```
//
//  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
//


// 基本的なヘッダファイルのインクルード
#ifdef  _WIN32
    #include <Windows.h>
    #include <mmsystem.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 <Vector3.h>
#include <Point3.h>
#include <Matrix3.h>
#include <Matrix4.h>
#include <Quat4.h>
#include <Color4.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"

// 標準算術関数・定数の定義
#define  _USE_MATH_DEFINES
#include <math.h>



//
//  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;

```

```
113
114 //
115 //  キーフレーム情報に関するグローバル変数
116 //
117
118 // キーフレーム情報
119 struct Keyframe
120 {
121     float     time; // 時刻
122     Point3f   pos;  // 位置
123     Matrix3f  ori;  // 向き
124 };
125
126 // 設定されている全キーフレーム情報 (可変長配列)
127 vector< Keyframe >   keyframes;
128
129
130 //
131 //  アニメーション関連のグローバル変数
132 //
133
134 // アニメーション中かどうかを表すフラグ
135 bool   on_animation = false;
136
137 // 全フレーム描画モード・軌道描画モード
138 bool   on_draw_frames = false;
139 bool   on_draw_trajectory = true;
140
141 // アニメーションの再生時間
142 float  animation_time = 0.0f;
143
144 // 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
145 float  model_mat[ 16 ];
146
147
148
149 //
150 //  キーフレームアニメーションのための処理
151 //
152
153
154 //
155 //  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
156 //
 H3 void  UpdateKeyframes()
158 {
159     Keyframe  key;
160
161     // キーフレーム数を設定
162     int  num_keyframes = layout->GetNumObjects();
163     keyframes.resize( num_keyframes );
164
165     // 各キーフレームの情報を設定
166     for ( int i=0; i<num_keyframes; i++ )
167     {
168         // i番目のキーフレームの時刻を i秒とする
169         key.time = (float) i;
170
171         // 位置・向きを設定
172         key.pos = layout->GetPosition( i );
173         key.ori = layout->GetOrientation( i );
174
175         // キーフレームの情報を設定
176         keyframes[ i ] = key;
177     }
178 }
179
180
181 //
182 //  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)
183 //
184 void  ConvMatToEular( const Matrix3f & m, float & yaw, float & pitch, float & roll )
185 {
186     Vector3f  y_axis, z_axis;
187     m.getColumn( 1, &y_axis );
188     m.getColumn( 2, &z_axis );
189
190     yaw = atan2( z_axis.x, z_axis.z );
191
192     float  cos_yaw = cos( yaw );
193     pitch = atan2( -z_axis.y, sqrt( z_axis.x * z_axis.x + z_axis.z * z_axis.z ) );
194
195     float  sin_yaw = sin( yaw );
196     float  cos_pitch = cos( pitch );
197     roll = atan2( cos_pitch * ( sin_yaw * y_axis.z - cos_yaw * y_axis.x ), y_axis.y );
198 }
199
200
201 //
202 //  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)
203 //
204 void  ConvMatToEular( const float m[9], float & yaw, float & pitch, float & roll )
205 {
206     struct Vector3f
207     {
208         float  x, y, z;
209     };
210     Vector3f  y_axis, z_axis;
211     z_axis.x = m[2];
212     z_axis.y = m[5];
213     z_axis.z = m[8];
214     y_axis.x = m[1];
215     y_axis.y = m[4];
216     y_axis.z = m[7];
217
218     yaw = atan2( z_axis.x, z_axis.z );
219
220     float  cos_yaw = cos( yaw );
221     pitch = atan2( cos_yaw * z_axis.y, fabs( z_axis.z ) );
222
223     float  sin_yaw = sin( yaw );
224     float  cos_pitch = cos( pitch );
```

```
225     roll = atan2( cos_pitch * ( sin_yaw * y_axis.z - cos_yaw * y_axis.x ), y_axis.y );
226 }
227
228
229 //
230 //  物体の位置・向きを更新
231 //  (キーフレーム数、キーフレーム配列、時刻を入力として、その時刻における位置・向きを表す変換行列を出力)
232 //
 H3 void  UpdateModelMat( int num_keyframes, const Keyframe * keyframes, float time, float mat[ 16 ] )
234 {
235     if ( num_keyframes <= 1 )
236         return;
237
238     // 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)
239     int  seg_no = -1;
240     float  t = 0.0f;
241
242     // 指定時刻に対応する区間の番号を取得
243     for ( int i=0; i<num_keyframes-1; i++ )
244     {
245         // 指定時刻が i番目の区間に対応するかを判定
246         if ( ( time >= keyframes[ i ].time ) && ( time <= keyframes[ i+1 ].time ) )
247         {
248             seg_no = i;
249
250             // 区間内での正規化時間を計算
251             t = ( time - keyframes[ i ].time ) / ( keyframes[ i+1 ].time  - keyframes[ i ].time );
252
253             break;
254         }
255     }
256     if ( seg_no == -1 )
257     {
258         // 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
259         if ( time < keyframes[ 0 ].time )
260         {
261             seg_no = 0;
262             t = 0.0f;
263         }
264         // 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
265         else
266         {
267             seg_no = num_keyframes - 2;
268             t = 1.0f;
269         }
270     }
271
272
273     // 指定時刻におけるオブジェクトの位置・向き
274     Vector3f  p;
275     Matrix3f  o;
276
277     // 位置を線形補間により計算
278     if ( pos_method == PI_LINEAR )
279     {
280         // 区間の両端点の位置を取得
281         const Point3f &  p0 = keyframes[ seg_no ].pos;
282         const Point3f &  p1 = keyframes[ seg_no + 1 ].pos;
283
284         // 両端点を線形に補間
285         p.scaleAdd( t, p1 - p0, p0 );
286
287         // 両端点を線形に補間 (下記の書き方でも可)
288 //      p = t * ( p1 - p0 ) + p0;
289     }
290
291     // エルミート補間
292     else if ( pos_method == PI_HERMIT )
293     {
294         // 区間の両端点の位置を取得
295         const Point3f &  p0 = keyframes[ seg_no ].pos;
296         const Point3f &  p1 = keyframes[ seg_no + 1 ].pos;
297
298         // 区間の両端点の傾きを取得
299         Vector3f  v0, v1;
300         const Matrix3f &  o0 = keyframes[ seg_no ].ori;
301         const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;
302         o0.getColumn( 2, &v0 );
303         o1.getColumn( 2, &v1 );
304         v0.negate();
305         v1.negate();
306
307         // Hermite関数の値を計算
308         // 各自実装 (式・プログラムは講義資料を参照)
309
310         // ※レポート課題
311
312
313     }
314
315     // Bezierによる補間
316     else if ( pos_method == PI_BEZIER )
317     {
318         // 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)
319         int  bezier_seg_no = -1;
320         float  s = 0.0f;
321
322         // Bezier補間の区間番号を計算
323         // 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
324         // 区間数×３＋１個のキーフレームが必要
325         bezier_seg_no = ( seg_no == 0 ) ? 0 : (int) floor( seg_no / 3 );
326
327         // 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
328         if ( ( bezier_seg_no + 1 ) * 3 + 1 > num_keyframes )
329         {
330             bezier_seg_no --;
331             s = 1.0f;
332         }
333         // 区間内での正規化時間 (0.0～1.0) を計算
334         else
335         {
336             s = ( time - keyframes[ bezier_seg_no * 3 ].time ) / ( keyframes[ bezier_seg_no * 3 + 3 ].time - keyframes[ bezier_seg_no * 3 ].time );
```

```
        }

        // 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
        if ( num_keyframes < 4 )
        {
            p = keyframes[ 0 ].pos;
        }
        // Bezier補間を計算
        else
        {
            // 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
            const Point3f &  p0 = keyframes[ bezier_seg_no * 3 ].pos;
            const Point3f &  p1 = keyframes[ bezier_seg_no * 3 + 1 ].pos;
            const Point3f &  p2 = keyframes[ bezier_seg_no * 3 + 2 ].pos;
            const Point3f &  p3 = keyframes[ bezier_seg_no * 3 + 3 ].pos;

            // Bezier関数の値を計算
            // 各自実装（式は講義資料を参照）
            // ※ 媒介変数は、t ではなく s を使うことに注意

            // ※レポート課題


        }
    }

    // B-Splineによる補間
    else if ( pos_method == PI_BSPLINE )
    {
        // 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
        int  k0, k1, k2, k3;
        k0 = ( seg_no > 0 ) ? ( seg_no - 1 ) : seg_no;
        k1 = seg_no;
        k2 = seg_no + 1;
        k3 = ( seg_no + 2 == num_keyframes ) ? ( seg_no + 1 ) : ( seg_no + 2);
        const Point3f &  p0 = keyframes[ k0 ].pos;
        const Point3f &  p1 = keyframes[ k1 ].pos;
        const Point3f &  p2 = keyframes[ k2 ].pos;
        const Point3f &  p3 = keyframes[ k3 ].pos;

        // B-Spline 関数の値を計算
        // 各自実装（式は講義資料を参照）

        // ※レポート課題


    }


    // 向きの補間なし
    if ( ori_method == OI_NONE )
    {
        o = layout->GetOrientation( seg_no );
    }

    // 向きをオイラー角で補間
    else if ( ori_method == OI_EULAR )
    {
        // 区間の両端点の向きを取得
        const Matrix3f &  o0 = keyframes[ seg_no ].ori;
        const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;

        // オイラー角に変換
        float  y0, p0, r0;
        float  y1, p1, r1;
        ConvMatToEular( o0, y0, p0, r0 );
        ConvMatToEular( o1, y1, p1, r1 );

        // 各回転角度を線形補間
        float  y, p, r;
        if ( y0 < y1 - M_PI )
            y0 += 2.0f * M_PI;
        else if ( y0 > y1 + M_PI )
            y0 -= 2.0f * M_PI;
        y = ( y1 - y0 ) * t + y0;
        p = ( p1 - p0 ) * t + p0;
        r = ( r1 - r0 ) * t + r0;

        // 行列に変換
        Matrix3f  rot;
        o.rotY( y );
        rot.rotX( p );
        o.mul( o, rot );
        rot.rotZ( r );
        o.mul( o, rot );
    }

    // 向きを四元数と球面線形補間により計算
    else if ( ori_method == OI_QUAT )
    {
        // 区間の両端点の向きを取得
        const Matrix3f &  o0 = keyframes[ seg_no ].ori;
        const Matrix3f &  o1 = keyframes[ seg_no + 1 ].ori;

        // 四元数を使って球面線形補間を計算
        // 各自実装（式は講義資料を参照）
        // vecmath の Quat4f.interpolate() メソッドを使用すれば、容易に計算できる

        // ※レポート課題


    }

    // 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
    Matrix4f  f;
    f.set( o, p, 1.0f );
    f.transpose();
    memcpy( mat, &f.m00, sizeof( float ) * 16 );
}
```

```
449 //
450 //  以下、プログラムのメイン処理
451 //
452
453
454 //
455 //  あらかじめ定義されたオブジェクト配置を設定
456 //
 H3 void  SetupScene( int no )
458 {
459     if ( !layout )
460         return;
461
462     Matrix3f  ori, rot;
463
464     if ( no == 1 )
465     {
466         layout->DeleteAllObjects();
467         layout->AddObject();
468         layout->SetObjectSize( 0, object_size );
469         layout->SetObjectPos( 0, Point3f( -1.0f, 0.5f, -1.5f ) );
470         ori.rotY( M_PI * 0.6f );
471         layout->SetObjectOri( 0, ori );
472
473         layout->AddObject();
474         layout->SetObjectSize( 1, object_size );
475         layout->SetObjectPos( 1, Point3f( 0.0f, 1.0f, 0.0f ) );
476         ori.rotY( M_PI * 1.2f );
477         rot.rotX( M_PI / 3.0f );
478         ori.mul( ori, rot );
479         rot.rotZ( M_PI / 4.0f );
480         ori.mul( ori, rot );
481         layout->SetObjectOri( 1, ori );
482
483         layout->AddObject();
484         layout->SetObjectSize( 2, object_size );
485         layout->SetObjectPos( 2, Point3f( -2.0f, 0.5f, 1.0f ) );
486         ori.rotY( M_PI );
487         rot.rotX( - M_PI / 6.0f );
488         ori.mul( ori, rot );
489         layout->SetObjectOri( 2, ori );
490
491         layout->AddObject();
492         layout->SetObjectSize( 3, object_size );
493         layout->SetObjectPos( 3, Point3f( 0.0f, 0.4f, 1.5f ) );
494         ori.rotY( M_PI * 1.2f );
495         layout->SetObjectOri( 3, ori );
496
497         layout->AddObject();
498         layout->SetObjectSize( 4, object_size );
499         layout->SetObjectPos( 4, Point3f( 1.0f, 0.35f, 1.75f ) );
500         ori.rotY( M_PI * -0.4f );
501         rot.rotX( M_PI / 12.0f );
502         ori.mul( ori, rot );
503         layout->SetObjectOri( 4, ori );
504
505         layout->AddObject();
506         layout->SetObjectSize( 5, object_size );
507         layout->SetObjectPos( 5, Point3f( 1.5f, 0.6f, 0.0f ) );
508         ori.rotY( 0.0f );
509         layout->SetObjectOri( 5, ori );
510
511         layout->AddObject();
512         layout->SetObjectSize( 6, object_size );
513         layout->SetObjectPos( 6, Point3f( 1.0f, 1.0f, -1.0f ) );
514         ori.rotY( M_PI / 3 );
515         layout->SetObjectOri( 6, ori );
516
517         camera_yaw = 15.0f;
518         camera_pitch = -20.0f;
519         camera_distance = 6.0f;
520     }
521     else if ( no == 2 )
522     {
523         layout->DeleteAllObjects();
524
525         layout->AddObject();
526         layout->SetObjectSize( 0, object_size );
527         layout->SetObjectPos( 0, Point3f(-1.0f, 0.5f, 0.0f ) );
528         ori.rotY( M_PI * 0.6f );
529         layout->SetObjectOri( 0, ori );
530
531         layout->AddObject();
532         layout->SetObjectSize( 1, object_size );
533         layout->SetObjectPos( 1, Point3f( 1.0f, 0.5f, 0.0f ) );
534         ori.rotY( M_PI * 1.2f );
535         rot.rotX( M_PI / 3.0f );
536         ori.mul( ori, rot );
537         rot.rotZ( M_PI / 4.0f );
538         ori.mul( ori, rot );
539         layout->SetObjectOri( 1, ori );
540
541         camera_yaw = 0.0f;
542         camera_pitch = -20.0f;
543         camera_distance = 5.0f;
544     }
545
546     // 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
547     UpdateKeyframes();
548 }
549
550
551 //
552 //  格子模様の床を描画
553 //
 H3 void  DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0, float b0, float r1, float g1, float b1 )
555 {
556     int  x, z;
557     float  ox, oz;
558
559     glBegin( GL_QUADS );
560     glNormal3d( 0.0, 1.0, 0.0 );
```

```
561
562     ox = - ( num_x * tile_size ) / 2;
563     for ( x=0; x<num_x; x++ )
564     {
565         oz = - ( num_z * tile_size ) / 2;
566         for ( z=0; z<num_z; z++ )
567         {
568             if ( ( (x + z) % 2 ) == 0 )
569                 glColor3f( r0, g0, b0 );
570             else
571                 glColor3f( r1, g1, b1 );
572
573             glTexCoord2d( 0.0f, 0.0f );
574             glVertex3d( ox, 0.0, oz );
575             glTexCoord2d( 0.0f, 1.0f );
576             glVertex3d( ox, 0.0, oz + tile_size );
577             glTexCoord2d( 1.0f, 1.0f );
578             glVertex3d( ox + tile_size, 0.0, oz + tile_size );
579             glTexCoord2d( 1.0f, 0.0f );
580             glVertex3d( ox + tile_size, 0.0, oz );
581
582             oz += tile_size;
583         }
584         ox += tile_size;
585     }
586     glEnd();
587 }
588
589
590 //
591 //  幾何形状モデル (Obj形状) の影の描画
592 //
H3 593 void  RenderShadow( Obj * obj, const float matrix[ 16 ] )
594 {
595     RenderObjShadow( obj, matrix, light_pos.x, light_pos.y, light_pos.z, shadow_color.x, shadow_color.y, shadow_color.z, shadow_color.w );
596 }
597
598 void  RenderShadow( Obj * obj, Matrix4f & mat )
599 {
600     Matrix4f  frame( mat );
601     frame.transpose();
602     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 );
603 }
604
605
606 //
607 //  テキストを描画
608 //
H3 609 void  DrawTextInformation( int line_no, const char * message )
610 {
611     if ( message == NULL )
612         return;
613
614     // 射影行列を初期化 (初期化の前に現在の行列を退避)
615     glMatrixMode( GL_PROJECTION );
616     glPushMatrix();
617     glLoadIdentity();
618     gluOrtho2D( 0.0, win_width, win_height, 0.0 );
619
620     // 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)
621     glMatrixMode( GL_MODELVIEW );
622     glPushMatrix();
623     glLoadIdentity();
624
625     // Ｚバッファ・ライティングはオフにする
626     glDisable( GL_DEPTH_TEST );
627     glDisable( GL_LIGHTING );
628
629     // メッセージの描画
630     glColor3f( 1.0, 0.0, 0.0 );
631     glRasterPos2i( 16, 40 + 24 * line_no );
632     for ( int i=0; message[i]!='¥0'; i++ )
633         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
634
635     // 設定を全て復元
636     glEnable( GL_DEPTH_TEST );
637     glEnable( GL_LIGHTING );
638     glMatrixMode( GL_PROJECTION );
639     glPopMatrix();
640     glMatrixMode( GL_MODELVIEW );
641     glPopMatrix();
642 }
643
644
645 //
646 //  画面描画時に呼ばれるコールバック関数
647 //
H3 648 void  DisplayCallback( void )
649 {
650     // 画面をクリア
651     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
652
653     // 変換行列を設定 (ワールド座標系→カメラ座標系)
654     glMatrixMode( GL_MODELVIEW );
655     glLoadIdentity();
656     glTranslatef( 0.0,  0.0, - camera_distance );
657     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
658     glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
659     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)
660
661     // 光源の位置を更新
662     float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
663     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
664
665     // 格子模様の床を描画
666     DrawFloor( 1.0f, 50, 50, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );
667
668     // オブジェクトの軌道を描画
669     if ( on_draw_trajectory )
670     {
671         // オブジェクトの軌道を描画
672         float  mat[ 16 ];
```

```
673         glDisable( GL_LIGHTING );
674         glLineWidth( 2.0 );
675         glColor3f( 1.0f, 0.0f, 0.0f );
676         glBegin( GL_LINE_STRIP );
677         for ( float t=0.0f; t<=layout->GetNumObjects()-1.0f+0.001f; t+=0.1f )
678         {
679             UpdateModelMat( keyframes.size(), &keyframes.front(), t, mat );
680             glVertex3f( mat[12], mat[13], mat[14] );
681         }
682         glEnd();
683         glEnable( GL_LIGHTING );
684 
685         // キーフレームのオブジェクトを描画
686         if ( on_animation )
687         {
688             for ( int i=0; i<layout->GetNumObjects(); i++ )
689             {
690                 // オブジェクトを描画
691                 glPushMatrix();
692                     glMultMatrixf( layout->GetFrame( i ) );
693                     RenderObj( object );
694                 glPopMatrix();
695 
696                 // オブジェクトの影を描画
697                 RenderShadow( object, layout->GetFrame( i ) );
698             }
699         }
700     }
701     // 全フレームのオブジェクトを描画
702     else if ( on_draw_frames )
703     {
704         float  mat[ 16 ];
705         for ( float t=0.0f; t<=layout->GetNumObjects()-1.0f+0.001f; t+=0.2f )
706         {
707             UpdateModelMat( keyframes.size(), &keyframes.front(), t, mat );
708 
709             // オブジェクトを描画
710             glPushMatrix();
711                 glMultMatrixf( mat );
712                 RenderObj( object );
713             glPopMatrix();
714 
715             // オブジェクトの影を描画
716             RenderShadow( object, mat );
717         }
718     }
719 
720     // アニメーション中のオブジェクトを描画
721     if ( on_animation && !on_draw_frames )
722     {
723         // アニメーション中のオブジェクトを描画
724         glPushMatrix();
725             glMultMatrixf( model_mat );
726             RenderObj( object );
727         glPopMatrix();
728 
729         // オブジェクトの影を描画
730         RenderShadow( object, model_mat );
731     }
732     // 編集モード中の描画
733     else
734     {
735         // 各オブジェクトを描画
736         for ( int i=0; i<layout->GetNumObjects(); i++ )
737         {
738             glPushMatrix();
739                 glMultMatrixf( layout->GetFrame( i ) );
740                 RenderObj( object );
741             glPopMatrix();
742 
743             // オブジェクトの影を描画
744             RenderShadow( object, layout->GetFrame( i ) );
745         }
746 
747         // 操作用の情報を描画
748         layout->Render();
749     }
750 
751     // 現在の描画モードを表示
752     if ( on_animation )
753         DrawTextInformation( 0, "Animation Mode" );
754     else
755         DrawTextInformation( 0, "Layout Mode" );
756 
757     // 現在の補間モードを表示
758     if ( on_draw_frames || on_draw_trajectory || on_animation )
759     {
760         char  message[ 64 ] = "";
761         sprintf( message, "Position: %s, Orientation: %s", pi_name[ pos_method ], oi_name[ ori_method ] );
762         DrawTextInformation( 1, message );
763     }
764     // 現在の操作モードを表示
765     else
766     {
767         DrawTextInformation( 1, layout->GetOperationMode() );
768     }
769 
770     // 現在の時刻を表示
771     if ( on_animation && !on_draw_frames )
772     {
773         char  message[ 64 ] = "";
774         sprintf( message, "Time: %2.2f", animation_time );
775         DrawTextInformation( 2, message );
776     }
777 
778     // 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
779     glutSwapBuffers();
780 }
781 
782 
783 //
784 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
```

```
785 //
 H3 void  ReshapeCallback( int w, int h )
787 {
788     // 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）
789     glViewport(0, 0, w, h);
790 
791     // カメラ座標系→スクリーン座標系への変換行列を設定
792     glMatrixMode( GL_PROJECTION );
793     glLoadIdentity();
794     gluPerspective( 45, (double)w/h, 1, 500 );
795 
796     // ウィンドウのサイズを記録（テキスト描画処理のため）
797     win_width = w;
798     win_height = h;
799 }
800 
801 
802 //
803 //  マウスクリック時に呼ばれるコールバック関数
804 //
 H3 void  MouseClickCallback( int button, int state, int mx, int my )
806 {
807     // 左ボタンが押されたらドラッグ開始
808     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
809         drag_mouse_l = 1;
810     // 左ボタンが離されたらドラッグ終了
811     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
812         drag_mouse_l = 0;
813 
814     // 右ボタンが押されたらドラッグ開始
815     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
816         drag_mouse_r = 1;
817     // 右ボタンが離されたらドラッグ終了
818     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
819         drag_mouse_r = 0;
820 
821     // 中ボタンが押されたらドラッグ開始
822     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
823         drag_mouse_m = 1;
824     // 中ボタンが離されたらドラッグ終了
825     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
826         drag_mouse_m = 0;
827 
828     // シーン配置機能に左クリックを通知
829     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
830         layout->OnMouseDown( mx, my );
831     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
832         layout->OnMouseUp( mx, my );
833 
834     // 再描画
835     glutPostRedisplay();
836 
837     // 現在のマウス座標を記録
838     last_mouse_x = mx;
839     last_mouse_y = my;
840 }
841 
842 
843 //
844 // マウス移動時に呼ばれるコールバック関数
845 //
 H3 void  MouseMotionCallback( int mx, int my )
847 {
848     // シーン配置機能にマウス移動を通知
849     if ( layout )
850         layout->OnMoveMouse( mx, my );
851 
852     // 再描画
853     glutPostRedisplay();
854 }
855 
856 
857 //
858 //  マウスドラッグ時に呼ばれるコールバック関数
859 //
 H3 void  MouseDragCallback( int mx, int my )
861 {
862     // 右ボタンのドラッグ中は視点を回転する
863     if ( drag_mouse_r )
864     {
865         // 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
866 
867         // マウスの横移動に応じてＹ軸を中心に回転
868         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
869         if ( camera_yaw < 0.0 )
870             camera_yaw += 360.0;
871         else if ( camera_yaw > 360.0 )
872             camera_yaw -= 360.0;
873 
874         // マウスの縦移動に応じてＸ軸を中心に回転
875         camera_pitch -= ( my - last_mouse_y ) * 1.0;
876         if ( camera_pitch < -90.0 )
877             camera_pitch = -90.0;
878         else if ( camera_pitch > 90.0 )
879             camera_pitch = 90.0;
880     }
881     // 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
882     if ( drag_mouse_m )
883     {
884         // 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
885 
886         // マウスの縦移動に応じて距離を移動
887         camera_distance += ( my - last_mouse_y ) * 0.2;
888         if ( camera_distance < 2.0 )
889             camera_distance = 2.0;
890     }
891 
892     // シーン配置機能にマウス移動を通知
893     if ( layout )
894     {
895         layout->Update();
896         layout->OnMoveMouse( mx, my );
```

```
897
898         // 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
899         UpdateKeyframes();
900     }
901
902     // 今回のマウス座標を記録
903     last_mouse_x = mx;
904     last_mouse_y = my;
905
906     // 再描画
907     glutPostRedisplay();
908 }
909
910
911 //
912 //  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
913 //
 H3 void  KeyboardCallback( unsigned char key, int mx, int my )
915 {
916     // s キーでアニメーションの停止・再開
917     if ( key == 's' )
918         on_animation = !on_animation;
919
920     // 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
921     if ( ( key >= '1' ) && ( key <= '9' ) )
922     {
923         SetupScene( key - '0' );
924     }
925
926     // スペースキーでアニメーションを開始
927     if ( key == ' ' )
928     {
929         on_animation = ! on_animation;
930         if ( on_animation )
931             animation_time = 0.0f;
932         on_draw_frames = false;
933     }
934
935     // pキーで位置補間方法を変更
936     if ( key == 'p' )
937     {
938         pos_method = (PositionInterpolationEnum)( ( pos_method + 1 ) % NUM_PI_METHOD );
939     }
940     // oキーで向き補間方法を変更
941     if ( key == 'o' )
942     {
943         ori_method = (OrientationInterpolationEnum)( ( ori_method + 1 ) % NUM_OI_METHOD );
944         if ( ori_method == OI_NONE )
945             ori_method = (OrientationInterpolationEnum)( ori_method + 1 );
946     }
947
948     // fキーで通常・描画全フレーム描画・軌道描画を切り替え
949     if ( key == 'f' )
950     {
951         if ( !on_draw_frames && !on_draw_trajectory )
952             on_draw_trajectory = true;
953         else if ( !on_draw_frames && on_draw_trajectory )
954         {
955             on_draw_frames = true;
956             on_draw_trajectory = false;
957         }
958         else
959             on_draw_frames = false;
960     }
961
962     // aキーでオブジェクトを追加
963     if ( key == 'a' )
964     {
965         layout->AddObject();
966
967         // 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
968         UpdateKeyframes();
969     }
970     // dキーでオブジェクトを削除
971     if ( key == 'd' )
972     {
973         layout->DeleteObject();
974
975         // 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
976         UpdateKeyframes();
977     }
978
979     // tキーで軸の描画モードを変更
980     if ( key == 't' )
981     {
982         bool &  flag = layout->GetRenderOption().enable_xray_mode;
983         flag = ! flag;
984     }
985
986     glutPostRedisplay();
987 }
988
989
990 //
991 //  アイドル時に呼ばれるコールバック関数
992 //
 H3 void  IdleCallback( void )
994 {
995     // アニメーション処理
996     if ( on_animation )
997     {
998 #ifdef  WIN32
999         // 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
1000         static DWORD  last_time = 0;
1001         DWORD  curr_time = timeGetTime();
1002         float  delta = ( curr_time - last_time ) * 0.001f;
1003         if ( delta > 0.1f )
1004             delta = 0.1f;
1005         last_time = curr_time;
1006         animation_time += delta;
1007 #else
1008         // 固定のΔｔを使用
```

```
1009|        animation_time += 0.02f;
1010|#endif
1011|        // 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）
1012|        if ( animation_time >= layout->GetNumObjects() - 1 )
1013|            animation_time = 0.0f;
1014|
1015|        // アニメーション中のオブジェクトの位置・向きを更新
1016|        UpdateModelMat( keyframes.size(), &keyframes.front(), animation_time, model_mat );
1017|
1018|        // 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）
1019|        glutPostRedisplay();
1020|    }
1021|}
1022|
1023|
1024|//
1025|//  環境初期化関数
1026|//
  H3|void  initEnvironment( void )
1028|{
1029|    // 光源を作成する
1030|    float  light0_position[] = { 0.0, 10.0, 0.0, 1.0 };
1031|    float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
1032|    float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
1033|    float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
1034|    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
1035|    glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
1036|    glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
1037|    glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
1038|    glEnable( GL_LIGHT0 );
1039|
1040|    // 光源計算を有効にする
1041|    glEnable( GL_LIGHTING );
1042|
1043|    // 物体の色情報を有効にする
1044|    glEnable( GL_COLOR_MATERIAL );
1045|
1046|    // Ｚテストを有効にする
1047|    glEnable( GL_DEPTH_TEST );
1048|
1049|    // 背面除去を有効にする
1050|    glCullFace( GL_BACK );
1051|    glEnable( GL_CULL_FACE );
1052|
1053|    // 背景色を設定
1054|    glClearColor( 0.5, 0.5, 0.8, 0.0 );
1055|
1056|
1057|    // オブジェクトの読み込み
1058|    object = LoadObj( "car.obj" );
1059|    if ( !object || ( object->num_triangles == 0 ) )
1060|    {
1061|        // 読み込みに失敗したら終了
1062|        printf( "Failed to load the object file." );
1063|        exit( -1 );
1064|    }
1065|    ScaleObj( object, 1.0f, &object_size.x, &object_size.y, &object_size.z );
1066|
1067|    // オブジェクト配置機能の初期化
1068|    layout = new ObjectLayout();
1069|
1070|    // あらかじめ定義されたオブジェクト配置を設定
1071|    SetupScene( 1 );
1072|}
1073|
1074|
1075|//
1076|//  メイン関数（プログラムはここから開始）
1077|//
  H3|int  main( int argc, char ** argv )
1079|{
1080|    // GLUTの初期化
1081|    glutInit( &argc, argv );
1082|    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
1083|    glutInitWindowSize( 640, 640 );
1084|    glutInitWindowPosition( 0, 0 );
1085|    glutCreateWindow("Keyframe Animation");
1086|
1087|    // コールバック関数の登録
1088|    glutDisplayFunc( DisplayCallback );
1089|    glutReshapeFunc( ReshapeCallback );
1090|    glutMouseFunc( MouseClickCallback );
1091|    glutMotionFunc( MouseDragCallback );
1092|    glutPassiveMotionFunc( MouseMotionCallback );
1093|    glutKeyboardFunc( KeyboardCallback );
1094|    glutIdleFunc( IdleCallback );
1095|
1096|    // 環境初期化
1097|    initEnvironment();
1098|
1099|    // GLUTのメインループに処理を移す
1100|    glutMainLoop();
1101|    return 0;
1102|}
1103|
1104|
```