COMP220: Graphics & Simulation

5: Textures & Models

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.

FALMOUTH
UNIVERSITY

Basic texture mapping

Shaders vs. textures

Shaders vs. textures

- **Shaders** determine surface appearance based on **interpolated per-vertex** properties.

Shaders vs. textures

- **Shaders** determine surface appearance based on **interpolated per-vertex** properties.
- **Textures** are 2D images (or arrays of values) that allow properties to be **varied across the surface**.

Shaders vs. textures

- **Shaders** determine surface appearance based on **interpolated per-vertex** properties.
- **Textures** are 2D images (or arrays of values) that allow properties to be **varied across the surface**.
- Values may represent colours, transparency, surface normals, surface displacements, light reflectance parameters etc.

Shaders vs. textures

- **Shaders** determine surface appearance based on **interpolated per-vertex** properties.
- **Textures** are 2D images (or arrays of values) that allow properties to be **varied across the surface**.
- Values may represent colours, transparency, surface normals, surface displacements, light reflectance parameters etc.
- **But** we can still only pass values at vertices...

Texture coordinates

- We use **UV coordinates** to refer to points in a texture

Texture coordinates

- We use **UV coordinates** to refer to points in a texture
- *u* axis is horizontal and ranges from 0 (left) to 1 (right)

Texture coordinates

- We use **UV coordinates** to refer to points in a texture
- *u* axis is horizontal and ranges from 0 (left) to 1 (right)
- *v* axis is vertical and ranges from 0 (bottom) to 1 (top)

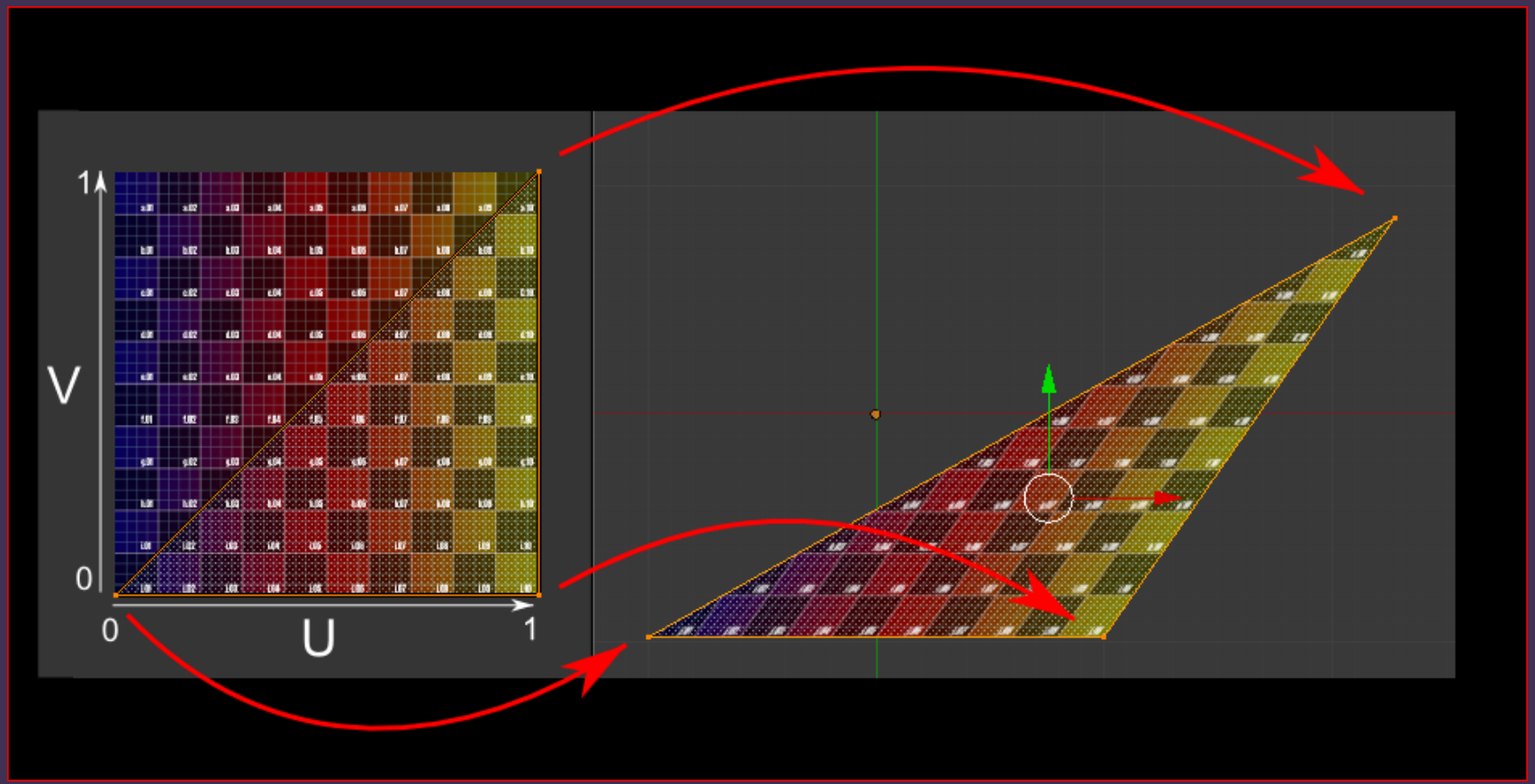
Texture coordinates

- We use **UV coordinates** to refer to points in a texture
- *u* axis is horizontal and ranges from 0 (left) to 1 (right)
- *v* axis is vertical and ranges from 0 (bottom) to 1 (top)
- (So really just another name for *xy* coordinates in texture space)

Texture coordinates

- We use **UV coordinates** to refer to points in a texture
- *u* axis is horizontal and ranges from 0 (left) to 1 (right)
- *v* axis is vertical and ranges from 0 (bottom) to 1 (top)
- (So really just another name for *xy* coordinates in texture space)
- Basic idea of texture mapping: give each vertex a *uv* coordinate, and interpolate across the triangle

UV coordinates



Character textures

FALMOUTH
UNIVERSITY

Texture parameters

Texture wrapping

What happens if we assign a coordinate outside the *uv* range?

Texture wrapping

What happens if we assign a coordinate outside the *uv* range?

- The **wrapping** parameter determines what happens for texture coordinates < 0 or > 1

Texture wrapping

What happens if we assign a coordinate outside the *uv* range?

- The **wrapping** parameter determines what happens for texture coordinates < 0 or > 1

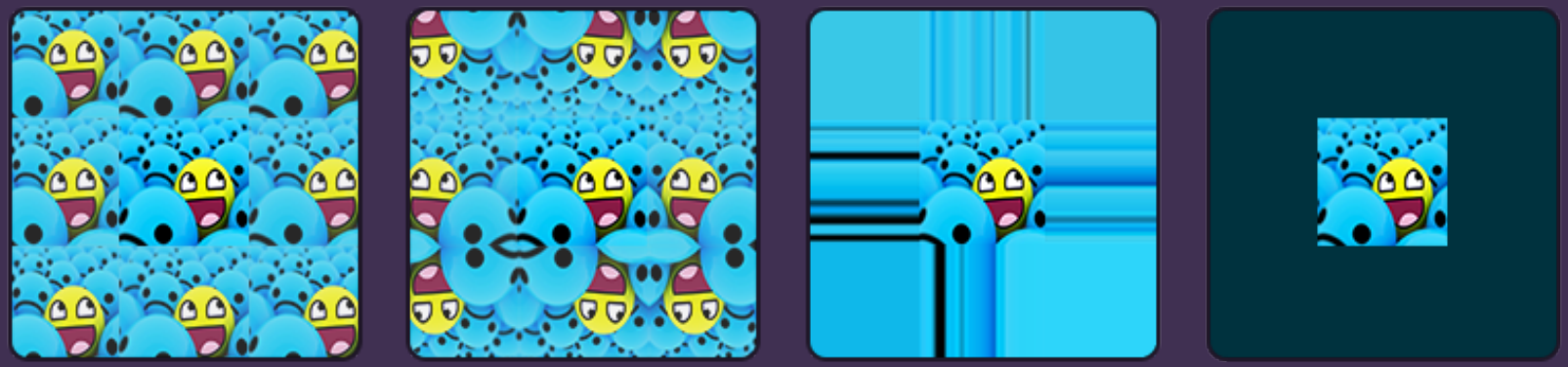


Image source: `https://learnopengl.com/Getting-started/Textures`

Texture filtering

How do we map a floating-point texture coordinate to the correct **texel** (texture element)?

Texture filtering

How do we map a floating-point texture coordinate to the correct **texel** (texture element)?

- **Nearest neighbour** selects the texel with the closest centre (Manhattan distance).

Texture filtering

How do we map a floating-point texture coordinate to the correct **texel** (texture element)?

- **Nearest neighbour** selects the texel with the closest centre (Manhattan distance).
- **(Bi)linear** takes an interpolated value from neighbouring texels.

Texture filtering

How do we map a floating-point texture coordinate to the correct **texel** (texture element)?

- **Nearest neighbour** selects the texel with the closest centre (Manhattan distance).
- **(Bi)linear** takes an interpolated value from neighbouring texels.
- Can specify different modes for scaling up and down.

Texture filtering

How do we map a floating-point texture coordinate to the correct **texel** (texture element)?

- **Nearest neighbour** selects the texel with the closest centre (Manhattan distance).
- **(Bi)linear** takes an interpolated value from neighbouring texels.
- Can specify different modes for scaling up and down.

Image source: `https://learnopengl.com/Getting-started/Textures`

Mipmaps

On faraway objects, a fragment may cover many texels...

Mipmaps

On faraway objects, a fragment may cover many texels...

- A **mipmap** is a set of textures with each scaled to half the size of the last.

Mipmaps

On faraway objects, a fragment may cover many texels...

- A **mipmap** is a set of textures with each scaled to half the size of the last.
- Chosen based on the distance to the camera.

Mipmaps

On faraway objects, a fragment may cover many texels...

- A **mipmap** is a set of textures with each scaled to half the size of the last.
- Chosen based on the distance to the camera.
- "Multum in parvo" = "much in a small space"

Mipmaps

On faraway objects, a fragment may cover many texels...

- A **mipmap** is a set of textures with each scaled to half the size of the last.
- Chosen based on the distance to the camera.
- "Multum in parvo" = "much in a small space"
- Pregenerated (on request) by OpenGL: faster processing/better quality for more memory.

Mipmaps

On faraway objects, a fragment may cover many texels...

- A **mipmap** is a set of textures with each scaled to half the size of the last.
- Chosen based on the distance to the camera.
- "Multum in parvo" = "much in a small space"
- Pregenerated (on request) by OpenGL: faster processing/better quality for more memory.

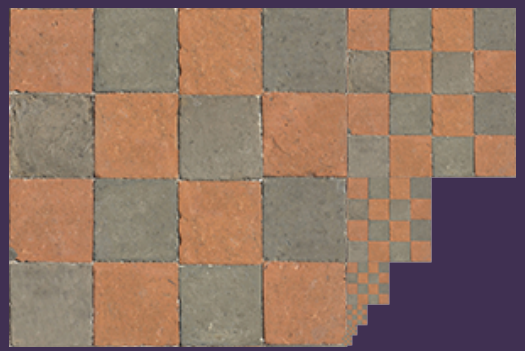


Image source: `https://learnopengl.com/Getting-started/Textures`

Texture dimensions

- In the old days, OpenGL required textures to have **power of two** dimensions

Texture dimensions

- In the old days, OpenGL required textures to have **power of two** dimensions
  - 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, . . .

Texture dimensions

- In the old days, OpenGL required textures to have **power of two** dimensions
  - $2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, \ldots$
- Nowadays **non-power of two (NPOT)** textures are widely supported

Texture dimensions

- In the old days, OpenGL required textures to have **power of two** dimensions
  - 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...
- Nowadays **non-power of two (NPOT)** textures are widely supported
- Still better to stick to powers of two as some things work better (e.g. mipmapping)

Texture dimensions

- In the old days, OpenGL required textures to have **power of two** dimensions
  - 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, . . .
- Nowadays **non-power of two (NPOT)** textures are widely supported
- Still better to stick to powers of two as some things work better (e.g. mipmapping)
- NB: **rectangular** textures are fine, but **square** textures make UV coordinates saner

FALMOUTH
UNIVERSITY

Transparency

Alpha

- We are used to working with colours in **RGB** space

Alpha

- We are used to working with colours in **RGB** space
- We can also work in **RGBA** space, where A = alpha = transparency

Alpha

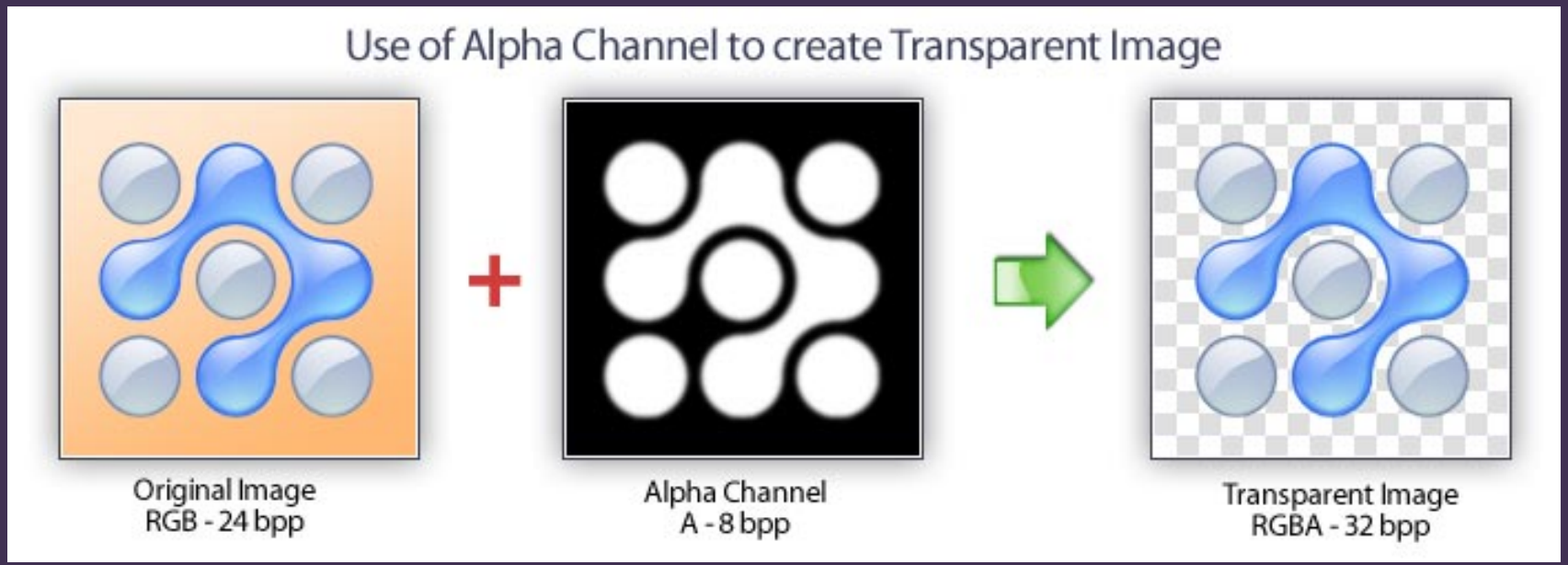
- We are used to working with colours in **RGB** space
- We can also work in **RGBA** space, where A = alpha = transparency
- $A = 0 \implies$ fully transparent

Alpha

- We are used to working with colours in **RGB** space
- We can also work in **RGBA** space, where A = alpha = transparency
- $A = 0 \implies$ fully transparent
- $A = 1$ (or $A = 255$) $\implies$ fully opaque

Alpha

- We are used to working with colours in **RGB** space
- We can also work in **RGBA** space, where A = alpha = transparency
- $A = 0 \implies$ fully transparent
- $A = 1$ (or $A = 255$) $\implies$ fully opaque



Transparency and depth testing

- Recall we are using **depth testing**

Transparency and depth testing

- Recall we are using **depth testing**
  - Each fragment on screen remembers its **depth** (distance from the camera)

Transparency and depth testing

- Recall we are using **depth testing**
  - Each fragment on screen remembers its **depth** (distance from the camera)
  - A new fragment is drawn **only if** its depth value is **less** than the current depth value

Transparency and depth testing

- Recall we are using **depth testing**
  - Each fragment on screen remembers its **depth** (distance from the camera)
  - A new fragment is drawn **only if** its depth value is **less** than the current depth value
  - I.e. don't draw objects that should be behind something that was already drawn

Transparency and depth testing

- Recall we are using **depth testing**
  - Each fragment on screen remembers its **depth** (distance from the camera)
  - A new fragment is drawn **only if** its depth value is **less** than the current depth value
  - I.e. don't draw objects that should be behind something that was already drawn
- But if the object in front is (semi-)transparent, we want to see the object behind it!

Transparency and depth testing

- Recall we are using **depth testing**
  - Each fragment on screen remembers its **depth** (distance from the camera)
  - A new fragment is drawn **only if** its depth value is **less** than the current depth value
  - I.e. don't draw objects that should be behind something that was already drawn
- But if the object in front is (semi-)transparent, we want to see the object behind it!
- Solution: draw semi-transparent objects **after** opaque objects, and in **back to front** order

FALMOUTH
UNIVERSITY

Complex meshes

Mesh Formats

- Typically we invent our own mesh format (see Doom's MD6, Valve's smd formats)

Mesh Formats

- Typically we invent our own mesh format (see Doom's MD6, Valve's smd formats)
- These formats are optimised for realtime rendering and are very efficient

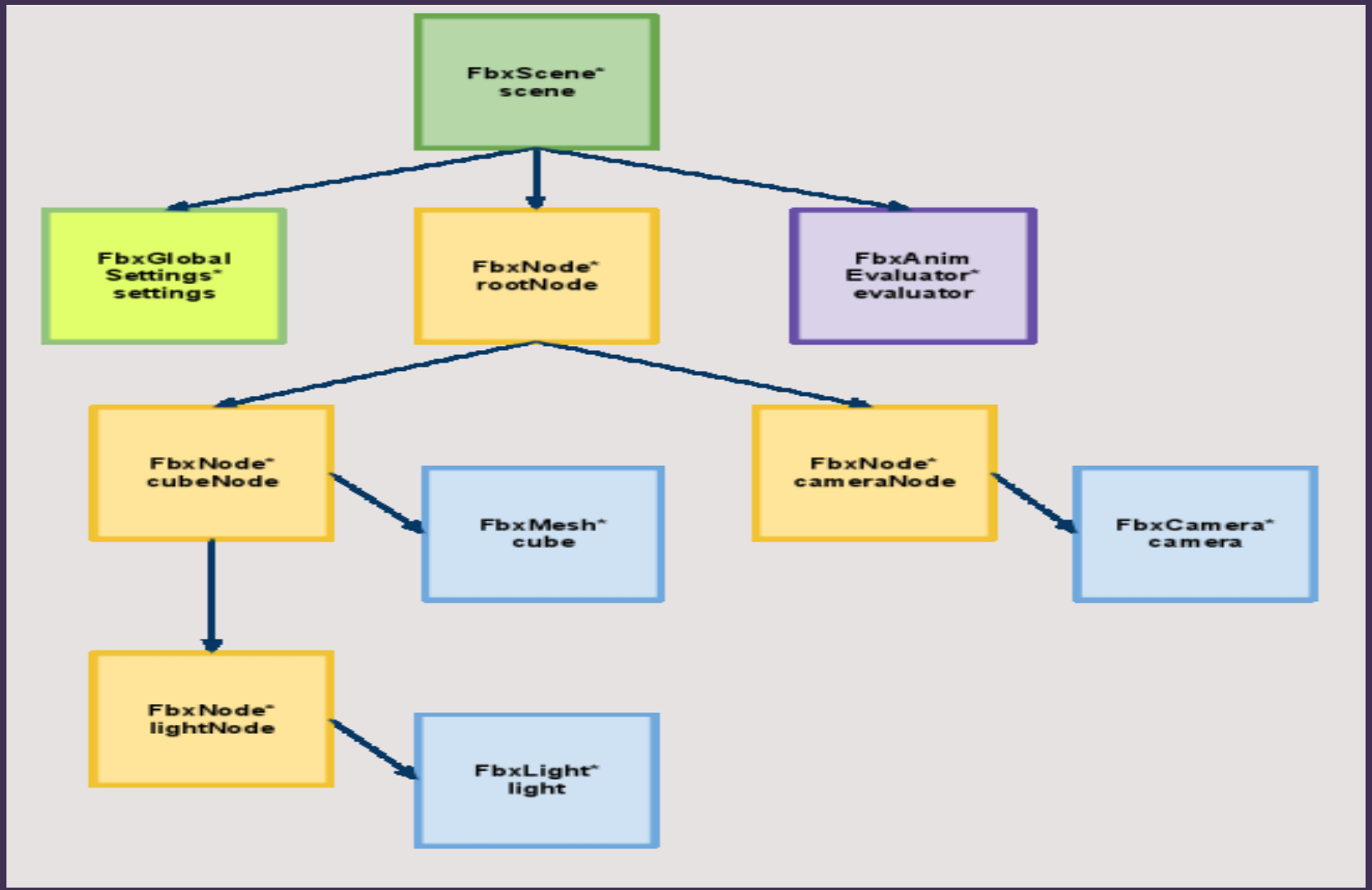
Mesh Formats

- Typically we invent our own mesh format (see Doom's MD6, Valve's smd formats)
- These formats are optimised for realtime rendering and are very efficient
- Usually developers write exporters for Maya or 3DSMax to support their format

Mesh Formats

- Typically we invent our own mesh format (see Doom's MD6, Valve's smd formats)
- These formats are optimised for realtime rendering and are very efficient
- Usually developers write exporters for Maya or 3DSMax to support their format
- We are going to use FBX (Autodesk Filmbox) as our model format, this known as an 'interchange' format

Quick Tour of the FBX Format



Next steps

- **Review** the additional asynchronous material for more background on texture properties and a preview of ways to structure code for importing meshes/models.
- **Attend** the workshop to practice loading textures and meshes from file.