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3回 幾何形状データの読み込み

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

幾何形状モデル（Obj 形式のファイル）を読み込んで表示するプログラムを作成する。キーボードの L キーを押すことで、Obj 形式のファイルを開いて、表示できる。マウスの右ボタン・左ボタンを押しながらドラッグすることで、視点を変更できる。

最初に、サンプルプログラム（obj_viewer.cpp, obj.h, obj.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を説明する。

ソースコード 1: obj_viewer.cpp

```
1  //
2  //  コンピュータアニメーション特論
3  //  幾何形状データ（Obj形式）の読み込み＆描画のサンプルプログラム
4  //
5
6
7  // 基本的なヘッダファイルのインクルード
8  #ifdef _WIN32
9     #include <windows.h>
10 #endif
11 #include <stdio.h>
12 #include <math.h>
13
14 // GLUTヘッダファイルのインクルード
15 #include <GL/glut.h>
16
17 // 幾何形状データ（Obj形式）のデータ構造と関数定義のヘッダファイルをインクルード
18 #include "obj.h"
19
20
21 // 視点操作のための変数
22 float  camera_yaw = -30.0;     // Y軸を中心とする回転角度
23 float  camera_pitch = -30.0;  // X軸を中心とする回転角度
24 float  camera_distance = 15.0;// 中心からカメラの距離
25
26 // マウスのドラッグのための変数
27 int  drag_mouse_r = 0;  // 右ボタンをドラッグ中かどうかのフラグ（0:非ドラッグ中,1:ドラッグ中）
28 int  drag_mouse_l = 0;  // 左ボタンをドラッグ中かどうかのフラグ（0:非ドラッグ中,1:ドラッグ中）
29 int  last_mouse_x;       // 最後に記録されたマウスカーソルのX座標
30 int  last_mouse_y;       // 最後に記録されたマウスカーソルのY座標
31
32 // 幾何形状オブジェクト
33 struct  Obj *  obj = NULL;
34 float  object_y = 0.0f;
35
```

```
36
37
38 //
39 //  格子模様の床を描画
40 //
41 void  DrawFloor( int tile_size, int num_x, int num_z, float r0, float g0, float b0,
       float r1, float g1, float b1 )
42 {
43    int  x, z;
44    float  ox, oz;
45
46    glBegin( GL_QUADS );
47    glNormal3d( 0.0, 1.0, 0.0 );
48
49    ox = -( num_x * tile_size ) / 2;
50    for ( x=0; x<num_x; x++ )
51    {
52       oz = -( num_z * tile_size ) / 2;
53       for ( z=0; z<num_z; z++ )
54       {
55          if ( ( (x + z) % 2 ) == 0 )
56             glColor3f( r0, g0, b0 );
57          else
58             glColor3f( r1, g1, b1 );
59
60          glTexCoord2d( 0.0f, 0.0f );
61          glVertex3d( ox, 0.0, oz );
62          glTexCoord2d( 0.0f, 1.0f );
63          glVertex3d( ox, 0.0, oz + tile_size );
64          glTexCoord2d( 1.0f, 1.0f );
65          glVertex3d( ox + tile_size, 0.0, oz + tile_size );
66          glTexCoord2d( 1.0f, 0.0f );
67          glVertex3d( ox + tile_size, 0.0, oz );
68
69          oz += tile_size;
70       }
71       ox += tile_size;
72    }
73    glEnd();
74 }
75
76
77 //
78 //  ウィンドウ再描画時に呼ばれるコールバック関数
79 //
80 void  DisplayCallback( void )
81 {
82    // 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）
83    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
84
85    // 変換行列を設定（ワールド座標系→カメラ座標系）
86    glMatrixMode( GL_MODELVIEW );
87    glLoadIdentity();
88    glTranslatef( 0.0,  0.0, - camera_distance );
89    glRotatef( - camera_pitch, 1.0, 0.0, 0.0 );
90    glRotatef( - camera_yaw, 0.0, 1.0, 0.0 );
91
92    // 光源位置を設定（モデルビュー行列の変更にあわせて再設定）
93    float  light0_position[] = { 10.0, 30.0, 10.0, 1.0 };
94    glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
95
96    // 格子模様の床を描画
97    DrawFloor( 1.0f, 10, 10, 1.0, 1.0, 1.0, 1.0, 0.8, 0.8 );
98
```

```
99      // 幾何形状を描画
100     if ( obj )
101     {
102        glColor3f( 1.0, 1.0, 1.0 );
103
104        // 変換行列を設定（物体のモデル座標系→カメラ座標系）
105        //（幾何形状が地面に接するように、幾何形状の中心の y座標を設定する）
106        glTranslatef( 0.0f, - object_y, 0.0f );
107
108        // 幾何形状を描画
109        RenderObj( obj );
110     }
111
112     // 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
113     glutSwapBuffers();
114 }
115
116
117 //
118 //  ウィンドウサイズ変更時に呼ばれるコールバック関数
119 //
120 void  ReshapeCallback( int w, int h )
121 {
122     // ウィンドウ内の描画を行う範囲を設定（ウィンドウ全体に描画するように設定）
123     glViewport(0, 0, w, h);
124
125     // カメラ座標系→スクリーン座標系への変換行列を設定
126     glMatrixMode( GL_PROJECTION );
127     glLoadIdentity();
128     gluPerspective( 45, (double)w/h, 1, 500 );
129 }
130
131
132 //
133 //  マウスクリック時に呼ばれるコールバック関数
134 //
135 void  MouseClickCallback( int button, int state, int mx, int my )
136 {
137     // 右ボタンが押されたらドラッグ開始のフラグを設定
138     if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_DOWN ) )
139        drag_mouse_l = 1;
140     // 右ボタンが離されたらドラッグ終了のフラグを設定
141     else if ( ( button == GLUT_LEFT_BUTTON ) && ( state == GLUT_UP ) )
142        drag_mouse_l = 0;
143
144     // 右ボタンが押されたらドラッグ開始のフラグを設定
145     if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_DOWN ) )
146        drag_mouse_r = 1;
147     // 右ボタンが離されたらドラッグ終了のフラグを設定
148     else if ( ( button == GLUT_RIGHT_BUTTON ) && ( state == GLUT_UP ) )
149        drag_mouse_r = 0;
150
151     // 現在のマウス座標を記録
152     last_mouse_x = mx;
153     last_mouse_y = my;
154 }
155
156
157 //
158 //  マウスドラッグ時に呼ばれるコールバック関数
159 //
160 void  MouseDragCallback( int mx, int my )
161 {
162     // 左ボタンのドラッグ中であれば、マウスの移動量に応じて視点を移動する
```

```
163     if ( drag_mouse_l == 1 )
164     {
165         // マウスの縦移動に応じて距離を移動
166         camera_distance += ( my - last_mouse_y ) * 0.2;
167         if ( camera_distance < 5.0 )
168             camera_distance = 5.0;
169     }
170
171     // 右ボタンのドラッグ中であれば、マウスの移動量に応じて視点を回転する
172     if ( drag_mouse_r == 1 )
173     {
174         // マウスの横移動に応じてY軸を中心に回転
175         camera_yaw -= ( mx - last_mouse_x ) * 1.0;
176         if ( camera_yaw < 0.0 )
177             camera_yaw += 360.0;
178         else if ( camera_yaw > 360.0 )
179             camera_yaw -= 360.0;
180
181         // マウスの縦移動に応じてX軸を中心に回転
182         camera_pitch -= ( my - last_mouse_y ) * 1.0;
183         if ( camera_pitch < -90.0 )
184             camera_pitch = -90.0;
185         else if ( camera_pitch > 0.0 )
186             camera_pitch = 0.0;
187     }
188
189     // 今回のマウス座標を記録
190     last_mouse_x = mx;
191     last_mouse_y = my;
192
193     // 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）
194     glutPostRedisplay();
195 }
196
197
198 //
199 //  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
200 //
201 void  KeyboardCallback( unsigned char key, int mx, int my )
202 {
203     // l キーで幾何形状を読み込み
204     if ( key == 'l' )
205     {
206         const int  file_name_len = 256;
207         char  file_name[ file_name_len ] = "";
208
209         // ファイルダイアログの設定
210         OPENFILENAME  open_file;
211         memset( &open_file, 0, sizeof(OPENFILENAME) );
212         open_file.lStructSize = sizeof(OPENFILENAME);
213         open_file.hwndOwner = NULL;
214         open_file.lpstrFilter = "Obj File (*.obj)\0*.obj\0すべて(*.*)\0*.*\0";
215         open_file.nFilterIndex = 1;
216         open_file.lpstrFile = file_name;
217         open_file.nMaxFile = file_name_len;
218         open_file.lpstrTitle = "Obj File";
219         open_file.lpstrDefExt = "obj";
220         open_file.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
221
222         // ファイルダイアログを表示
223         BOOL  ret = GetOpenFileName( &open_file );
224
225         // ファイルが指定されたら新しい幾何形状を読み込み
226         if( ret )
```

```
227         {
228             // 幾何形状ファイルの読み込み
229             obj = LoadObj( file_name );
230 
231             // 読み込んだ幾何形状をスケーリングする（スケーリング後の中心の高さを記録）
232             object_y = ScaleObj( obj, 5.0f );
233 
234             // 画面の再描画
235             glutPostRedisplay();
236         }
237     }
238 }
239 
240 
241 //
242 //  環境初期化関数
243 //
244 void  initEnvironment( void )
245 {
246     // 光源を作成する
247     float  light0_position[] = { 10.0, 30.0, 10.0, 1.0 };
248     float  light0_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
249     float  light0_specular[] = { 1.0, 1.0, 1.0, 1.0 };
250     float  light0_ambient[] = { 0.1, 0.1, 0.1, 1.0 };
251     glLightfv( GL_LIGHT0, GL_POSITION, light0_position );
252     glLightfv( GL_LIGHT0, GL_DIFFUSE, light0_diffuse );
253     glLightfv( GL_LIGHT0, GL_SPECULAR, light0_specular );
254     glLightfv( GL_LIGHT0, GL_AMBIENT, light0_ambient );
255     glEnable( GL_LIGHT0 );
256 
257     // 光源計算を有効にする
258     glEnable( GL_LIGHTING );
259 
260     // 物体の色情報を有効にする
261     glEnable( GL_COLOR_MATERIAL );
262 
263     // Ｚテストを有効にする
264     glEnable( GL_DEPTH_TEST );
265 
266     // 背面除去を有効にする
267     glCullFace( GL_BACK );
268     glEnable( GL_CULL_FACE );
269 
270     // 背景色を設定
271     glClearColor( 0.5, 0.5, 0.8, 0.0 );
272 }
273 
274 
275 //
276 //  メイン関数（プログラムはここから開始）
277 //
278 int  main( int argc, char ** argv )
279 {
280     // GLUTの初期化
281     glutInit( &argc, argv );
282     glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA );
283     glutInitWindowSize( 640, 640 );
284     glutInitWindowPosition( 0, 0 );
285     glutCreateWindow( "Obj Viewer" );
286 
287     // コールバック関数の登録
288     glutDisplayFunc( DisplayCallback );
289     glutReshapeFunc( ReshapeCallback );
290     glutMouseFunc( MouseClickCallback );
```

```
291     glutMotionFunc( MouseDragCallback );
292     glutKeyboardFunc( KeyboardCallback );
293
294     // 環境初期化
295     initEnvironment();
296
297     // GLUTのメインループに処理を移す
298     glutMainLoop();
299
300     // プログラムを終了
301     return 0;
302 }
```

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  //  幾何形状データ（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 // 幾何形状データの例（本プログラムでは使用しない）
25 struct  SampleGeometry
26 {
27     int        num_vertices;  // 頂点数
28     Vector *   vertices;      // 頂点座標配列 [num_vertices]
29     Vector *   normals;       // 法線ベクトル配列 [num_vertices]
30     Color *    colors;        // カラー配列 [num_vertices]
31
32     int        num_triangles; // 三角面数
33     int *      triangles;     // 三角面の頂点番号配列 [num_triangles*3]
34 };
35
36
37 // マテリアルデータ（Mtl形式用）
38 struct  Mtl
39 {
40     char *      name;           // マテリアル名
41
42     Color       kd;             // 拡散反射光（とりあえず拡散反射光を glColor3f() で使用する）
43
44     char *      texture_name;   // テクスチャ画像のファイル名
45 };
46
47
48 // 幾何形状データ（Obj形式用）
```

```
49 struct  Obj
50 {
51     int         num_vertices;  // 頂点数
52     Vector *    vertices;      // 頂点座標配列 [num_vertices]
53
54     int         num_normals;
55     Vector *    normals;       // 法線ベクトル配列 [num_normals]
56
57     int         num_tex_coords;
58     Vector *    tex_coords;    // テクスチャ座標配列 [num_tex_coords]
59
60     int         num_triangles; // 三角面数
61     int *       tri_v_no;      // 三角面の各頂点の頂点座標番号配列 [num_triangles*3]
62     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]
63     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]
64     Mtl **      tri_material;  // 三角面の素材 [num_triangles]
65
66     int         num_materials; // マテリアル数
67     Mtl **      materials;     // マテリアルの配列 [num_materials]
68 };
69
70
71
72 // Objファイルの読み込み
73 Obj *  LoadObj( const char * filename );
74
75 // Mtlファイルの読み込み
76 void  LoadMtl( const char * filename, Obj * obj );
77
78 // オブジェクトのスケーリング（スケーリング後の中心の高さを返す）
79 float  ScaleObj( Obj * obj, float max_size );
80
81 // Obj形状データの描画
82 void  RenderObj( Obj * obj );
83
84
85 #endif // _OBJ_H_
```

ソースコード 3: obj.cpp

```
1  //
2  //  コンピュータアニメーション特論
3  //  幾何形状データ（Obj形式）の読み込み&描画のサンプルプログラム
4  //
5
6
7  // 基本的なヘッダファイルのインクルード
8  #ifdef _WIN32
9     #include <windows.h>
10    #pragma warning( disable: 4996 )
11 #endif
12 #include <stdio.h>
13 #include <GL/gl.h>
14
15 // 幾何形状データの定義のインクルード
16 #include "obj.h"
17
18 // バッファ長（サイズは適当）
19 #define  BUFFER_LENGTH       1024
20 #define  MAX_VECTOR_SIZE     4096
21 #define  MAX_TRIANGLE_SIZE   4096
22 #define  MAX_MTL_SIZE          32
23
```

```
24
25 //
26 //  Objファイルの読み込み
27 //
28 Obj *  LoadObj( const char * filename )
29 {
30    FILE *  fp;
31    char    line[ BUFFER_LENGTH ];
32    char    name[ BUFFER_LENGTH ];
33    int  i, j;
34    Vector  vec;
35    int  v_no[4], vt_no[4], vn_no[4];
36    int  count;
37    Mtl *  curr_mtl = NULL;
38
39    // ファイルを開く
40    fp = fopen( filename, "r" );
41    if ( fp == NULL )
42       return  NULL;
43
44    // Obj構造体を初期化（ひとまず固定サイズの配列を割り当てる）
45    Obj *  obj = new Obj();
46    obj->num_vertices = 0;
47    obj->num_normals = 0;
48    obj->num_tex_coords = 0;
49    obj->vertices = new Vector[ MAX_VECTOR_SIZE ];
50    obj->normals = new Vector[ MAX_VECTOR_SIZE ];
51    obj->tex_coords = new Vector[ MAX_VECTOR_SIZE ];
52    obj->num_triangles = 0;
53    obj->tri_v_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
54    obj->tri_vn_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
55    obj->tri_vt_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
56    obj->tri_material = new Mtl*[ MAX_TRIANGLE_SIZE ];
57    obj->num_materials = 0;
58    obj->materials = NULL;
59
60    // ファイルから1行ずつ読み込み
61    while ( fgets( line, BUFFER_LENGTH, fp ) != NULL )
62    {
63       // マテリアルの読み込み
64       if ( strncmp( line, "mtllib", 6 ) == 0 )
65       {
66          // テキストを解析
67          sscanf( line, "mtllib %s", name );
68
69          // 指定されたファイル名のマテリアルデータを読み込み
70          if ( strlen( name ) > 0 )
71             LoadMtl( name, obj );
72       }
73
74       // マテリアルの変更
75       if ( strncmp( line, "usemtl", 6 ) == 0 )
76       {
77          // テキストを解析
78          sscanf( line, "usemtl %s", name );
79
80          // 指定された名前のマテリアルデータを探索して記録
81          for ( i=0; i<obj->num_materials; i++ )
82          {
83             if ( strcmp( name, obj->materials[ i ]->name ) == 0 )
84             {
85                curr_mtl = obj->materials[ i ];
86                break;
87             }
```

```
88            }
89        }
90
91        // 頂点データの読み込み
92        if ( line[0] == 'v' )
93        {
94            // 法線ベクトル（vn）
95            if ( line[1] == 'n' )
96            {
97                // テキストを解析
98                sscanf( line, "vn %f %f %f", &vec.x, &vec.y, &vec.z );
99
100               // 法線ベクトル配列の末尾に格納
101               if ( obj->num_normals < MAX_VECTOR_SIZE )
102               {
103                   obj->normals[ obj->num_normals ] = vec;
104                   obj->num_normals ++;
105               }
106           }
107           // テクスチャ座標（vt）
108           else if ( line[1] == 't' )
109           {
110               // テキストを解析
111               sscanf( line, "vt %f %f %f", &vec.x, &vec.y, &vec.z );
112
113               // テクスチャ座標配列の末尾に格納
114               if ( obj->num_tex_coords < MAX_VECTOR_SIZE )
115               {
116                   obj->tex_coords[ obj->num_tex_coords ] = vec;
117                   obj->num_tex_coords ++;
118               }
119           }
120           // 頂点座標（v）
121           else
122           {
123               // テキストを解析
124               sscanf( line, "v %f %f %f", &vec.x, &vec.y, &vec.z );
125
126               // 法線ベクトル配列の末尾に格納
127               if ( obj->num_vertices < MAX_VECTOR_SIZE )
128               {
129                   obj->vertices[ obj->num_vertices ] = vec;
130                   obj->num_vertices ++;
131               }
132           }
133       }
134
135       // ポリゴンデータの読み込み
136       if ( line[0] == 'f' )
137       {
138           // 未実装（各自実装）
139
140           // テキストを解析（三角形・テクスチャ座標なしの場合）
141           count = sscanf( line, "f %i//%i %i//%i %i//%i", &v_no[0], &vn_no[0], &v_no[1],
                  &vn_no[1], &v_no[2], &vn_no[2] );
142
143           // 解析に成功したらポリゴンデータを記録
144           if ( count == 6 )
145           {
146               i = obj->num_triangles * 3;
147               for ( j=0; j<3; j++ )
148               {
149                   obj->tri_v_no[ i+j ] = v_no[ j ] - 1;  //
                          Obj形式ではインデックス番号は1から始まるので、－1して0から始まるようにする
```

```
150                 obj->tri_vn_no[ i+j ] = vn_no[ j ] - 1;
151                 obj->tri_vt_no[ i+j ] = vt_no[ j ] - 1;
152             }
153             obj->tri_material[ obj->num_triangles ] = curr_mtl;
154             obj->num_triangles ++;
155         }
156         // 解析に失敗したら別のフォーマットを試す
157         {
158             // ......
159 
160             // 未実装（各自実装）
161         }
162 
163         // ポリゴン数が確保した配列の大きさを超えたら強制終了
164         if ( obj->num_triangles >= MAX_TRIANGLE_SIZE )
165             break;
166     }
167 };
168 
169 // 必要な配列を確保しなおす
170 Vector *  new_array;
171 int *  new_nums;
172 Mtl **  new_mtls;
173 
174 new_array = new Vector[ obj->num_vertices ];
175 memcpy( new_array, obj->vertices, sizeof( Vector ) * obj->num_vertices );
176 delete[]  obj->vertices;
177 obj->vertices = new_array;
178 
179 new_array = new Vector[ obj->num_normals ];
180 memcpy( new_array, obj->normals, sizeof( Vector ) * obj->num_normals );
181 delete[]  obj->normals;
182 obj->normals = new_array;
183 
184 new_array = new Vector[ obj->num_tex_coords ];
185 memcpy( new_array, obj->tex_coords, sizeof( Vector ) * obj->num_tex_coords );
186 delete[]  obj->tex_coords;
187 obj->tex_coords = new_array;
188 
189 new_nums = new int[ obj->num_triangles * 3 ];
190 memcpy( new_nums, obj->tri_v_no, sizeof( int ) * obj->num_triangles * 3 );
191 delete  obj->tri_v_no;
192 obj->tri_v_no = new_nums;
193 
194 new_nums = new int[ obj->num_triangles * 3 ];
195 memcpy( new_nums, obj->tri_vn_no, sizeof( int ) * obj->num_triangles * 3 );
196 delete  obj->tri_vn_no;
197 obj->tri_vn_no = new_nums;
198 
199 new_nums = new int[ obj->num_triangles * 3 ];
200 memcpy( new_nums, obj->tri_vt_no, sizeof( int ) * obj->num_triangles * 3 );
201 delete  obj->tri_vt_no;
202 obj->tri_vt_no = new_nums;
203 
204 new_mtls = new Mtl*[ obj->num_triangles ];
205 memcpy( new_mtls, obj->tri_material, sizeof( Mtl* ) * obj->num_triangles );
206 delete[]  obj->tri_material;
207 obj->tri_material = new_mtls;
208 
209 // ファイルを閉じる
210 fclose( fp );
211 
212 // 読み込んだオブジェクトデータを返す
```

```
213     return  obj;
214 }
215
216
217 //
218 //  Mtlファイルの読み込み
219 //
220 void  LoadMtl( const char * filename, Obj * obj )
221 {
222     FILE *  fp;
223     char    line[ BUFFER_LENGTH ];
224     char    name[ BUFFER_LENGTH ];
225     Color   color;
226     Mtl *   curr_mtl = NULL;
227
228     // ファイルを開く
229     fp = fopen( filename, "r" );
230     if ( fp == NULL )
231         return;
232
233     // Mtl配列を初期化（ひとまず固定サイズの配列を割り当てる）
234     obj->num_materials = 0;
235     obj->materials = new Mtl*[ MAX_MTL_SIZE ];
236
237     // ファイルから1行ずつ読み込み
238     while ( fgets( line, BUFFER_LENGTH, fp ) != NULL )
239     {
240         // マテリアルデータの追加
241         if ( strncmp( line, "newmtl", 6 ) == 0 )
242         {
243             // テキストを解析
244             sscanf( line, "newmtl %s", name );
245             if ( strlen( name ) == 0 )
246                 continue;
247
248             // マテリアルデータの作成
249             curr_mtl = new Mtl();
250             curr_mtl->name = new char[ strlen( name ) + 1 ];
251             strcpy( curr_mtl->name, name );
252             curr_mtl->kd.r = 0.8f;
253             curr_mtl->kd.g = 0.8f;
254             curr_mtl->kd.b = 0.8f;
255
256             // マテリアルデータを配列に記録
257             obj->materials[ obj->num_materials ] = curr_mtl;
258             obj->num_materials ++;
259         }
260
261         // 反射特性データの読み込み
262         if ( line[0] == 'K' )
263         {
264             // 拡散反射特性（Kd）
265             if ( line[1] == 'd' )
266             {
267                 // テキストを解析
268                 sscanf( line, "Kd %f %f %f", &color.r, &color.g, &color.b );
269
270                 // 拡散反射特性（Kd）を記録
271                 if ( curr_mtl )
272                     curr_mtl->kd = color;
273             }
274         }
275
276     }
```

```
277
278     // ファイルを閉じる
279     fclose( fp );
280 }
281
282
283 //
284 //  オブジェクトのスケーリング（スケーリング後の中心の高さを返す）
285 //
286 float  ScaleObj( Obj * obj, float max_size )
287 {
288     if ( !obj || ( obj->num_vertices == 0 ) )
289         return  0.0;
290
291     // サイズ計算
292     Vector  min, max;
293     min = max = obj->vertices[ 0 ];
294     int  i;
295     for ( i=0; i<obj->num_vertices; i++ )
296     {
297         const Vector &  p = obj->vertices[ i ];
298         if ( p.x < min.x )  min.x = p.x;
299         if ( p.y < min.y )  min.y = p.y;
300         if ( p.z < min.z )  min.z = p.z;
301         if ( p.x > max.x )  max.x = p.x;
302         if ( p.y > max.y )  max.y = p.y;
303         if ( p.z > max.z )  max.z = p.z;
304     }
305
306     // スケーリング
307     float  size, scale;
308     size = ( ( max.x - min.x ) > ( max.z - min.z ) ) ? ( max.x - min.x ) : ( max.z - min
            .z );
309     scale = max_size / size;
310     for ( i=0; i<obj->num_vertices; i++ )
311     {
312         obj->vertices[ i ].x *= scale;
313         obj->vertices[ i ].y *= scale;
314         obj->vertices[ i ].z *= scale;
315     }
316
317     // スケーリング後の中心の高さを返す
318     float  object_y = min.y * scale;
319     return  object_y;
320 }
321
322
323 //
324 //  Obj形状データの描画
325 //
326 void  RenderObj( Obj * obj )
327 {
328     int  i, j, no;
329     Mtl *  curr_mtl = NULL;
330
331     glBegin( GL_TRIANGLES );
332     for ( i=0; i<obj->num_triangles; i++ )
333     {
334         // マテリアルの切り替え
335         if ( ( obj->num_materials > 0 ) && ( obj->tri_material[ i ] != curr_mtl ) )
336         {
337             curr_mtl = obj->tri_material[ i ];
338             glColor3f( curr_mtl->kd.r, curr_mtl->kd.g, curr_mtl->kd.b );
339         }
```

```
340
341         // 三角面の各頂点データの指定
342         for ( j=0; j<3; j++ )
343         {
344             // 法線ベクトル
345             no = obj->tri_vn_no[ i*3 + j ];
346             const Vector &  vn = obj->normals[ no ];
347             glNormal3f( vn.x, vn.y, vn.z );
348
349             // 頂点座標
350             no = obj->tri_v_no[ i*3 + j ];
351             const Vector &  v = obj->vertices[ no ];
352             glVertex3f( v.x, v.y, v.z );
353         }
354     }
355     glEnd();
356 }
```

1.1 幾何形状データの定義

サンプルプログラム（obj.h, obj.cpp）で、Obj 形式の幾何形状モデルの管理・読み込み・描画を行うための構造体や関数を定義・実装している。

サンプルプログラム（obj.h）では、3 次元座標やベクトルを表す Vector 構造体、色を表す Color 構造体、Obj 形式の幾何形状データを表す Obj 構造体などを定義している。また、Obj ファイルの読み込みを行う LoadObj 関数や、描画を行う RenderObj 関数を定義している。

1.2 幾何形状モデルの読み込み

サンプルプログラム（obj.cpp）の 28〜181 行に、Obj 形式の幾何形状モデルの読み込みを行う LoadObj 関数が記述されている。LoadObj 関数は、ファイル名を入力として受け取り、読み込んだ Obj 構造体のオブジェクトを戻り値として出力する。本関数では、C 言語の標準関数である stdio を使ったファイルの読み込みや、scanf 系の標準関数を使った文字列の解析を行っている。また、可変長のデータの読み込みに対しては、あらかじめ十分大きな配列を確保しておくことで対応しており、確保済みの配列の大きさを超えるサイズには未対応である。

以降、ファイル読み込み処理の主な手順を説明する。

45〜58 行で、Obj 構造体のオブジェクトを生成し、そのメンバ変数として、読み込み結果の格納に使用する、頂点に関する配列（頂点座標 vetices、法線ベクトル normals、テクスチャ座標 tex_coords）、三角面に関する配列（頂点番号 tri_v_no、法線ベクトル番号 tri_vn_no、テクスチャ座標番号 tri_vt_no、素材番号 tri_material）を、最初に定義された定数分の大きめなサイズで確保している。これらの配列は暫定的なもので、最終的には、読み込んだデータの量に合わせて、適切な大きさの配列を生成して使用する。

40 行で、stdio ライブラリの fopen 関数を呼び出して、アスキー形式の読み込みモードで、ファイルを開いている。fopen 関数の戻り値として、ファイルポインタ（fp）が得られるので、以降、このファイルポインタに対して、読み込みの処理を行っていく。

61 行以降、fgets 関数を呼び出して、ファイルから 1 行ずつ読み込みながら、読み込んだ文字列を解析して幾何形状データに格納する処理を行っていく。64 行以降の if 文では、strncmp 関数を使って、読み込んだ行の先頭が、Obj 形式の各コマンドに対応するかを判定する。頂点データ（92 行以降）の場合は、1 文字目が v になるため、まずは、1 文字目が v であるかを判定して、その後、2 文字目の文字で、頂点座標、法線ベクトル、テクスチャ座標のどの情報を表すかを判定する。

頂点データやポリゴンデータを表す文字列の解析には、sscanf 関数を使用する。sscanf 関数を使って文字列の解析を行うときには、固定の書式を使うことになる。ポリゴンデータは、頂点数や指定される情報（頂点座標番号、テクスチャ座標番号、法線ベクトル番号）によって、異なる形式になる。本プログラムでは、処理を簡単にする

ために、全てのポリゴンが三角形で、テクスチャ座標は省略される場合を想定した、「f v0//n0 v1//n1 v2//n2」（v0〜v1 には頂点座標番号、n0〜n2 には法線ベクトル番号が入る）の形式の入力のみ対応している。

マテリアル情報の読み込みは、220 行〜280 行で定義されている、Mtl 形式のファイルを読み込んで Obj 構造体のメンバ変数にマテリアル情報を追加する LoatMtl 関数を呼び出している。

170 行以降、読み込んだデータサイズに合わせて、新しい配列を確保して、読み込んだデータをそちらにコピーし、読み込みに使用した大きな配列は解放している。

1.3 幾何形状モデルの描画

サンプルプログラム（obj.cpp）の 29〜182 行に、Obj 形式の幾何形状モデルの描画を行う RenderObj 関数が記述されている。本関数では、図形の種類として三角形（GL_TRIANGLES）を指定して、OpenGL の glBegin 関数・glEnd 関数を呼び出して、各三角面の頂点情報（法線ベクトルと頂点座標）を順番に渡すことで、描画を行っている。各三角面が素材の情報を持っているが、無駄な設定変更は省略するために、前の三角面とは異なる素材情報を使用する三角面の場合のみ、素材情報（色）の設定を変更している。

1.4 メインプログラム

サンプルプログラム（obj_viewer.cpp）の大まかな構成は、前回の GLUT を用いた視点操作のサンプルプログラムと同様である。

21〜30 行で、視点操作のためのグローバル変数の定義を行っている。78〜238 行で、5 つの GLUT コールバック関数を定義しており、マウスクリック時に呼ばれるコールバック関数（MouseClickCallback 関数）、マウスドラッグ時に呼ばれるコールバック関数（MouseDragCallback 関数）の中で、媒介変数による視点操作の処理を行っている。

33 行で、読み込んだ幾何形状モデルを格納するためのグルーバル変数（obj）の定義を行っている。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（KeyboardCallback 関数）で、L キーが押されたときに、幾何形状モデルの読み込みを行う LoadObj 関数を呼び出して、グルーバル変数 obj に格納している。また、細かい説明は省略するが、読み込んだ幾何形状モデルを適当な大きさで表示するために、幾何形状のスケーリングを行う ScaleObj 関数を呼び出して、中心の高さをグローバル変数 object_y に格納している。中心の高さは、ウィンドウ再描画時に呼ばれるコールバック関数（DisplayCallback 関数）で、幾何形状モデルの描画処理（RenderObj 関数）を呼び出す際に、幾何形状モデルの下端が地面に接する高さに描画するために使用する。

2 サンプルプログラム 2

C++を使用した幾何形状モデル（Obj 形式のファイル）を読み込み処理のプログラムの例を示す。

最初に、サンプルプログラム（WavefrontObj.h, WavefrontObj.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を説明する。

ソースコード 4: WavefrontObj.h

```
1  //
2  //   コンピュータアニメーション特論
3  //   幾何形状データ（Obj形式）の読み込み＆描画のサンプルプログラム
4  //
5
6
7  #ifndef   _WAVEFRONT_OBJ_H_
8  #define   _WAVEFRONT_OBJ_H_
9
10
11 #include <vector>
12 #include <map>
```

```
13 #include <string>
14
15 using namespace  std;
16
17
18
19 //
20 //  Alias|Wavefront Obj形式 の幾何形状データ
21 //
22 class  WavefrontObj
23 {
24   public:
25     /*  構造体  */
26
27     // 頂点
28     struct  Vertex
29     {
30        double   x, y, z;
31     };
32
33     // カラー
34     struct  Color
35     {
36        double   r, g, b;
37     };
38
39     struct  Face;
40
41     // グループ
42     struct  Group
43     {
44        string            name;
45        vector< int >     faces;
46     };
47
48     // マテリアル
49     struct  Material
50     {
51        string            name;
52        Color             ambient;
53        Color             diffuse;
54        Color             specular;
55        string            diffuse_map;
56        vector< int >     faces;
57     };
58
59     // エレメント
60     struct  Element
61     {
62        Group  *    group;
63        Material *  material;
64     };
65
66     // 頂点
67     struct  Face : public Element
68     {
69        vector< int >   data;
70
71        int  NumVertex() const          { return  data.size() / 3; }
72        int  GetVertex( int n ) const   { return  data[ n*3 ]; }
73        int  GetTexture( int n ) const  { return  data[ n*3+1 ]; }
74        int  GetNormal( int n ) const   { return  data[ n*3+2 ]; }
75     };
76
```

```
77
78    private:
79     /*  内部データ  */
80     vector< Group * >      groups;
81     vector< Material * >   materials;
82     vector< Vertex >       vertices;
83     vector< Vertex >       normals;
84     vector< Vertex >       t_coords;
85     vector< Face * >       faces;
86
87     map< string, Group * >  group_index;
88     map< string, Material * >  mtl_index;
89
90
91    public:
92     /*  コンストラクタ・デストラクタ  */
93     WavefrontObj( const char * file_name );
94     ~WavefrontObj();
95
96    protected:
97     // マテリアルファイルの読み込み
98     void  LoadMaterialFile( const char * file_name );
99
100   public:
101    /*  アクセサ  */
102    const int  NumGroups() const  { return groups.size(); }
103    const Group * GetGroup( int n ) const  { return groups[n]; }
104    const Group * GetGroup( const string & s ) const  {
105       return ( (group_index.find( s ) != group_index.end()) ? (*group_index.find( s )).
           second : NULL ); }
106    const int  NumMaterials() const  { return materials.size(); }
107    const Material * GetMaterial( int n ) const  { return materials[n]; }
108    const int  NumVertices() const  { return vertices.size(); }
109    const Vertex & GetVertex( int n ) const  { return vertices[n]; }
110    const int  NumNormals() const  { return normals.size(); }
111    const Vertex & GetNormal( int n ) const  { return normals[n]; }
112    const int  NumTextureCoords() const  { return t_coords.size(); }
113    const Vertex & GetTextureCoords( int n ) const  { return t_coords[n]; }
114    const int  NumFaces() const  { return faces.size(); }
115    const Face * GetFace( int n ) const  { return faces[n]; }
116
117    // OpenGLを使用してオブジェクトを描画
118    void  Draw();
119 };
120
121
122
123 #endif // _WAVEFRONT_OBJ_H_
```

ソースコード 5: WavefrontObj.cpp

```
1  //
2  //  コンピュータアニメーション特論
3  //  幾何形状データ（Obj形式）の読み込み＆描画のサンプルプログラム
4  //
5
6
7  #include <fstream>
8  #pragma warning( disable: 4996 )
9
10 #include "WavefrontObj.h"
11
12
13 // バッファ長
```

```
14 #define  BUFFER_LENGTH  1024
15
16
17 //
18 //  コンストラクタ（ローダ）
19 //
20 WavefrontObj::WavefrontObj( const char * file_name )
21 {
22    ifstream  file;
23    char      line[ BUFFER_LENGTH ];
24    char *    token;
25    char *    data;
26    Group *     curr_group = NULL;
27    Material *  curr_mtl = NULL;
28    vector< int >   face_data;
29
30    // ファイルのオープン
31    file.open( file_name, ios::in );
32    if ( file.is_open() == 0 )  return; // ファイルが開けなかったら終了
33
34    // ファイルを先頭から1行ずつ順に読み込み
35    while ( ! file.eof() )
36    {
37       // 1行読み込み、先頭の単語を取得
38       file.getline( line, BUFFER_LENGTH );
39       token = strtok( line, " " );
40
41       // 空行・コメント行の場合は次の行へ
42       if ( (token == NULL ) || ( *token == '#' ) )
43          continue;
44
45       // 点データの読み込み
46       if ( *token == 'v' )
47       {
48          Vertex  v;
49          data = strtok( NULL, " " );  v.x = data ? atof( data ) : 0.0;
50          data = strtok( NULL, " " );  v.y = data ? atof( data ) : 0.0;
51          data = strtok( NULL, " " );  v.z = data ? atof( data ) : 0.0;
52
53          token ++;
54          if ( *token == 't' )
55             t_coords.push_back( v );
56          else if ( *token == 'n' )
57             normals.push_back( v );
58          else
59             vertices.push_back( v );
60          continue;
61       }
62
63       // 面データの読み込み
64       if ( *token == 'f' )
65       {
66          face_data.clear();
67          while ( (data = strtok( NULL, " /" )) != NULL )
68          {
69             // テクスチャ番号が省略された時（'/' が続いた時）は -1 を設定
70             if ( *(data - 1) == '/' )
71                face_data.push_back( -1 );
72
73             // Objファイルでは頂点番号は1から始まっているので -1して0から開始にする
74             face_data.push_back( atoi( data ) - 1 );
75
76             // 法線データがない場合は、自動的に法線ベクトル番号に -1 を設定
77             if ( ( ( face_data.size() % 3 ) == 2 ) && ( normals.size() == 0 ) )
```

```
78                face_data.push_back( face_data[ face_data.size() - 2 ] );
79            }
80
81            Face *  face = new Face();
82            face->group = curr_group;
83            face->material = curr_mtl;
84            face->data = face_data;
85            faces.push_back( face );
86
87            if ( curr_group )  curr_group->faces.push_back( faces.size() - 1 );
88            if ( curr_mtl )  curr_mtl->faces.push_back( faces.size() - 1 );
89            continue;
90        }
91
92        // グループ名の読み込み
93        if ( *token == 'g' )
94        {
95            data = strtok( NULL, " " );
96            if ( data == NULL )  { curr_group = NULL;  continue; }
97
98            map< string, Group * >::iterator  it;
99            it = group_index.find( data );
100           if ( it != group_index.end() )
101               curr_group = (*it).second;
102           else
103           {
104               Group *  group = new Group();
105               group->name = data;
106               groups.push_back( group );
107               group_index[ group->name ] = group;
108               curr_group = group;
109           }
110           continue;
111       }
112
113       // マテリアル名の読み込み
114       if ( strcmp( token, "usemtl" ) == 0 )
115       {
116           data = strtok( NULL, " " );
117           if ( data == NULL )  { curr_mtl = NULL;  continue; }
118
119           map< string, Material * >::iterator  it;
120           it = mtl_index.find( data );
121           if ( it != mtl_index.end() )
122               curr_mtl = (*it).second;
123           else
124           {
125               Material *  material = new Material();
126               material->name = data;
127               material->ambient.r = 0.6f;
128               material->ambient.g = 0.6f;
129               material->ambient.b = 0.6f;
130               material->diffuse.r = 0.6f;
131               material->diffuse.g = 0.6f;
132               material->diffuse.b = 0.6f;
133               material->specular.r = 0.1f;
134               material->specular.g = 0.1f;
135               material->specular.b = 0.1f;
136               materials.push_back( material );
137               mtl_index[ material->name ] = material;
138               curr_mtl = material;
139           }
140           continue;
141       }
```

```
142
143         // マテリアルファイルの読み込み
144         if ( strcmp( token, "mtllib" ) == 0 )
145         {
146             data = strtok( NULL, " " );
147             if ( data == NULL )  { curr_mtl = NULL;  continue; }
148
149             string  mtl_file_name( data );
150             const char *  path = strrchr( file_name, '\\' );
151             if ( path != NULL )
152             {
153                 mtl_file_name.assign( file_name, path + 1 );
154                 mtl_file_name.append( data );
155             }
156
157             LoadMaterialFile( mtl_file_name.c_str() );
158         }
159
160     }
161 }
162
163
164 //
165 //  マテリアルファイルの読み込み
166 //
167 void WavefrontObj::LoadMaterialFile( const char * file_name )
168 {
169     // 省略（各自作成）
170 }
171
172
173 //
174 //  OpenGLを使用してオブジェクトを描画
175 //
176 void WavefrontObj::Draw()
177 {
178     // 省略（各自作成）
179 }
```

2.1 幾何形状データの定義

サンプルプログラム（WavefrontObj.h）では、Obj 形式の幾何形状データを表す WavefrontObj クラスを定義している。クラス内の構造体として、3 次元座標やベクトルを表す Vector 構造体、色を表す Color 構造体、ポリゴンのグループを表す Group 構造体、素材情報を表す Material 構造体、面の情報を表す Element 構造体と Face 構造体などを定義している。幾何形状のデータは、Standard Template Library (STL) の可変長配列 vector クラスを使ったメンバ変数として、定義している。また、Obj ファイルの読み込みを行うコンストラクタや、描画を行う Draw 関数などのメンバ関数を定義している。

2.2 幾何形状モデルの読み込み

サンプルプログラム（WavefrontObj.cpp）の 28〜161 行に、Obj 形式の幾何形状モデルの読み込みを行う WavefrontObj クラスのコンストラクタ（WavefrontObj メンバ関数）が記述されている。WavefrontObj メンバ関数は、ファイル名を入力として受け取り、読み込んだ幾何形状モデルの情報を、メンバ変数に設定する。本関数では、C++言語の標準関数である iostream を使ったファイルの読み込みや、tokenizer を使った文字列の解析を行っている。また、可変長のデータの読み込みに対しては、STL の可変長配列 vector クラスを用いることで対応している。

以降、ファイル読み込み処理の主な手順を説明する。

31 行で、iostream ライブラリの ifstream オブジェクトの open メンバ関数を呼び出して、読み込みモードでファイルを開いている。以降、この ifstream オブジェクトに対して、読み込みの処理を行っていく。

35 行以降、getline メンバ関数を呼び出して、ファイルから 1 行ずつ読み込みながら、読み込んだ文字列を解析して幾何形状データとして格納する処理を行っていく。39 行で、tokenizer 関数ょ呼び出し、読み込んだ文字列から先頭の単語を取り出している。42 行以降の if 文では、取得した単語が、Obj 形式の各コマンドに対応するかを判定する。頂点データ（92 行以降）の場合は、1 文字目が v になるため、まずは、1 文字目が v であるかを判定して、その後、2 文字目の文字で、頂点座標、法線ベクトル、テクスチャ座標のどの情報を表すかを判定する。

頂点データやポリゴンデータを表す文字列の解析には、引き続き、tokenizer 関数を使用する。tokenizer 関数を呼び出すことで、2 つ目以降の単語を取得し、数値を表す文字列は、atof 関数や atoi 関数により実数や整数に変換して、メンバ変数に格納する。各可変長配列のメンバ変数には、vector クラスの push_back メンバ関数を呼び出すことで、可変長配列の末尾にデータを追加する。このとき、自動的に配列のサイズの拡張が行われる。

なお、マテリアルファイルの読み込みを行う LoadMaterialFile メンバ関数や、描画を行う Draw メンバ関数は、処理の記述は省略されているため、実際に本クラスをプログラムから利用する場合は、各自でこれらの処理を追加する必要がある。