```
//
//  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 <vecmath.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"


//
//  カメラ・GLUTの入力処理に関するグローバル変数
//

// π
#define  M_PI  3.14159268

// 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
//

```

```
109
110 //
111 //  全オブジェクトの画面上の位置を更新
112 //
 H3 void  UpdateObjectProjection()
114 {
115     // 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
116
117     // OpenGL の変換行列を取得
118     double  model_view_matrix[ 16 ];
119     double  projection_matrix[ 16 ];
120     int  viewport_param[ 4 ];
121     Point3d  projected_pos;
122     glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
123     glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
124     glGetIntegerv( GL_VIEWPORT, viewport_param );
125
126     // 各オブジェクトの画面上の位置を計算
127     for ( int i=0; i<num_objects; i++ )
128     {
129         // i番目のオブジェクトの情報を取得
130         ObjectInfo *  obj = &objects[ i ];
131
132         // オブジェクトの画面上の位置を計算
133         // (ワールド座標系での位置をスクリーン座標に投影)
134
135         // ※レポート課題
136
137         // obj->screen_pos.x = ???;
138         // obj->screen_pos.y = ???;
139     }
140 }
141
142
143 //
144 //  ピッキング処理 (スクリーン座標系)
145 //
 H3 int  PickObjectScreen( int mouse_x, int mouxe_y )
147 {
148     // 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
149     // 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
150     const float  threshold = 20.0f;
151
152     // 全オブジェクトの画面上の位置を更新
153     // (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)
154     UpdateObjectProjection();
155
156     // ※レポート課題
157
158     // 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、
159     // 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
160
161     // 選択されたオブジェクトの番号を返す
162     // 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
163
164     return  -1;
165 }
166
167
168 //
169 //  三角形と半直線の交差判定
170 //
171 bool  CheckCross( const Point3f & tri_p0, const Point3f & tri_p1, const Point3f & tri_p2, const Point3f & seg_org, const Vector3f & seg_ve
    c, Point3f & cross_point )
172 {
173     // ※レポート課題
174
175     // 三角形と直線が交差する場合は、戻り値として true を返す
176     // 交差しない場合は、false を返す
177     // また、交差する場合は、交点の座標を cross_point に格納して返す
178
179     return  false;
180 }
181
182
183 //
184 //  ピッキング処理 (ワールド座標系)
185 //
 H3 int  PickObjectWorld( int mouse_x, int mouse_y )
187 {
188     // OpenGL の変換行列を取得
189     double  model_view_matrix[ 16 ];
190     double  projection_matrix[ 16 ];
191     int  viewport_param[ 4 ];
192     glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
193     glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
194     glGetIntegerv( GL_VIEWPORT, viewport_param );
195
196     // マウス位置に対応する３次元空間の直線を求める
197     double  wx, wy, wz, dx, dy, dz;
198
199     // ※レポート課題
200     // 視点位置 (wx,wy,wz) と視線ベクトル (dx,dy,dz) を計算
201
202
203     // 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
204     Point3f  line_org;
205     Vector3f  line_vec;
206     line_org.set( wx, wy, wz );
207     line_vec.set( dx, dy, dz );
208
209     // ピッキングを行ったときの視線ベクトルを記録 (表示用)
210     enable_eye_line = true;
211     eye_line_org = line_org;
212     eye_line_vec = line_vec;
213
214     // 計算用変数
215     Point3f  p0, p1, p2;
```

```
216     Point3f  cross_point, closest_cross_point;
217     Vector3f  vec;
218     bool  cross;
219     int  dist;
220 
221     // 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）
222     int  closeset_object_no = -1;
223     float  closest_dist = -1.0f;
224 
225     // 各オブジェクトと直線の交差判定
226     for ( int i=0; i<num_objects; i++ )
227     {
228         const ObjectInfo &  obj = objects[ i ];
229 
230         // オブジェクトの各ポリゴンとの交差判定
231         for ( int j=0; j<object->num_triangles; j++ )
232         {
233             // 三角面の頂点座標を取得（モデル座標系）
234             p0.set( &object->vertices[ object->tri_v_no[ j*3 + 0 ] ].x );
235             p1.set( &object->vertices[ object->tri_v_no[ j*3 + 1 ] ].x );
236             p2.set( &object->vertices[ object->tri_v_no[ j*3 + 2 ] ].x );
237 
238             // 三角面の頂点座標を計算（ワールド座標系）
239             obj.frame.transform( &p0 );
240             obj.frame.transform( &p1 );
241             obj.frame.transform( &p2 );
242 
243             // 半直線と三角面の交差判定
244             cross = CheckCross( p0, p1, p2, line_org, line_vec, cross_point );
245 
246             // 交差する場合の処理
247             if ( cross )
248             {
249                 // 交点と視点の距離を計算
250                 vec.sub( cross_point, line_org );
251                 dist = vec.length();
252 
253                 // 最も視点に近い交点とそのオブジェクト番号を記録
254                 if ( ( closeset_object_no == -1 ) || ( dist < closest_dist ) )
255                 {
256                     closeset_object_no = i;
257                     closest_dist = dist;
258 
259                     // 交点の位置を記録
260                     enable_slected_point = true;
261                     selected_point = cross_point;
262                 }
263             }
264         }
265     }
266 
267     // 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
268     if ( closeset_object_no == -1 )
269         enable_slected_point = false;
270 
271     // 選択されたオブジェクトの番号を返す
272     return  closeset_object_no;
273 }
274 
275 
276 //
277 //  ピッキング処理
278 //
 H3 int  PickObject( int mouse_x, int mouse_y )
280 {
281     // 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
282     enable_slected_point = false;
283     enable_eye_line = false;
284 
285     // スクリーン座標系で判定
286     if ( pick_mode == PICK_SCREEN )
287         return  PickObjectScreen( mouse_x, mouse_y );
288 
289     // ワールド座標系で判定
290     else if ( pick_mode == PICK_WORLD )
291         return  PickObjectWorld( mouse_x, mouse_y );
292 
293     // オブジェクトが選択されなかった場合は、-1 を返す
294     return  -1;
295 }
296 
297 
298 
299 //
300 //  以下、プログラムのメイン処理
301 //
302 
303 
304 //
305 //  シーン初期化（オブジェクトをランダムに配置）
306 //
 H3 void  InitScene( int n )
308 {
309     // 全オブジェクトの情報を格納する配列を初期化
310     num_objects = n;
311     if ( !objects )
312         delete[]  objects;
313     objects = new ObjectInfo[ num_objects ];
314 
315     // 全オブジェクトの位置・向きをランダムに設定
316     ObjectInfo *  obj = NULL;
317     float  yaw, pitch, roll;
318     Matrix3f  rot;
319     for ( int i=1; i<num_objects; i++ )
320     {
321         obj = &objects[ i ];
322         obj->pos.x = ( (float) rand() / RAND_MAX ) * 10.0f - 5.0f;
323         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();
```

```
}


//
//  画面描画時に呼ばれるコールバック関数
//
void  DisplayCallback( void )
{
    // 画面をクリア
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );

    // 変換行列を設定（ワールド座標系→カメラ座標系）
    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 );

    // 光源の位置を更新
    float  light0_position[] = { light_pos.x, light_pos.y, light_pos.z, 1.0 };
    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );

    // 格子模様の床を描画
    DrawFloor( 1.5f, 10, 10, 1.0f, 1.0f, 1.0f, 1.0f, 0.8f, 0.8f );

    // 全オブジェクトを描画
    for ( int i=0; i<num_objects; i++ )
    {
        glPushMatrix();

            // 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）
            glMultMatrixf( objects[ i ].frame );

            // 選択されているオブジェクトを描画
            if ( i == selected_object_no )
                RenderObj( object_selected );
            // 通常のオブジェクトを描画
            else
                RenderObj( object );

        glPopMatrix();

        // オブジェクトの影を描画
        RenderShadow( object, objects[ i ].frame );
    }

    // 選択点に球を描画
    if ( enable_slected_point )
    {
        glPushMatrix();
            glTranslatef( selected_point );
            glColor3f( 1.0, 0.0, 0.0 );
            glutSolidSphere( 0.05f, 16, 16 );
        glPopMatrix();
    }

    // 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）
    if ( draw_eye_line && enable_eye_line )
    {
        float  s = camera_distance * 2.0f;
        glBegin( GL_LINES );
            glColor3f( 1.0, 0.0, 0.0 );
            glVertex3f( eye_line_org );
            glVertex3f( eye_line_org + s * eye_line_vec );
        glEnd();
    }

    // 現在の選択モードを表示
    if ( pick_mode == PICK_SCREEN )
        DrawTextInformation( 0, "Picking on Screen" );
    else if ( pick_mode == PICK_WORLD )
        DrawTextInformation( 0, "Picking in World" );

    // 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
    glutSwapBuffers();
}


//
//  ウィンドウサイズ変更時に呼ばれるコールバック関数
//
void  ReshapeCallback( int w, int h )
{
    // 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）
    glViewport(0, 0, w, h);

    // カメラ座標系→スクリーン座標系への変換行列を設定
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 45, (double)w/h, 1, 500 );

    // ウィンドウのサイズを記録（テキスト描画処理のため）
    win_width = w;
    win_height = h;
}


//
//  マウスクリック時に呼ばれるコールバック関数
//
void  MouseClickCallback( int button, int state, int mx, int my )
{
    // 左ボタンが押されたらドラッグ開始
    if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
        drag_mouse_l = 1;
    // 左ボタンが離されたらドラッグ終了
    else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
        drag_mouse_l = 0;
```

```
539
540     // 右ボタンが押されたらドラッグ開始
541     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
542         drag_mouse_r = 1;
543     // 右ボタンが離されたらドラッグ終了
544     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
545         drag_mouse_r = 0;
546
547     // 中ボタンが押されたらドラッグ開始
548     if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_DOWN ) )
549         drag_mouse_m = 1;
550     // 中ボタンが離されたらドラッグ終了
551     else if ( ( button == GLUT_MIDDLE_BUTTON ) && ( state == GLUT_UP ) )
552         drag_mouse_m = 0;
553
554     // 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）
555     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
556         selected_object_no = PickObject( mx, my );
557
558     // 再描画
559     glutPostRedisplay();
560
561     // 現在のマウス座標を記録
562     last_mouse_x = mx;
563     last_mouse_y = my;
564 }
565
566
567 //
568 //  マウスドラッグ時に呼ばれるコールバック関数
569 //
 H3 void  MouseDragCallback( int mx, int my )
571 {
572     // 右ボタンのドラッグ中は視点を回転する
573     if ( drag_mouse_r )
574     {
575         // 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
576
577         // マウスの横移動に応じてＹ軸を中心に回転
578         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
579         if ( camera_yaw < 0.0 )
580             camera_yaw += 360.0;
581         else if ( camera_yaw > 360.0 )
582             camera_yaw -= 360.0;
583
584         // マウスの縦移動に応じてＸ軸を中心に回転
585         camera_pitch -= ( my - last_mouse_y ) * 1.0;
586         if ( camera_pitch < -90.0 )
587             camera_pitch = -90.0;
588         else if ( camera_pitch > 90.0 )
589             camera_pitch = 90.0;
590     }
591     // 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
592     if ( drag_mouse_m )
593     {
594         // 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
595
596         // マウスの縦移動に応じて距離を移動
597         camera_distance += ( my - last_mouse_y ) * 0.2;
598         if ( camera_distance < 2.0 )
599             camera_distance = 2.0;
600     }
601
602     // 今回のマウス座標を記録
603     last_mouse_x = mx;
604     last_mouse_y = my;
605
606     // 再描画
607     glutPostRedisplay();
608 }
609
610
611 //
612 //  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
613 //
 H3 void  KeyboardCallback( unsigned char key, int mx, int my )
615 {
616     // ピッキング判定方法の変更
617     if ( key == 'p' )
618     {
619         if ( pick_mode == PICK_SCREEN )
620             pick_mode = PICK_WORLD;
621         else
622             pick_mode = PICK_SCREEN;
623     }
624
625     // 視線ベクトルを描画するかの設定を変更
626     if ( key == 'd' )
627         draw_eye_line = !draw_eye_line;
628
629     glutPostRedisplay();
630 }
631
632
633 //
634 //  環境初期化関数
635 //
 H3 void  initEnvironment( void )
637 {
638     // 光源を作成する
639     float  light0_position[] = { 0.0, 10.0, 0.0, 1.0 };
640     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
641     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
642     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
643     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
644     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
645     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
646     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
```

```
647     glEnable( GL_LIGHT0 );
648
649     // 光源計算を有効にする
650     glEnable( GL_LIGHTING );
651
652     // 物体の色情報を有効にする
653     glEnable( GL_COLOR_MATERIAL );
654
655     // Ｚテストを有効にする
656     glEnable( GL_DEPTH_TEST );
657
658     // 背面除去を有効にする
659     glCullFace( GL_BACK );
660     glEnable( GL_CULL_FACE );
661
662     // 背景色を設定
663     glClearColor( 0.5, 0.5, 0.8, 0.0 );
664
665     // オブジェクトの幾何形状モデルの読み込み
666     object = LoadObj( "car.obj" );
667     if ( !object || ( object->num_triangles == 0 ) )
668     {
669         // 読み込みに失敗したら終了
670         printf( "Failed to load the object file." );
671         exit( -1 );
672     }
673     ScaleObj( object, 1.0f, &object_size.x, &object_size.y, &object_size.z );
674
675     // 選択表示用のオブジェクトの幾何形状モデルの作成
676     // (同じオブジェクトを読み込んで、色を変更)
677     object_selected = LoadObj( "car.obj" );
678     ScaleObj( object_selected, 1.0f, &object_size.x, &object_size.y, &object_size.z );
679     for ( int i=0; i<object_selected->num_materials; i++ )
680     {
681         Mtl *  mtl = object_selected->materials[ i ];
682         mtl->kd.r = 1.0f - mtl->kd.r;
683         mtl->kd.g = 1.0f - mtl->kd.g;
684         mtl->kd.b = 1.0f - mtl->kd.b;
685     }
686
687     // シーンの初期化 (オブジェクトをランダムに配置)
688     InitScene( 30 );
689 }
690
691
692 //
693 //  メイン関数 (プログラムはここから開始)
694 //
 H3 int  main( int argc, char ** argv )
696 {
697     // GLUTの初期化
698     glutInit( &argc, argv );
699     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_STENCIL );
700     glutInitWindowSize( 480, 480 );
701     glutInitWindowPosition( 0, 0 );
702     glutCreateWindow("Picking");
703
704     // コールバック関数の登録
705     glutDisplayFunc( DisplayCallback );
706     glutReshapeFunc( ReshapeCallback );
707     glutMouseFunc( MouseClickCallback );
708     glutMotionFunc( MouseDragCallback );
709     glutKeyboardFunc( KeyboardCallback );
710
711     // 環境初期化
712     initEnvironment();
713
714     // GLUTのメインループに処理を移す
715     glutMainLoop();
716     return 0;
717 }
718
719
```