Week 10: 3D Assets

Digital Asset Development

Contents

- Managing 3D assets
- Materials and textures
- UE materials

3D Assets

- So far in the module we have covered image, video and audio assets
- The other major asset type in the games and animation industries is 3D data
- Developing 3D assets is a specialist area with a pipeline of its own
  - Not covered in this module
- When creating 3D assets, the nature of the target output is vitally important

3D Asset Types

- Main asset types generated from within a 3D application are:
  - 3D meshes (models)
  - Skeletal rigs (usually for characters)
  - Animation data
  - Surface materials
- Can also generate associated 2D assets
  - Rendered stills and video
  - Textures (different from materials)

Meshes

- The mesh defines the basic geometry of any 3D object
  - A collection of vertices combined (usually) into polygons
  - Most outputs prefer meshes restricted to 4 or 3-sided polygons (quads and triangles)
  - Game engines usually require triangles only
  - Polygon flow will affect ability to texture and animate a model effectively
  - Polygon count will affect render times

Rigs and Animation Data

- When exporting a 3D character mesh, additional data is usually needed
- Rigging data identifies joint positions and the range of movement
- Animation data defines actual character movements
  - eg. run/walk cycles, jump, punch, kick,...
- For game characters, all of this data would usually be required

Materials

- To be visible in a render, a 3D mesh must be associated with a material
- The material defines how the mesh surface interacts with light
- Two aspects to defining a material:
  - Shading properties: overall way in which surface responds to incoming light
  - Texture: variation of shading properties across a surface

Shading Parameters

- Different applications allow different sets of shading parameters to be defined
  - Depends on rendering model used
  - For example, materials in Maya can have either Maya software or mental ray shaders
- As a rule, parameters include:
  - Overall colour
  - Reflectivity
  - Proportion of diffuse / specular response
  - Some means of defining surface roughness

Surface Roughness

- The large scale shape of a surface is defined by its mesh geometry
- However, most realistic surfaces are not entirely smooth at small scales
- We can build such small scale variation into a material in various ways
- Best known are bump and normal maps
  - Both involve using a texture image map to specify surface roughness

Textures

- “Texture” describes any spatial change in a material’s properties
- Can be defined procedurally (using maths) or via an image map
- Can apply textures to any parameter
  - Colour the is most obvious
  - Some applications allow many parameters to be mapped (diffuse, specularity,...)
  - Can also texture map surface roughness

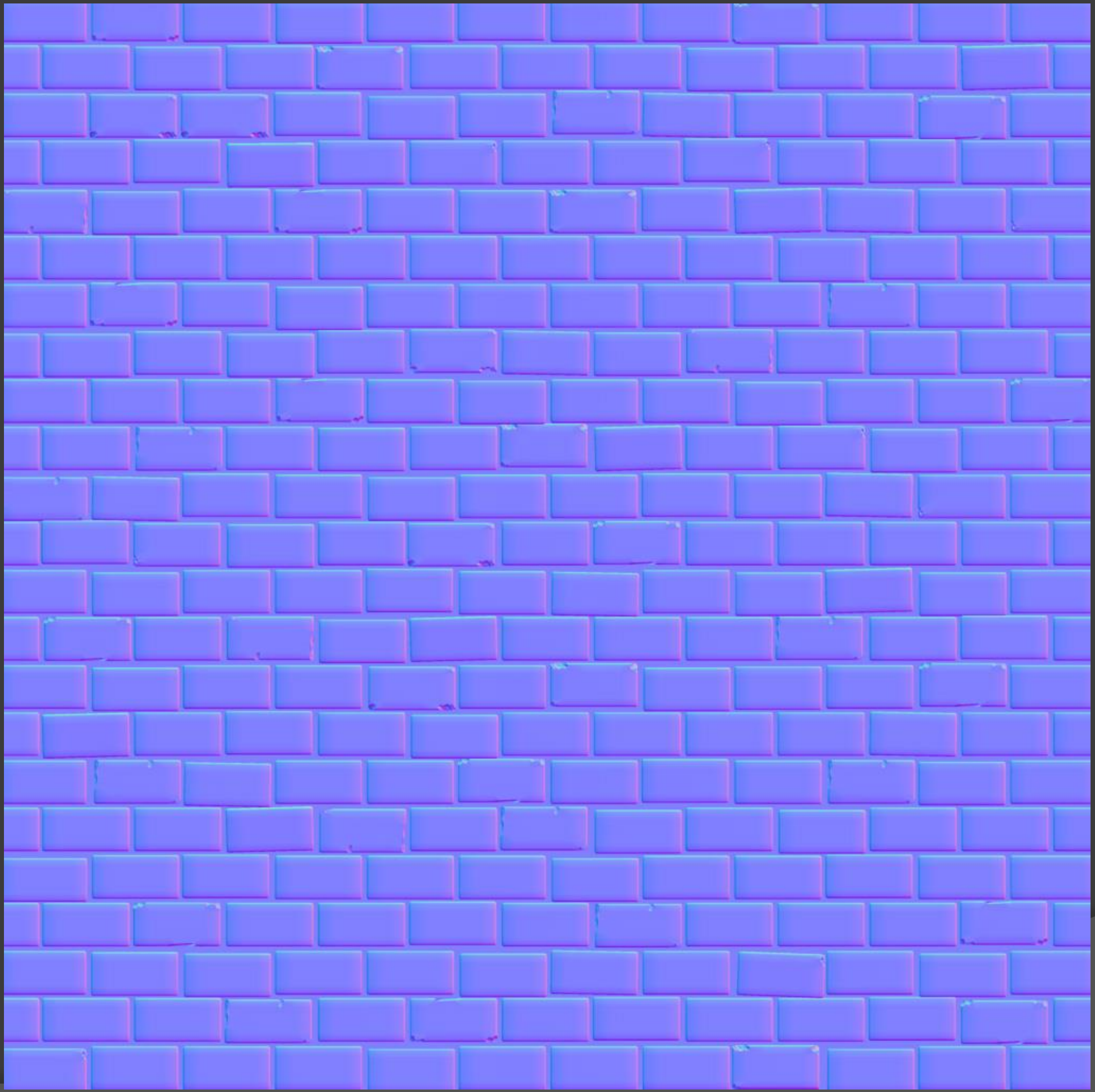
Bump Maps

- The traditional method of “faking” a rough surface is a bump map
  - A bump map is a greyscale image
- Brightness of each pixel indicates its height relative to the overall surface
  - Lighter = higher
  - Darker = lower
- A bump mapped surface appears lumpy and exhibits shadowing effects

Normal Maps

- Most modern renderer engines use normal maps to simulate roughness
- Pixel values describe surface normal direction rather than height
  - Normals are defined as 3-vectors, so normal maps are RGB images
  - Red, green and blue values represent the X, Y and Z components of the normal vector
- Normal maps typically give better quality results than bump mapping

A Typical Normal Map



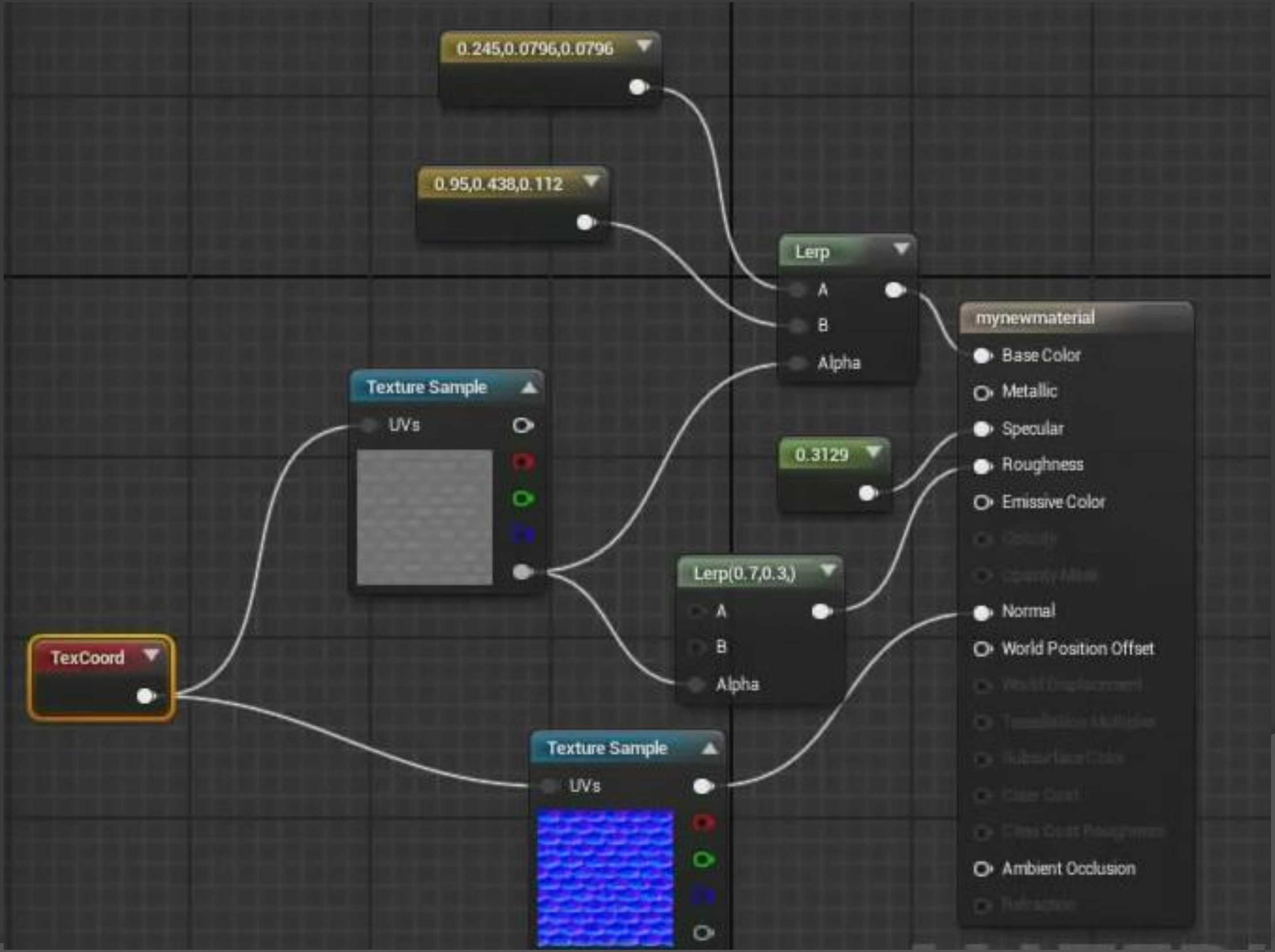
Materials in Unreal Engine

- Unreal Engine (UE) has a simpler model for materials than most 3D applications
  - Real time rendering requires this
- Main surface parameters are:
  - Base colour
  - Metallic (usually set to 0 or 1)
  - Specular
  - Roughness
  - Normal map

Material Nodes

- Like many applications, UE uses a node system for configuring materials
- Nodes may include:
  - Texture images
  - Texture coordinates (for positioning)
  - Vector and scalar constants (eg. colours)
  - Mathematical operations (including add, multiply, linear interpolate,...)
- These are linked into a node network

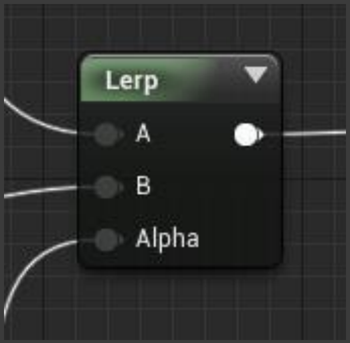
UE Node Network



Why Use Nodes?

- As well as UE, most major applications use nodes to control materials
  - Maya’s Hypershade window is an example
- Has some advantages over layers:
  - Easier to organise (when you know how!)
  - Nodes can be combined to give complex behaviour and functionality
  - Nodes can also be reused within a material
  - Chunks of a node network can be replicated for different materials

Linear Interpolate Node



- Known as “lerp” node
  - Acts rather like a Photoshop layer mask
- Takes two constants as input (A and B), plus an alpha mask
  - If the alpha mask is white, use value A
  - If the mask is black, use value B
  - Blend A and B for intermediate values

Combining Textures

- A general issue when using textures is how they can be combined
  - How to combine a material-based texture (eg. wood) with a structural one (panelling)
- Even using layers in Photoshop we have different ways of blending images
- Things get more complicated for non-colour parts of a texture
  - How do you combine two normal maps?

Combining Colour Maps

- The standard technique for combining colour textures is to multiply them
- This relates to the way material colours are rendered
  - The renderer multiplies the light and surface colours together
  - This is why red light on a green surface (for example) looks very dark
- Multiplying colours together therefore gives the appropriate result

Combining Normal Maps

- With normal maps it is small scale geometry we are trying to combine
- The combined normal map is the result of adding that geometry together
  - Theoretically, we should add two normal maps when combining their textures
  - However, this actually is wrong – it is only the X and Y components we should add
  - The correct method is thus to add the red and green channels of the normal maps