コンピュータアニメーション特論 プログラミング演習資料
第7回 ピッキング

九州工業大学 情報工学研究院　尾下 真樹

2022 年度

1　サンプルプログラム

ピッキングのサンプルプログラム（picking_sample.cpp）をもとに、2 種類のピッキング手法（スクリーン座標でのピッキング、ワールド座標でのピッキング）を実現するプログラムを作成する。マウス左クリックで、オブジェクトの選択（ピッキング）を行う。p キーで、選択の判定方法をスクリーン座標とワールド座標のどちらで行うかの切り替えを行う。d キーで、視線ベクトルの表示・非表示を切り替えを行う。ワールド座標系での判定時のみ、最後にピッキングを行ったときの視線ベクトルを表示する。

本サンプルプログラムは、keyframe_sample.cpp, obj.h, obj.cpp, vecmath_gl.h の 4 つのファイルから構成される。外部ライブラリとして、OpenGL と GLUT に加えて、行列・ベクトルなどを扱うための vecmath C++ ライブラリを使用する。最初に、メインとなるサンプルプログラム（picking_sample.cpp）のソースコード全体を示す。その後、サンプルプログラムの主要な処理や他のソースコードを説明する。

ソースコード 1: picking_sample.cpp

```
1  //
2  //  コンピュータアニメーション特論
3  //  ピッキング サンプルプログラム
4  //
5
6
7  // GLUTヘッダファイルのインクルード
8  #include <GL/glut.h>
9
10 // ベクトル・行列の表現・計算に vecmath を使用
11 #include <Vector3.h>
12 #include <Point3.h>
13 #include <Matrix3.h>
14 #include <Matrix4.h>
15 #include <Color4.h>
16 #include "vecmath_gl.h"
17
18 // 幾何形状オブジェクト、及び、読み込み・描画関数
19 #include "Obj.h"
20
21 // 標準算術関数・定数の定義
22 #define  _USE_MATH_DEFINES
23 #include <math.h>
24
25
26 //
27 //  カメラ・GLUTの入力処理に関するグローバル変数
28 //
29
30 // カメラの回転のための変数
31 float    camera_yaw = 15.0f;       // Y軸を中心とする回転角度
```

```
32 float    camera_pitch = -20.0f;    // X軸を中心とする回転角度
33 float    camera_distance = 15.0f; // 中心からカメラの距離
34
35 // マウスのドラッグのための変数
36 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）
37 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）
38 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）
39 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
40
41 // ウィンドウのサイズ
42 int      win_width, win_height;
43
44
45 //
46 //  オブジェクトの配置・表示に関するグローバル変数
47 //
48
49 // 表示用の幾何形状モデル
50 Obj *  object;
51 Obj *  object_selected;
52 Vector3f  object_size;
53
54 // 点光源の位置（影の投影方向）
55 Vector3f  light_pos( 0.0f, 10.0f, 0.0f );
56
57 // 影の色
58 Color4f   shadow_color( 0.2f, 0.2f, 0.2f, 0.5f );
59
60 // オブジェクトの配置情報
61 struct  ObjectInfo
62 {
63    // 位置・向き
64    Point3f   pos;
65    Matrix3f  ori;
66
67    // 変換行列（位置・向きから描画用に計算）
68    Matrix4f  frame;
69
70    // 画面上での位置（ピッキングのために計算）
71    Point2f   screen_pos;
72 };
73
74 // 全オブジェクトの配列
75 int            num_objects = 0;
76 ObjectInfo *   objects = NULL;
77
78
79 //
80 //  ピッキング処理に関するグローバル変数
81 //
82
83 // ピッキング判定方法を表す列挙型
84 enum  PickModeEnum
85 {
86    PICK_SCREEN,
87    PICK_WORLD,
88 };
89
90 // ピッキング判定方法
91 PickModeEnum  pick_mode = PICK_SCREEN;
92
```

```
93  // オブジェクトの選択情報（選択中のオブジェクト番号）
94  int       selected_object_no = -1;
95
96  // 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）
97  bool      enable_slected_point = false;
98  Point3f   selected_point;
99
100 // 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）
101 bool      enable_eye_line = false;
102 Point3f   eye_line_org;
103 Vector3f  eye_line_vec;
104
105 // 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）
106 bool      draw_eye_line = false;
107
108
109
110 //
111 //  ピッキング処理
112 //
113
114
115 //
116 //  全オブジェクトの画面上の位置を更新
117 //
118 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 //
153 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_point )
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 //
193 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 //
292 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 //
320 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;
336         obj->pos.z = ( (float)rand() / RAND_MAX ) * 10.0f - 5.0f;
337         obj->pos.y = ( (float)rand() / RAND_MAX ) * 5.0f + 0.2f;
338         pitch = ( (float) rand() / RAND_MAX ) * 0.5f * M_PI - 0.25f * M_PI;
339         yaw = ( (float) rand() / RAND_MAX ) * 2.0f * M_PI;
340         roll = ( (float) rand() / RAND_MAX ) * 0.5f * M_PI - 0.25f * M_PI;
341         obj->ori.setIdentity();
342         rot.rotZ( roll );
343         obj->ori.mul( obj->ori, rot );
344         rot.rotX( pitch );
345         obj->ori.mul( obj->ori, rot );
```

```
346         rot.rotY( yaw );
347         obj->ori.mul( obj->ori, rot );
348         obj->frame.set( obj->ori, obj->pos, 1.0f );
349     }
350
351     // １つ目のオブジェクトは、必ず原点に配置する。
352     objects[ 0 ].pos.set( 0.0f, 0.3f, 0.0f );
353     objects[ 0 ].ori.setIdentity();
354     objects[ 0 ].frame.set( objects[ 0 ].ori, objects[ 0 ].pos, 1.0f );
355 }
356
357
358 //
359 //  格子模様の床を描画
360 //
361 void  DrawFloor( float tile_size, int num_x, int num_z, float r0, float g0, float b0,
        float r1, float g1, float b1 )
362 {
363     int  x, z;
364     float  ox, oz;
365
366     glBegin( GL_QUADS );
367     glNormal3d( 0.0, 1.0, 0.0 );
368
369     ox = - ( num_x * tile_size ) / 2;
370     for ( x=0; x<num_x; x++ )
371     {
372         oz = - ( num_z * tile_size ) / 2;
373         for ( z=0; z<num_z; z++ )
374         {
375             if ( ( (x + z) % 2 ) == 0 )
376                 glColor3f( r0, g0, b0 );
377             else
378                 glColor3f( r1, g1, b1 );
379
380             glTexCoord2d( 0.0f, 0.0f );
381             glVertex3d( ox, 0.0, oz );
382             glTexCoord2d( 0.0f, 1.0f );
383             glVertex3d( ox, 0.0, oz + tile_size );
384             glTexCoord2d( 1.0f, 1.0f );
385             glVertex3d( ox + tile_size, 0.0, oz + tile_size );
386             glTexCoord2d( 1.0f, 0.0f );
387             glVertex3d( ox + tile_size, 0.0, oz );
388
389             oz += tile_size;
390         }
391         ox += tile_size;
392     }
393     glEnd();
394 }
395
396
397 //
398 //  幾何形状モデル（Obj形状）の影の描画
399 //
400 void  RenderShadow( Obj * obj, Matrix4f & mat )
401 {
402     Matrix4f  frame( mat );
403     frame.transpose();
404     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 );
405 }
406
407
```

```
408 //
409 //  テキストを描画
410 //
411 void  DrawTextInformation( int line_no, const char * message )
412 {
413     if ( message == NULL )
414         return;
415 
416     // 射影行列を初期化（初期化の前に現在の行列を退避）
417     glMatrixMode( GL_PROJECTION );
418     glPushMatrix();
419     glLoadIdentity();
420     gluOrtho2D( 0.0, win_width, win_height, 0.0 );
421 
422     // 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）
423     glMatrixMode( GL_MODELVIEW );
424     glPushMatrix();
425     glLoadIdentity();
426 
427     // Zバッファ・ライティングはオフにする
428     glDisable( GL_DEPTH_TEST );
429     glDisable( GL_LIGHTING );
430 
431     // メッセージの描画
432     glColor3f( 1.0, 0.0, 0.0 );
433     glRasterPos2i( 16, 40 + 24 * line_no );
434     for ( int i=0; message[i]!='\0'; i++ )
435         glutBitmapCharacter( GLUT_BITMAP_HELVETICA_18, message[i] );
436 
437     // 設定を全て復元
438     glEnable( GL_DEPTH_TEST );
439     glEnable( GL_LIGHTING );
440     glMatrixMode( GL_PROJECTION );
441     glPopMatrix();
442     glMatrixMode( GL_MODELVIEW );
443     glPopMatrix();
444 }
445 
446 
447 //
448 //  画面描画時に呼ばれるコールバック関数
449 //
450 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 //
525 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 //
544 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 //
583 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 //
627 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 //
649 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 //
708 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( 640, 640 );
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 }
```

1.1 幾何形状モデルの読み込みと描画

本プログラムでは、キーフレームアニメーションを行う物体やキーフレームを表す物体を表示するために、Obj 形式の幾何形状モデルを読み込んで描画する。Obj 形式の幾何形状モデルの読み込みや描画には、過去に作成したプログラム（obj.h, obj.cpp）を使用する。ここでは、obj.h のソースコードのみを示し、obj.cpp のソースコードは省略する。obj.h では、幾何形状モデルを表す Obj 構造体、幾何形状モデルの読み込みを行う LoadObj 関数、幾何形状モデルの描画を行う RenderObj 関数、幾何形状モデルの影の描画を行う RenderObjShadow 関数などが定義されている。

メインのサンプルプログラム（picking_sample.cpp）から、上記の関数を呼び出して利用する。プログラムの開始時に、initEnvironment 関数の中で、LoadObj 関数を呼び出して、幾何形状モデルの読み込みを行い、グローバル変数として定義された obj 変数に、読み込んだ幾何形状モデルの情報を格納する。画面描画時に呼ばれる DisplayCallback 関数の中で、空間に配置されている物体を描画するために、RenderObj 関数や RenderObjShadow 関数を呼び出して、読み込んだ幾何形状モデルを描画する。

ソースコード 2: obj.h

```
1  //
2  //  コンピュータアニメーション特論
3  //  幾何形状モデル（Obj形式）の読み込み＆描画のサンプルプログラム
4  //
5
6  #ifndef  _OBJ_H_
7  #define  _OBJ_H_
8
9
10 // ベクトルデータ
11 struct  Vector
12 {
13     float   x, y, z;
14 };
15
16
17 // カラーデータ
18 struct  Color
19 {
20     float  r, g, b;
21 };
22
23
24 //
25 //  幾何形状モデルの表現例（本プログラムでは使用しない）
26 //
27 struct  SampleGeometry
28 {
29     int        num_vertices;  // 頂点数
30     Vector *   vertices;      // 頂点座標配列 [num_vertices]
31     Vector *   normals;       // 法線ベクトル配列 [num_vertices]
32     Color *    colors;        // カラー配列 [num_vertices]
33
34     int        num_triangles; // 三角面数
35     int *      triangles;     // 三角面の頂点番号配列 [num_triangles*3]
36 };
37
```

```
38
39 //
40 //  幾何形状モデルの素材情報（Mtl形式）
41 //
42 struct  Mtl
43 {
44     char *      name;           // マテリアル名
45
46     Color       kd;             // 拡散反射光（とりあえず拡散反射光を glColor3f() で使用する）
47
48     char *      texture_name;   // テクスチャ画像のファイル名
49 };
50
51
52 //
53 //  幾何形状モデル（Obj形式）
54 //
55 struct  Obj
56 {
57     int         num_vertices;   // 頂点数
58     Vector *    vertices;       // 頂点座標配列 [num_vertices]
59
60     int         num_normals;
61     Vector *    normals;        // 法線ベクトル配列 [num_normals]
62
63     int         num_tex_coords;
64     Vector *    tex_coords;     // テクスチャ座標配列 [num_tex_coords]
65
66     int         num_triangles;  // 三角面数
67     int *       tri_v_no;       // 三角面の各頂点の頂点座標番号配列 [num_triangles*3]
68     int *       tri_vn_no;      // 三角面の各頂点の法線ベクトル番号配列 [num_triangles*3]
69     int *       tri_vt_no;      // 三角面の各頂点のテクスチャ座標番号配列 [num_triangles*3]
70     Mtl **      tri_material;   // 三角面の素材 [num_triangles]
71
72     int         num_materials;  // マテリアル数
73     Mtl **      materials;      // マテリアルの配列 [num_materials]
74 };
75
76
77
78 // Objファイルの読み込み
79 Obj *  LoadObj( const char * filename );
80
81 // Mtlファイルの読み込み
82 void  LoadMtl( const char * filename, Obj * obj );
83
84 // 幾何形状モデルのスケーリング（スケーリング後のサイズを返す）
85 void  ScaleObj( Obj * obj, float max_size, float * size_x = NULL, float * size_y = NULL, float * size_z = NULL );
86
87 // 幾何形状モデル（Obj形状）の描画
88 void  RenderObj( Obj * obj );
89
90 // 幾何形状モデル（Obj形状）の描画（固定色で描画）
91 void  RenderObjUnicolor( const Obj * obj, float color_r, float color_g, float color_b, float color_a );
92
93 // 幾何形状モデル（Obj形状）の影の描画（ポリゴン投影による影の描画）
94 void  RenderObjShadow( const Obj * obj, const float obj_matrix[ 16 ], float light_dir_x, float light_dir_y, float light_dir_z, float color_r, float color_g, float color_b, float color_a );
95
```

```
96  // 幾何形状モデルを頂点配列を使って描画可能なモデルに変換
97  void  ConvertObjForVertexArrays( Obj * obj );
98
99  // 幾何形状モデルを頂点配列を使って描画
100 void  RenderObjWithVertexArrays( Obj * obj );
101
102 // Objファイルの書き出し
103 int  SaveObj( const Obj * obj, const char * filename );
104
105
106
107 #endif // _OBJ_H_
```

1.2 vecmath 補助関数

本サンプルプログラムでは、行列・ベクトルなどを扱うために vecmath C++ ライブラリを使用する。OpenGL の関数は、行列やベクトルなどの入力を関数の引数として渡す場合、複数の数値型や数値型の配列によって渡す形になっており、vecmath の行列やベクトルを渡すときには、変換が必要になる。毎回このような変換を行わなくとも良いように、OpenGL の関数と同名の関数で、vecmath の行列やベクトルを引数として受け取り、内部で変換を行って OpenGL の関数を呼び出す関数を定義して、利用できるようにする。メインのサンプルプログラム（picking_sample.cpp）から、これらの関数を呼び出して利用する。

ソースコード 3: vecmath_gl.h

```
1   //
2   //  コンピュータアニメーション特論
3   //  vecmathオブジェクトを引数としてOpenGL関数を呼び出すための補助関数
4   //
5
6
7   #ifndef  _VECMATH_GL_H_
8   #define  _VECMATH_GL_H_
9
10
11  #include <Vector3.h>
12  #include <Point3.h>
13  #include <Point2.h>
14  #include <Color3.h>
15  #include <Matrix3.h>
16  #include <Matrix4.h>
17
18
19  inline void  glVertex3f( const Tuple3f & v )
20  {
21     glVertex3f( v.x, v.y, v.z );
22  }
23
24  inline void  glTexCoord2f( const Tuple2f & t )
25  {
26     glTexCoord2f( t.x, t.y );
27  }
28
29  inline void  glNormal3f( const Vector3f & n )
30  {
31     glNormal3f( n.x, n.y, n.z );
32  }
33
34  inline void  glColor3f( const Color3f & c )
35  {
36     glColor3f( c.x, c.y, c.z );
```

```
37 }
38
39 inline void  glColor4f( const Color4f & c )
40 {
41    glColor4f( c.x, c.y, c.z, c.w );
42 }
43
44
45 inline void  glTranslatef( const Tuple3f & t )
46 {
47    glTranslatef( t.x, t.y, t.z );
48 }
49
50 inline void  glMultMatrixf( const Matrix4f & m )
51 {
52    Matrix4f  mat;
53    mat.transpose( m );
54    glMultMatrixf( &mat.m00 );
55 }
56
57 inline void  glMultMatrixf( const Matrix3f & m )
58 {
59    Matrix4f  mat;
60    mat.set( m );
61    mat.transpose();
62    glMultMatrixf( &mat.m00 );
63 }
64
65
66 #endif // _VECMATH_GL_H_
```

1.3 オブジェクトの位置・向きの情報

サンプルプログラム（picking_sample.cpp）の 61~72 行で、オブジェクトの配置情報（位置・向きの情報）を表す ObjectInfo 構造体が定義されている。本構造体は、位置（Point3f pos）と向き（Matrix3f ori）を表すメンバ変数を持つ。また、内部処理で使用するためのメンバ変数として、描画時に使用する位置と向きをまとめた 4×4 変換行列（Matrix4f frame）とスクリーン座標系での位置（Point2f screen_pos）を持つ。

75~76 行で、ObjectInfo 構造体の配列として、全てのオブジェクトの情報を格納するための変数（int num_objects, ObjectInfo * objects）が定義されている。変数 objects には、num_objects 個分の ObjectInfo 構造体の配列の領域を割り当てる。

320~355 行の、シーン初期化を行う InitScene 関数で、オブジェクトを初期化してランダムに配置（pos, ori, frame）を決定する。118~147 行の、全オブジェクトの画面上の位置を更新する UpdateObjectProjection 関数で、全てのオブジェクトのスクリーン座標系での位置（screen_pos メンバ変数）を計算する。

1.4 ピッキング処理

ピッキング方法に関する情報として、サンプルプログラム（picking_sample.cpp）の 84~88 行で、2 種類のピッキング方法を表す列挙型が定義されている。また、91 行で、この列挙型を使って、現在使用するピッキング方法を表すグローバル変数 pick_mode が定義されている。

94 行で、ピッキングにより選択されたオブジェクトの番号を格納するグローバル変数 selected_object_no が定義されている。どのオブジェクトも選択されていないときには、この変数には -1 が格納されているものとする。

さらに、ワールド座標系でのピッキングで使用する変数として、97~98 行で、選択されたオブジェクトの表面上での選択された点の位置の情報を表すグローバル変数 selected_point が定義されている。101~103 行で、画面がクリックされたときのマウス位置に対応する視線ベクトルを表すグローバル変数が定義されている。

ピッキング処理の流れとしては、544~577 行の マウスクリック時に呼ばれる MouseClickCallback 関数の中で、ピッキング処理を行う PickObject 関数を呼び出している。そして、292~308 行の PickObject 関数の中で、現在のピッキング方法に応じて、スクリーン座標でのピッキングを行う PickObjectScreen 関数か、ワールド座標系でのピッキングを行う PickObjectWorld 関数を呼び出している。

PickObject 関数、PickObjectScreen 関数、PickObjectWorld 関数は、いずれも、入力としてマウス座標を受け取り、そのマウス位置に存在するオブジェクトの番号を変えす。マウス位置に対応するオブジェクトが存在しない場合は、-1 を返す。この出力が、グローバル変数 selected_object_no に格納される。さらに、PickObjectScreen 関数（から呼び出される UpdateObjectProjection 関数）では、全てのオブジェクトのスクリーン座標系での位置を求めて、グローバル変数に格納する。また、PickObjectWorld 関数では、選択されたオブジェクトの表面上での選択された点の位置や、マウス位置に対応する視線ベクトルも求めて、グローバル変数に格納する。

2 レポート課題

レポート課題として、スクリーン座標でのピッキングとワールド座標でのピッキングの 2 つの処理を作成する。各処理は、さらに 2 段階の処理に分かれているため、全体で 4 つの処理を作成する。

2.1 スクリーン座標でのピッキング

サンプルプログラム（picking_sample.cpp）の 84~88 行で、スクリーン座標でのピッキングを行う PickObjectScreen 関数が定義されている。この関数では、全てのオブジェクトの位置をワールド座標系からスクリーン座標系に投影変換する処理と、スクリーン座標系での位置比較を行いマウス位置に最も近いオブジェクトを探索する処理の 2 段階の処理で、ピッキングを実現する。このうち、前者の処理は、118~147 行の UpdateObjectProjection 関数を呼び出すことで実現する。以降で、それぞれの処理を作成する。

2.1.1 ワールド座標系からスクリーン座標系への投影変換

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

全てのオブジェクトの画面上での位置を計算する。

ObjectInfo 型の配列 objects の全ての要素に対して、ワールド座標系の位置を表すメンバ変数 空欄 A から、スクリーン座標系の位置を表すメンバ変数 空欄 B を計算する。

OpenGL の gluProjection 関数を呼び出して（ 空欄 C ~ 空欄 F ）、スクリーン座標系での位置をローカル変数 projected_pos に格納し、その値を、スクリーンの左下を原点とする座標系から、スクリーンの左上を原点とする座標系の値に変換して、メンバ変数 空欄 B に格納する（ 空欄 G ~ 空欄 H ）。

ソースコード 4: ワールド座標系からスクリーン座標系への投影変換

```
void  UpdateObjectProjection()
{
   // 計算用変数
   Point3d  projected_pos;

   // OpenGL の変換行列を取得
   double  model_view_matrix[ 16 ];
   double  projection_matrix[ 16 ];
   int  viewport_param[ 4 ];
   glGetDoublev( GL_MODELVIEW_MATRIX, model_view_matrix );
   glGetDoublev( GL_PROJECTION_MATRIX, projection_matrix );
   glGetIntegerv( GL_VIEWPORT, viewport_param );
```

```
    // 各オブジェクトの画面上の位置を計算
    for ( int i=0; i<num_objects; i++ )
    {
        ObjectInfo *  obj = &objects[ i ];

        // ※レポート課題（ここに自分が作成したプログラムを記述する）

        gluProject( 空欄C,
            空欄D : model_view_matrix, 空欄E : projection_matrix, 空欄F : viewport_param,
            &projected_pos.x, &projected_pos.y, &projected_pos.z );

        obj->screen_pos.x = 空欄G;
        obj->screen_pos.y = 空欄H;

    }
}
```

2.1.2 マウス位置に最も近いオブジェクトの探索

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

各オブジェクトの画面上の位置とマウス位置の距離を計算して、距離が一定の閾値以下で、最もマウス位置に近いオブジェクトを探索する（ 空欄 A ～ 空欄 F ）。

ソースコード 5: マウス位置に最も近いオブジェクトの探索

```
int  PickObjectScreen( int mouse_x, int mouxe_y )
{
    // 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
    // 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
    const float  threshold = 20.0f;

    // 全オブジェクトの画面上の位置を更新
    UpdateObjectProjection();

    // ※レポート課題（ここに自分が作成したプログラムを記述する）

    // 計算用変数
    int  closest_no = -1;
    int  closest_dist;
    int  dist, dx, dy;

    // 各オブジェクトに対して繰り返し
    for ( int i=0; i<num_objects; i++ )
    {
        const ObjectInfo &  o = objects[ i ];

        dx = 空欄A;
        dy = 空欄B;
        dist = dx * dx + dy * dy;

        if ( 空欄C )
            continue;

        if ( ( closest_no == -1 ) || ( 空欄D ) )
        {
            closest_no = 空欄E;
            空欄F
        }
    }
```

```
    // 見つかったオブジェクト番号を返す
    return  closest_no;

}
```

2.2 ワールド座標系でのピッキング

サンプルプログラム（picking_sample.cpp）の 193~286 行で、ワールド座標系でのピッキングを行う PickObjectWorld 関数が定義されている。この関数では、マウス位置に対応する視点位置と視線ベクトルを求める処理と、マウス位置に対応する半直線と交差するオブジェクト・三角面を探索する処理（三角面と半直線の交差判定の処理）の 2 段階の処理で、ピッキングを実現する。以降で、それぞれの処理を作成する。

2.2.1 スクリーン座標系からワールド座標系への逆投影変換

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

マウス位置に対応する、視点位置と視線ベクトルを求める。そのために、OpenGL の gluUnProject 関数を呼び出して、マウス位置に対応するワールド座標系での位置 (dx, dy, dz) を計算する（ 空欄 A ~ 空欄 C ）。

ソースコード 6: スクリーン座標系からワールド座標系への逆投影変換

```
int  PickObjectWorld( int mouse_x, int mouse_y )
{
    // 省略

    // マウス位置に対応する 3 次元空間の直線
    double  wx, wy, wz, dx, dy, dz;

    // ※レポート課題（ここに自分が作成したプログラムを記述する）

    gluUnProject( 0.0, 0.0, 10000000000.0, model_view_matrix, projection_matrix,
        viewport_param, &wx, &wy, &wz );
    gluUnProject( 空欄A, 空欄B, 空欄C, model_view_matrix, projection_matrix,
        viewport_param, &dx, &dy, &dz );
    dx = dx - wx;
    dy = dy - wy;
    dz = dz - wz;

    // 省略
}
```

2.2.2 三角面と半直線の交差判定

以下の説明文やプログラムの空欄に入るコードや語句を考えて、プログラムを作成せよ。

マウス位置に対応する半直線と交差するオブジェクト・三角面を探索するための、三角面と半直線の交差判定を行う関数の処理を作成する（ 空欄 A ~ 空欄 G ）。

ソースコード 7: 三角面と半直線の交差判定

```
bool  CheckCross( const Point3f & p0, const Point3f & p1, const Point3f & p2, const
    Point3f & seg_org, const Vector3f & seg_vec, Point3f & cross_point )
```

```
{
    // ※レポート課題（ここに自分が作成したプログラムを記述する）

    // 計算用変数
    Vector3f  norm, x;
    Point3f  p;
    float  t;

    // 三角面を含む超平面の法線を計算
    空欄A

    // 半直線を含む直線と三角面を含む超平面の交点を計算
    t = 空欄B / 空欄C;
    空欄D

    // 交点が半直線の上にあるかを判定
    if ( 空欄E )
        return  false;

    // 交点が三角面の内側にあるかを、p0とp1の辺に対して判定
    空欄F
    if ( 空欄G )
        return  false;

    // 交点が三角面の内側にあるかを、p1とp2の辺に対して判定
    // 省略（上の処理と同じ）

    // 交点が三角面の内側にあるかを、p2とp0の辺に対して判定
    // 省略（上の処理と同じ）

    // 交点を返す
    cross_point.set( p );

    return  true;
}
```