```
//
//  コンピュータグラフィックス特論II
//  影の描画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>

// ヘッダファイルのインクルード
#include "Obj.h"
#include "bitmap.h"

#ifdef  WIN32
    #include <mmsystem.h>
#endif


// 視点操作のためのグローバル変数

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

// カメラの回転のための変数
float   camera_yaw = 15.0f; // 30.0;     // Ｙ軸を中心とする回転角度
float   camera_pitch = -20.0f; // -30.0;   // Ｘ軸を中心とする回転角度
float   camera_distance = 15.0; // 中心からカメラの距離

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

// 物体の回転アニメーションのための変数
bool    on_animation = true;


// 影の描画に関するグローバル変数

// 影の描画方法
enum  ShadowModeEnum
{
    SHADOW_NONE,
    SHADOW_TEXTURE,
    SHADOW_PROJECTION,
    SHADOW_VOLUME,
    NUM_SHADOW_MODE
};

// 現在の影の描画方法
ShadowModeEnum    shadow_mode = SHADOW_TEXTURE; // SHADOW_PROJECTION;

// 影の描画方法の名前
const char *  shadow_mode_name[ NUM_SHADOW_MODE ] = {
    "No Shadow", "Texture Shadow", "Polygon Projection Shadow", "Shadow Volume" };

// 点光源の位置 (影の投影方向)
Vector    light_pos;

// 影のテクスチャ画像の番号
unsigned int    shadow_texture = 0;

// ポリゴン投影による影の描画色
float    shadow_color_rgb = 0.2f;
float    shadow_color_alpha = 0.5f;

// 影の描画の設定 (ブレンディングやステンシルバッファの使用の有無の切り替え)
bool    shadow_blend_off = false;
bool    shadow_stencil_off = false;


// 幾何形状オブジェクトに関するグローバル変数

// 幾何形状オブジェクトの数
#define  NUM_OBJECTS  2

// 幾何形状オブジェクト
Obj *     object[ NUM_OBJECTS ] = { NULL, NULL };

// 位置
Vector    object_pos[ NUM_OBJECTS];

// 水平向き
float     object_ori[ NUM_OBJECTS ];

// 大きさ (テクスチャマッピングによる影の描画用)
Vector    object_size[ NUM_OBJECTS ];

// 描画フラグ
bool      object_display[ NUM_OBJECTS ] = { true, true };

// アニメーションフラグ
bool      object_animation[ NUM_OBJECTS ] = { false, true };



////////////////////////////////////////////////////////////////////////////////
//
//  テクスチャマッピングによる影の描画
//


//
//  影のテクスチャ画像の読み込み・設定
//
bool  LoadShadowTexture( const char * filename = NULL )
{
    // デフォルトのテクスチャ画像のファイル名
```

```
109     static const char *  default_filename = "shadow.bmp";
110
111     // 読み込みに失敗したかどうかのフラグ
112     static bool  try_default_file = false;
113
114     // ファイル名が省略されたらデフォルトの画像ファイルを使用
115     if ( filename == NULL )
116     {
117         // 既にデフォルトの画像ファイルの読み込みに失敗していれば終了
118         if ( try_default_file )
119             return  false;
120
121         // デフォルトの画像ファイル名を設定
122         filename = default_filename;
123     }
124
125     // テクスチャ画像の読み込み
126     int  result;
127     unsigned char *  shadow_image = NULL;
128     int  shadow_width, shadow_height;
129     result = loadBitmap( filename, &shadow_image, &shadow_width, &shadow_height );
130
131     // 読み込みに失敗したら終了
132     if ( result != 0 )
133     {
134         if ( filename == default_filename )
135             try_default_file = true;
136         return  false;
137     }
138
139     // テクスチャマッピングの設定
140     glGenTextures( 1, &shadow_texture );
141     glBindTexture( GL_TEXTURE_2D, shadow_texture );
142     glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, shadow_width, shadow_height, 0,
143                   GL_RGB, GL_UNSIGNED_BYTE, shadow_image );
144     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
145     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
146     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
147     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
148
149     return  true;
150 }
151
152
153 //
154 //  テクスチャマッピングによる影の描画
155 //
 H3 void  RenderTextureShadow( float obj_matrix[ 16 ], float size_x, float size_z, float shadow_y )
157 {
158     // テクスチャ画像の読み込みと設定
159     // テクスチャ画像が読み込まれていなければ、最初にデフォルトの画像を読み込み
160     if ( shadow_texture == 0 )
161     {
162         // 読み込みに失敗したら終了
163         if ( ! LoadShadowTexture() )
164             return;
165     }
166
167     // 影のテクスチャ画像を描画する四隅の水平位置＋高さ
168     float  x0, z0, x1, z1, x2, z2, x3, z3, y;
169
170     // ※レポート課題
171
172
173     // 現在の描画設定を取得（描画終了後に元の設定に戻すため）
174     GLboolean  b_texture, b_blend, b_lighting;
175     glGetBooleanv( GL_TEXTURE_2D, &b_texture );
176     glGetBooleanv( GL_BLEND, &b_blend );
177     glGetBooleanv( GL_LIGHTING, &b_lighting );
178
179     // 描画設定の変更
180     glDisable( GL_LIGHTING );
181     glEnable( GL_BLEND );
182     glEnable( GL_TEXTURE_2D );
183
184     // 動作確認のための描画設定の変更
185     if ( shadow_blend_off )
186         glDisable( GL_BLEND );
187
188     // テクスチャマッピングの設定
189     glBindTexture( GL_TEXTURE_2D, shadow_texture );
190     glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL );
191
192     // ※レポート課題
193
194
195     // 描画設定を復元
196     if ( b_lighting )
197         glEnable( GL_LIGHTING );
198     if ( !b_blend )
199         glDisable( GL_BLEND );
200     if ( !b_texture )
201         glDisable( GL_TEXTURE_2D );
202 }
203
204
205
206 ////////////////////////////////////////////////////////////////////////////
207 //
208 //  ポリゴン投影による影の描画
209 //
210
211
212 //
213 //  幾何形状モデル（Obj形状）の描画（固定色で描画）
214 //
 H3 void  RenderObjUnicolor( const Obj * obj, float color_r, float color_g, float color_b, float color_a )
216 {
```

```
217     // ※レポート課題
218
219 }
220
221
222 //
223 //  ポリゴン投影による影の描画
224 //
225 void  RenderProjectionShadow( const Obj * obj, const float obj_matrix[ 16 ], const Vector & light_dir, float color_r, float color_g, float
    color_b, float color_a )
226 {
227     // 現在の描画設定を取得（描画終了後に元の設定に戻すため）
228     GLboolean  b_cull_face, b_blend, b_lighting, b_stencil;
229     glGetBooleanv( GL_CULL_FACE, &b_cull_face );
230     glGetBooleanv( GL_BLEND, &b_blend );
231     glGetBooleanv( GL_LIGHTING, &b_lighting );
232     glGetBooleanv( GL_STENCIL_TEST, &b_stencil );
233
234     // 描画設定の変更
235     if ( b_lighting )
236         glDisable( GL_LIGHTING );
237     if ( !b_cull_face )
238         glEnable( GL_CULL_FACE );
239     if ( !b_blend )
240         glEnable( GL_BLEND );
241
242     // ブレンディングの設定
243     glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
244
245     // ステンシルバッファの設定
246
247     // ※レポート課題
248
249
250     // 動作確認のための描画設定の変更
251     if ( shadow_blend_off )
252         glDisable( GL_BLEND );
253     if ( shadow_stencil_off )
254         glDisable( GL_STENCIL_TEST );
255
256
257     // 現在の変換行列を一時保存
258     glPushMatrix();
259
260     // ポリゴンモデルを地面に投影して描画するための変換行列を設定
261     //（この時点で、ワールド座標系からカメラ座標系への変換行列が設定されているものとする）
262
263     // ※レポート課題
264
265
266     // 影の描画、幾何形状モデルを指定色で描画
267     RenderObjUnicolor( obj, color_r, color_g, color_b, color_a );
268
269     // 一時保存しておいた変換行列を復元
270     glPopMatrix();
271
272     // 描画設定を復元
273     if ( b_lighting )
274         glEnable( GL_LIGHTING );
275     if ( b_cull_face )
276         glEnable( GL_CULL_FACE );
277     if ( !b_blend )
278         glDisable( GL_BLEND );
279     if ( !b_stencil )
280         glDisable( GL_STENCIL_TEST );
281 }
282
283
284
285 ////////////////////////////////////////////////////////////////////////////
286 //
287 //  シャドウ・ヴォリュームによる影の描画
288 //
289
290
291 //
292 //  シャドウ・ヴォリュームを描画
293 //
294 void  DrawShadowVolume( const Obj * obj, const float model_world[ 16 ], const Vector & light_vec )
295 {
296 }
297
298
299 //
300 //  シャドウ・ヴォリュームを塗りつぶす
301 //
 H3 void  FillShadowVolume( float color_r, float color_g, float color_b, float color_a )
303 {
304 }
305
306
307 ////////////////////////////////////////////////////////////////////////////
308 //
309 //  以下、プログラムのメイン処理
310 //
311
312
313 //
314 //  幾何形状オブジェクトの読み込み・初期化
315 //
 H3 void  InitObjects()
317 {
318     // 幾何形状オブジェクトの読み込み
319     if ( !object[ 0 ] )
320     {
321         object[ 0 ] = LoadObj( "Car.obj" );
322         if ( object[ 0 ] )
323
```

```
        {
            ScaleObj( object[ 0 ], 5.0f, &object_size[ 0 ].x, &object_size[ 0 ].y, &object_size[ 0 ].z );
            object_size[ 0 ].x *= 1.5;
            object_size[ 0 ].z *= 1.5;
        }
    }
    if ( !object[ 1 ] )
    {
        object[ 1 ] = LoadObj( "Pyramid.obj" );
        if ( object[ 1 ] )
        {
            ScaleObj( object[ 1 ], 2.0f, &object_size[ 1 ].x, &object_size[ 1 ].y, &object_size[ 1 ].z );
            object_size[ 1 ].x *= 1.5;
            object_size[ 1 ].z *= 1.5;
        }
    }

    // 幾何形状オブジェクトの初期位置・向きの設定
    object_pos[ 0 ].x = 0.0f;
    object_pos[ 0 ].y = 2.0f;
    object_pos[ 0 ].z = 0.0f;
    object_ori[ 0 ] = 180.0f;

    object_pos[ 1 ].x = 2.5f;
    object_pos[ 1 ].y = 4.0f;
    object_pos[ 1 ].z = 1.5f;
    object_ori[ 1 ] = 0.0f;
}



//
//  格子模様の床を描画
//
H3 void  DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0, float b0, float r1, float g1, float b1 )
{
    int  x, z;
    float  ox, oz;

    glBegin( GL_QUADS );
    glNormal3d( 0.0, 1.0, 0.0 );

    ox = - ( num_x * tile_size ) / 2;
    for ( x=0; x<num_x; x++ )
    {
        oz = - ( num_z * tile_size ) / 2;
        for ( z=0; z<num_z; z++ )
        {
            if ( ( (x + z) % 2 ) == 0 )
                glColor3f( r0, g0, b0 );
            else
                glColor3f( r1, g1, b1 );

            glTexCoord2d( 0.0f, 0.0f );
            glVertex3d( ox, 0.0, oz );
            glTexCoord2d( 0.0f, 1.0f );
            glVertex3d( ox, 0.0, oz + tile_size );
            glTexCoord2d( 1.0f, 1.0f );
            glVertex3d( ox + tile_size, 0.0, oz + tile_size );
            glTexCoord2d( 1.0f, 0.0f );
            glVertex3d( ox + tile_size, 0.0, oz );

            oz += tile_size;
        }
        ox += tile_size;
    }
    glEnd();
}


//
//  文字情報（現在のモード名）を描画
//
H3 void  DrawTextInformation( const char * message )
{
    // 表示するメッセージ
    int   i;

    // 射影行列を初期化（初期化の前に現在の行列を退避）
    glMatrixMode( GL_PROJECTION );
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D( 0.0, win_width, win_height, 0.0 );

    // 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）
    glMatrixMode( GL_MODELVIEW );
    glPushMatrix();
    glLoadIdentity();

    // Ｚバッファ・ライティングはオフにする
    glDisable( GL_DEPTH_TEST );
    glDisable( GL_LIGHTING );

    // メッセージの描画
    glColor3f( 1.0, 0.0, 0.0 );
    glRasterPos2i( 16, 16 + 18 );
    for ( i=0; message[i]!='¥0'; i++ )
        glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );

    // 設定を全て復元
    glEnable( GL_DEPTH_TEST );
    glEnable( GL_LIGHTING );
    glMatrixMode( GL_PROJECTION );
    glPopMatrix();
    glMatrixMode( GL_MODELVIEW );
    glPopMatrix();
}
```

```
432
433 //
434 //  画面描画時に呼ばれるコールバック関数
435 //
 H3 void  DisplayCallback()
437 {
438     // 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）
439     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
440
441     // 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）を設定
442     glMatrixMode( GL_MODELVIEW );
443     glLoadIdentity();
444     glTranslatef( 0.0,  0.0, - camera_distance );
445     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
446     glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
447
448     // 光源位置を設定
449 //  float  light0_position[] = { light_pos.x * 10.0f, light_pos.y * 10.0f, light_pos.z * 10.0f, 1.0 };
450     float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
451     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
452
453     // 格子模様の床を描画
454     DrawFloor( 1.5f, 10, 10, 1.0, 1.0, 1.0, 1.0, 0.8, 0.8 );
455
456     // それぞれの幾何形状モデル＋影を描画
457     for ( int i=0; i<NUM_OBJECTS; i++ )
458     {
459         if ( ! object[ i ] )
460             continue;
461         if ( ! object_display[ i ] )
462             continue;
463
464         // モデル座標系からワールド座標系への変換行列を計算
465         float  matrix[ 16 ];
466
467         // 変換行列を計算するために、現在の変換行列を一時保存
468         glPushMatrix();
469
470             // 変換行列を単位行列で初期化
471             glLoadIdentity();
472
473             // モデルの位置・水平向きの変換行列をかける
474             glTranslatef( object_pos[ i ].x, object_pos[ i ].y, object_pos[ i ].z );
475             glRotatef( object_ori[ i ], 0.0f, 1.0f, 0.0f );
476
477             // 変換行列を取得（モデル座標系からワールド座標系への変換行列）
478             glGetFloatv( GL_MODELVIEW_MATRIX, matrix );
479
480         // 一時保存しておいた変換行列を復元
481         glPopMatrix();
482
483
484         glPushMatrix();
485
486             // 現在の変換行列に、モデル座標系からワールド座標系への変換行列をかける
487             glMultMatrixf( matrix );
488
489             // オブジェクトを描画
490             RenderObj( object[ i ] );
491
492         glPopMatrix();
493
494         // 影を描画
495         if ( shadow_mode == SHADOW_PROJECTION )
496         {
497             // ポリゴン投影による影の描画
498             RenderProjectionShadow( object[ i ], matrix, light_pos, shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_alph
    a );
499         }
500         else if ( shadow_mode == SHADOW_TEXTURE )
501         {
502             // 影テクスチャを描画する高さ（地面や他の影と重ならないように微妙に高さを変化させる）
503             float  height = 0.01f * ( i + 1 );
504
505             // テクスチャマッピングによる影の描画
506             RenderTextureShadow( matrix, object_size[ i ].x, object_size[ i ].z, height );
507         }
508         else if ( shadow_mode == SHADOW_VOLUME )
509         {
510             // １つ目のオブジェクトは、ポリゴン投影による影の描画
511             if ( i == 0 )
512                 RenderProjectionShadow( object[ i ], matrix, light_pos, shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_
    alpha );
513             else
514             // ２つ目のオブジェクトは、シャドウ・ヴォリュームによる影の描画
515             {
516                 DrawShadowVolume( object[ i ], matrix, light_pos );
517                 FillShadowVolume( shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_alpha );
518             }
519
520         }
521     }
522
523     // 文字情報（現在のモード名）を描画
524     DrawTextInformation( shadow_mode_name[ shadow_mode ] );
525
526     // 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
527     glutSwapBuffers();
528 }
529
530
531 //
532 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
533 //
 H3 void  ReshapeCallback( int w, int h )
535 {
536     // 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）
537     glViewport(0, 0, w, h);
```

```
538
539     // カメラ座標系→スクリーン座標系への変換行列を設定
540     glMatrixMode( GL_PROJECTION );
541     glLoadIdentity();
542     gluPerspective( 45, (double)w/h, 1, 500 );
543
544     // ウィンドウのサイズを記録（テキスト描画処理のため）
545     win_width = w;
546     win_height = h;
547 }
548
549
550 //
551 // マウスクリック時に呼ばれるコールバック関数
552 //
 H3 void  MouseClickCallback( int button, int state, int mx, int my )
554 {
555     // 右ボタンが押されたらドラッグ開始
556     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
557         drag_mouse_r = 1;
558     // 右ボタンが離されたらドラッグ終了
559     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
560         drag_mouse_r = 0;
561
562     // 左ボタンが押されたらドラッグ開始
563     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
564         drag_mouse_l = 1;
565     // 左ボタンが離されたらドラッグ終了
566     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
567         drag_mouse_l = 0;
568
569     // 中ボタンが押されたらドラッグ開始
570     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
571         drag_mouse_m = 1;
572     // 中ボタンが離されたらドラッグ終了
573     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
574         drag_mouse_m = 0;
575
576     // 現在のマウス座標を記録
577     last_mouse_x = mx;
578     last_mouse_y = my;
579 }
580
581
582 //
583 // マウスドラッグ時に呼ばれるコールバック関数
584 //
 H3 void  MouseDragCallback( int mx, int my )
586 {
587     // 右ボタンのドラッグ中は視点を回転する
588     if ( drag_mouse_r && !( glutGetModifiers() & GLUT_ACTIVE_CTRL ) )
589     {
590         // 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
591
592         // マウスの横移動に応じてＹ軸を中心に回転
593         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
594         if ( camera_yaw < 0.0 )
595             camera_yaw += 360.0;
596         else if ( camera_yaw > 360.0 )
597             camera_yaw -= 360.0;
598
599         // マウスの縦移動に応じてＸ軸を中心に回転
600         camera_pitch -= ( my - last_mouse_y ) * 1.0;
601         if ( camera_pitch < -90.0 )
602             camera_pitch = -90.0;
603         else if ( camera_pitch > 90.0 )
604             camera_pitch = 90.0;
605     }
606
607     // 中ボタン（もしくは ctrlキーを押しながら右ボタン）のドラッグ中は視点とカメラの距離を変更する
608     if ( drag_mouse_m || ( drag_mouse_r && ( glutGetModifiers() & GLUT_ACTIVE_CTRL ) ) )
609     {
610         // 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
611
612         // マウスの縦移動に応じて距離を移動
613         camera_distance += ( my - last_mouse_y ) * 0.2;
614         if ( camera_distance < 5.0 )
615             camera_distance = 5.0;
616     }
617
618     // 左ボタンのドラッグ中は影の方向を変更する
619     if ( drag_mouse_l )
620     {
621         // 前回のマウス座標と今回のマウス座標の差に応じて影の方向を変更
622
623         // マウスの縦移動に応じて距離を移動
624         float  delta_x = ( mx - last_mouse_x ) * 0.05f;
625         float  delta_z = ( my - last_mouse_y ) * 0.05f;
626
627         light_pos.x += delta_x;
628         if ( light_pos.x < -5.0f )
629             light_pos.x = -5.0f;
630         else if ( light_pos.x > 5.0f )
631             light_pos.x = 5.0f;
632
633         light_pos.z += delta_z;
634         if ( light_pos.z < -5.0f )
635             light_pos.z = -5.0f;
636         else if ( light_pos.z > 5.0f )
637             light_pos.z = 5.0f;
638     }
639
640     // 今回のマウス座標を記録
641     last_mouse_x = mx;
642     last_mouse_y = my;
643
644     // 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）
645     glutPostRedisplay();
```

```
646 }
647
648
649 //
650 //  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
651 //
 H3 void  KeyboardCallback( unsigned char key, int mx, int my )
653 {
654     // mキーで影の描画モードを変更
655     if ( key == 'm' )
656     {
657         shadow_mode = (ShadowModeEnum)( ( shadow_mode + 1 ) % NUM_SHADOW_MODE );
658         glutPostRedisplay();
659     }
660
661     // oキーでオブジェクトの描画を変更
662     if ( key == 'o' )
663     {
664         if ( object_display[ 0 ] && object_display[ 1 ] )
665         {
666             object_display[ 0 ] = false;
667         }
668         else if ( !object_display[ 0 ] && object_display[ 1 ] )
669         {
670             object_display[ 0 ] = true;
671             object_display[ 1 ] = false;
672         }
673         else
674         {
675             object_display[ 1 ] = true;
676         }
677     }
678
679     // sキーでオブジェクトのアニメーションを変更
680     if ( key == 's' )
681     {
682         if ( object_animation[ 0 ] && object_animation[ 1 ] )
683         {
684             object_animation[ 0 ] = false;
685             object_animation[ 1 ] = false;
686         }
687         else if ( !object_animation[ 0 ] && object_animation[ 1 ] )
688         {
689             object_animation[ 0 ] = true;
690             object_animation[ 1 ] = true;
691         }
692         else
693         {
694             object_animation[ 0 ] = false;
695             object_animation[ 1 ] = true;
696         }
697         on_animation = ( object_animation[ 0 ] && object_animation[ 1 ] );
698     }
699
700     // dキーでデバッグモードを変更
701     if ( key == 'd' )
702     {
703         if ( !shadow_blend_off && !shadow_stencil_off )
704         {
705             shadow_blend_off = true;
706         }
707         else if ( shadow_blend_off && !shadow_stencil_off )
708         {
709             shadow_blend_off = false;
710             shadow_stencil_off = true;
711         }
712         else
713         {
714             shadow_blend_off = false;
715             shadow_stencil_off = false;
716         }
717     }
718
719     // 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）
720     glutPostRedisplay();
721 }
722
723
724 //
725 //  アイドル時に呼ばれるコールバック関数
726 //
 H3 void  IdleCallback( void )
728 {
729     if ( on_animation )
730     {
731 #ifdef  WIN32
732         // 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
733         static DWORD  last_time = 0;
734         DWORD  curr_time = timeGetTime();
735         float  delta = ( curr_time - last_time ) * 0.001f;
736         if ( delta > 0.01f )
737             delta = 0.01f;
738         last_time = curr_time;
739 #else
740         // 固定のΔｔを使用
741         delta = 0.01f;
742 #endif
743         // オブジェクトを回転
744         for ( int i=0; i<NUM_OBJECTS; i++ )
745         {
746             if ( !object_animation[ i ] )
747                 continue;
748
749             object_ori[ i ] += 100.0 * delta;
750             if ( object_ori[ i ] > 360.0f )
751                 object_ori[ i ] -= 360.0f;
752             if ( object_ori[ i ] < 0.0f )
753                 object_ori[ i ] += 360.0f;
```

```
754         }
755
756         // 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）
757         glutPostRedisplay();
758     }
759 }
760
761
762 //
763 //  環境初期化関数
764 //
 H3 void  InitEnvironment()
766 {
767     // 光源を作成する
768     float  light0_position[] = { 10.0, 10.0, 10.0, 1.0 };
769     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
770     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
771     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
772     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
773     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
774     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
775     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
776     glEnable( GL_LIGHT0 );
777
778     // 光源計算を有効にする
779     glEnable( GL_LIGHTING );
780
781     // 物体の色情報を有効にする
782     glEnable( GL_COLOR_MATERIAL );
783
784     // Ｚテストを有効にする
785     glEnable( GL_DEPTH_TEST );
786
787     // 背面除去を有効にする
788     glCullFace( GL_BACK );
789     glEnable( GL_CULL_FACE );
790
791     // 背景色を設定
792     glClearColor( 0.5, 0.5, 0.8, 0.0 );
793
794     // 光源位置を初期化
795     light_pos.x = 4.0f;
796     light_pos.y = 5.0f;
797     light_pos.z = 1.0f;
798 }
799
800
801 //
802 //  メイン関数（プログラムはここから開始）
803 //
 H3 int  main( int argc, char ** argv )
805 {
806     // GLUTの初期化
807     glutInit( &argc, argv );
808     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
809     glutInitWindowSize( 640, 640 );
810     glutInitWindowPosition( 0, 0 );
811     glutCreateWindow("Shadow Sample");
812
813     // コールバック関数の登録
814     glutDisplayFunc( DisplayCallback );
815     glutReshapeFunc( ReshapeCallback );
816     glutMouseFunc( MouseClickCallback );
817     glutMotionFunc( MouseDragCallback );
818     glutKeyboardFunc( KeyboardCallback );
819     glutIdleFunc( IdleCallback );
820
821     // 環境初期化
822     InitEnvironment();
823
824     // オブジェクトの読み込み・初期化
825     InitObjects();
826
827     // GLUTのメインループに処理を移す
828     glutMainLoop();
829     return 0;
830 }
831
832
```