```
//
//  コンピュータグラフィックス特論Ⅱ
//  ピッキング 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>

// ベクトル・行列の表現・計算に vecmath を使用
#include <Vector3.h>
#include <Point3.h>
#include <Matrix3.h>
#include <Matrix4.h>
#include <Color4.h>
#include "vecmath_gl.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
float   camera_yaw = 15.0f;      // Y軸を中心とする回転角度
float   camera_pitch = -20.0f;   // X軸を中心とする回転角度
float   camera_distance = 15.0f; // 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

// ウィンドウのサイズ
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
//

// 表示用の幾何形状モデル
Obj *  object;
Obj *  object_selected;
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 );

// オブジェクトの配置情報
struct  ObjectInfo
{
    // 位置・向き
    Point3f   pos;
    Matrix3f  ori;

    // 変換行列 (位置・向きから描画用に計算)
    Matrix4f  frame;

    // 画面上での位置 (ピッキングのために計算)
    Point2f   screen_pos;
};

// 全オブジェクトの配列
int           num_objects = 0;
ObjectInfo *   objects = NULL;


//
//  ピッキング処理に関するグローバル変数
//

// ピッキング判定方法を表す列挙型
enum  PickModeEnum
{
    PICK_SCREEN,
    PICK_WORLD,
};

// ピッキング判定方法
PickModeEnum  pick_mode = PICK_SCREEN;

// オブジェクトの選択情報 (選択中のオブジェクト番号)
int      selected_object_no = -1;

// オブジェクトの選択情報 (選択された点の位置) (ワールド座標系での判定時のみ有効)
bool     enable_slected_point = false;
Point3f  selected_point;

// 最後にピッキングを行ったときの視線ベクトル (ワールド座標系での判定時のみ)
bool    enable_eye_line = false;
Point3f  eye_line_org;
Vector3f  eye_line_vec;

// 視線ベクトルを描画するかどうかの設定 (ワールド座標系での判定時のみ有効)
bool    draw_eye_line = false;



//
//  ピッキング処理
//
```

```
113
114
115 //
116 //  全オブジェクトの画面上の位置を更新
117 //
 H3 void  UpdateObjectProjection()
119 {
120     // 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
121
122     // 計算用変数
123     Point3d  projected_pos;
124
125     // OpenGL の変換行列を取得
126     double  model_view_matrix[ 16 ];
127     double  projection_matrix[ 16 ];
128     int  viewport_param[ 4 ];
129     glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
130     glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
131     glGetIntegerv( GL_VIEWPORT, viewport_param );
132
133     // 各オブジェクトの画面上の位置を計算
134     for ( int i=0; i<num_objects; i++ )
135     {
136         // i番目のオブジェクトの情報を取得
137         ObjectInfo *  obj = &objects[ i ];
138
139         // オブジェクトの画面上の位置を計算
140         // (ワールド座標系での位置をスクリーン座標に投影)
141
142         // ※レポート課題
143
144         // obj->screen_pos.x = ???;
145         // obj->screen_pos.y = ???;
146     }
147 }
148
149
150 //
151 //  ピッキング処理 (スクリーン座標系)
152 //
 H3 int  PickObjectScreen( int mouse_x, int mouxe_y )
154 {
155     // 選択されたかどうかを判定するための、画面上での距離の閾値
156     // オブジェクトの中心位置とマウス位置の距離が閾値以下であれば、選択されたと判定する
157     const float  threshold = 20.0f;
158
159     // 全オブジェクトの画面上の位置を更新
160     // (本来は、前回の計算時から視点が更新されていなければ再計算の必要はないが、毎回計算を行っている)
161     UpdateObjectProjection();
162
163     // ※レポート課題
164
165     // 各オブジェクトの画面上の位置 (objects[ i ].screen_pos) とマウス位置の間の距離を計算して、
166     // 距離が閾値以下で、最もマウス位置に近いオブジェクトを探索
167
168     // 選択されたオブジェクトの番号を返す
169     // マウス座標の近くにオブジェクトがない場合は、-1 を返す
170
171     return  -1;
172 }
173
174
175 //
176 //  三角形と半直線の交差判定
177 //
178 bool  CheckCross( const Point3f & p0, const Point3f & p1, const Point3f & p2, const Point3f & seg_org, const Vector3f & seg_vec, Point3f & cross_po
    int )
179 {
180     // ※レポート課題
181
182     // 三角形と直線が交差する場合は、戻り値として true を返す
183     // 交差しない場合は、false を返す
184     // また、交差する場合は、交点の座標を cross_point に格納して返す
185
186     return  false;
187 }
188
189
190 //
191 //  ピッキング処理 (ワールド座標系)
192 //
 H3 int  PickObjectWorld( int mouse_x, int mouse_y )
194 {
195     // OpenGL の変換行列を取得
196     double  model_view_matrix[ 16 ];
197     double  projection_matrix[ 16 ];
198     int  viewport_param[ 4 ];
199     glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
200     glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
201     glGetIntegerv( GL_VIEWPORT, viewport_param );
202
203     // マウス位置に対応する３次元空間の半直線
204     double  wx, wy, wz, dx, dy, dz;
205
206     // ※レポート課題
207     // 視点位置 (wx,wy,wz) と視線ベクトル (dx,dy,dz) を計算
208     wx = 0.0f;
209     wy = 0.0f;
210     wz = 0.0f;
211     dx = 0.0f;
212     dy = 0.0f;
213     dz = 0.0f;
214
215
216     // 半直線の始点 (視点位置) と方向ベクトル (視線ベクトル) を設定
217     Point3f  line_org;
218     Vector3f  line_vec;
219     line_org.set( wx, wy, wz );
220     line_vec.set( dx, dy, dz );
221
222     // ピッキングを行ったときの視線ベクトルを記録 (表示用)
223     enable_eye_line = true;
```

```
224     eye_line_org = line_org;
225     eye_line_vec = line_vec;
226 
227     // 計算用変数
228     Point3f  p0, p1, p2;
229     Point3f  cross_point, closest_cross_point;
230     Vector3f  vec;
231     bool  cross;
232     int  dist;
233 
234     // 最も視点に近い交点のオブジェクト番号と距離（最初は -1 で初期化）
235     int  closeset_object_no = -1;
236     float  closest_dist = -1.0f;
237 
238     // 各オブジェクトと直線の交差判定
239     for ( int i=0; i<num_objects; i++ )
240     {
241         const ObjectInfo &  obj = objects[ i ];
242 
243         // オブジェクトの各ポリゴンとの交差判定
244         for ( int j=0; j<object->num_triangles; j++ )
245         {
246             // 三角面の頂点座標を取得（モデル座標系）
247             p0.set( &object->vertices[ object->tri_v_no[ j*3 + 0 ] ].x );
248             p1.set( &object->vertices[ object->tri_v_no[ j*3 + 1 ] ].x );
249             p2.set( &object->vertices[ object->tri_v_no[ j*3 + 2 ] ].x );
250 
251             // 三角面の頂点座標を計算（ワールド座標系）
252             obj.frame.transform( &p0 );
253             obj.frame.transform( &p1 );
254             obj.frame.transform( &p2 );
255 
256             // 半直線と三角面の交差判定
257             cross = CheckCross( p0, p1, p2, line_org, line_vec, cross_point );
258 
259             // 交差する場合の処理
260             if ( cross )
261             {
262                 // 交点と視点の距離を計算
263                 vec.sub( cross_point, line_org );
264                 dist = vec.length();
265 
266                 // 最も視点に近い交点とそのオブジェクト番号を記録
267                 if ( ( closeset_object_no == -1 ) || ( dist < closest_dist ) )
268                 {
269                     closeset_object_no = i;
270                     closest_dist = dist;
271 
272                     // 交点の位置を記録
273                     enable_slected_point = true;
274                     selected_point = cross_point;
275                 }
276             }
277         }
278     }
279 
280     // オブジェクトが選択されなかった場合は、交点の位置は無効とする
281     if ( closeset_object_no == -1 )
282         enable_slected_point = false;
283 
284     // 選択されたオブジェクトの番号を返す
285     return  closeset_object_no;
286 }
287 
288 
289 //
290 //  ピッキング処理
291 //
H3  int  PickObject( int mouse_x, int mouse_y )
293 {
294     // 選択された点の位置の情報、視線ベクトルの情報をクリア
295     enable_slected_point = false;
296     enable_eye_line = false;
297 
298     // スクリーン座標系で判定
299     if ( pick_mode == PICK_SCREEN )
300         return  PickObjectScreen( mouse_x, mouse_y );
301 
302     // ワールド座標系で判定
303     else if ( pick_mode == PICK_WORLD )
304         return  PickObjectWorld( mouse_x, mouse_y );
305 
306     // オブジェクトが選択されなかった場合は、-1 を返す
307     return  -1;
308 }
309 
310 
311 
312 //
313 //  以下、プログラムのメイン処理
314 //
315 
316 
317 //
318 //  シーン初期化（オブジェクトをランダムに配置）
319 //
H3  void  InitScene( int n )
321 {
322     // 全オブジェクトの情報を格納する配列を初期化
323     num_objects = n;
324     if ( !objects )
325         delete[]  objects;
326     objects = new ObjectInfo[ num_objects ];
327 
328     // 全オブジェクトの位置・向きをランダムに設定
329     ObjectInfo *  obj = NULL;
330     float  yaw, pitch, roll;
331     Matrix3f  rot;
332     for ( int i=1; i<num_objects; i++ )
333     {
334         obj = &objects[ i ];
335         obj->pos.x = ( (float) rand() / RAND_MAX ) * 10.0f - 5.0f;
```

```
        obj->pos.z = ( (float)rand() / RAND_MAX ) * 10.0f - 5.0f;
        obj->pos.y = ( (float)rand() / RAND_MAX ) * 5.0f + 0.2f;
        pitch = ( (float) rand() / RAND_MAX ) * 0.5f * M_PI - 0.25f * M_PI;
        yaw = ( (float) rand() / RAND_MAX ) * 2.0f * M_PI;
        roll = ( (float) rand() / RAND_MAX ) * 0.5f * M_PI - 0.25f * M_PI;
        obj->ori.setIdentity();
        rot.rotZ( roll );
        obj->ori.mul( obj->ori, rot );
        rot.rotX( pitch );
        obj->ori.mul( obj->ori, rot );
        rot.rotY( yaw );
        obj->ori.mul( obj->ori, rot );
        obj->frame.set( obj->ori, obj->pos, 1.0f );
    }

    // １つ目のオブジェクトは、必ず原点に配置する。
    objects[ 0 ].pos.set( 0.0f, 0.3f, 0.0f );
    objects[ 0 ].ori.setIdentity();
    objects[ 0 ].frame.set( objects[ 0 ].ori, objects[ 0 ].pos, 1.0f );
}


//
//  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();
}


//
//  幾何形状モデル (Obj形状) の影の描画
//
void  RenderShadow( Obj * obj, Matrix4f & mat )
{
    Matrix4f  frame( mat );
    frame.transpose();
    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 );
}


//
//  テキストを描画
//
void  DrawTextInformation( int line_no, const char * message )
{
    if ( message == NULL )
        return;

    // 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, 40 + 24 * line_no );
    for ( int 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();
}


//
```

```
448 //  画面描画時に呼ばれるコールバック関数
449 //
 H3 void  DisplayCallback( void )
451 {
452     // 画面をクリア
453     glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
454 
455     // 変換行列を設定（ワールド座標系→カメラ座標系）
456     glMatrixMode( GL_MODELVIEW );
457     glLoadIdentity();
458     glTranslatef( 0.0,  0.0, - camera_distance );
459     glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
460     glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
461 
462     // 光源の位置を更新
463     float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
464     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
465 
466     // 格子模様の床を描画
467     DrawFloor( 1.5f, 10, 10, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );
468 
469     // 全オブジェクトを描画
470     for ( int i=0; i<num_objects; i++ )
471     {
472         glPushMatrix();
473 
474             // オブジェクトの位置・向きにもとづく変換行列を設定（モデル座標系→ワールド座標系）
475             glMultMatrixf( objects[ i ].frame );
476 
477             // 選択されているオブジェクトを描画
478             if ( i == selected_object_no )
479                 RenderObj( object_selected );
480             // 通常のオブジェクトを描画
481             else
482                 RenderObj( object );
483 
484         glPopMatrix();
485 
486         // オブジェクトの影を描画
487         RenderShadow( object, objects[ i ].frame );
488     }
489 
490     // 選択点に球を描画
491     if ( enable_slected_point )
492     {
493         glPushMatrix();
494             glTranslatef( selected_point );
495             glColor3f( 1.0, 0.0, 0.0 );
496             glutSolidSphere( 0.05f, 16, 16 );
497         glPopMatrix();
498     }
499 
500     // 最後にピッキングを行ったときの視線ベクトルを描画（確認用）
501     if ( draw_eye_line && enable_eye_line )
502     {
503         float  s = camera_distance * 2.0f;
504         glBegin( GL_LINES );
505             glColor3f( 1.0, 0.0, 0.0 );
506             glVertex3f( eye_line_org );
507             glVertex3f( eye_line_org + s * eye_line_vec );
508         glEnd();
509     }
510 
511     // 現在の選択モードを表示
512     if ( pick_mode == PICK_SCREEN )
513         DrawTextInformation( 0, "Picking on Screen" );
514     else if ( pick_mode == PICK_WORLD )
515         DrawTextInformation( 0, "Picking in World" );
516 
517     // 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
518     glutSwapBuffers();
519 }
520 
521 
522 //
523 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
524 //
 H3 void  ReshapeCallback( int w, int h )
526 {
527     // 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）
528     glViewport(0, 0, w, h);
529 
530     // カメラ座標系→スクリーン座標系への変換行列を設定
531     glMatrixMode( GL_PROJECTION );
532     glLoadIdentity();
533     gluPerspective( 45, (double)w/h, 1, 500 );
534 
535     // ウィンドウのサイズを記録（テキスト描画処理のため）
536     win_width = w;
537     win_height = h;
538 }
539 
540 
541 //
542 //  マウスクリック時に呼ばれるコールバック関数
543 //
 H3 void  MouseClickCallback( int button, int state, int mx, int my )
545 {
546     // 左ボタンが押されたらドラッグ開始
547     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
548         drag_mouse_l = 1;
549     // 左ボタンが離されたらドラッグ終了
550     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
551         drag_mouse_l = 0;
552 
553     // 右ボタンが押されたらドラッグ開始
554     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
555         drag_mouse_r = 1;
556     // 右ボタンが離されたらドラッグ終了
557     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
558         drag_mouse_r = 0;
559 
```

```
560     // 中ボタンが押されたらドラッグ開始
561     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
562         drag_mouse_m = 1;
563     // 中ボタンが離されたらドラッグ終了
564     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
565         drag_mouse_m = 0;
566
567     // 左ボタンが押されたら、オブジェクトを選択（ピッキング処理）
568     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
569         selected_object_no = PickObject( mx, my );
570
571     // 再描画
572     glutPostRedisplay();
573
574     // 現在のマウス座標を記録
575     last_mouse_x = mx;
576     last_mouse_y = my;
577 }
578
579
580 //
581 //  マウスドラッグ時に呼ばれるコールバック関数
582 //
 H3 void  MouseDragCallback( int mx, int my )
584 {
585     // 右ボタンのドラッグ中は視点を回転する
586     if ( drag_mouse_r )
587     {
588         // 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
589
590         // マウスの横移動に応じてＹ軸を中心に回転
591         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
592         if ( camera_yaw < 0.0 )
593             camera_yaw += 360.0;
594         else if ( camera_yaw > 360.0 )
595             camera_yaw -= 360.0;
596
597         // マウスの縦移動に応じてＸ軸を中心に回転
598         camera_pitch -= ( my - last_mouse_y ) * 1.0;
599         if ( camera_pitch < -90.0 )
600             camera_pitch = -90.0;
601         else if ( camera_pitch > 90.0 )
602             camera_pitch = 90.0;
603     }
604     // 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
605     if ( drag_mouse_m )
606     {
607         // 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
608
609         // マウスの縦移動に応じて距離を移動
610         camera_distance += ( my - last_mouse_y ) * 0.2;
611         if ( camera_distance < 2.0 )
612             camera_distance = 2.0;
613     }
614
615     // 今回のマウス座標を記録
616     last_mouse_x = mx;
617     last_mouse_y = my;
618
619     // 再描画
620     glutPostRedisplay();
621 }
622
623
624 //
625 //  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
626 //
 H3 void  KeyboardCallback( unsigned char key, int mx, int my )
628 {
629     // ピッキング判定方法の変更
630     if ( key == 'p' )
631     {
632         if ( pick_mode == PICK_SCREEN )
633             pick_mode = PICK_WORLD;
634         else
635             pick_mode = PICK_SCREEN;
636     }
637
638     // 視線ベクトルを描画するかの設定を変更
639     if ( key == 'd' )
640         draw_eye_line = !draw_eye_line;
641
642     glutPostRedisplay();
643 }
644
645
646 //
647 //  環境初期化関数
648 //
 H3 void  initEnvironment( void )
650 {
651     // 光源を作成する
652     float  light0_position[] = { 0.0, 10.0, 0.0, 1.0 };
653     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
654     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
655     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
656     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
657     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
658     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
659     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
660     glEnable( GL_LIGHT0 );
661
662     // 光源計算を有効にする
663     glEnable( GL_LIGHTING );
664
665     // 物体の色情報を有効にする
666     glEnable( GL_COLOR_MATERIAL );
667
668     // Ｚテストを有効にする
669     glEnable( GL_DEPTH_TEST );
670
671     // 背面除去を有効にする
```

```
672     glCullFace( GL_BACK );
673     glEnable( GL_CULL_FACE );
674 
675     // 背景色を設定
676     glClearColor( 0.5, 0.5, 0.8, 0.0 );
677 
678     // オブジェクトの幾何形状モデルの読み込み
679     object = LoadObj( "car.obj" );
680     if ( !object || ( object->num_triangles == 0 ) )
681     {
682         // 読み込みに失敗したら終了
683         printf( "Failed to load the object file." );
684         exit( -1 );
685     }
686     ScaleObj( object, 1.0f, &object_size.x, &object_size.y, &object_size.z );
687 
688     // 選択表示用のオブジェクトの幾何形状モデルの作成
689     // (同じオブジェクトを読み込んで、色を変更)
690     object_selected = LoadObj( "car.obj" );
691     ScaleObj( object_selected, 1.0f, &object_size.x, &object_size.y, &object_size.z );
692     for ( int i=0; i<object_selected->num_materials; i++ )
693     {
694         Mtl *  mtl = object_selected->materials[ i ];
695         mtl->kd.r = 1.0f - mtl->kd.r;
696         mtl->kd.g = 1.0f - mtl->kd.g;
697         mtl->kd.b = 1.0f - mtl->kd.b;
698     }
699 
700     // シーンの初期化 (オブジェクトをランダムに配置)
701     InitScene( 30 );
702 }
703 
704 
705 //
706 //  メイン関数 (プログラムはここから開始)
707 //
 H3 int  main( int argc, char ** argv )
709 {
710     // GLUTの初期化
711     glutInit( &argc, argv );
712     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
713     glutInitWindowSize( 480, 480 );
714     glutInitWindowPosition( 0, 0 );
715     glutCreateWindow("Picking");
716 
717     // コールバック関数の登録
718     glutDisplayFunc( DisplayCallback );
719     glutReshapeFunc( ReshapeCallback );
720     glutMouseFunc( MouseClickCallback );
721     glutMotionFunc( MouseDragCallback );
722     glutKeyboardFunc( KeyboardCallback );
723 
724     // 環境初期化
725     initEnvironment();
726 
727     // GLUTのメインループに処理を移す
728     glutMainLoop();
729     return 0;
730 }
731 
732 
```