Week 9: Introduction to VFX
Part 3: Geometry Meshes

COMP270: Mathematics for 3D Worlds and Simulations

Objectives

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

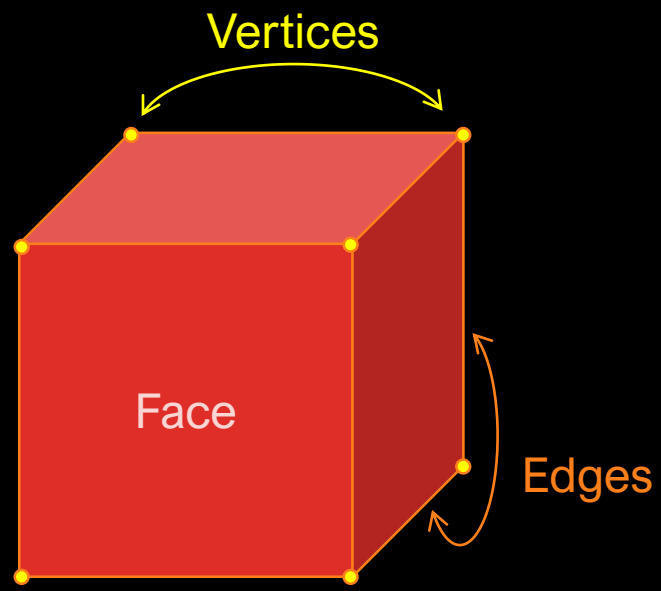
Packaging Data

- 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:
  - How this data is represented in memory
  - How to operate on the data in shaders to achieve certain effects

Meshes

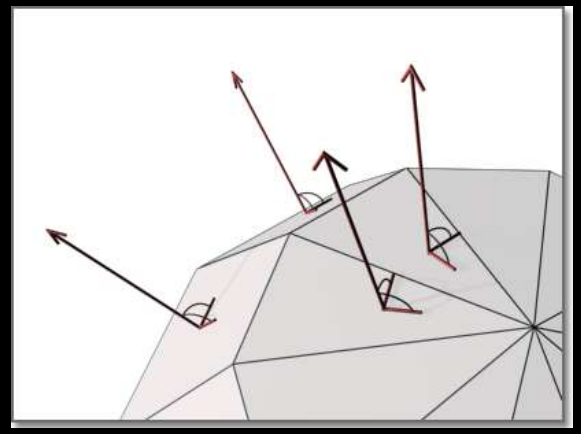
- A mesh is a collection of vertices which define polygons on the surface of an object
  - Each vertex contains the (x, y, z) coordinates of its position in local space – which is required for the shader to run
  - May contain other data
- Usually created by an artist in applications such as Maya, and then exported in a file format such as FBX
- We can also create custom meshes in code e.g.
  - Procedural Mesh in UE4 (via Blueprints or in C++)
  - Mesh Class in Unity

Procedural generation

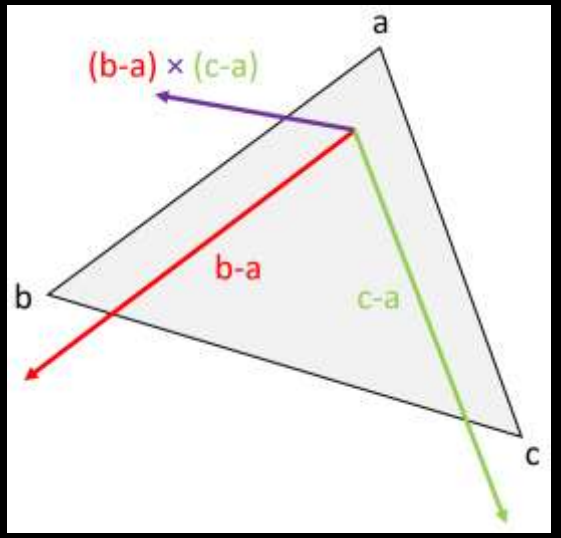


Surface Normals

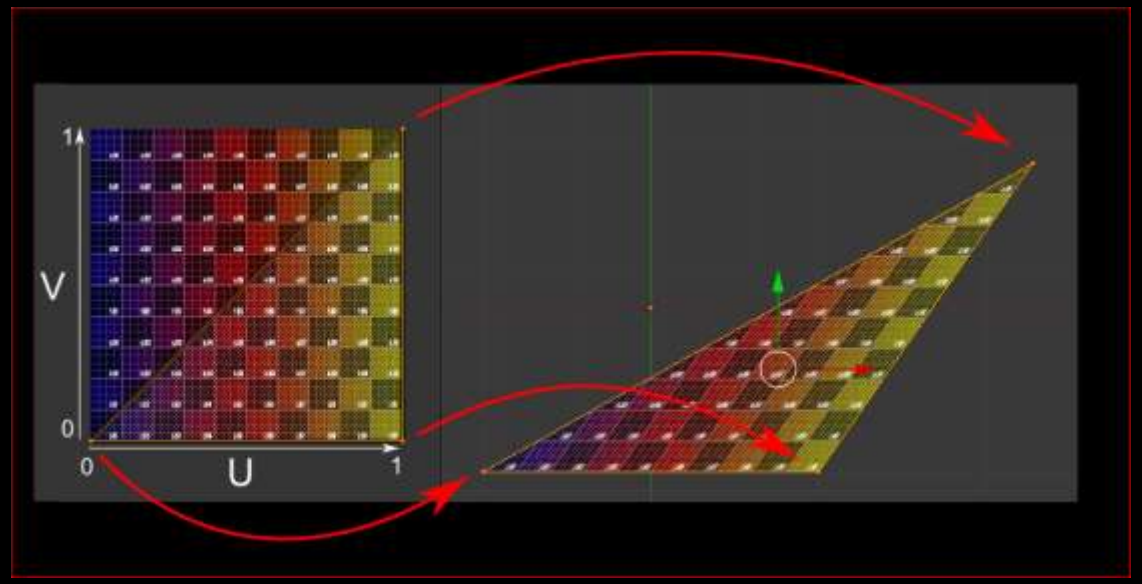
- The normal to a surface is a unit vector that is perpendicular to the surface
- If we have two non-parallel vectors that are tangent to the surface, we can use the cross product to find the normal
- For a triangle with vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, two such vectors are $\mathbf{b} - \mathbf{a}$ and $\mathbf{c} - \mathbf{a}$
- So the normal is $\frac{\mathbf{n}}{\|\mathbf{n}\|}$ where $\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})$



Order is important..



Texture Coordinates



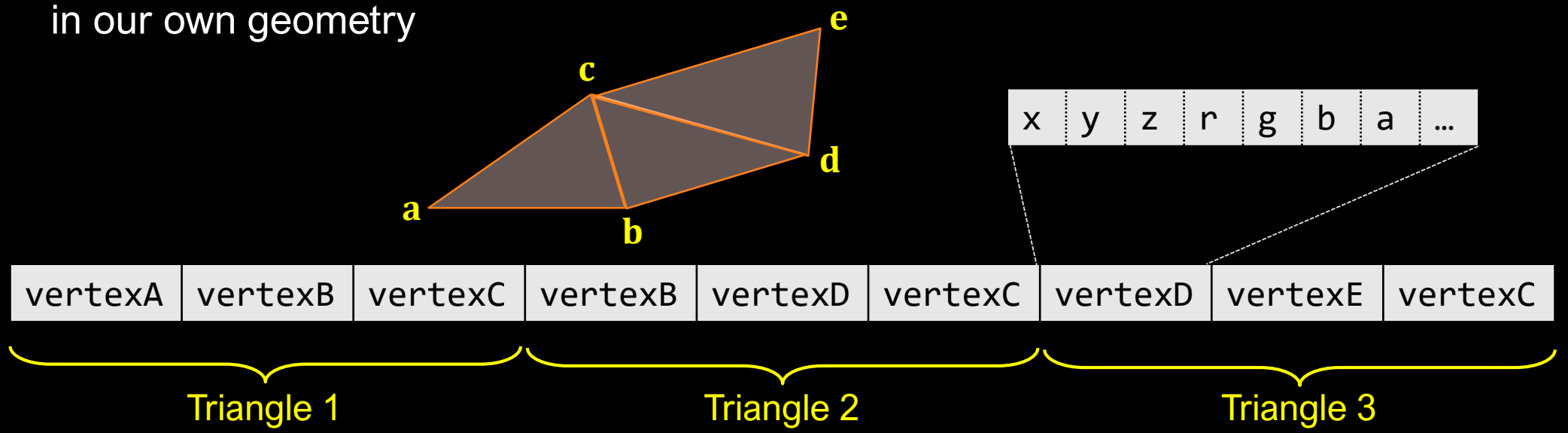
- We use UV coordinates to refer to points in a texture:
  - u axis is horizontal and ranges from 0 (left) to 1 (right)
  - v axis is vertical and ranges from 0 (bottom) to 1 (top)
- Each vertex is associated with a $(u.v)$ value, with the texture interpolated in between

Flexible Vertex Format

- In addition to the above vertex elements there can be other attributes, e.g.
  - Vertex Colours – r, g, b, a
  - Vertex Tangent – tangent to the surface (x, y, z)
  - Blend Weights – index of a bone and a float weight for skinning
- There is nothing stopping you encoding any information into these
  - For example, you could use the vertex colours to hold target positions for animations

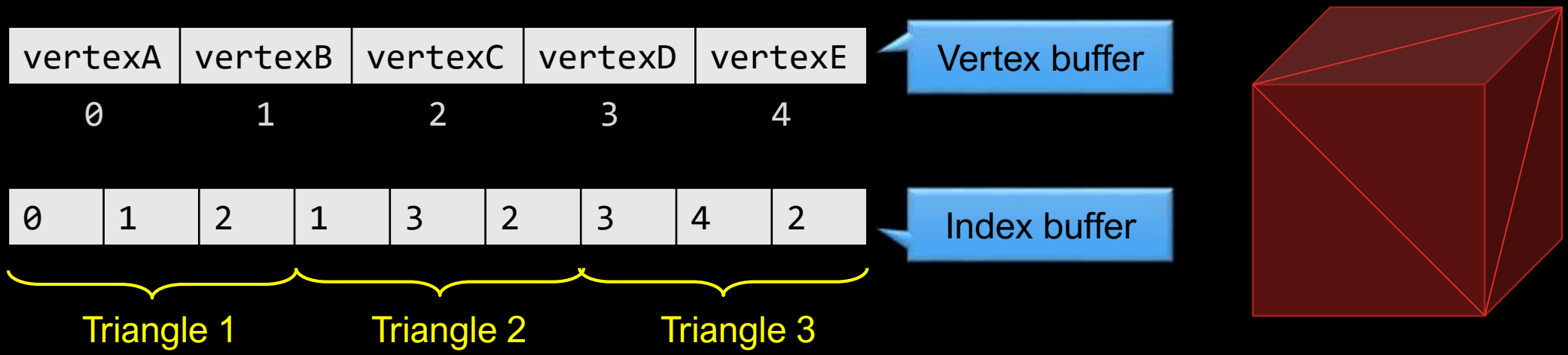
Vertex Buffer

- The vertices are stored in a Vertex Buffer
- In OpenGL and Direct3D, we fill these up and tell the pipeline which 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 are integers which specify the vertices that make up the triangles of a mesh
- In rendering this allows us to send less data to the pipeline
  - A triangulated 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 an Index Buffer (Direct3D) or Element Buffer (OpenGL)
- In OpenGL and Direct3D, we fill these up and tell the pipeline which 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

Generating Meshes

- Why?
  - Variety: possible to incorporate random variations
  - Scalability: choose the appropriate level of detail (number of vertices)
  - Convenience: allows developers to create geometry, to produce an effect or perhaps for visual debugging
- How?
  - Calculate the vertex positions (and other data) to store in the vertex buffer
  - Identify with the relevant triangles in the index buffer
  - Start with a basic shape and modify it…