 H3 void  MouseDragCallback( int mx, int my )
845 {
846     // 右ボタンのドラッグ中は視点を回転する
847     if ( drag_mouse_r )
848     {
849         // 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
850 
851         // マウスの横移動に応じてＹ軸を中心に回転
852         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
853         if ( camera_yaw < 0.0 )
854             camera_yaw += 360.0;
855         else if ( camera_yaw > 360.0 )
856             camera_yaw -= 360.0;
857 
858         // マウスの縦移動に応じてＸ軸を中心に回転
859         camera_pitch -= ( my - last_mouse_y ) * 1.0;
860         if ( camera_pitch < -90.0 )
861             camera_pitch = -90.0;
862         else if ( camera_pitch > 90.0 )
```

```
            camera_pitch = 90.0;
    }
    // 中ボタンのドラッグ中は視点とカメラの距離を変更する
    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

        // マウスの縦移動に応じて距離を移動
        camera_distance += ( my - last_mouse_y ) * 0.2;
        if ( camera_distance < 2.0 )
            camera_distance = 2.0;
    }

    // シーン配置機能にマウス移動を通知
    if ( layout )
    {
        layout->Update();
        layout->OnMoveMouse( mx, my );

        // オブジェクト配置にもとづいて全キーフレーム情報を更新
        UpdateKeyframes();
    }

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

    // 再描画
    glutPostRedisplay();
}


//
//  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
//
void  KeyboardCallback( unsigned char key, int mx, int my )
{
    // s キーでアニメーションの停止・再開
    if ( key == 's' )
        on_animation = !on_animation;

    // 数字キーであらかじめ定義されたオブジェクト配置を設定
    if ( ( key >= '1' ) && ( key <= '9' ) )
    {
        SetupScene( key - '0' );
    }

    // スペースキーでアニメーションを開始
    if ( key == ' ' )
    {
        on_animation = ! on_animation;
        if ( on_animation )
            animation_time = 0.0f;
        on_draw_frames = false;
    }

    // pキーで位置補間方法を変更
    if ( key == 'p' )
    {
        pos_method = (PositionInterpolationEnum)( ( pos_method + 1 ) % NUM_PI_METHOD );
    }
    // oキーで向き補間方法を変更
    if ( key == 'o' )
    {
        ori_method = (OrientationInterpolationEnum)( ( ori_method + 1 ) % NUM_OI_METHOD );
        if ( ori_method == OI_NONE )
            ori_method = (OrientationInterpolationEnum)( ori_method + 1 );
    }

    // fキーで通常・描画全フレーム描画・軌道描画を切り替え
    if ( key == 'f' )
    {
        if ( !on_draw_frames && !on_draw_trajectory )
            on_draw_trajectory = true;
        else if ( !on_draw_frames && on_draw_trajectory )
        {
            on_draw_frames = true;
            on_draw_trajectory = false;
        }
        else
            on_draw_frames = false;
    }

    // aキーでオブジェクトを追加
    if ( key == 'a' )
    {
        layout->AddObject();

        // オブジェクト配置にもとづいて全キーフレーム情報を更新
        UpdateKeyframes();
    }
    // dキーでオブジェクトを削除
    if ( key == 'd' )
    {
        layout->DeleteObject();

        // オブジェクト配置にもとづいて全キーフレーム情報を更新
        UpdateKeyframes();
    }

    // tキーで軸の描画モードを変更
    if ( key == 't' )
    {
        bool &  flag = layout->GetRenderOption().enable_xray_mode;
        flag = ! flag;
    }

    glutPostRedisplay();
```

```
}


//
//  アイドル時に呼ばれるコールバック関数
//
void  IdleCallback( void )
{
    // アニメーション処理
    if ( on_animation )
    {
#ifdef  WIN32
        // システム時間を取得し、前回からの経過時間に応じてΔｔを決定
        static DWORD  last_time = 0;
        DWORD  curr_time = timeGetTime();
        float  delta = ( curr_time - last_time ) * 0.001f;
        if ( delta > 0.01f )
            delta = 0.01f;
        last_time = curr_time;
        animation_time += delta;
#else
        // 固定のΔｔを使用
        animation_time += 0.02f;
#endif
        // アニメーションの繰り返し（最後まで再生が終わったら時間を０に戻す）
        if ( animation_time >= layout->GetNumObjects() - 1 )
            animation_time = 0.0f;

        // アニメーション中のオブジェクトの位置・向きを更新
        UpdateModelMat( animation_time, model_mat );

        // 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）
        glutPostRedisplay();
    }
}


//
//  環境初期化関数
//
void  initEnvironment( void )
{
    // 光源を作成する
    float  light0_position[] = { 0.0, 10.0, 0.0, 1.0 };
    float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
    float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
    glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
    glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
    glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
    glEnable( GL_LIGHT0 );

    // 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 );


    // オブジェクトの読み込み
    object = LoadObj( "car.obj" );
    if ( !object || ( object->num_triangles == 0 ) )
    {
        // 読み込みに失敗したら終了
        printf( "Failed to load the object file." );
        exit( -1 );
    }
    ScaleObj( object, 1.0f, &object_size.x, &object_size.y, &object_size.z );

    // オブジェクト配置機能の初期化
    layout = new ObjectLayout();

    // あらかじめ定義されたオブジェクト配置を設定
    SetupScene( 1 );
}


//
//  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_STENCIL );
    glutInitWindowSize( 640, 640 );
    glutInitWindowPosition( 0, 0 );
    glutCreateWindow("Keyframe Animation");

    // コールバック関数の登録
    glutDisplayFunc( DisplayCallback );
    glutReshapeFunc( ReshapeCallback );
    glutMouseFunc( MouseClickCallback );
    glutMotionFunc( MouseDragCallback );
    glutPassiveMotionFunc( MouseMotionCallback );
    glutKeyboardFunc( KeyboardCallback );
    glutIdleFunc( IdleCallback );
```

```
1079|
1080|    // 環境初期化
1081|    initEnvironment();
1082|
1083|    // GLUTのメインループに処理を移す
1084|    glutMainLoop();
1085|    return 0;
1086|}
1087|
1088|
```