ASSIMP drawing textured model

cireNeikual   388

Never mind, I figured it out on my own. For those out there that would like to know, this is how I did it:

Model.h:

```
#ifndef MODEL_H
#define MODEL_H

#include <SDL.h>
#include <SDL_opengl.h>
#include <assimp.hpp>
#include <assimp.h>
#include <aiScene.h>       // Output data structure
#include <aiPostProcess.h> // Post processing flags

#include <FreeImage.h>
#include <vector>

#include <iostream>

#define aisgl_min(x,y) (x<y?x:y)
#define aisgl_max(x,y) (y>x?y:x)

struct TextureAndPath
{
        GLuint hTexture;
        aiString pathName;
};

class Model
{
private:
        std::vector<TextureAndPath> texturesAndPaths;
        const struct aiScene* scene;

        void recursiveTextureLoad(const struct aiScene *sc, const struct aiNode* nd);
        void recursive_render(const struct aiScene *sc, const struct aiNode* nd);

        void get_bounding_box_for_node(const struct aiNode* nd, struct aiVector3D* min, struct aiVector3D* max, struct aiMatrix4x4* trafo);
        void get_bounding_box(struct aiVector3D* min, struct aiVector3D* max);
public:
        Model();
        ~Model();

        void LoadModel(const char* fileName);
        void Draw();
};

void color4_to_float4(const struct aiColor4D *c, float f[4]);

void set_float4(float f[4], float a, float b, float c, float d);

void apply_material(const struct aiMaterial *mtl);

// Can't send color down as a pointer to aiColor4D because AI colors are ABGR.
void Color4f(const struct aiColor4D *color);

#endif
```

Implementation:

```
#include "Model.h"

Model::Model()
        : scene(NULL)
{
}

Model::~Model()
{
}

void Model::LoadModel(const char* fileName)
{
        scene = aiImportFile(fileName, aiProcessPreset_TargetRealtime_Quality);
        recursiveTextureLoad(scene, scene->mRootNode);
}

void Model::recursiveTextureLoad(const struct aiScene *sc, const struct aiNode* nd)
{
        int i;
        unsigned int n = 0, t;
        struct aiMatrix4x4 m = nd->mTransformation;

        // update transform
        aiTransposeMatrix4(&m);
        glPushMatrix();
        glMultMatrixf((float*)&m);

        // draw all meshes assigned to this node
        for (; n < nd->mNumMeshes; ++n)
        {
                const struct aiMesh* mesh = sc->mMeshes[nd->mMeshes[n]];
                unsigned int cont = aiGetMaterialTextureCount(sc->mMaterials[mesh->mMaterialIndex], aiTextureType_DIFFUSE);
                struct aiString* str = (aiString*)malloc(sizeof(struct aiString));

                if(cont > 0)
                {
                        //aiGetMaterialString(sc->mMaterials[mesh->mMaterialIndex],AI_MATKEY_TEXTURE_DIFFUSE(0),str);
                        aiGetMaterialTexture(sc->mMaterials[mesh->mMaterialIndex],aiTextureType_DIFFUSE,0,str,0,0,0,0,0,0);
                        // See if another mesh is already using this texture, if so, just copy GLuint instead of remaking entire texture
                        bool newTextureToBeLoaded = true;
                        for(int x = 0; x < texturesAndPaths.size(); x++)
                        {
                                if(texturesAndPaths[x].pathName == *str)
                                {
                                        TextureAndPath reusedTexture;
                                        reusedTexture.hTexture = texturesAndPaths[x].hTexture;
                                        reusedTexture.pathName = *str;
                                        texturesAndPaths.push_back(reusedTexture);
                                        newTextureToBeLoaded = false;

                                        std::cout << "Texture reused." << std::endl;

                                        break;
                                }
                        }

                        if(newTextureToBeLoaded)
                        {
                                FREE_IMAGE_FORMAT formato = FreeImage_GetFileType(str->data,0);
                                //Automatocally detects the format(from over 20 formats!)
                                FIBITMAP* imagen = FreeImage_Load(formato, str->data);
                                FIBITMAP* temp = imagen;
                                imagen = FreeImage_ConvertTo32Bits(imagen);
                                FreeImage_Unload(temp);
                                int w = FreeImage_GetWidth(imagen);
                                int h = FreeImage_GetHeight(imagen);

                                                        //Some debugging code
                                char* pixeles = (char*)FreeImage_GetBits(imagen);
                                //FreeImage loads in BGR format, so you need to swap some bytes(Or use GL_BGR).
                                //Now generate the OpenGL texture object
                                TextureAndPath newTexture;
                                newTexture.pathName = *str;
                                glGenTextures(1, &newTexture.hTexture);

                                                        glBindTexture(GL_TEXTURE_2D, newTexture.hTexture);
                                glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA, w, h, 0, GL_BGRA_EXT,GL_UNSIGNED_BYTE,(GLvoid*)pixeles );
                                //glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
                                glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
                                glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
                                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
                                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

                                glBindTexture(GL_TEXTURE_2D, newTexture.hTexture);

                                                        GLenum huboError = glGetError();

                                                        if(huboError)
                                {
                                        std::cout<<"There was an error loading the texture"<<std::endl;
                                }

                                std::cout << "texture loaded." << std::endl;

                                texturesAndPaths.push_back(newTexture);
                        }
                }
        }

        // Get textures from all children
        for (n = 0; n < nd->mNumChildren; ++n)
                recursiveTextureLoad(sc, nd->mChildren[n]);
}

void Model::get_bounding_box_for_node(const struct aiNode* nd, struct aiVector3D* min, struct aiVector3D* max, struct aiMatrix4x4* trafo)
{
        struct aiMatrix4x4 prev; // Use struct keyword to show you want struct version of this, not normal typedef?
        unsigned int n = 0, t;

        prev = *trafo;
        aiMultiplyMatrix4(trafo,&nd->mTransformation);

        for (; n < nd->mNumMeshes; ++n)
        {
                const struct aiMesh* mesh = scene->mMeshes[nd->mMeshes[n]];
                for (t = 0; t < mesh->mNumVertices; ++t)
                {
                        struct aiVector3D tmp = mesh->mVertices[t];
                        aiTransformVecByMatrix4(&tmp,trafo);

                        min->x = aisgl_min(min->x,tmp.x);
                        min->y = aisgl_min(min->y,tmp.y);
                        min->z = aisgl_min(min->z,tmp.z);

                        max->x = aisgl_max(max->x,tmp.x);
                        max->y = aisgl_max(max->y,tmp.y);
                        max->z = aisgl_max(max->z,tmp.z);
                }
        }

        for (n = 0; n < nd->mNumChildren; ++n)
                get_bounding_box_for_node(nd->mChildren[n],min,max,trafo);

        *trafo = prev;
}

void Model::get_bounding_box(struct aiVector3D* min, struct aiVector3D* max)
{
        struct aiMatrix4x4 trafo;
        aiIdentityMatrix4(&trafo);

        min->x = min->y = min->z =  1e10f;
        max->x = max->y = max->z = -1e10f;
        get_bounding_box_for_node(scene->mRootNode,min,max,&trafo);
}

void Model::recursive_render(const struct aiScene *sc, const struct aiNode* nd)
{
        int i;
        unsigned int n = 0, t;
        struct aiMatrix4x4 m = nd->mTransformation;

        // update transform
        aiTransposeMatrix4(&m);
        glPushMatrix();
        glMultMatrixf((float*)&m);

        // draw all meshes assigned to this node
        for (; n < nd->mNumMeshes; ++n)
        {
                const struct aiMesh* mesh = sc->mMeshes[nd->mMeshes[n]];

                                if(n < texturesAndPaths.size())
                        glBindTexture(GL_TEXTURE_2D, texturesAndPaths[n].hTexture);

                apply_material(sc->mMaterials[mesh->mMaterialIndex]);

                if(mesh->mNormals == NULL)
                        glDisable(GL_LIGHTING);
                else
                        glEnable(GL_LIGHTING);

                if(mesh->mColors[0] != NULL)
                        glEnable(GL_COLOR_MATERIAL);
                else
                        glDisable(GL_COLOR_MATERIAL);

                for (t = 0; t < mesh->mNumFaces; ++t)
                {
                        const struct aiFace* face = &mesh->mFaces[t];
                        GLenum face_mode;

                        switch(face->mNumIndices)
                        {
                                case 1:
                                        face_mode = GL_POINTS;
                                        break;
                                case 2:
                                        face_mode = GL_LINES;
                                        break;
                                case 3:
                                        face_mode = GL_TRIANGLES;
                                        break;
                                default:
                                        face_mode = GL_POLYGON;
                                        break;
                        }

                        glBegin(face_mode);

                        for(i = 0; i < face->mNumIndices; i++)
                        {
                                int index = face->mIndices;
                                if(mesh->mColors[0] != NULL)
                                        Color4f(&mesh->mColors[0][index]);
                                if(mesh->mNormals != NULL)
                                        glNormal3fv(&mesh->mNormals[index].x);
                                if(mesh->HasTextureCoords(0))
                                        glTexCoord2f(mesh->mTextureCoords[0][index].x, mesh->mTextureCoords[0][index].y);
                                glVertex3fv(&mesh->mVertices[index].x);
                        }

                        glEnd();
                }
        }

        // draw all children
        for (n = 0; n < nd->mNumChildren; ++n)
                recursive_render(sc, nd->mChildren[n]);

        glPopMatrix();
}

void Model::Draw()
{
        recursive_render(scene, scene->mRootNode);
}

void color4_to_float4(const struct aiColor4D *c, float f[4])
{
        f[0] = c->r;
        f[1] = c->g;
        f[2] = c->b;
        f[3] = c->a;
}

void set_float4(float f[4], float a, float b, float c, float d)
{
        f[0] = a;
        f[1] = b;
        f[2] = c;
        f[3] = d;
}

void apply_material(const struct aiMaterial *mtl)
{
        float c[4];

        GLenum fill_mode;
        int ret1, ret2;
        struct aiColor4D diffuse;
        struct aiColor4D specular;
        struct aiColor4D ambient;
        struct aiColor4D emission;
        float shininess, strength;
        int two_sided;
        int wireframe;
        unsigned int max;

        set_float4(c, 0.8f, 0.8f, 0.8f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_DIFFUSE, &diffuse))
```

```
                color4_to_float4(&diffuse, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, c);

        set_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_SPECULAR, &specular))
                color4_to_float4(&specular, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, c);

        set_float4(c, 0.2f, 0.2f, 0.2f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_AMBIENT, &ambient))
                color4_to_float4(&ambient, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, c);

        set_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_EMISSIVE, &emission))
                color4_to_float4(&emission, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, c);

        max = 1;
        ret1 = aiGetMaterialFloatArray(mtl, AI_MATKEY_SHININESS, &shininess, &max);
        max = 1;
        ret2 = aiGetMaterialFloatArray(mtl, AI_MATKEY_SHININESS_STRENGTH, &strength, &max);
        if((ret1 == AI_SUCCESS) && (ret2 == AI_SUCCESS))
                glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shininess * strength);
        else {
                glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0f);
                set_float4(c, 0.0f, 0.0f, 0.0f, 0.0f);
                glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, c);
        }

        max = 1;
        if(AI_SUCCESS == aiGetMaterialIntegerArray(mtl, AI_MATKEY_ENABLE_WIREFRAME, &wireframe, &max))
                fill_mode = wireframe ? GL_LINE : GL_FILL;
        else
                fill_mode = GL_FILL;
        glPolygonMode(GL_FRONT_AND_BACK, fill_mode);

        max = 1;
        if((AI_SUCCESS == aiGetMaterialIntegerArray(mtl, AI_MATKEY_TWOSIDED, &two_sided, &max)) && two_sided)
                glEnable(GL_CULL_FACE);
        else
                glDisable(GL_CULL_FACE);
}

void Color4f(const struct aiColor4D *color)
{
        glColor4f(color->r, color->g, color->b, color->a);
}
```

As you can see, it is still based off of the original demo code, but adds textures and a texture manager.