COMP220: Graphics & Simulation

5: Textures & Models

Worksheet Schedule

Worksheet	Start	Formative deadline
1: Framework	Week 2	Mon **15th Feb** 4pm (Week 4)
2: Basic scene	Week 4	Mon **1st Mar** 4pm (Week 6)
3: Plan/prototype	Week 6	Mon **15th Mar** 4pm (Week 8)
4: Final iteration	Week 8	Mon **12th Apr** 4pm (Week 10)

Learning outcomes

By the end of this week, you should be able to:

- **Explain** how a 2D texture image can be wrapped onto a 3D model.
- **Explain** how a complex 3D model is represented in memory.
- **Write** programs which draw textured meshes to the screen.

Agenda

Agenda

- Lecture (async):
  - **Explore** options and settings for applying textures to meshes.
  - **Introduce** mesh file formats and the FBX structure.

Agenda

- Lecture (async):
  - **Explore** options and settings for applying textures to meshes.
  - **Introduce** mesh file formats and the FBX structure.
- Workshop (sync):
  - **Apply** simple textures to our OpenGL primitives.
  - **Implement** code to load models from file using Assimp.

Schedule

16:00-16:10	Arrival, sign-in & overview
16:10-16:40	Demo & Exercise: Loading Textures
16:40-16:50	Introduction to Assimp
16:50-17:30	Demo & Exercise: Loading a Mesh from File
17:30-18:00	Parsing a Scene and Storing Data

FALMOUTH
UNIVERSITY

Applying textures in OpenGL

Texture loading

Texture loading

Load with SDL Image:

```
SDL_Surface* image = IMG_Load("Crate.jpg");
```

Texture loading

Load with SDL Image:

```
SDL_Surface* image = IMG_Load("Crate.jpg");
```

Set up in OpenGL:

```
GLuint textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_2D, textureID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->w, image->h,
             0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
```

Texture loading

Load with SDL Image:

```
SDL_Surface* image = IMG_Load("Crate.jpg");
```

Set up in OpenGL:

```
GLuint textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_2D, textureID);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->w, image->h,
             0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
```

Assign texture coordinates to each vertex - as for any other attribute.

Texture filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

Texture filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

- **Linear interpolation** (`GL_LINEAR`) smooths between pixels

Texture filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

- **Linear interpolation** (`GL_LINEAR`) smooths between pixels
- **Nearest neighbour** (`GL_NEAREST`) is pixelated but may be slightly faster

Texture filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

- **Linear interpolation** (`GL_LINEAR`) smooths between pixels
- **Nearest neighbour** (`GL_NEAREST`) is pixelated but may be slightly faster
- **Mip-mapping** pre-calculates scaled down versions of the texture — improves quality but costs memory

Texture filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

- **Linear interpolation** (`GL_LINEAR`) smooths between pixels
- **Nearest neighbour** (`GL_NEAREST`) is pixelated but may be slightly faster
- **Mip-mapping** pre-calculates scaled down versions of the texture — improves quality but costs memory

```
glGenerateMipmap(GL_TEXTURE_2D);
```

Textures in GLSL

Fragment shader:

```
in vec2 textureCoords;
uniform sampler2D textureSampler;

void main()
{
    fragmentColour = texture(textureSampler, textureCoords);
}
```

Alpha in OpenGL

- Use `vec4` instead of `vec3` for colours

Alpha in OpenGL

- Use `vec4` instead of `vec3` for colours
- Textures can have an **alpha channel**

Alpha in OpenGL

- Use `vec4` instead of `vec3` for colours
- Textures can have an **alpha channel**
  - PNG supports alpha channels, JPG and BMP do not

Alpha in OpenGL

- Use `vec4` instead of `vec3` for colours
- Textures can have an **alpha channel**
  - PNG supports alpha channels, JPG and BMP do not
- Need to enable **alpha blending**

Alpha in OpenGL

- Use `vec4` instead of `vec3` for colours
- Textures can have an **alpha channel**
  - PNG supports alpha channels, JPG and BMP do not
- Need to enable **alpha blending**

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- Other values can be passed to `glBlendFunc` for special effects (e.g. **additive blending** is often used for particle effects simulating light, fire, explosions etc.)

FALMOUTH
UNIVERSITY

Importing models

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including
  - FBX

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including
  - FBX
  - OBJ

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including
  - FBX
  - OBJ
  - DAE (aka Collada)

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including
  - FBX
  - OBJ
  - DAE (aka Collada)
  - MD5 (DOOM3)

Open Asset Import Library

- There is an FBX SDK published by Autodesk, this can be used to load FBX files
- We will use Asset Import Library to load FBX files
- This allows us to support multiple file formats, including
  - FBX
  - OBJ
  - DAE (aka Collada)
  - MD5 (DOOM3)
  - SMD (Half Life 2, Portal etc)

Overview of an Assimp scene

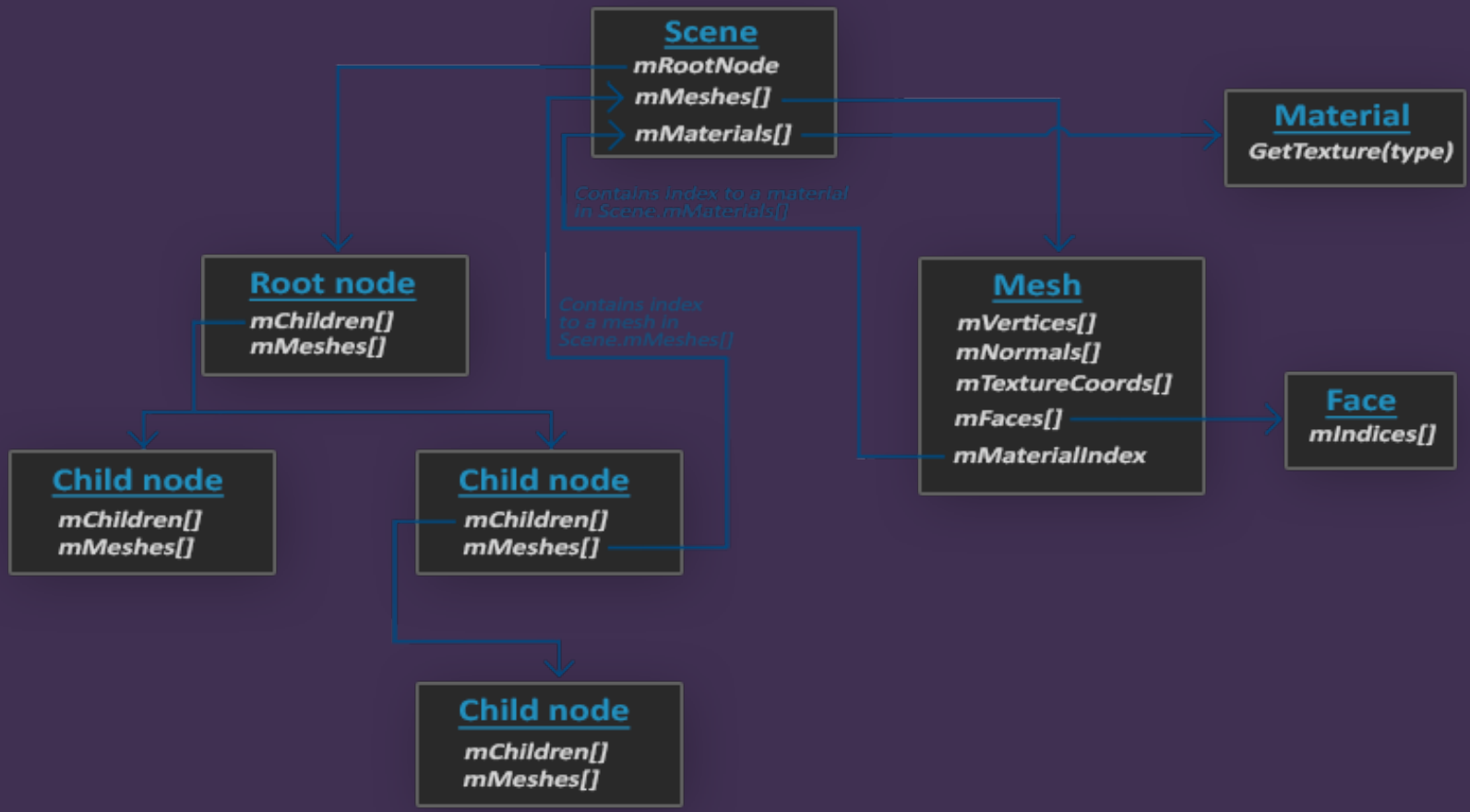

Image source: `https://learnopengl.com/Model-Loading/Assimp`

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

- An `aiScene` has (amongst other things):

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

- An `aiScene` has (amongst other things):
  - `aiMesh** mMeshes` - an array of pointers to the meshes in the scene

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

- An `aiScene` has (amongst other things):
  - `aiMesh** mMeshes` - an array of pointers to the meshes in the scene
  - `aiNode* mRootNode` - a pointer to the root node of the scene (provides access to all child nodes)

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

- An `aiScene` has (amongst other things):
  - `aiMesh** mMeshes` - an array of pointers to the meshes in the scene
  - `aiNode* mRootNode` - a pointer to the root node of the scene (provides access to all child nodes)
- An `aiNode` has pointers to its **parent** and **child** nodes, the **indices** of its meshes, and a **transform** relative to its parent.

Loading a model from file

```
Assimp::Importer importer;
const aiScene *scene = importer.ReadFile(path, flags);
```

- An `aiScene` has (amongst other things):
  - `aiMesh** mMeshes` - an array of pointers to the meshes in the scene
  - `aiNode* mRootNode` - a pointer to the root node of the scene (provides access to all child nodes)
- An `aiNode` has pointers to its **parent** and **child** nodes, the **indices** of its meshes, and a **transform** relative to its parent.
- Optional `flags` can be supplied to specify post-processing steps, e.g. `aiProcess_Triangulate`, `aiProcess_GenNormals`