<https://blog.csdn.net/shenshen211/article/details/51740988>

obj文件时一种文本文件格式，比起二进制文件为主、连每个块的用途都得猜测的3ds，文本文件为主的obj对我们更友好。与3ds文件的树状[块结构]不同，obj文件只是很单纯的字典状结构，没有块ID来表征名字而是简单地用易懂的表意字符来表示。

**obj文件特点**

1. obj是一种3D模型文件，因此不包含动画、材质特性、贴图路径、动力学、粒子信息
2. obj文件主要支持多边形模型，虽然obj文件也支持曲线、表面、点组材质，但Maya导出的obj文件并不包括这些信息
3. obj文件支持三个点以上的面，这一点很有用，很多其他的模型文件格式只支持三个点的面，所有我们导入Maya的模型经常被三角化了，这对于模型的再加工不利
4. obj文件支持法线和贴图坐标

**obj文件基本结构**

obj文件不需要任何头文件，尽管经常使用几行文件信息的注释作为文件的开头

obj文件由一行行文本组成，注释行以一个“#”号为开头，空格和空行可以随意加到文件中以增加文件的可读性。有字的行都由一两个标记字母也就是关键字开头，关键字可以说明这一行是什么样的数据。多行可以逻辑地连接在一起表示一行，方法是在每一行最后添加一个连接符(\)，注意连接符(\)后面不能出现空格或者tab键，否则将导致文件出错，在这个列表中，关键字根据数据类型排列，每个关键字有一段简短描述：

1. 顶点数据

v 几何体顶点

vt 贴图坐标点

vn 顶点法线

vp 参数空格顶点

1. 自由形态曲线/表面属性

deg 度

bmat 基础矩阵

step 步尺寸

cstype 曲线或表面类型

1. 元素

p 点

l 线

f 面

curv 曲线

curv2 2d曲线

surf 表面

4）自由形态曲线(Free-form curve)/表面主体陈述(surface body statements):

parm 参数值 (Parameter values )

trim 外部修剪循环 (Outer trimming loop)

hole 内部整修循环 (Inner trimming loop)

scrv 特殊曲线 (Special curve)

sp 特殊的点 (Special point)

end 结束陈述 (End statement)

5）自由形态表面之间的连接(Connectivity between free-form surfaces):

con 连接 (Connect)

成组(Grouping):

g 组名称 (Group name)

s 光滑组 (Smoothing group)

mg 合并组 (Merging group)

o 对象名称 (Object name)

6）显示(Display)/渲染属性(render attributes):

bevel 导角插值 (Bevel interpolation)   
　　c\_interp 颜色插值 (Color interpolation)   
　　d\_interp 溶解插值 (Dissolve interpolation)   
　　lod 细节层次 (Level of detail)   
　　usemtl 材质名称 (Material name)   
　　mtllib 材质库 (Material library)   
　　shadow\_obj 投射阴影 (Shadow casting)   
　　trace\_obj 光线跟踪 (Ray tracing)   
　　ctech 曲线近似技术 (Curve approximation technique)   
　　stech 表面近似技术 (Surface approximation technique)

mtllib boo1.mtl 材质库，会直接加载该材质

usemtl \_bossdefault

看一个例子（在Maya中创建一个正方体导出）：

　　# The units used in this file are centimeters.   
　　g default   
　　v -0.500000 -0.500000 0.500000   
　　v 0.500000 -0.500000 0.500000   
　　v -0.500000 0.500000 0.500000   
　　v 0.500000 0.500000 0.500000   
　　v -0.500000 0.500000 -0.500000   
　　v 0.500000 0.500000 -0.500000   
　　v -0.500000 -0.500000 -0.500000   
　　v 0.500000 -0.500000 -0.500000   
　　vt 0.000000 0.000000   
　　vt 1.000000 0.000000   
　　vt 0.000000 1.000000   
　　vt 1.000000 1.000000   
　　vt 0.000000 2.000000   
　　vt 1.000000 2.000000   
　　vt 0.000000 3.000000   
　　vt 1.000000 3.000000   
　　vt 0.000000 4.000000   
　　vt 1.000000 4.000000   
　　vt 2.000000 0.000000   
　　vt 2.000000 1.000000   
　　vt -1.000000 0.000000   
　　vt -1.000000 1.000000   
　　vn 0.000000 0.000000 1.000000   
　　vn 0.000000 0.000000 1.000000   
　　vn 0.000000 0.000000 1.000000   
　　vn 0.000000 0.000000 1.000000   
　　vn 0.000000 1.000000 0.000000   
　　vn 0.000000 1.000000 0.000000   
　　vn 0.000000 1.000000 0.000000   
　　vn 0.000000 1.000000 0.000000   
　　vn 0.000000 0.000000 -1.000000   
　　vn 0.000000 0.000000 -1.000000   
　　vn 0.000000 0.000000 -1.000000   
　　vn 0.000000 0.000000 -1.000000   
　　vn 0.000000 -1.000000 0.000000   
　　vn 0.000000 -1.000000 0.000000   
　　vn 0.000000 -1.000000 0.000000   
　　vn 0.000000 -1.000000 0.000000   
　　vn 1.000000 0.000000 0.000000   
　　vn 1.000000 0.000000 0.000000   
　　vn 1.000000 0.000000 0.000000   
　　vn 1.000000 0.000000 0.000000   
　　vn -1.000000 0.000000 0.000000   
　　vn -1.000000 0.000000 0.000000   
　　vn -1.000000 0.000000 0.000000   
　　vn -1.000000 0.000000 0.000000   
　　s off   
　　g pCube1   
　　usemtl initialShadingGroup   
　　f 1/1/1 2/2/2 4/4/3 3/3/4   
　　f 3/3/5 4/4/6 6/6/7 5/5/8   
　　f 5/5/9 6/6/10 8/8/11 7/7/12   
　　f 7/7/13 8/8/14 2/10/15 1/9/16   
　　f 2/2/17 8/11/18 6/12/19 4/4/20   
　　f 7/13/21 1/1/22 3/3/23 5/14/24

解释：

vt 1.000000 0.000000 这句代表点的贴图坐标

vn 0.000000 0.000000 -1.000000 这句代表点的法线

s off 表示关闭光滑组

usemtl initialShadingGroup 表示使用的材质

f 7/13/21 这时在面的数据中多了贴图坐标uv点和法线的索引号，索引号分别作用左斜线(/)隔开

格式：“f 顶点索引/uv点索引/法线索引”

G pCube1表示组，这里的成组合Maya中的成组不一样，这里的成组是指把g pCube1后出现的面都结合到一起，组成一个整的多边形几何体

**Cocos2d-x的obj模型文件解析：**

Sprite3D的数据成员：

Skeleton3D\* \_skeleton; //skeleton

**Vector<MeshVertexData\*> \_meshVertexDatas;**

std::unordered\_map<std::string, AttachNode\*> \_attachments;

BlendFunc \_blend;

Vector<Mesh\*> \_meshes;

mutable AABB \_aabb; // cache current aabb

mutable Mat4 \_nodeToWorldTransform; // cache the matrix

mutable bool \_aabbDirty;

unsigned int \_lightMask;

bool \_shaderUsingLight; // is current shader using light ?

bool \_forceDepthWrite; // Always write to depth buffer

bool \_usingAutogeneratedGLProgram;

struct AsyncLoadParam

{

std::function<void(Sprite3D\*, void\*)> afterLoadCallback; // callback after load

void\* callbackParam;

bool result; // sprite load result

std::string modlePath;

std::string texPath; //

MeshDatas\* meshdatas;

MaterialDatas\* materialdatas;

NodeDatas\* nodeDatas;

};

AsyncLoadParam \_asyncLoadParam;

初始化：

bool Sprite3D::initWithFile(const std::string& path)

{

\_aabbDirty = true;

\_meshes.clear();

\_meshVertexDatas.clear();

CC\_SAFE\_RELEASE\_NULL(\_skeleton);

removeAllAttachNode();

if (loadFromCache(path))

return true;

**MeshDatas\* meshdatas = new (std::nothrow) MeshDatas();**

**MaterialDatas\* materialdatas = new (std::nothrow) MaterialDatas();**

**NodeDatas\* nodeDatas = new (std::nothrow) NodeDatas();**

**if (loadFromFile(path, nodeDatas, meshdatas, materialdatas))**

{

if (initFrom(\*nodeDatas, \*meshdatas, \*materialdatas))

{

//add to cache

auto data = new (std::nothrow) Sprite3DCache::Sprite3DData();

data->materialdatas = materialdatas;

data->nodedatas = nodeDatas;

data->meshVertexDatas = \_meshVertexDatas;

for (const auto mesh : \_meshes) {

data->glProgramStates.pushBack(mesh->getGLProgramState());

}

Sprite3DCache::getInstance()->addSprite3DData(path, data);

CC\_SAFE\_DELETE(meshdatas);

\_contentSize = getBoundingBox().size;

return true;

}

}

CC\_SAFE\_DELETE(meshdatas);

CC\_SAFE\_DELETE(materialdatas);

CC\_SAFE\_DELETE(nodeDatas);

return false;

}

Obj加载：

bool Sprite3D::loadFromFile(const std::string& path, NodeDatas\* nodedatas, MeshDatas\* meshdatas, MaterialDatas\* materialdatas)

{

std::string fullPath = FileUtils::getInstance()->fullPathForFilename(path);

std::string ext = FileUtils::getInstance()->getFileExtension(path);

if (ext == ".obj")

{

return **Bundle3D::loadObj(\*meshdatas, \*materialdatas, \*nodedatas, fullPath);**

}

else if (ext == ".c3b" || ext == ".c3t")

{

//load from .c3b or .c3t

auto bundle = Bundle3D::createBundle();

if (!bundle->load(fullPath))

{

Bundle3D::destroyBundle(bundle);

return false;

}

auto ret = bundle->loadMeshDatas(\*meshdatas)

&& bundle->loadMaterials(\*materialdatas) && bundle->loadNodes(\*nodedatas);

Bundle3D::destroyBundle(bundle);

return ret;

}

return false;

}

bool Bundle3D::loadObj(MeshDatas& meshdatas, MaterialDatas& materialdatas, NodeDatas& nodedatas, const std::string& fullPath, const char\* mtl\_basepath)

{

meshdatas.resetData();

materialdatas.resetData();

nodedatas.resetData();

std::string mtlPath = "";

if (mtl\_basepath)

mtlPath = mtl\_basepath;

else

mtlPath = fullPath.substr(0, fullPath.find\_last\_of("\\/") + 1).c\_str();

std::vector<tinyobj::shape\_t> shapes;

std::vector<tinyobj::material\_t> materials;

auto ret = **tinyobj::LoadObj(shapes, materials, fullPath.c\_str(), mtlPath.c\_str());// 这个函数会生成shapes，实际上会填充mesh\_t，这里面包含了position、normal、texcoords、indices以及material\_ids，需要注意的是，即使obj文件中的面f是多边形，它也会将它变为三角形，同时添加了indices的三角形索引，同时在变换三角形的时候，会有顶点缓存，用于求索引，在position、normal、texcoords都相同的情况下会使用之前缓存的索引，否则新添加position、normal及texcoord，也就是说mesh中会有相同的position，但这个点的normal或者texcoord会不同，对于每一个新的三角形索引都会添加一个material\_id：shape.mesh.material\_ids.push\_back(material\_id);实际上所有的材质都来自于mtllib，然后对于每个object会用usemtl来获取当前object使用的材质，如果没有找到该材质则material\_id=-1**

if (ret.empty())

{

//fill data

//convert material

int i = 0;

char str[20];

std::string dir = "";

auto last = fullPath.rfind("/");

if (last != -1)

dir = fullPath.substr(0, last + 1);

for (auto& material : materials) {

NMaterialData materialdata;

NTextureData tex;

tex.filename = material.diffuse\_texname.empty() ? material.diffuse\_texname : dir + material.diffuse\_texname;

tex.type = NTextureData::Usage::Diffuse;

tex.wrapS = GL\_CLAMP\_TO\_EDGE;

tex.wrapT = GL\_CLAMP\_TO\_EDGE;

sprintf(str, "%d", i++);

materialdata.textures.push\_back(tex);

materialdata.id = str;

material.name = str;

materialdatas.materials.push\_back(materialdata);

}

//convert mesh

i = 0;

for (auto& shape : shapes) {

auto mesh = shape.mesh;

MeshData\* meshdata = new (std::nothrow) MeshData();

MeshVertexAttrib attrib;

attrib.size = 3;

attrib.type = GL\_FLOAT;

**// 下面这段代码是将所有的之前封装在shape中的mesh\_t填充到MeshData中，MeshData中的vertices就是所有的顶点（包括顶点坐标、法线坐标、纹理坐标），MeshData中的attribs就是每个坐标类型的属性**

**struct MeshData**

**{**

**typedef std::vector<unsigned short> IndexArray;**

**std::vector<float> vertex;**

**int vertexSizeInFloat;**

**std::vector<IndexArray> subMeshIndices;**

**std::vector<std::string> subMeshIds; //subMesh Names (since 3.3)**

**std::vector<AABB> subMeshAABB;**

**int numIndex;**

**std::vector<MeshVertexAttrib> attribs;**

**int attribCount;**

**}**

if (mesh.positions.size())

{

attrib.vertexAttrib = GLProgram::VERTEX\_ATTRIB\_POSITION;

attrib.attribSizeBytes = attrib.size \* sizeof(float);

meshdata->**attribs**.push\_back(attrib);

}

bool hasnormal = false, hastex = false;

if (mesh.normals.size())

{

hasnormal = true;

attrib.vertexAttrib = GLProgram::VERTEX\_ATTRIB\_NORMAL;

attrib.attribSizeBytes = attrib.size \* sizeof(float);;

meshdata->**attribs**.push\_back(attrib);

}

if (mesh.texcoords.size())

{

hastex = true;

attrib.size = 2;

attrib.vertexAttrib = GLProgram::VERTEX\_ATTRIB\_TEX\_COORD;

attrib.attribSizeBytes = attrib.size \* sizeof(float);

meshdata->**attribs**.push\_back(attrib);

}

auto vertexNum = mesh.positions.size() / 3;

for(unsigned int k = 0; k < vertexNum; k++)

{

meshdata->**vertex**.push\_back(mesh.positions[k \* 3]);

meshdata->**vertex**.push\_back(mesh.positions[k \* 3 + 1]);

meshdata->**vertex**.push\_back(mesh.positions[k \* 3 + 2]);

if (hasnormal)

{

meshdata->**vertex**.push\_back(mesh.normals[k \* 3]);

meshdata->**vertex**.push\_back(mesh.normals[k \* 3 + 1]);

meshdata->**vertex**.push\_back(mesh.normals[k \* 3 + 2]);

}

if (hastex)

{

meshdata->**vertex**.push\_back(mesh.texcoords[k \* 2]);

meshdata->**vertex**.push\_back(mesh.texcoords[k \* 2 + 1]);

}

}

//split into submesh according to material

std::map<int, std::vector<unsigned short> > subMeshMap;

for (size\_t k = 0; k < mesh.material\_ids.size(); k++) {

int id = mesh.material\_ids[k];

size\_t idx = k \* 3;

subMeshMap[id].push\_back(mesh.indices[idx]);

subMeshMap[id].push\_back(mesh.indices[idx + 1]);

subMeshMap[id].push\_back(mesh.indices[idx + 2]);

}

auto node = new (std::nothrow) NodeData();

node->id = shape.name;

for (auto& submesh : subMeshMap) {

meshdata->subMeshIndices.push\_back(submesh.second);

meshdata->subMeshAABB.push\_back(calculateAABB(meshdata->vertex, meshdata->getPerVertexSize(), submesh.second));

sprintf(str, "%d", i++);

meshdata->subMeshIds.push\_back(str);

auto modelnode = new (std::nothrow) ModelData();

modelnode->matrialId = submesh.first == -1 ? "" : materials[submesh.first].name;

modelnode->subMeshId = str;

node->modelNodeDatas.push\_back(modelnode);

}

nodedatas.nodes.push\_back(node);

meshdatas.meshDatas.push\_back(meshdata);

}

return true;

}

CCLOG("warning: load %s file error: %s", fullPath.c\_str(), ret.c\_str());

return false;

}

看看的到的shapes和materials

typedef struct {

std::string name;

mesh\_t mesh;

} shape\_t;

typedef struct {

std::vector<float> positions;

std::vector<float> normals;

std::vector<float> texcoords;

std::vector<unsigned short> indices;

std::vector<int> material\_ids; // per-mesh material ID

} mesh\_t;

材质参数：

typedef struct {

std::string name;

float ambient[3];

float diffuse[3];

float specular[3];

float transmittance[3];

float emission[3];

float shininess;

float ior; // index of refraction

float dissolve; // 1 == opaque; 0 == fully transparent

// illumination model (see http://www.fileformat.info/format/material/)

int illum;

std::string ambient\_texname;

std::string diffuse\_texname;

std::string specular\_texname;

std::string normal\_texname;

std::map<std::string, std::string> unknown\_parameter;

} material\_t;