```
//
//  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
//

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

// 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"


// 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
    static const char *  default_filename = "shadow.bmp";

    // 読み込みに失敗したかどうかのフラグ
    static bool  try_default_file = false;

```

```
113     // 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
114     if ( filename == NULL )
115     {
116         // 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
117         if ( try_default_file )
118             return  false;
119 
120         // デフォルトの画像ファイル名を設定
121         filename = default_filename;
122     }
123 
124     // テクスチャ画像の読み込み
125     int  result;
126     unsigned char *  shadow_image = NULL;
127     int  shadow_width, shadow_height;
128     result = loadBitmap( filename, &shadow_image, &shadow_width, &shadow_height );
129 
130     // 読み込みに失敗したら終了
131     if ( result != 0 )
132     {
133         if ( filename == default_filename )
134             try_default_file = true;
135         return  false;
136     }
137 
138     // テクスチャマッピングの設定
139     glGenTextures( 1, &shadow_texture );
140     glBindTexture( GL_TEXTURE_2D, shadow_texture );
141     glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, shadow_width, shadow_height, 0,
142                 GL_RGB, GL_UNSIGNED_BYTE, shadow_image );
143     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
144     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
145     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
146     glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
147 
148     return  true;
149 }
150 
151 
152 //
153 //  テクスチャマッピングによる影の描画
154 //
 H3 void  RenderTextureShadow( float obj_matrix[ 16 ], float size_x, float size_z, float shadow_y )
156 {
157     // テクスチャ画像の読み込みと設定
158     // 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
159     if ( shadow_texture == 0 )
160     {
161         // 読み込みに失敗したら終了
162         if ( ! LoadShadowTexture() )
163             return;
164     }
165 
166     // 影のテクスチャ画像を描画する四隅の水平位置＋高さ
167     float  x0, z0, x1, z1, x2, z2, x3, z3, y;
168 
169     // ※レポート課題
170 
171 
172     // 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）
173     GLboolean  b_texture, b_blend, b_lighting;
174     glGetBooleanv( GL_TEXTURE_2D, &b_texture );
175     glGetBooleanv( GL_BLEND, &b_blend );
176     glGetBooleanv( GL_LIGHTING, &b_lighting );
177 
178     // 描画設定の変更
179     glDisable( GL_LIGHTING );
180     glEnable( GL_BLEND );
181     glEnable( GL_TEXTURE_2D );
182 
183     // 動作確認のための描画設定の変更
184     if ( shadow_blend_off )
185         glDisable( GL_BLEND );
186 
187     // テクスチャマッピングの設定
188     glBindTexture( GL_TEXTURE_2D, shadow_texture );
189     glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL );
190 
191     // ※レポート課題
192 
193 
194     // 描画設定を復元
195     if ( b_lighting )
196         glEnable( GL_LIGHTING );
197     if ( !b_blend )
198         glDisable( GL_BLEND );
199     if ( !b_texture )
200         glDisable( GL_TEXTURE_2D );
201 }
202 
203 
204 
205 ////////////////////////////////////////////////////////////////////////////
206 //
207 //  ポリゴン投影による影の描画
208 //
209 
210 
211 //
212 //  幾何形状モデル（Obj形状）の描画（固定色で描画）
213 //
 H3 void  RenderObjUnicolor( const Obj * obj, float color_r, float color_g, float color_b, float color_a )
215 {
216     // ※レポート課題
217 
218 
219 }
220 
221 
222 //
223 //  ポリゴン投影による影の描画
224 //
```

```
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 )
{
    // 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）
    GLboolean  b_cull_face, b_blend, b_lighting, b_stencil;
    glGetBooleanv( GL_CULL_FACE, &b_cull_face );
    glGetBooleanv( GL_BLEND, &b_blend );
    glGetBooleanv( GL_LIGHTING, &b_lighting );
    glGetBooleanv( GL_STENCIL_TEST, &b_stencil );

    // 描画設定の変更
    if ( b_lighting )
        glDisable( GL_LIGHTING );
    if ( !b_cull_face )
        glEnable( GL_CULL_FACE );
    if ( !b_blend )
        glEnable( GL_BLEND );

    // ブレンディングの設定
    glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );

    // ステンシルバッファの設定

    // ※レポート課題


    // 動作確認のための描画設定の変更
    if ( shadow_blend_off )
        glDisable( GL_BLEND );
    if ( shadow_stencil_off )
        glDisable( GL_STENCIL_TEST );


    // 現在の変換行列を一時保存
    glPushMatrix();

    // 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
    //（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）

    // ※レポート課題


    // 影の描画、幾何形状モデルを指定色で描画
    RenderObjUnicolor( obj, color_r, color_g, color_b, color_a );

    // 一時保存しておいた変換行列を復元
    glPopMatrix();

    // 描画設定を復元
    if ( b_lighting )
        glEnable( GL_LIGHTING );
    if ( b_cull_face )
        glEnable( GL_CULL_FACE );
    if ( !b_blend )
        glDisable( GL_BLEND );
    if ( !b_stencil )
        glDisable( GL_STENCIL_TEST );
}



////////////////////////////////////////////////////////////////////////////////
//
//  シャドウ・ヴォリュームによる影の描画
//


//
//  シャドウ・ヴォリュームを描画
//
void  DrawShadowVolume( const Obj * obj, const float model_world[ 16 ], const Vector & light_vec )
{
    // 省略
}

//
//  シャドウ・ヴォリュームを塗りつぶす
//
### void  FillShadowVolume( float color_r, float color_g, float color_b, float color_a )
{
    // 省略
}



////////////////////////////////////////////////////////////////////////////////
//
//  以下、プログラムのメイン処理
//


//
//  幾何形状オブジェクトの読み込み・初期化
//
### void  InitObjects()
{
    // 幾何形状オブジェクトの読み込み
    if ( !object[ 0 ] )
    {
        object[ 0 ] = LoadObj( "Car.obj" );
        if ( object[ 0 ] )
        {
            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
//
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
//
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();
}


//
//  画面描画時に呼ばれるコールバック関数
//
void  DisplayCallback()
{
    // 画面をクリア（ピクセルデータとＺバッファの両方をクリア）
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );

    // 視点パラメタに応じて変換行列（カメラ座標系からワールド座標系への変換行列）を設定
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    glTranslatef( 0.0,  0.0, - camera_distance );
    glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
    glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );

```

```
448     // 光源位置を設定
449     float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
450     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
451 
452     // 格子模様の床を描画
453     DrawFloor( 1.5f, 10, 10, 1.0, 1.0, 1.0, 1.0, 0.8, 0.8 );
454 
455     // それぞれの幾何形状モデル＋影を描画
456     for ( int i=0; i<NUM_OBJECTS; i++ )
457     {
458         if ( ! object[ i ] )
459             continue;
460         if ( ! object_display[ i ] )
461             continue;
462 
463         // モデル座標系からワールド座標系への変換行列を計算
464         float  matrix[ 16 ];
465 
466         // 変換行列を計算するために、現在の変換行列を一時保存
467         glPushMatrix();
468 
469             // 変換行列を単位行列で初期化
470             glLoadIdentity();
471 
472             // モデルの位置・水平向きの変換行列をかける
473             glTranslatef( object_pos[ i ].x, object_pos[ i ].y, object_pos[ i ].z );
474             glRotatef( object_ori[ i ], 0.0f, 1.0f, 0.0f );
475 
476             // 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）
477             glGetFloatv( GL_MODELVIEW_MATRIX, matrix );
478 
479         // 一時保存しておいた変換行列を復元
480         glPopMatrix();
481 
482 
483         glPushMatrix();
484 
485             // 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
486             glMultMatrixf( matrix );
487 
488             // オブジェクトを描画
489             RenderObj( object[ i ] );
490 
491         glPopMatrix();
492 
493         // 影を描画
494         if ( shadow_mode == SHADOW_PROJECTION )
495         {
496             // ポリゴン投影による影の描画
497             RenderProjectionShadow( object[ i ], matrix, light_pos, shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_alpha );
498         }
499         else if ( shadow_mode == SHADOW_TEXTURE )
500         {
501             // 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）
502             float  height = 0.01f * ( i + 1 );
503 
504             // テクスチャマッピングによる影の描画
505             RenderTextureShadow( matrix, object_size[ i ].x, object_size[ i ].z, height );
506         }
507         else if ( shadow_mode == SHADOW_VOLUME )
508         {
509             // １つ目のオブジェクトは、ポリゴン投影による影の描画
510             if ( i == 0 )
511                 RenderProjectionShadow( object[ i ], matrix, light_pos, shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_alpha );
512             else
513             // 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
514             {
515                 DrawShadowVolume( object[ i ], matrix, light_pos );
516                 FillShadowVolume( shadow_color_rgb, shadow_color_rgb, shadow_color_rgb, shadow_color_alpha );
517             }
518 
519         }
520     }
521 
522     // 文字情報（現在のモード名）を描画
523     DrawTextInformation( shadow_mode_name[ shadow_mode ] );
524 
525     // 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
526     glutSwapBuffers();
527 }
528 
529 
530 //
531 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
532 //
 H3 void  ReshapeCallback( int w, int h )
534 {
535     // 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）
536     glViewport(0, 0, w, h);
537 
538     // カメラ座標系→スクリーン座標系への変換行列を設定
539     glMatrixMode( GL_PROJECTION );
540     glLoadIdentity();
541     gluPerspective( 45, (double)w/h, 1, 500 );
542 
543     // ウィンドウのサイズを記録（テキスト描画処理のため）
544     win_width = w;
545     win_height = h;
546 }
547 
548 
549 //
550 // マウスクリック時に呼ばれるコールバック関数
551 //
 H3 void  MouseClickCallback( int button, int state, int mx, int my )
553 {
554     // 右ボタンが押されたらドラッグ開始
555     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
556         drag_mouse_r = 1;
557     // 右ボタンが離されたらドラッグ終了
558     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
559         drag_mouse_r = 0;
```

```
560
561     // 左ボタンが押されたらドラッグ開始
562     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
563         drag_mouse_l = 1;
564     // 左ボタンが離されたらドラッグ終了
565     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
566         drag_mouse_l = 0;
567
568     // 中ボタンが押されたらドラッグ開始
569     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
570         drag_mouse_m = 1;
571     // 中ボタンが離されたらドラッグ終了
572     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
573         drag_mouse_m = 0;
574
575     // 現在のマウス座標を記録
576     last_mouse_x = mx;
577     last_mouse_y = my;
578 }
579
580
581 //
582 // マウスドラッグ時に呼ばれるコールバック関数
583 //
 H3 void  MouseDragCallback( int mx, int my )
585 {
586     // 右ボタンのドラッグ中は視点を回転する
587     if ( drag_mouse_r && !( glutGetModifiers() & GLUT_ACTIVE_CTRL ) )
588     {
589         // 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
590
591         // マウスの横移動に応じてＹ軸を中心に回転
592         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
593         if ( camera_yaw < 0.0 )
594             camera_yaw += 360.0;
595         else if ( camera_yaw > 360.0 )
596             camera_yaw -= 360.0;
597
598         // マウスの縦移動に応じてＸ軸を中心に回転
599         camera_pitch -= ( my - last_mouse_y ) * 1.0;
600         if ( camera_pitch < -90.0 )
601             camera_pitch = -90.0;
602         else if ( camera_pitch > 90.0 )
603             camera_pitch = 90.0;
604     }
605
606     // 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
607     if ( drag_mouse_m || ( drag_mouse_r && ( glutGetModifiers() & GLUT_ACTIVE_CTRL ) ) )
608     {
609         // 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
610
611         // マウスの縦移動に応じて距離を移動
612         camera_distance += ( my - last_mouse_y ) * 0.2;
613         if ( camera_distance < 5.0 )
614             camera_distance = 5.0;
615     }
616
617     // 左ボタンのドラッグ中は影の方向を変更する
618     if ( drag_mouse_l )
619     {
620         // 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
621
622         // マウスの縦移動に応じて距離を移動
623         float  delta_x = ( mx - last_mouse_x ) * 0.05f;
624         float  delta_z = ( my - last_mouse_y ) * 0.05f;
625
626         light_pos.x += delta_x;
627         if ( light_pos.x < -5.0f )
628             light_pos.x = -5.0f;
629         else if ( light_pos.x > 5.0f )
630             light_pos.x = 5.0f;
631
632         light_pos.z += delta_z;
633         if ( light_pos.z < -5.0f )
634             light_pos.z = -5.0f;
635         else if ( light_pos.z > 5.0f )
636             light_pos.z = 5.0f;
637     }
638
639     // 今回のマウス座標を記録
640     last_mouse_x = mx;
641     last_mouse_y = my;
642
643     // 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）
644     glutPostRedisplay();
645 }
646
647
648 //
649 //  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
650 //
 H3 void  KeyboardCallback( unsigned char key, int mx, int my )
652 {
653     // mキーで影の描画モードを変更
654     if ( key == 'm' )
655     {
656         shadow_mode = (ShadowModeEnum)( ( shadow_mode + 1 ) % NUM_SHADOW_MODE );
657         glutPostRedisplay();
658     }
659
660     // oキーでオブジェクトの描画を変更
661     if ( key == 'o' )
662     {
663         if ( object_display[ 0 ] && object_display[ 1 ] )
664         {
665             object_display[ 0 ] = false;
666         }
667         else if ( !object_display[ 0 ] && object_display[ 1 ] )
668         {
669             object_display[ 0 ] = true;
670             object_display[ 1 ] = false;
671         }
```

```
672         else
673         {
674             object_display[ 1 ] = true;
675         }
676     }
677 
678     // sキーでオブジェクトのアニメーションを変更
679     if ( key == 's' )
680     {
681         if ( object_animation[ 0 ] && object_animation[ 1 ] )
682         {
683             object_animation[ 0 ] = false;
684             object_animation[ 1 ] = false;
685         }
686         else if ( !object_animation[ 0 ] && object_animation[ 1 ] )
687         {
688             object_animation[ 0 ] = true;
689             object_animation[ 1 ] = true;
690         }
691         else
692         {
693             object_animation[ 0 ] = false;
694             object_animation[ 1 ] = true;
695         }
696         on_animation = ( object_animation[ 0 ] && object_animation[ 1 ] );
697     }
698 
699     // dキーでデバッグモードを変更
700     if ( key == 'd' )
701     {
702         if ( !shadow_blend_off && !shadow_stencil_off )
703         {
704             shadow_blend_off = true;
705         }
706         else if ( shadow_blend_off && !shadow_stencil_off )
707         {
708             shadow_blend_off = false;
709             shadow_stencil_off = true;
710         }
711         else
712         {
713             shadow_blend_off = false;
714             shadow_stencil_off = false;
715         }
716     }
717 
718     // 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）
719     glutPostRedisplay();
720 }
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
```