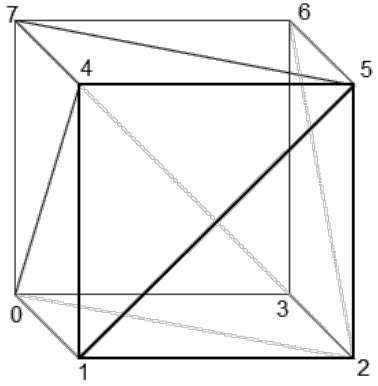
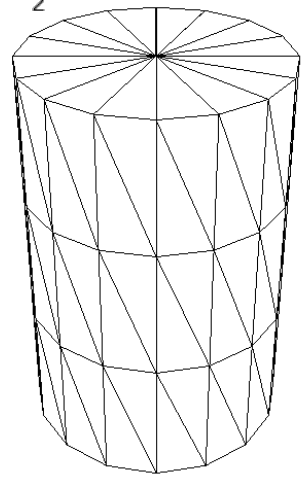
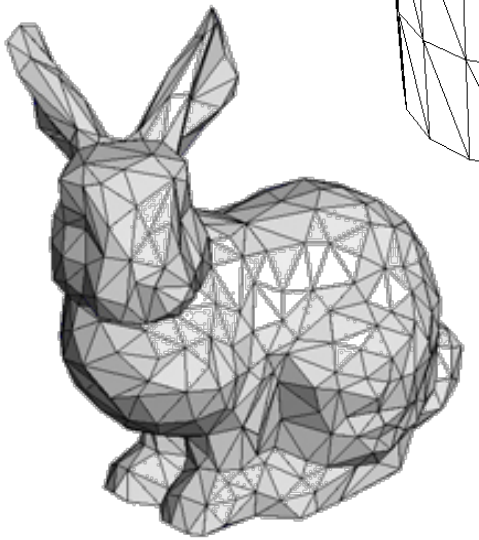
3D Model Basics

What are 3D models?

- Just more triangles
  - Enough to represent a full 3D shape
  - Generally the outer “shell” of the shape
- Vertices contain several sets of data
  - Position, normal, uv, etc.
- Ways to get these triangles:
  - Hardcode them (…ugh)
  - Procedurally generate them
  - Load from a file







Simple model format: .OBJ

- Advantages:
  - Purely text-based (easy to read with code)
  - Stores positions, uvs, normals and indices
  - Can export from almost every modeling package

- If you can read a text file in C++
  - You can load a model from an OBJ file
  - http://en.wikipedia.org/wiki/Wavefront_.obj_file

.Obj model files and Git

- By default, Git ***ignores .obj files!***
  - .obj is also a temporary VS file
- Solution 1
  - Alter .gitignore to never ignore files in an Asset folder
  - (Or just never ignore .obj in general)
- Solution 2
  - Rename .obj models with a different extension
  - Example: cube.obj → cube.ggp_model

OBJ file example: Quad

```
v -0.500000 -0.500000 0.000000
v  0.500000 -0.500000 0.000000
v -0.500000  0.500000 0.000000
v  0.500000  0.500000 0.000000

vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 1.000000

vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 0.000000 1.000000

f 1/1/1 2/2/2 3/3/3
f 3/3/3 2/2/2 4/4/4
```

v – vertex (position)
x, y, z

vt – vertex texture coords
u, v

vn – vertex normal (direction)
x, y, z

f – face (component indices)

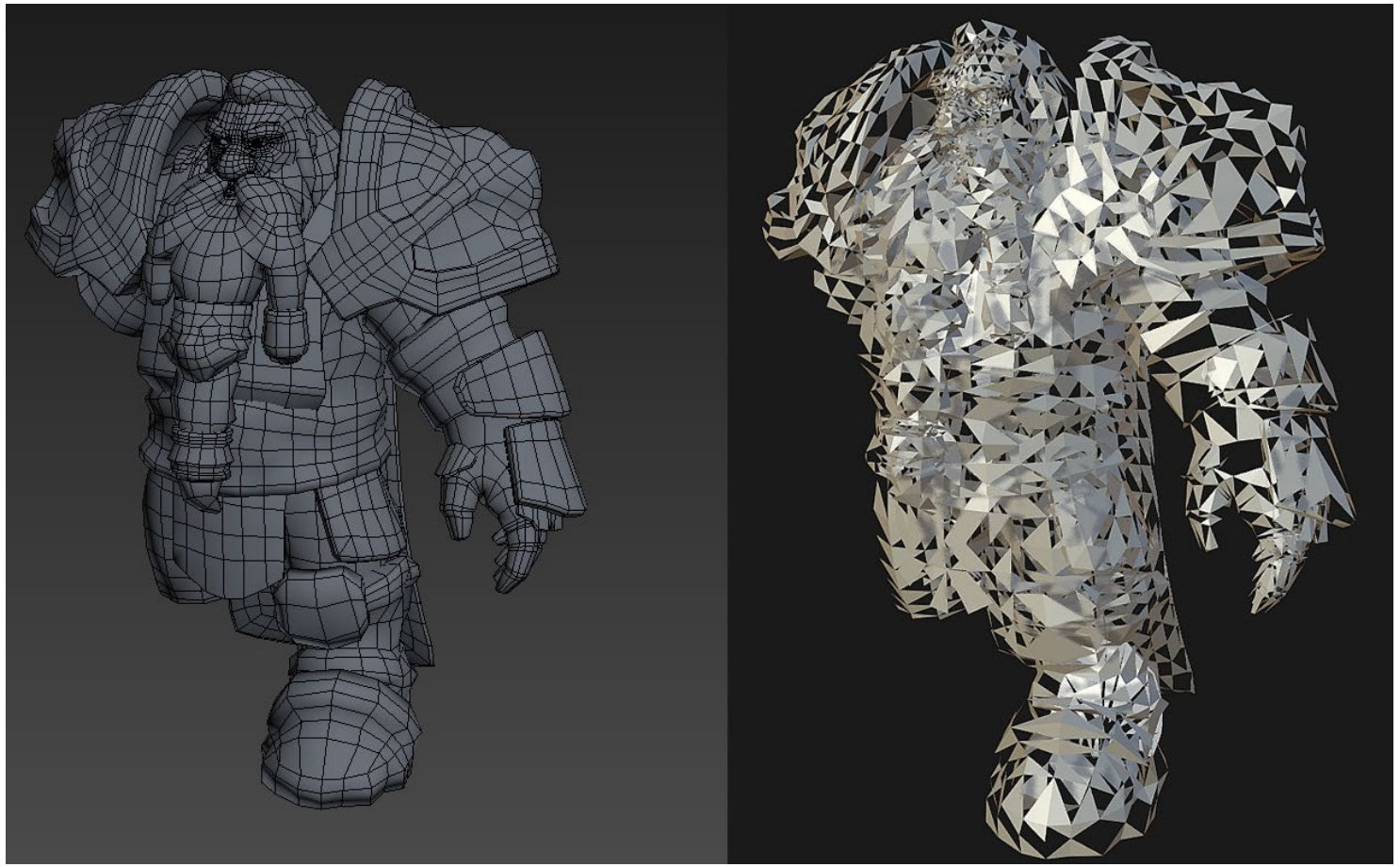
v/vt/vn v/vt/vn v/vt/vn

Model loading code

- I'll be providing basic .OBJ model loading code
  - Reads positions, uvs, normals and face definitions
  - Handles both 3 and 4-vertex faces
  - (Chops up quads into 2 triangles)
  - Does *not* handle external material definitions

- Other .OBJ loading libraries exist
  - TinyOBJLoader: https://github.com/tinyobjloader/tinyobjloader
  - Unnecessary for this course, but feel free to play around

Side note: What if we don’t triangulate quads?

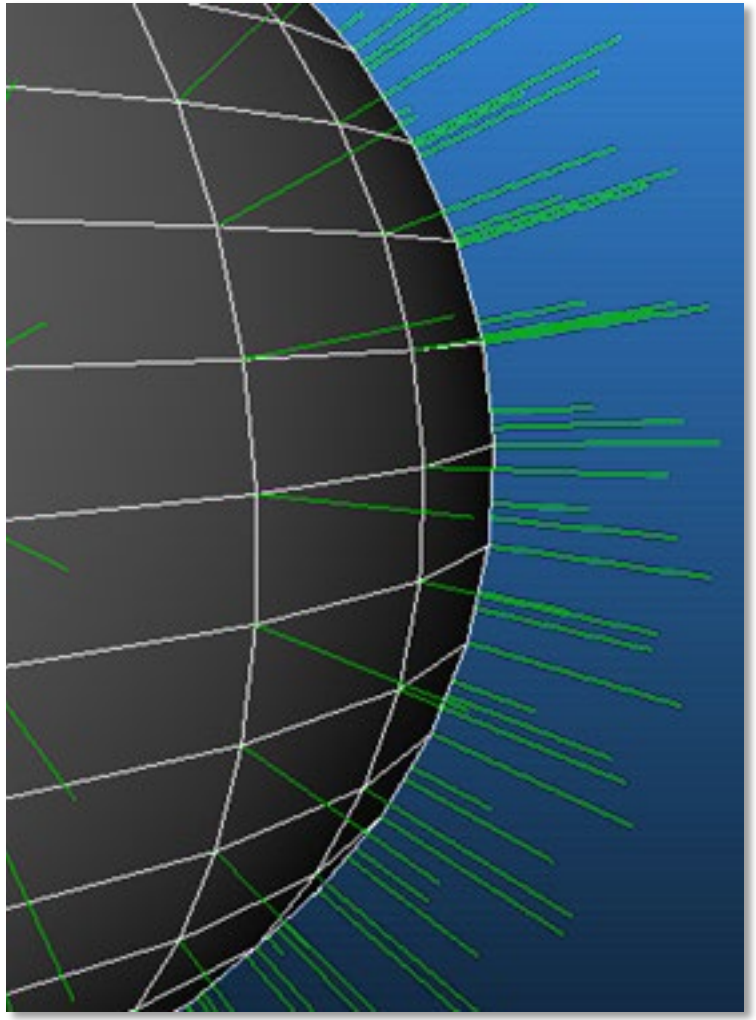


Side note: Other 3D model formats?

- .FBX
  - Probably best all-around format
  - Many advanced features (materials, embedded textures, animations)
  - But reading .FBX files manually is gnarly
- Want to load FBX files?
  - Use the *Open Asset Importer Library*
  - http://assimp.org
  - Can read 40+ different model formats
- Also unnecessary for this course (but useful to know about)

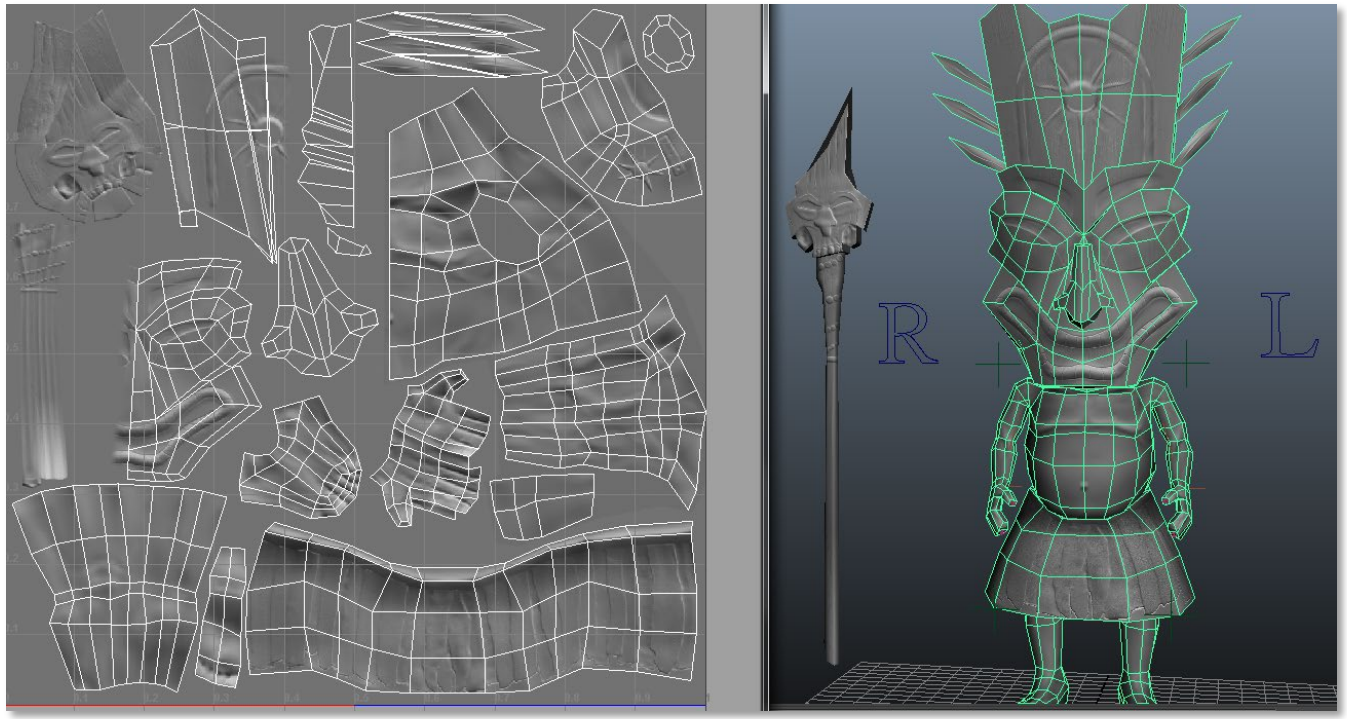
Normals

- Float3 stored at each vertex
- Define which way the surface “faces”
- Normal of each *pixel* is interpolated from normals at *vertices*



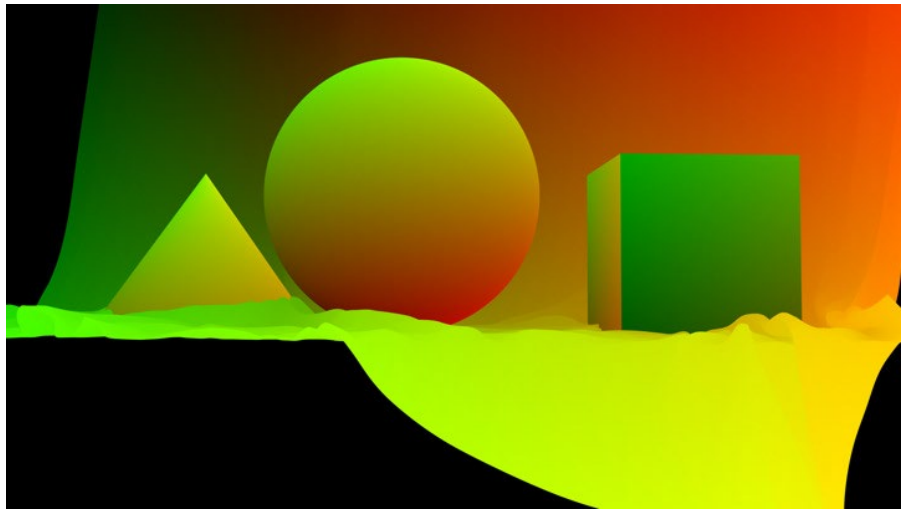
UV coordinates

- Float2 stored at each vertex
- Maps a texture region to individual triangles



UV coordinates at the pixel level

- Vertex UV is interpolated by rasterizer
- Each pixel gets a unique UV (float2)
- Used to look up a color in a texture



UV's returned as Red/Green colors

Handling 3D Models

Options and changes to starter code

Update your vertex format

- Vertex declaration should now have
  - **Position** Location in space
  - **UV** For texture mapping
  - **Normal** For lighting calculations

- Notice that *color* is removed!
  - Per-vertex color rarely used once we have textures
  - Most 3D models don't contain that information

Input Layout? SimpleShader has you covered!

- The pipeline needs to know the layout of our vertex buffer
- Since we're now using SimpleShader, we don't need to create an Input Layout ourselves anymore!
- But for your reference…

```
D3D11_INPUT_ELEMENT_DESC inputElements[3] = {};

// Set up the first element - a position, which is 3 float values
inputElements[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
inputElements[0].SemanticName = "POSITION";
inputElements[0].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;

// Set up the second element - UV coord, which is 2 more floats
inputElements[1].Format = DXGI_FORMAT_R32G32_FLOAT;
inputElements[1].SemanticName = "TEXCOORD";
inputElements[1].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;

// Set up a third - vertex normal, another 3 floats
inputElements[2].Format = DXGI_FORMAT_R32G32B32_FLOAT;
inputElements[2].SemanticName = "NORMAL";
inputElements[2].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;

device->CreateInputLayout(
    inputElements,
    3,
    shaderBlob->GetBufferPointer(),
    shaderBlob->GetBufferSize(),
    inputLayout.GetAddressOf());
```

Update Vertex Shader input struct

```
struct VertexShaderInput
{
    float3 position  : POSITION;
    float2 uv        : TEXCOORD;
    float3 normal    : NORMAL;
};
```

- Must match vertex definition
- Don’t forget semantics!

Update Vertex Shader output struct

```
struct VertexToPixel
{
    float4 position  : SV_POSITION;
    float2 uv        : TEXCOORD;
    float3 normal    : NORMAL;
};
```

- Need to get the uv and normal to the pixel shader
  - That's where we do texturing and lighting
  - We have to pass this data through

Update Pixel Shader input struct

```
struct VertexToPixel
{
    float4 position  : SV_POSITION;
    float2 uv        : TEXCOORD;
    float3 normal    : NORMAL;
};
```

- Should match vertex output!
  - These could even be stored in another HLSL file
  - And #included in both shaders

Materials

What is a material? (In the real world)

- Physical matter (elements or substances)
  - From which an object is made
  - Wood, stone, plant, cotton, iron, etc.

What is a material? (In a game engine)

- What our 3D models “look like”
  - Data associated with the surface of a mesh

- Entities should *appear* to be made of various materials

Material data

- Need to define surface properties to distinguish materials
  - What does it look like?
  - How does light interact with it?

- Here are some common material properties
  - **Tint** – a color applied to entire object
  - **Texture** – an image to apply to the surface
  - **Texture Scale** – how much to scale/repeat image on surface
  - **Shininess/roughness** – how precise are reflections?

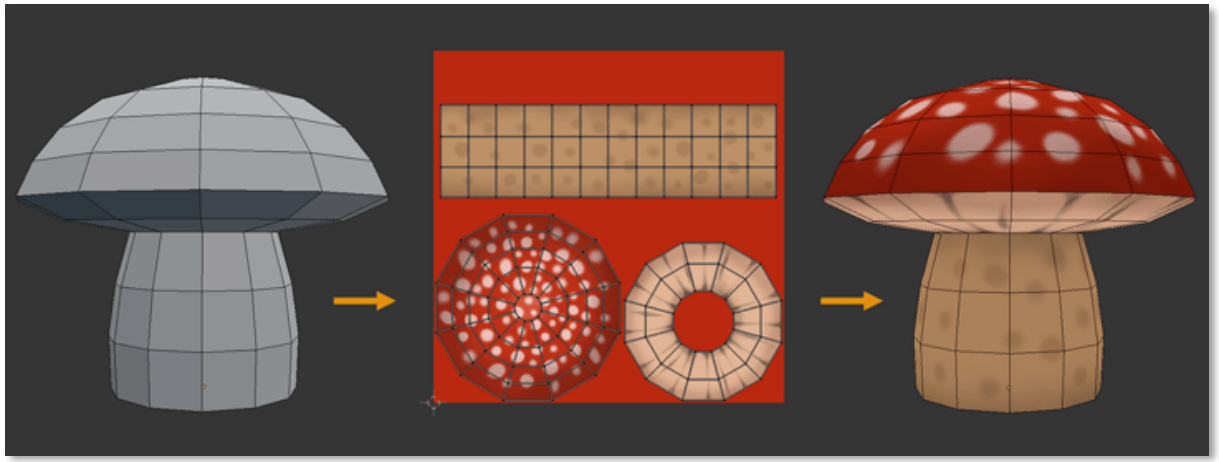
Material class

- We'll need a class to represent each unique material

- Then we can share materials among entities
  - GameEntity needs a material pointer
  - In addition to its mesh pointer

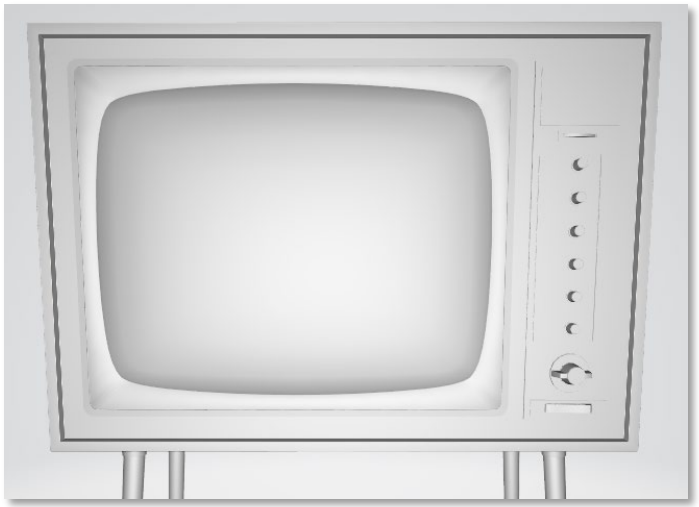
Materials & meshes

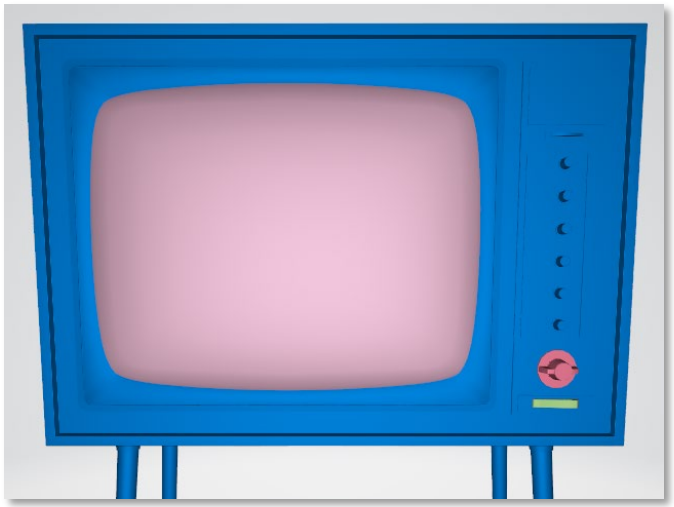
- A single mesh can have a single material associated with it



- However, some modeling programs allow multiple materials
  - Applied to different sections of a single model

Models with multiple materials?





- In a game engine, those are *multiple entities*
  - One entity = one set of triangles (mesh) + one material
- Each must be drawn separately!

Ready for 3D?

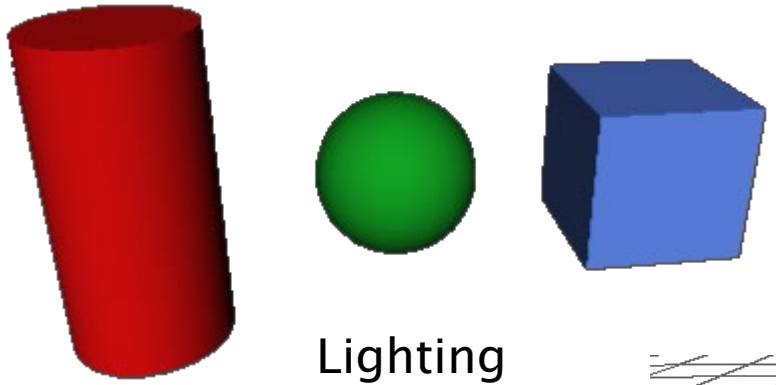
Ready for 3D?

- You've spent time creating or finding the perfect models
- You've got triangles loading from a file
- You go to draw them and…

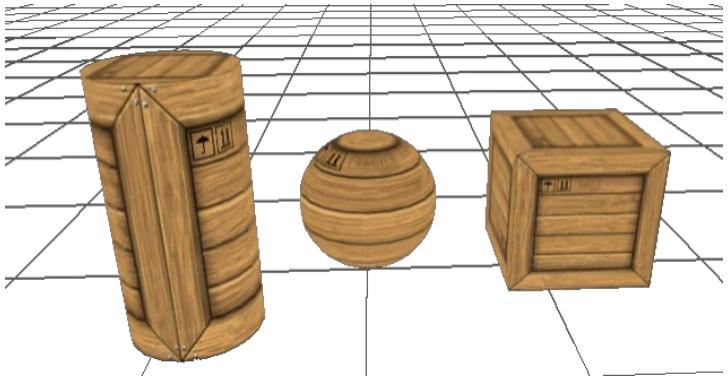
They don’t look very good

- That’s ok – next week is lighting, followed by texturing!

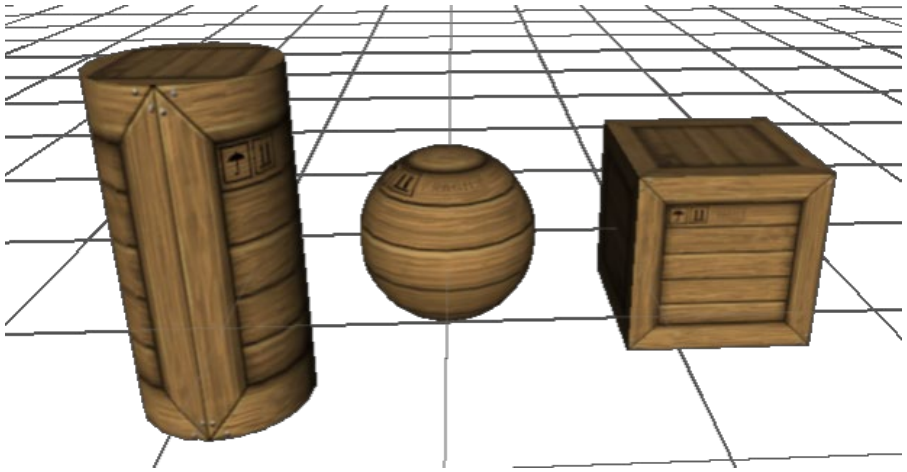
Previews of things to come



Lighting



Texturing



Lighting + Texturing