```
/**
 * @author mrdoob / http://mrdoob.com/
 */

THREE.OBJLoader = function ( manager ) {

	this.manager = ( manager !== undefined ) ? manager :
THREE.DefaultLoadingManager;

	this.materials = null;

	this.regexp = {
		// v float float float
		vertex_pattern           : /^v\s+([\d|\.|\+|\-
|e|E]+)\s+([\d|\.|\+|\-|e|E]+)\s+([\d|\.|\+|\-|e|E]+)/,
		// vn float float float
		normal_pattern           : /^vn\s+([\d|\.|\+|\-
|e|E]+)\s+([\d|\.|\+|\-|e|E]+)\s+([\d|\.|\+|\-|e|E]+)/,
		// vt float float
		uv_pattern               : /^vt\s+([\d|\.|\+|\-
|e|E]+)\s+([\d|\.|\+|\-|e|E]+)/,
		// f vertex vertex vertex
		face_vertex              : /^f\s+(-?\d+)\s+(-?\d+)\s+
(-?\d+)(?:\s+(-?\d+))?/,
		// f vertex/uv vertex/uv vertex/uv
		face_vertex_uv           : /^f\s+(-?\d+)\/(-?\d+)\s+(-
?\d+)\/(-?\d+)\s+(-?\d+)\/(-?\d+)(?:\s+(-?\d+)\/(-?\d+))?/,
		// f vertex/uv/normal vertex/uv/normal
vertex/uv/normal
		face_vertex_uv_normal    : /^f\s+(-?\d+)\/(-?\d+)\/(-?
\d+)\s+(-?\d+)\/(-?\d+)\/(-?\d+)\s+(-?\d+)\/(-?\d+)\/(-?\d+)(?:\s+(-?
\d+)\/(-?\d+)\/(-?\d+))?/,
		// f vertex//normal vertex//normal vertex//normal
		face_vertex_normal       : /^f\s+(-?\d+)\/\/(-?\d+)\s+
(-?\d+)\/\/(-?\d+)\s+(-?\d+)\/\/(-?\d+)(?:\s+(-?\d+)\/\/(-?\d+))?/,
		// o object_name | g group_name
		object_pattern           : /^[og]\s*(.+)?/,
		// s boolean
		smoothing_pattern        : /^s\s+(\d+|on|off)/,
		// mtllib file_reference
		material_library_pattern : /^mtllib /,
		// usemtl material_name
		material_use_pattern     : /^usemtl /
	};

};

THREE.OBJLoader.prototype = {

	constructor: THREE.OBJLoader,

	load: function ( url, onLoad, onProgress, onError ) {

		var scope = this;

		var loader = new THREE.XHRLoader( scope.manager );
```

```
                loader.setPath( this.path );
                loader.load( url, function ( text ) {

                        onLoad( scope.parse( text ) );

                }, onProgress, onError );

        },

        setPath: function ( value ) {

                this.path = value;

        },

        setMaterials: function ( materials ) {

                this.materials = materials;

        },

        _createParserState : function () {

                var state = {
                        objects  : [],
                        object   : {},

                        vertices : [],
                        normals  : [],
                        uvs      : [],

                        materialLibraries : [],

                        startObject: function ( name, fromDeclaration
) {

                                // If the current object (initial from
reset) is not from a g/o declaration in the parsed
                                // file. We need to use it for the
first parsed g/o to keep things in sync.
                                if ( this.object &&
this.object.fromDeclaration === false ) {

                                        this.object.name = name;
                                        this.object.fromDeclaration =
( fromDeclaration !== false );
                                        return;

                                }

                                if ( this.object && typeof
this.object._finalize === 'function' ) {

                                        this.object._finalize();

                                }
```

```
                                var previousMaterial = ( this.object
&& typeof this.object.currentMaterial === 'function' ?
this.object.currentMaterial() : undefined );

                                this.object = {
                                        name : name || '',
                                        fromDeclaration : (
fromDeclaration !== false ),

                                        geometry : {
                                                vertices : [],
                                                normals  : [],
                                                uvs      : []
                                        },
                                        materials : [],
                                        smooth : true,

                                        startMaterial : function(
name, libraries ) {

                                                var previous =
this._finalize( false );

                                                // New usemtl
declaration overwrites an inherited material, except if faces were
declared
                                                // after the material,
then it must be preserved for proper MultiMaterial continuation.
                                                if ( previous && (
previous.inherited || previous.groupCount <= 0 ) ) {


this.materials.splice( previous.index, 1 );

                                                }

                                                var material = {
                                                        index      :
this.materials.length,
                                                        name       :
name || '',
                                                        mtllib     : (
Array.isArray( libraries ) && libraries.length > 0 ? libraries[
libraries.length - 1 ] : '' ),
                                                        smooth     : (
previous !== undefined ? previous.smooth : this.smooth ),
                                                        groupStart : (
previous !== undefined ? previous.groupEnd : 0 ),
                                                        groupEnd   :
-1,
                                                        groupCount :
-1,
                                                        inherited  :
false,

                                                        clone :
function( index ) {
```

```
                                                                    return
{

index      : ( typeof index === 'number' ? index : this.index ),

name       : this.name,

mtllib     : this.mtllib,

smooth     : this.smooth,

groupStart : this.groupEnd,

groupEnd   : -1,

groupCount : -1,

inherited  : false
                                                                        };
                                                                }
                                                        };

                                                        this.materials.push(
material );

                                                        return material;

                                                },

                                                currentMaterial : function() {

                                                        if (
this.materials.length > 0 ) {
                                                                return
this.materials[ this.materials.length - 1 ];
                                                        }

                                                        return undefined;

                                                },

                                                _finalize : function( end ) {

                                                        var lastMultiMaterial
= this.currentMaterial();
                                                        if ( lastMultiMaterial
&& lastMultiMaterial.groupEnd === -1 ) {


lastMultiMaterial.groupEnd = this.geometry.vertices.length / 3;

lastMultiMaterial.groupCount = lastMultiMaterial.groupEnd -
lastMultiMaterial.groupStart;

lastMultiMaterial.inherited = false;

                                                        }
```

```
                                                // Guarantee at least
one empty material, this makes the creation later more straight
forward.
                                                if ( end !== false &&
this.materials.length === 0 ) {

this.materials.push({
                                                                name
: '',
                                                                smooth
: this.smooth
                                                        });
                                                }

                                                return
lastMultiMaterial;

                                        }
                                };

                                // Inherit previous objects material.
                                // Spec tells us that a declared
material must be set to all objects until a new material is declared.
                                // If a usemtl declaration is
encountered while this new object is being parsed, it will
                                // overwrite the inherited material.
Exception being that there was already face declarations
                                // to the inherited material, then it
will be preserved for proper MultiMaterial continuation.

                                if ( previousMaterial &&
previousMaterial.name && typeof previousMaterial.clone === "function"
) {

                                        var declared =
previousMaterial.clone( 0 );
                                        declared.inherited = true;
                                        this.object.materials.push(
declared );

                                }

                                this.objects.push( this.object );

                        },

                        finalize : function() {

                                if ( this.object && typeof
this.object._finalize === 'function' ) {

                                        this.object._finalize();

                                }

                        },
```

```
                        parseVertexIndex: function ( value, len ) {

                                var index = parseInt( value, 10 );
                                return ( index >= 0 ? index - 1 :
index + len / 3 ) * 3;

                        },

                        parseNormalIndex: function ( value, len ) {

                                var index = parseInt( value, 10 );
                                return ( index >= 0 ? index - 1 :
index + len / 3 ) * 3;

                        },

                        parseUVIndex: function ( value, len ) {

                                var index = parseInt( value, 10 );
                                return ( index >= 0 ? index - 1 :
index + len / 2 ) * 2;

                        },

                        addVertex: function ( a, b, c ) {

                                var src = this.vertices;
                                var dst =
this.object.geometry.vertices;

                                dst.push( src[ a + 0 ] );
                                dst.push( src[ a + 1 ] );
                                dst.push( src[ a + 2 ] );
                                dst.push( src[ b + 0 ] );
                                dst.push( src[ b + 1 ] );
                                dst.push( src[ b + 2 ] );
                                dst.push( src[ c + 0 ] );
                                dst.push( src[ c + 1 ] );
                                dst.push( src[ c + 2 ] );

                        },

                        addVertexLine: function ( a ) {

                                var src = this.vertices;
                                var dst =
this.object.geometry.vertices;

                                dst.push( src[ a + 0 ] );
                                dst.push( src[ a + 1 ] );
                                dst.push( src[ a + 2 ] );

                        },

                        addNormal : function ( a, b, c ) {
```

```
                                var src = this.normals;
                                var dst =
this.object.geometry.normals;

                                dst.push( src[ a + 0 ] );
                                dst.push( src[ a + 1 ] );
                                dst.push( src[ a + 2 ] );
                                dst.push( src[ b + 0 ] );
                                dst.push( src[ b + 1 ] );
                                dst.push( src[ b + 2 ] );
                                dst.push( src[ c + 0 ] );
                                dst.push( src[ c + 1 ] );
                                dst.push( src[ c + 2 ] );

                        },

                        addUV: function ( a, b, c ) {

                                var src = this.uvs;
                                var dst = this.object.geometry.uvs;

                                dst.push( src[ a + 0 ] );
                                dst.push( src[ a + 1 ] );
                                dst.push( src[ b + 0 ] );
                                dst.push( src[ b + 1 ] );
                                dst.push( src[ c + 0 ] );
                                dst.push( src[ c + 1 ] );

                        },

                        addUVLine: function ( a ) {

                                var src = this.uvs;
                                var dst = this.object.geometry.uvs;

                                dst.push( src[ a + 0 ] );
                                dst.push( src[ a + 1 ] );

                        },

                        addFace: function ( a, b, c, d, ua, ub, uc,
ud, na, nb, nc, nd ) {

                                var vLen = this.vertices.length;

                                var ia = this.parseVertexIndex( a,
vLen );
                                var ib = this.parseVertexIndex( b,
vLen );
                                var ic = this.parseVertexIndex( c,
vLen );
                                var id;

                                if ( d === undefined ) {

                                        this.addVertex( ia, ib, ic );
```

```
                                } else {

                                        id = this.parseVertexIndex( d,
vLen );

                                        this.addVertex( ia, ib, id );
                                        this.addVertex( ib, ic, id );

                                }

                                if ( ua !== undefined ) {

                                        var uvLen = this.uvs.length;

                                        ia = this.parseUVIndex( ua,
uvLen );
                                        ib = this.parseUVIndex( ub,
uvLen );
                                        ic = this.parseUVIndex( uc,
uvLen );

                                        if ( d === undefined ) {

                                                this.addUV( ia, ib, ic
);

                                        } else {

                                                id =
this.parseUVIndex( ud, uvLen );

                                                this.addUV( ia, ib, id
);
                                                this.addUV( ib, ic, id
);

                                        }

                                }

                                if ( na !== undefined ) {

                                        // Normals are many times the
same. If so, skip function call and parseInt.
                                        var nLen =
this.normals.length;
                                        ia = this.parseNormalIndex(
na, nLen );

                                        ib = na === nb ? ia :
this.parseNormalIndex( nb, nLen );
                                        ic = na === nc ? ia :
this.parseNormalIndex( nc, nLen );

                                        if ( d === undefined ) {

                                                this.addNormal( ia,
```

```
ib, ic );

                                        } else {

                                                id =
this.parseNormalIndex( nd, nLen );

                                                this.addNormal( ia,
ib, id );
                                                this.addNormal( ib,
ic, id );

                                        }

                                }

                        },

                        addLineGeometry: function ( vertices, uvs ) {

                                this.object.geometry.type = 'Line';

                                var vLen = this.vertices.length;
                                var uvLen = this.uvs.length;

                                for ( var vi = 0, l = vertices.length;
vi < l; vi ++ ) {

                                        this.addVertexLine(
this.parseVertexIndex( vertices[ vi ], vLen ) );

                                }

                                for ( var uvi = 0, l = uvs.length; uvi
< l; uvi ++ ) {

                                        this.addUVLine(
this.parseUVIndex( uvs[ uvi ], uvLen ) );

                                }

                        }

                };

                state.startObject( '', false );

                return state;

        },

        parse: function ( text ) {

                console.time( 'OBJLoader' );

                var state = this._createParserState();
```

```
		if ( text.indexOf( '\r\n' ) !== - 1 ) {

			// This is faster than String.split with regex that splits on both
			text = text.replace( '\r\n', '\n' );

		}

		var lines = text.split( '\n' );
		var line = '', lineFirstChar = '', lineSecondChar = '';
		var lineLength = 0;
		var result = [];

		// Faster to just trim left side of the line. Use if available.
		var trimLeft = ( typeof ''.trimLeft === 'function' );

		for ( var i = 0, l = lines.length; i < l; i ++ ) {

			line = lines[ i ];

			line = trimLeft ? line.trimLeft() : line.trim();

			lineLength = line.length;

			if ( lineLength === 0 ) continue;

			lineFirstChar = line.charAt( 0 );

			// @todo invoke passed in handler if any
			if ( lineFirstChar === '#' ) continue;

			if ( lineFirstChar === 'v' ) {

				lineSecondChar = line.charAt( 1 );

				if ( lineSecondChar === ' ' && ( result = this.regexp.vertex_pattern.exec( line ) ) !== null ) {

					// 0                  1      2      3
					// ["v 1.0 2.0 3.0", "1.0", "2.0", "3.0"]

					state.vertices.push(
						parseFloat( result[ 1 ] ),
						parseFloat( result[ 2 ] ),
						parseFloat( result[ 3 ] )
					);

				} else if ( lineSecondChar === 'n' && ( result = this.regexp.normal_pattern.exec( line ) ) !== null ) {
```

```
                                        // 0                    1
2      3
                                        // ["vn 1.0 2.0 3.0", "1.0",
"2.0", "3.0"]

                                        state.normals.push(
                                                parseFloat( result[ 1
] ),
                                                parseFloat( result[ 2
] ),
                                                parseFloat( result[ 3
] )
                                        );

                                } else if ( lineSecondChar === 't' &&
( result = this.regexp.uv_pattern.exec( line ) ) !== null ) {

                                        // 0                1      2
                                        // ["vt 0.1 0.2", "0.1",
"0.2"]

                                        state.uvs.push(
                                                parseFloat( result[ 1
] ),
                                                parseFloat( result[ 2
] )
                                        );

                                } else {

                                        throw new Error( "Unexpected
vertex/normal/uv line: '" + line  + "'" );

                                }

                        } else if ( lineFirstChar === "f" ) {

                                if ( ( result =
this.regexp.face_vertex_uv_normal.exec( line ) ) !== null ) {

                                        // f vertex/uv/normal
vertex/uv/normal vertex/uv/normal
                                        // 0                           1
2    3    4    5    6    7    8    9   10          11          12
                                        // ["f 1/1/1 2/2/2 3/3/3",
"1", "1", "1", "2", "2", "2", "3", "3", "3", undefined, undefined,
undefined]

                                        state.addFace(
                                                result[ 1 ], result[ 4
], result[ 7 ], result[ 10 ],
                                                result[ 2 ], result[ 5
], result[ 8 ], result[ 11 ],
                                                result[ 3 ], result[ 6
], result[ 9 ], result[ 12 ]
                                        );
```

```
                                } else if ( ( result =
this.regexp.face_vertex_uv.exec( line ) ) !== null ) {

                                        // f vertex/uv vertex/uv
vertex/uv
                                        // 0                  1    2
3    4    5    6   7          8
                                        // ["f 1/1 2/2 3/3", "1", "1",
"2", "2", "3", "3", undefined, undefined]

                                        state.addFace(
                                                result[ 1 ], result[ 3
], result[ 5 ], result[ 7 ],
                                                result[ 2 ], result[ 4
], result[ 6 ], result[ 8 ]
                                        );

                                } else if ( ( result =
this.regexp.face_vertex_normal.exec( line ) ) !== null ) {

                                        // f vertex//normal
vertex//normal vertex//normal
                                        // 0                     1
2    3    4    5    6   7          8
                                        // ["f 1//1 2//2 3//3", "1",
"1", "2", "2", "3", "3", undefined, undefined]

                                        state.addFace(
                                                result[ 1 ], result[ 3
], result[ 5 ], result[ 7 ],
                                                undefined, undefined,
undefined, undefined,
                                                result[ 2 ], result[ 4
], result[ 6 ], result[ 8 ]
                                        );

                                } else if ( ( result =
this.regexp.face_vertex.exec( line ) ) !== null ) {

                                        // f vertex vertex vertex
                                        // 0            1    2    3
4
                                        // ["f 1 2 3", "1", "2", "3",
undefined]

                                        state.addFace(
                                                result[ 1 ], result[ 2
], result[ 3 ], result[ 4 ]
                                        );

                                } else {

                                        throw new Error( "Unexpected
face line: '" + line  + "'" );

                                }
```

```
                        } else if ( lineFirstChar === "l" ) {

                                var lineParts = line.substring( 1
).trim().split( " " );
                                var lineVertices = [], lineUVs = [];

                                if ( line.indexOf( "/" ) === - 1 ) {

                                        lineVertices = lineParts;

                                } else {

                                        for ( var li = 0, llen =
lineParts.length; li < llen; li ++ ) {

                                                var parts = lineParts[
li ].split( "/" );

                                                if ( parts[ 0 ] !== ""
) lineVertices.push( parts[ 0 ] );
                                                if ( parts[ 1 ] !== ""
) lineUVs.push( parts[ 1 ] );

                                        }

                                }
                                state.addLineGeometry( lineVertices,
lineUVs );

                        } else if ( ( result =
this.regexp.object_pattern.exec( line ) ) !== null ) {

                                // o object_name
                                // or
                                // g group_name

                                var name = result[ 0 ].substr( 1
).trim();
                                state.startObject( name );

                        } else if (
this.regexp.material_use_pattern.test( line ) ) {

                                // material

                                state.object.startMaterial(
line.substring( 7 ).trim(), state.materialLibraries );

                        } else if (
this.regexp.material_library_pattern.test( line ) ) {

                                // mtl file

                                state.materialLibraries.push(
line.substring( 7 ).trim() );
```

```
                        } else if ( ( result =
this.regexp.smoothing_pattern.exec( line ) ) !== null ) {

                                // smooth shading

                                // @todo Handle files that have
varying smooth values for a set of faces inside one geometry,
                                // but does not define a usemtl for
each face set.
                                // This should be detected and a dummy
material created (later MultiMaterial and geometry groups).
                                // This requires some care to not
create extra material on each smooth value for "normal" obj files.
                                // where explicit usemtl defines
geometry groups.
                                // Example asset:
examples/models/obj/cerberus/Cerberus.obj

                                var value = result[ 1
].trim().toLowerCase();
                                state.object.smooth = ( value === '1'
|| value === 'on' );

                                var material =
state.object.currentMaterial();
                                if ( material ) {

                                        material.smooth =
state.object.smooth;

                                }

                        } else {

                                // Handle null terminated files
without exception
                                if ( line === '\0' ) continue;

                                throw new Error( "Unexpected line: '"
+ line  + "'" );

                        }

                }

                state.finalize();

                var container = new THREE.Group();
                container.materialLibraries = [].concat(
state.materialLibraries );

                for ( var i = 0, l = state.objects.length; i < l; i ++
) {

                        var object = state.objects[ i ];
                        var geometry = object.geometry;
                        var materials = object.materials;
```

```
                        var isLine = ( geometry.type === 'Line' );

                        // Skip o/g line declarations that did not
follow with any faces
                        if ( geometry.vertices.length === 0 )
continue;

                        var buffergeometry = new
THREE.BufferGeometry();

                        buffergeometry.addAttribute( 'position', new
THREE.BufferAttribute( new Float32Array( geometry.vertices ), 3 ) );

                        if ( geometry.normals.length > 0 ) {

                                buffergeometry.addAttribute( 'normal',
new THREE.BufferAttribute( new Float32Array( geometry.normals ), 3 )
);

                        } else {

                                buffergeometry.computeVertexNormals();

                        }

                        if ( geometry.uvs.length > 0 ) {

                                buffergeometry.addAttribute( 'uv', new
THREE.BufferAttribute( new Float32Array( geometry.uvs ), 2 ) );

                        }

                        // Create materials

                        var createdMaterials = [];

                        for ( var mi = 0, miLen = materials.length; mi
< miLen ; mi++ ) {

                                var sourceMaterial = materials[mi];
                                var material = undefined;

                                if ( this.materials !== null ) {

                                        material =
this.materials.create( sourceMaterial.name );

                                        // mtl etc. loaders probably
can't create line materials correctly, copy properties to a line
material.
                                        if ( isLine && material && ! (
material instanceof THREE.LineBasicMaterial ) ) {

                                                var materialLine = new
THREE.LineBasicMaterial();
                                                materialLine.copy(
material );
```

```
                                                material =
materialLine;

                                        }

                                }

                                if ( ! material ) {

                                        material = ( ! isLine ? new
THREE.MeshPhongMaterial() : new THREE.LineBasicMaterial() );
                                        material.name =
sourceMaterial.name;

                                }

                                material.shading =
sourceMaterial.smooth ? THREE.SmoothShading : THREE.FlatShading;

                                createdMaterials.push(material);

                        }

                        // Create mesh

                        var mesh;

                        if ( createdMaterials.length > 1 ) {

                                for ( var mi = 0, miLen =
materials.length; mi < miLen ; mi++ ) {

                                        var sourceMaterial =
materials[mi];
                                        buffergeometry.addGroup(
sourceMaterial.groupStart, sourceMaterial.groupCount, mi );

                                }

                                var multiMaterial = new
THREE.MultiMaterial( createdMaterials );
                                mesh = ( ! isLine ? new THREE.Mesh(
buffergeometry, multiMaterial ) : new THREE.Line( buffergeometry,
multiMaterial ) );

                        } else {

                                mesh = ( ! isLine ? new THREE.Mesh(
buffergeometry, createdMaterials[ 0 ] ) : new THREE.Line(
buffergeometry, createdMaterials[ 0 ] ) );
                        }

                        mesh.name = object.name;

                        container.add( mesh );

                }
```

```
                console.timeEnd( 'OBJLoader' );

                return container;

        }

};
```