MODEL

Don't be afraid to look at the code for this section. It is mostly Assimp-specific; unless you know how to implement Assimp, you will likely miss some things. If you want to know more about using Assimp, consult the GitHub.

A model contains multiple meshes, possibly with multiple textures.
- Example: A house that has a wooden balcony, a tower, and a swimming pool could still be loaded with a single model.
- We'll load the model via Assimp and translate it to multiple mesh objects from the `Mesh` class we created in the previous chapter.

Create a Model class header file that contains a constructor and functions to store model data.
- Model Class

Implement the Draw function.
- This is very simple. Just remember that a model is merely a collection of meshes.
- Draw Function

Importing a 3D Model into OpenGL

Implement the loadModel and processNode functions.
- Loading the Model and Processing Assimp Nodes

Assimp to Mesh

We need to translate the `aiMesh` object to our own `Mesh` object. All we need to do is access each of the mesh's relevant properties and store them in our own object. The general structure of the `processMesh` function then becomes: processMesh Structure

Processing a mesh is a 3-part process:
1. Retrieve all the vertex data
2. Retrieve the mesh's indices
3. Retrieve the relevant material data

Retrieve all the mesh data in the above steps and use that data to construct your own Mesh object.
- Processing Assimp Mesh Data

You should now be able to load 3D models into your application.

An Optimization

As it stands, our program loads a generates a new texture for each mesh, and since loading textures is not cheap, this quickly becomes the bottleneck of our model loading implementation.
- We can fix this by keeping track of whether we've loaded a texture already. We'll store the loaded textures globally and first check if it hasn't been loaded already. If it has, we just take that texture and skip the loading process.

Optimize the texture loading process to not load textures that have been previously loaded.
- Texture Loading Optimization

Loading a Model

With our model-loading implementation finished, loading 3D models into our application is very easy.

Load this 3D backpack model into your application and render it.
- Sample 3D Model Rendering Code

NOTE: Assimp cannot interpret ALL file formats. Also, all the texture data must be in the same folder.

FURTHER READING

How-To Texture Wavefront (.obj) Models for OpenGL: Great video guide by Matthew Early on how to set up 3D models in Blender so they directly work with the current model loader (as the texture setup we've chosen doesn't always work out of the box).

Model Class

Friday, April 15, 2022 3:01 PM

Model.h

```
#pragma once

#include <vector>

#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>

#include "Shader.h"
#include "Mesh.h"

class Model {
public:
   Model(char* path);

   void Draw(Shader& shader);

private:
   // model data
   std::vector<Mesh> meshes;
   std::string directory;

   void loadModel(std::string path);
   void processNode(aiNode* node, const aiScene* scene);
   Mesh processMesh(aiMesh* mesh, const aiScene* scene);
   std::vector<Texture> loadMaterialTextures(aiMaterial* mat, aiTextureType, std::string typeName);
};
```

Draw Function

Friday, April 15, 2022 3:45 PM

Model.cpp

```
// Draws the model from its meshes
void Model::Draw(Shader& shader) {
    for (int i = 0; i < meshes.size(); ++i)
        meshes[i].Draw(shader);
}
```

Loading the Model and Processing Assimp Nodes

Friday, April 15, 2022 4:25 PM

Model.cpp

```
// Uses assimp to load the model into an assimp scene object.
void Model::loadModel(std::string path) {
    Assimp::Importer importer;
    // Creates the assimp scene object, which contains all the assimp nodes of a model
    const aiScene* scene = importer.ReadFile(path, aiProcess_Triangulate | aiProcess_FlipUVs);

    // Error checking for scene object
    if (!scene || scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE || !scene->mRootNode) {
        std::cout << "ERROR::ASSIMP::" << importer.GetErrorString() << std::endl;
        return;
    }

    directory = path.substr(0, path.find_last_of('/'));
    processNode(scene->mRootNode, scene);
}

void Model::processNode(aiNode* node, const aiScene* scene) {
    // Process all the node's meshes (if any)
    for (int i = 0; i < node->mNumMeshes; ++i) {
        aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
        meshes.push_back(processMesh(mesh, scene));
    }
    // Then do the same for each of its children
    for (int i = 0; i < node->mNumChildren; ++i) {
        processNode(node->mChildren[i], scene);
    }
}
```

In the `Readfile` function, `aiProcess_Triangulate` tells assimp to transform all primitive shapes to triangles first, and `aiProcess_FlipUVs` flips the texture coordinates on the y-axis where necessary during processing (like `stbi_set_flip_vertically_on_load`).

After we load the model, we check if the scene and the root node of the scene are not null and check one of its flags to see if the returned data is incomplete and, if so, print the error message to the console and return. We then begin processing the data at the root node of the scene.

Each mesh of a node is stored in our meshes vector (after being reformatted to our `Mesh` object). Then, we process each child node of the current node recursively. This creates a parent-child hierarchy of meshes.

- This hierarchy means that we can translate, for example, a car mesh, which has child meshes (like an engine mesh, steering wh eel mesh, tire meshes, etc.), and each child mesh will move sequential with the parent mesh. Very useful.

processMesh Structure

Friday, April 15, 2022 9:32 PM

```
Mesh processMesh(aiMesh *mesh, const aiScene *scene) {
    vector<Vertex> vertices;
    vector<unsigned int> indices;
    vector<Texture> textures;

    for(unsigned int i = 0; i < mesh->mNumVertices; i++) {
        Vertex vertex;
        // process vertex positions, normals and texture coordinates
        [...]
        vertices.push_back(vertex);
    }
    // process indices
    [...]
    // process material
    if(mesh->mMaterialIndex >= 0) {
        [...]
    }

    return Mesh(vertices, indices, textures);
}
```

Processing Assimp Mesh Data

Monday, April 18, 2022 6:50 PM

```
Mesh Model::processMesh(aiMesh* mesh, const aiScene* scene) {
    std::vector<Vertex> vertices;
    std::vector<unsigned int> indices;
    std::vector<Texture> textures;

    Vertex vertex;

    // 1. Store the vertex data from the assimp mesh
    for (int i = 0; i < mesh->mNumVertices; ++i) {
        vertex.Position = glm::vec3(mesh->mVertices[i].x, mesh->mVertices[i].y, mesh->mVertices[i].z);
        vertex.Normal = glm::vec3(mesh->mNormals[i].x, mesh->mNormals[i].y, mesh->mNormals[i].z);

        // We only care about the first set of texture coodinates. Assimp allows 0 - 8 texture coordinates.
        if (mesh->mTextureCoords[0]) // some meshes don't contain texture coordinates
            vertex.TexCoords = glm::vec2(mesh->mTextureCoords[0][i].x, mesh->mTextureCoords[0][i].y);
        else
            vertex.TexCoords = glm::vec2(0.0f);

        vertices.push_back(vertex);
    }

    // 2. Store the indices from the assimp mesh
    for (int i = 0; i < mesh->mNumFaces; ++i) {
        for (int j = 0; j < mesh->mFaces[i].mNumIndices; ++j) {
            // For each face of the mesh, push each vertex index of the face to the indices vector.
            indices.push_back(mesh->mFaces[i].mIndices[j]);
        }
    }

    // 3. Store the materials (textures) from the assimp mesh
    if (mesh->mMaterialIndex >= 0) {
        aiMaterial* material = scene->mMaterials[mesh->mMaterialIndex];

        // Pushes all the diffuse maps into the textures vector
        std::vector<Texture> diffuseMaps = loadMaterialTextures(material, aiTextureType_DIFFUSE, "texture_diffuse");
        textures.insert(textures.end(), diffuseMaps.begin(), diffuseMaps.end());

        // Pushes all the specular maps into the textures vector
        std::vector<Texture> specularMaps = loadMaterialTextures(material, aiTextureType_SPECULAR, "texture_specular");
        textures.insert(textures.end(), specularMaps.begin(), specularMaps.end());
    }

    return Mesh(vertices, indices, textures);
}

// Loads the material data, creates a texture from the data, and store it into a Texture struct
std::vector<Texture> Model::loadMaterialTextures(aiMaterial* mat, aiTextureType type, std::string typeName) {
    std::vector<Texture> textures;

    for (int i = 0; i < mat->GetTextureCount(type); ++i) {
        aiString str; // texture name
        mat->GetTexture(type, i, &str);
        Texture texture;
        texture.id = textureFromFile(str.C_Str(), directory);
        texture.type = typeName;
        texture.path = str; // this is explained in the next part of this chapter
        textures.push_back(texture);
    }

    return textures;
}

unsigned int textureFromFile(const char* name, std::string dir) {
    Texture2D tex{ dir + '/' + name, 0 };
    return tex.id;
}
```

Texture Loading Optimization

Monday, April 18, 2022 7:42 PM

Mesh.h

```
struct Texture {
   unsigned int id;
   std::string type;
   std::string path; // used to check if a texture being loaded has already been loaded
};
```

Model.h

```
...

class Model {
public:
    std::vector<Texture> textures_loaded;

    ...
};
```

`textures_loaded` stores the textures that have been loaded, and is compared to the texture currently being loaded to check if they are both the same texture. If so, the texture loading routine is skipped.

Model.cpp

```
// Loads the material data, creates a texture from the data, and store it into a Texture vector
std::vector<Texture> Model::loadMaterialTextures(aiMaterial* mat, aiTextureType type, std::string typeName) {
    std::vector<Texture> textures;

    for (int i = 0; i < mat->GetTextureCount(type); ++i) {
        aiString str; // texture name
        mat->GetTexture(type, i, &str);
        bool skip = false;

        // Checks if the texture has been loaded previously, and if so, skips the loading process
        for (int j = 0; j < textures_loaded.size(); ++j) {
            if (std::strcmp(textures_loaded[j].path.data(), str.C_Str()) == 0) {
                textures.push_back(textures_loaded[j]);
                return textures;
            }
        }

        // Texture has not been loaded previously. Loads and generates the texture
        Texture texture;
        texture.id = textureFromFile(str.C_Str(), directory);
        texture.type = typeName;
        texture.path = str.C_Str();
        textures.push_back(texture);
        textures_loaded.push_back(texture);
    }

    return textures;
}
```

Sample 3D Model Rendering Code

Monday, April 18, 2022 9:49 PM

This is sample code you can use to easily test if your 3D model-loading implementation works. You can expand on this to implement what you've learned, but you should only do so after you make sure your model load correctly.

basic.frag

```
#version 330 core

struct Material {
    sampler2D texture_diffuse1;
};
uniform Material material;

out vec4 FragColor;

in vec2 TexCoords;

void main()
{
    FragColor = texture(material.texture_diffuse1, TexCoords);
}
```

basic.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
   gl_Position = projection * view * model * vec4(aPos, 1.0);
   TexCoords = aTexCoords;
}
```

main.cpp

```
// GLFW Configuration
...

Shader shaderProgram{ "Shaders/basic.vert", "Shaders/basic.frag" };

Model backpackModel{ "Models/backpack/backpack.obj" };

stbi_set_flip_vertically_on_load(true);

glEnable(GL_DEPTH_TEST);

// 3D model render loop
while (!glfwWindowShouldClose(window)) {
   float currentFrame = glfwGetTime();
   deltaTime = currentFrame - lastFrame;
   lastFrame = currentFrame;

   processInput(window);

   glClearColor(0.2f, 0.2f, 0.2f, 1.0f);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   shaderProgram.use();

   glm::mat4 view = camera.GetViewMatrix();
   glm::mat4 projection = glm::perspective(glm::radians(camera.fov), (float)WIDTH / (float)HEIGHT, 0.1f, 100.0f);

   glm::mat4 model{ 1.0f };
   model = translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
   model = scale(model, glm::vec3(1.0f, 1.0f, 1.0f));

   glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "model"), 1, GL_FALSE, glm::value_ptr(model));
   glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
```

```
   glUniformMatrix4fv(glGetUniformLocation(shaderProgram.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

   backpackModel.Draw(shaderProgram);

   glfwSwapBuffers(window);
   glfwPollEvents();
}

...
```