

NVMeshMender User Guide

The [NVMeshMender library](#) is designed to prepare meshes for per-pixel lighting, by generating normals, tangents and binormals.

Some of the issues that the library can address are:

- 1) Generating Tangents and Binormals for Per-Pixel Lighting
- 2) Duplicating vertices to avoid needing Cylindrical Texture Wrapping.
- 3) Intelligently smoothing across texture discontinuities
- 4) Generating normals, or using existing normals
- 5) Respecting existing split vertices, or collapsing similar vertices

Input

NVMeshMender requires 2-manifold triangular geometry, where each edge has one or two triangle neighbors. Meshes do not need to be closed, although triangles around a vertex won't be smoothed across holes.

NVMeshMender gathers all triangles around a vertex, and tries to walk clockwise, and then counter-clockwise around the vertex until all triangles are visited.

If a single one of these triangles is missing, all remaining triangles can be smoothed, depending on the angle of each with its neighbor. If two or more disjoint triangles are present, there will be at least two groups of triangles to be smoothed.

If geometry is not 2-manifold, then the smoothing operations will not function properly.

Output

NVMeshMender never creates additional triangles, but rather may create new vertices, and/or change the index list, in order to split edges that are geometrically shared, but must be split for the sake of a discontinuous tangent space, or to fix cylindrical wrapping requirements.

Smoothing Groups

NVMeshMender does not understand pre-defined smoothing groups, although it is flexible in how it groups neighboring triangles, using user-defined crease angles, and weighting based on a user-specified lerp factor between area-weighted normals and non-weighted normals.

If you have a certain part of geometry that you want in its own smoothing group, pass that in it a separate call to mend().

Interface

```
bool Mend( std::vector< Vertex >& theVerts,
           std::vector< unsigned int >& theIndices,
           std::vector< unsigned int >& mappingNewToOldVert,
           const float minNormalsCreaseCosAngle = 0.0f,
           const float minTangentsCreaseCosAngle = 0.0f ,
           const float minBinormalsCreaseCosAngle = 0.0f,
           const float weightNormalsByArea = 1.0f,
           const NormalCalcOption computeNormals = CALCULATE_NORMALS,
           const ExistingSplitOptions respectExistingSplits =
               DONT_RESPECT_SPLITS,
           const CylindricalFixOption fixCylindricalWrapping =
               DONT_FIX_CYLINDRICAL );
```

RETURNS true on success, false on failure

Each parameter is explained in NVMeshMender.h, as well as below.

theVerts - Should be initialized with your mesh data, NOTE that when mesh mender is done with it, the number of vertices may grow and it will be filled with normals, tangents and binormals

theIndices - Should be initialized with your mesh indices will contain the new indices..we are not adding triangles, so the number of indices passed back should be the same as the number of indices passed in, but they may point to new vertices now.

mappingNewToOldVert - This should be passed in as an empty vector. after mending it will contain a mapping of newvertexindex -> oldvertexindex so it could be used to map any per vertex data you had in your original mesh to the new mesh like so:

```
for each new vertex index
newVert[index]->myData =
    oldVert[ mappingNewToOldVert[index]]->myData;
```

where myData is some custom vertex data in your original mesh.

minNormalsCreaseCosAngle - The minimum cosine of the angle between normals so that they are allowed to be smoothed together. Ranges between -1.0 and +1.0. This is ignored if computeNormals is set to DONT_CALCULATE_NORMALS

minTangentsCreaseCosAngle - The minimum cosine of the angle between

tangents so that they are allowed to be smoothed together
Ranges between -1.0 and +1.0.

minBinormalsCreaseCosAngle - The minimum cosine of the angle between
binormals so that they are allowed to be smoothed together
Ranges between -1.0 and +1.0.

weightNormalsByArea - An amount to blend the normalized face normal,
and the un-normalized face normal together. Thus weighting the
normal by the face area by a given amount. Ranges between 0.0
and +1.0.

0.0 means use the normalized face normals (not weighted by area).
1.0 means use the unnormalized face normal (weighted by area).
0.5 means lerp between the two resulting normals & re-normalize.
This is ignored if computeNormals is set to
DONT_CALCULATE_NORMALS.

computeNormals - Should mesh mender calculate normals?
If this is set to DONT_CALCULATE_NORMALS. Then the vertex
normals after meshmender is called will be the same ones you pass
in. If you are automatically calculating normals yourself, you
may find that meshmender provides greater control over how
normals are smoothed together. We've been able to get better
results using the Crease angle with meshmender's smoothing
groups.

respectExistingSplits - DONT_RESPECT_SPLITS means that neighboring
triangles for smoothing will be determined based on position and
not on indices.

RESPECT_SPLITS means that neighboring triangles will be
determined based on the indices of the triangle and not the
positions of the vertices. You can usually get better smoothing
by not respecting existing splits.
Only respect them if you know they should be respected.

fixCylindricalWrapping - DONT_FIX_CYLINDRICAL means take the texture
coordinates as they come in.
FIX_CYLINDRICAL means we might need to split the verts at that
point and generate the proper texture coordinates. For instance,
if we have texcoords 0.9 -> 0.0 -> 0.2 we would need to add
a new vert so that we have 0.9 -> 1.0 < split > 0.0-> 0.2.
This is only supported for texture coordinates in the range
[0.0f , 1.0f]

NOTE: don't leave this on for all meshes, only use it when you
know you need it. If you have polygons that map to a large area
in texture space, this option could distort the texture
coordinates

Here is an example piece of code that uses the nvmeshmender to generate tangents &
collapse some non-shared triangles into indexed lists.

```
std::vector< uint32 > indices;
```

```

std::vector< uint32 > remap;

std::vector< MeshMender::Vertex > verts;

for ( size_t t = 0; t < mTriVector.size(); ++t )
{
    indices.push_back( mTriVector[ t ].a );
    indices.push_back( mTriVector[ t ].b );
    indices.push_back( mTriVector[ t ].c );
}

MeshMender::Vertex inv;

for ( size_t t = 0; t < mVertexVector.size(); ++t )
{
    const WorldVertex& wv = mVertexVector[ t ];

    inv.pos = wv.pos;
    inv.s = wv.s;
    inv.t = wv.t;
    verts.push_back( inv );
}

const float32 minNormalCreaseCos = 0.2f;
const float32 minTangentCreaseCos = 0.2f;
const float32 minBinormalCreaseCos = 0.2f;
const float32 weightNormalsByArea = 0.5f;

MeshMender mender;

mender.Mend( verts,
            indices,
            remap,
            minNormalCreaseCos,
            minTangentCreaseCos,
            minBinormalCreaseCos,
            weightNormalsByArea,
            MeshMender::CALCULATE_NORMALS,
            MeshMender::DONT_RESPECT_SPLITS,
            MeshMender::FIX_CYLINDRICAL );

for ( size_t t = 0; t < mTriVector.size(); ++t )
{
    Tri32& aTri = mTriVector[ t ];
    aTri.a = remap[ indices[ t * 3 + 0 ] ];
    aTri.b = remap[ indices[ t * 3 + 1 ] ];
    aTri.c = remap[ indices[ t * 3 + 2 ] ];
}

mVertexVector.resize(0);

WorldVertex wv;

for ( size_t t = 0; t < verts.size(); ++t )
{
    const MeshMender::Vertex& ov = verts[ t ];

```

```
wv.position = ov.pos;
wv.normal   = -ov.normal;
wv.s        = ov.s;
wv.t        = ov.t;
wv.tangent  = ov.tangent;
wv.binormal = ov.binormal;

if ( wv.tangent == ZeroVector )
{
    wv.tangent = XAxis;
}
if ( wv.binormal == ZeroVector )
{
    wv.binormal = ZAxis;
}

mVertexVector.push_back( wv );
}
```