网格及材质合并

作者：xu365082218
原文链接：https://ld246.com/article/1514990553992
来源网站：链滴

最近弄 unity 地图导出 pmd,发现好多网格合并代码用不了,我无法理解那些合并代码里的网格合到底是什么意思.我说说我理解的网格合并
一般一个地图场景,有很多个 gameobject,每个有网格的 gameobject 上都有一个 meshfilter 组件和个 meshrenderer 组件
而对 meshrenderer 组件来说,他可能具有一系列材质来对应渲染 meshfilter 里的每个 submesh,而个材质都有自己的 shader 参数等.但是使用同一材质的,这些 shader 参数都一样
而对于每个 meshfilter 组件来说,他拥有顶点缓冲区,索引缓冲区,uv,color,normal,以及每个 submesh 对应了哪些顶点,以及这个 meshfilter 所在的 gameobject 的坐标(世界变换矩阵)
所以要合并,应该是先要统计这些信息,然后把所有信息存储到一个在原点,无旋转,无缩放的 gameobject 上,把所有网格信息,为其添加 meshfilter 组件放到 mesh 里,把所有材质放到 meshrenderer 里的 materials 里
而这个主要麻烦,就是网格和材质的对应关系,以及网格顶点信息的重新计算(要把原来 meshfilter 里的点算出世界坐标然后放到这里).这整个流程只要心里有数了,就可以写代码来实现了.
最后放一个组件代码,用于合并一系列的 gameobject 到一个 gameobject 里，主要是合并网格息，材质等
这个组件没有合并同材质的 submesh,事实上，使用了同一材质的不同网格，可以合并到一个 submesh 里，这个点可以继续优化，但是现在是没处理的，我只是想把地图场景保存为 pmd 的，所以这部忽略了
如果有不对的请指教
在一个 Editor 类里添加一个类
[CustomEditor(typeof(PMDSave))]
public class PMDSaveInspector : Editor
{
public override void OnInspectorGUI()
{
base.OnInspectorGUI();
if (GUILayout.Button("Combine"))
{
var obj = (PMDSave)target;
obj.CombineMapMesh();
}
}
}
然后在 Asset 目录下新增此组件
然后在地图的根节点处，挂入此脚本组件，点组件 Inspector 栏下的 CombineMesh 按钮，这个单格就可以替代全部地图下的所有 gameobejct 了。
public class PMDSave : MonoBehaviour {
void CombineMapMesh()
{
Transform GenMapRoot = new GameObject("GenMapRoot").transform;
GenMapRoot.position = Vector3.zero;
GenMapRoot.localScale = Vector3.one;
GenMapRoot.rotation = Quaternion.identity;
MeshFilter mf = GenMapRoot.gameObject.AddComponent();
MeshRenderer mr = GenMapRoot.gameObject.AddComponent();
MeshFilter[] mfs = GetComponentsInChildren();
MeshRenderer[] mrs = GetComponentsInChildren();
//统计材质
List mat = new List();
for (int i = 0; i < mrs.Length; i++)
{
if (!mrs[i].enabled)
continue;
mat.AddRange(mrs[i].sharedMaterials);

}
mr.sharedMaterials = mat.ToArray();
int subMeshIndex = 0;
List vertex = new List();
List uv = new List();
List normal = new List();
List co = new List();
Dictionary<int, int[]> subMeshIndices = new Dictionary<int, int[]>();
//统计顶点.
for (int i = 0; i < mfs.Length; i++)
{
//隐藏了的不要
if (!mrs[i].enabled)
continue;
//推入顶点
int vertexIndex = vertex.Count;//记录推入之前的索引起始
for (int j = 0; j < mfs[i].sharedMesh.vertexCount; j++)
vertex.Add(mfs[i].transform.localToWorldMatrix * mfs[i].sharedMesh.vertices[j]);</p>
<pre><code class="highlight-chroma"><span class="highlight
cl">   //推入颜色,UV,法线
      int uv_len = m
s[i].sharedMesh.uv.Length;
      for (int j = 0; j
< uv_len; j++)
          uv.Add(mfs[
].sharedMesh.uv[j]);
      int normal_len
= mfs[i].sharedMesh.normals.Length;
      for (int j = 0; j
< normal_len; j++)
          normal.Add
mfs[i].sharedMesh.normals[j]);
      int color_len =
mfs[i].sharedMesh.colors.Length;
      for (int j = 0; j
< color_len; j++)
          co.Add(mfs[
].sharedMesh.colors[j]);
      for (int j = 0; j
< mfs[i].sharedMesh.subMeshCount; j++)
      {
          int[] indices
= mfs[i].sharedMesh.GetIndices(j);
          //把这些索
，都加上全局索引下标
          for (int k = 0
 k < indices.Length; k++)
              indices[k]
+= vertexIndex;
          subMeshInd
ces.Add(subMeshIndex, indices);
          subMeshIn
ex++;
      }

```
</span></span><span class="highlight-line"><span class="highlight-cl">  }
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh
 new Mesh();
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh.
etVertices(vertex);
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh.
etColors(co);
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh.
v = uv.ToArray();
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh.
etNormals(normal);
</span></span><span class="highlight-line"><span class="highlight-cl">  mf.sharedMesh.s
bMeshCount = subMeshIndices.Count;
</span></span><span class="highlight-line"><span class="highlight-cl">  vertexCnt = vert
x.Count;
</span></span><span class="highlight-line"><span class="highlight-cl">  foreach (var each
in subMeshIndices)
</span></span><span class="highlight-line"><span class="highlight-cl">      mf.sharedMes
.SetIndices(each.Value, MeshTopology.Triangles, each.Key);
</span></span></code></pre>
<p>}<br>
}</p>
<p>之前做这个 PMDSave 主要是来保存和官方 PMD 骨骼结构调整一致的流星.net 人物角色的，因想让.net 流星角色也跳极乐净土，后来朋友想把场景也弄进去，就让我导出下场景<br>
其实导出角色的时候，要导出骨骼结构，以及部分的腿部 IK 设置,以及蒙皮数据，所以导出角色，是导出地图更麻烦的。<br>
而一个 pmd 貌似只能保存一个 meshfilter 里的顶点和材质，所以后来就找代码合并网格和材质到一物件上，找了半天都是些不能用的，我就想不通到底哪里理解不一样<br>
于是自己折腾了一个，过几天朋友就会发出一个流星蝴蝶剑.net 角色跳舞蹈的视频。其实转换骨骼结为官方 mmd 的标准骨架后，舞蹈动作都可以通用了，想想一群.net 的角色一起跳舞<br>
就觉得好滑稽。</p>
<p>最后合并网格是用来减少 drawcall 的,相同材质的网格合并能减少 drawcall，所以在这个组件基上，只要把材质相同的 submesh 合并到一个 submesh 里，drawcall 也就会被优化掉<br>
不过合并后，成为一个整体，是无法控制地图上的任何元素的，因为都在一起了，要显示就一起显示要隐藏就一起隐藏</p>
<iframe src="https://www.bilibili.com/video/av18124919/" width="800" height="600" sandbox="allow-scripts allow-same-origin">
</iframe>
```