COMP280: Specialisms in Creative Computing

10: Geometry

Learning outcomes

- **Understand** how a mesh is represented in memory
- **Implement** custom meshes in UE4 or Unity
- **Manipulate** these meshes in a shader

Intro

- One of the most important points in 3D Graphics is that we are manipulating data on the GPU
- This means we need to understand how to package that data on the Application side
- You need to understand how this data is represented in memory
- Add how to operate on the data in shaders to achieve certain effects

Meshes

Mesh

- A mesh is a collection of vertices and indices which are collected together to form an object
- This is usually created by an artist in applications such as Maya
- And then exported in a file format such as FBX
- We can also create custom meshes in code e.g. Procedural Mesh in UE4 or Mesh Class in Unity
- Creating meshes in code are useful for certain effects and visual debugging

Vertices

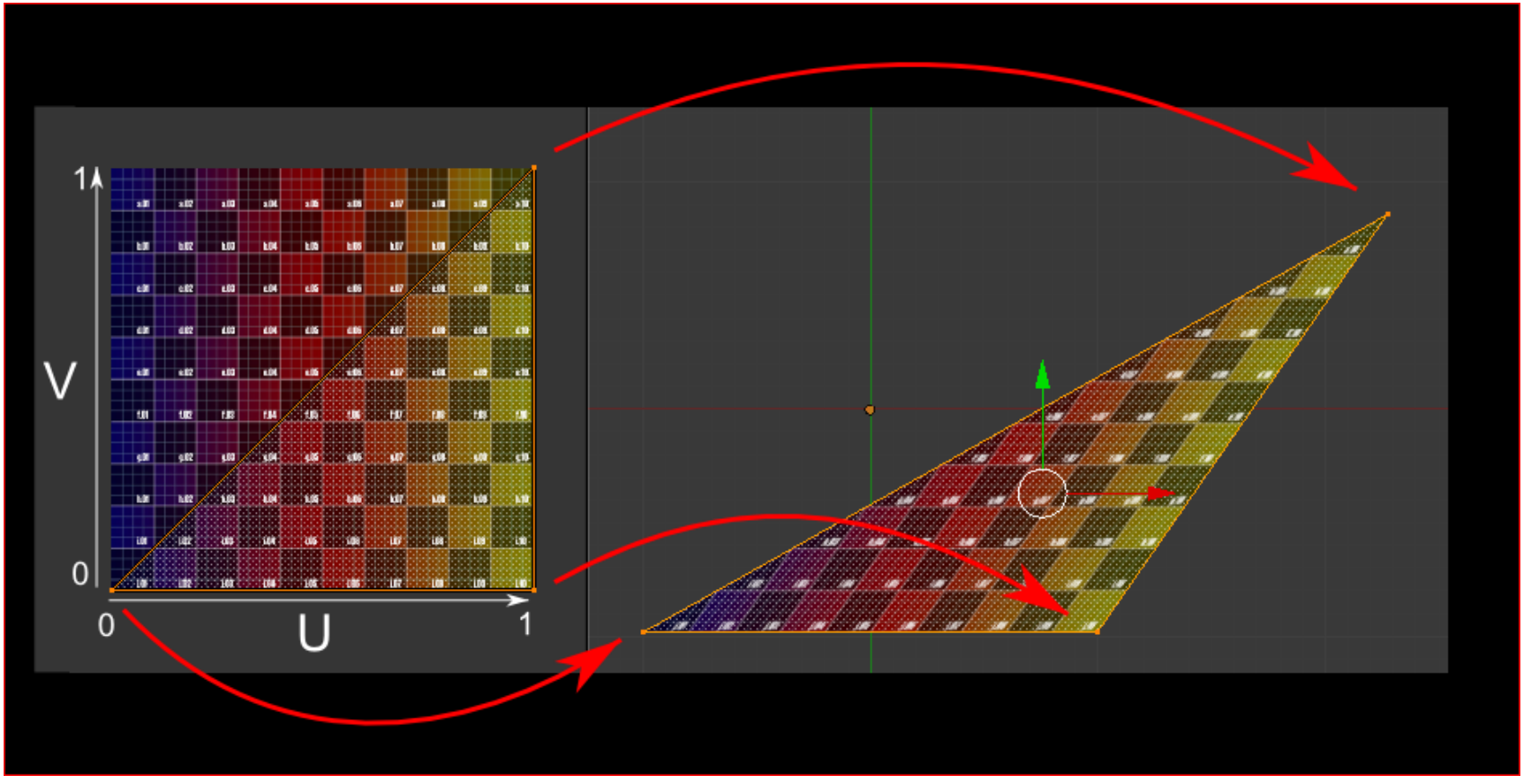
Position

- Vertices are the basic building blocks of any object
- It consists of at least one set of x, y, z coordinates
- This represents the position of a vertex in local space
- You must provide this x, y, z or the vertex shader will not run

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)

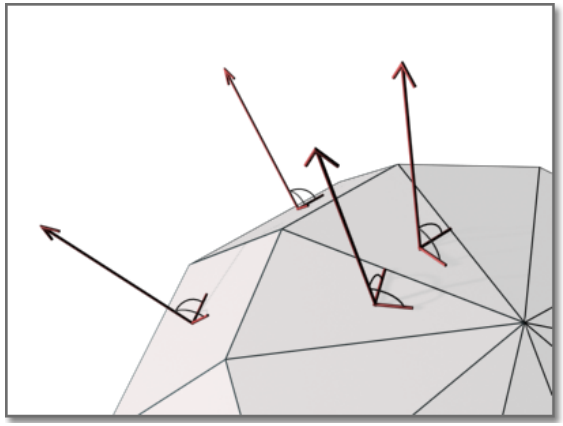
UV coordinates

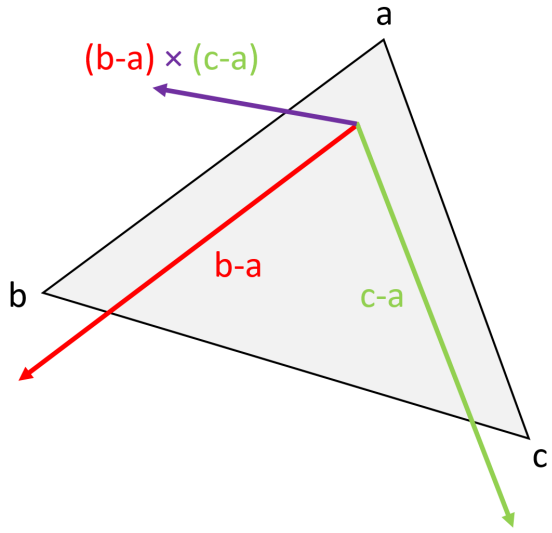


Normals

- Normals are used to specify the direction of a vertex
- This is represented as a unit vector (x, y, z)
- You can use the dot product of this normal and the light direction, to work out how much light is cast on the surface

Surface normals





- The **normal** to a surface is a **unit vector** that is **perpendicular** to the surface
- If we have two non-parallel vectors that are **tangent** to the surface, we can use the **cross product** to find the normal
- For a triangle with vertices a, b, c, two such vectors are $b - a$ and $c - a$
- So the normal is

$$\frac{n}{|n|} \quad \text{where} \quad n = (b - a) \times (c - a)$$

Flexible Vertex Format

- In addition to the above vertex elements there can be
  - Vertex Colours - r, g, b, a
  - Blend Weights - Index of the bone and a float weight
  - Tangent Normals - x, y, z
- There is nothing stopping you encoding any information into these
- You could use the vertex colours to hold target positions for animations

Vertex Buffer

- These vertices are stored in what is know as a Vertex Buffer
- In OpenGL and Directx, we fill these up and tell the pipeline what Buffer to use
- In UE4 and Unity, these buffers are usually hidden away from us
- We typically use higher level classes such as C# Mesh and UE4 Procedural Mesh to add in our own geometry

Indices

Indices and Rendering

- Indices are integers which specify the vertices that make up a mesh
- In rendering this allows us to send less data to the pipeline
- A cube would have 36 vertices, this will be at least 432 bytes (12 bytes per vertex)
- With indices, we used 8 vertices and 36 indices, which is around 240 bytes in total.

Index Buffer

- These indices are stored in what is know as a Index Buffer(Directx) or Element Buffer(OpenGL)
- In OpenGL and Directx, we fill these up and tell the pipeline what Buffer to use
- In UE4 and Unity, these buffers are usually hidden away from us
- We typically use higher level classes such as C# Mesh and UE4 Procedural Mesh to add in our own indices

Vertex Shader

Reminder

- As a reminder, the vertex shader takes in exactly one vertex
- This vertex will contain data we
- We then carry out operations on that vertex
- Then return that vertex back to the pipeline

Unity's Vertex

- Unity has built in vertex types
  - **appdata_base**: position, normal and one texture coordinate
  - **appdata_tan**: position, tangent, normal and one texture coordinate
  - **appdata_full**: position, tangent, normal, four texture coordinates and colour
- `https://docs.unity3d.com/Manual/SL-VertexProgramInputs.html`

UE4 Vertex - Expressions

- UE4 has a bunch of Expression nodes which allow you to interact with Vertices
  - Vector Expressions: `https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/ExpressionReference/Vector/index.html`
  - Coordinate Expressions: `https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/ExpressionReference/Coordinates/index.html`

Vertex Shader - GLSL Example

```
#version 330 core

layout(location = 0) in vec3 vertexPosition;
layout(location = 1) in vec2 vertexTextureCoord;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

out vec2 vertexTextureCoordOut;

void main(){

    mat4 mvpMatrix=projectionMatrix*viewMatrix*modelMatrix;

    vec4 mvpPosition=mvpMatrix*vec4(vertexPosition,1.0f);

    vertexTextureCoordOut=vertexTextureCoord;

    gl_Position=mvpPosition;
}
```

Fragment Shader

Reminder

- Takes in a pixel fragment (see rasterization)
- Outputs colour and depth values
- Typically used for shading calculations and texturing

Fragment Shader - GLSL Example

```
#version 330 core

in vec2 vertexTextureCoordOut;

out vec4 colour;

uniform sampler2D diffuseTexture;

void main()
{
    colour=texture2D(diffuseTexture,vertexTextureCoordOut);
}
```

Meshes Example

Unity3D - Meshes

- Mesh Class - `https://docs.unity3d.com/ScriptReference/Mesh.html`

UE4 - Meshes

- Procedural Mesh Blueprints - https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/index.html https://www.youtube.com/watch?v=dKlMEmVgbvg
- Procedural Mesh C++ - http://wlosok.cz/procedural-mesh-in-ue4-1-triangle/