obj.cpp

```
//
//  コンピュータグラフィックス特論Ⅱ
//  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
//


// 基本的なヘッダファイルのインクルード
#ifdef _WIN32
    #include <windows.h>
#endif
#include <stdio.h>
#include <GL/gl.h>

// 幾何形状データの定義のインクルード
#include "obj.h"


// バッファ長（サイズは適当）
#define  BUFFER_LENGTH      1024
#define  MAX_VECTOR_SIZE    4096
#define  MAX_TRIANGLE_SIZE  4096
#define  MAX_MTL_SIZE         32


//
//  Objファイルの読み込み
//
Obj *  LoadObj( const char * filename )
{
    FILE *  fp;
    char    line[ BUFFER_LENGTH ];
    char    name[ BUFFER_LENGTH ];
    int  i, j;
    Vector  vec;
    int  v_no[4], vt_no[4], vn_no[4];
    int  count;
    Mtl *  curr_mtl = NULL;

    // ファイルを開く
    fp = fopen( filename, "r" );
    if ( fp == NULL )
        return  NULL;

    // 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）
    Obj *  obj = new Obj();
    obj->num_vertices = 0;
    obj->num_normals = 0;
    obj->num_tex_coords = 0;
    obj->vertices = new Vector[ MAX_VECTOR_SIZE ];
    obj->normals = new Vector[ MAX_VECTOR_SIZE ];
    obj->tex_coords = new Vector[ MAX_VECTOR_SIZE ];
    obj->num_triangles = 0;
    obj->tri_v_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
    obj->tri_vn_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
    obj->tri_vt_no = new int[ MAX_TRIANGLE_SIZE * 3 ];
    obj->tri_material = new Mtl*[ MAX_TRIANGLE_SIZE * 3 ];
    obj->num_materials = 0;
    obj->materials = NULL;

    // ファイルから１行ずつ読み込み
    while ( fgets( line, BUFFER_LENGTH, fp ) != NULL )
    {
        // マテリアルの読み込み
        if ( strncmp( line, "mtllib", 6 ) == 0 )
        {
            // テキストを解析
            sscanf( line, "mtllib %s", name );

            // 指定されたファイル名のマテリアルデータを読み込み
            if ( strlen( name ) > 0 )
                LoadMtl( name, obj );
        }

        // マテリアルの変更
        if ( strncmp( line, "usemtl", 6 ) == 0 )
        {
            // テキストを解析
            sscanf( line, "usemtl %s", name );

            // 指定された名前のマテリアルデータを探索して記録
            for ( i=0; i<obj->num_materials; i++ )
            {
                if ( strcmp( name, obj->materials[ i ]->name ) == 0 )
                {
                    curr_mtl = obj->materials[ i ];
                    break;
                }
            }
        }

        // 頂点データの読み込み
        if ( line[0] == 'v' )
        {
            // 法線ベクトル（vn）
            if ( line[1] == 'n' )
            {
                // テキストを解析
                sscanf( line, "vn %f %f %f", &vec.x, &vec.y, &vec.z );

                // 法線ベクトル配列の末尾に格納
                if ( obj->num_normals < MAX_VECTOR_SIZE )
                {
                    obj->normals[ obj->num_normals ] = vec;
                    obj->num_normals ++;
                }
            }
            // テクスチャ座標（vt）
            else if ( line[1] == 't' )
            {
                // テキストを解析
                sscanf( line, "vt %f %f %f", &vec.x, &vec.y, &vec.z );

```

```
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 (面倒なのでとりあえず頂点座標配列のみ、後は各自追加)
170     Vector *  new_array;
171     new_array = new Vector[ obj->num_vertices ];
172     memcpy( new_array, obj->vertices, sizeof( Vector ) * obj->num_vertices );
173     delete[]  obj->vertices;
174     obj->vertices = new_array;
175 
176     // ファイルを閉じる
177     fclose( fp );
178 
179     // 読み込んだオブジェクトデータを返す
180     return  obj;
181 }
182 
183 
184 //
185 //  Mtlファイルの読み込み
186 //
187 void  LoadMtl( const char * filename, Obj * obj )
188 {
189     FILE *  fp;
190     char    line[ BUFFER_LENGTH ];
191     char    name[ BUFFER_LENGTH ];
192     Color   color;
193     Mtl *   curr_mtl = NULL;
194 
195     // ファイルを開く
196     fp = fopen( filename, "r" );
197     if ( fp == NULL )
198         return;
199 
200     // 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)
201     obj->num_materials = 0;
202     obj->materials = new Mtl*[ MAX_MTL_SIZE ];
203 
204     // ファイルから１行ずつ読み込み
205     while ( fgets( line, BUFFER_LENGTH, fp ) != NULL )
206     {
207         // マテリアルデータの追加
208         if ( strncmp( line, "newmtl", 6 ) == 0 )
209         {
210             // テキストを解析
211             sscanf( line, "newmtl %s", name );
212             if ( strlen( name ) == 0 )
213                 continue;
214 
215             // マテリアルデータの作成
216             curr_mtl = new Mtl();
217             curr_mtl->name = new char[ strlen( name ) + 1 ];
218             strcpy( curr_mtl->name, name );
219             curr_mtl->kd.r = 0.8f;
220             curr_mtl->kd.g = 0.8f;
221             curr_mtl->kd.b = 0.8f;
222 
223             // マテリアルデータを配列に記録
224             obj->materials[ obj->num_materials ] = curr_mtl;
```

```
225             obj->num_materials ++;
226         }
227 
228         // 反射特性データの読み込み
229         if ( line[0] == 'K' )
230         {
231             // 拡散反射特性 (Kd)
232             if ( line[1] == 'd' )
233             {
234                 // テキストを解析
235                 sscanf( line, "Kd %f %f %f", &color.r, &color.g, &color.b );
236 
237                 // 拡散反射特性 (Kd) を記録
238                 if ( curr_mtl )
239                     curr_mtl->kd = color;
240             }
241         }
242 
243     }
244 
245     // ファイルを閉じる
246     fclose( fp );
247 }
248 
249 
250 //
251 //  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)
252 //
253 float  ScaleObj( Obj * obj, float max_size )
254 {
255     if ( !obj || ( obj->num_vertices == 0 ) )
256         return  0.0;
257 
258     // サイズ計算
259     Vector  min, max;
260     min = max = obj->vertices[ 0 ];
261     int  i;
262     for ( i=0; i<obj->num_vertices; i++ )
263     {
264         const Vector &  p = obj->vertices[ i ];
265         if ( p.x < min.x )  min.x = p.x;
266         if ( p.y < min.y )  min.y = p.y;
267         if ( p.z < min.z )  min.z = p.z;
268         if ( p.x > max.x )  max.x = p.x;
269         if ( p.y > max.y )  max.y = p.y;
270         if ( p.z > max.z )  max.z = p.z;
271     }
272 
273     // スケーリング
274     float  size, scale;
275     size = ( ( max.x - min.x ) > ( max.z - min.z ) ) ? ( max.x - min.x ) : ( max.z - min.z );
276     scale = max_size / size;
277     for ( i=0; i<obj->num_vertices; i++ )
278     {
279         obj->vertices[ i ].x *= scale;
280         obj->vertices[ i ].y *= scale;
281         obj->vertices[ i ].z *= scale;
282     }
283 
284     // スケーリング後の中心の高さを返す
285     float  object_y = min.y * scale;
286     return  object_y;
287 }
288 
289 
290 //
291 //  Obj形状データの描画
292 //
293 void  RenderObj( Obj * obj )
294 {
295     int  i, j, no;
296     Mtl *  curr_mtl = NULL;
297 
298     glBegin( GL_TRIANGLES );
299     for ( i=0; i<obj->num_triangles; i++ )
300     {
301         // マテリアルの切り替え
302         if ( ( obj->num_materials > 0 ) && ( obj->tri_material[ i ] != curr_mtl ) )
303         {
304             curr_mtl = obj->tri_material[ i ];
305             glColor3f( curr_mtl->kd.r, curr_mtl->kd.g, curr_mtl->kd.b );
306         }
307 
308         // 三角面の各頂点データの指定
309         for ( j=0; j<3; j++ )
310         {
311             // 法線ベクトル
312             no = obj->tri_vn_no[ i*3 + j ];
313             const Vector &  vn = obj->normals[ no ];
314             glNormal3f( vn.x, vn.y, vn.z );
315 
316             // 頂点座標
317             no = obj->tri_v_no[ i*3 + j ];
318             const Vector &  v = obj->vertices[ no ];
319             glVertex3f( v.x, v.y, v.z );
320         }
321     }
322     glEnd();
323 }
324 
```