Computer Graphics

COMP3421/9415
2021 Term 3 Lecture 7

What did we learn last week?

3D Graphics!

- 2D to 3D (what we can reuse)
- 3D Objects
- Cameras
- Model/View/Projection Transform(s)

What are we covering today?

Expanding our knowledge of 3D Techniques

- Scene Graph - organising objects
- Depth Testing - What's in front of what?
- Blending - Rendering Transparency

The Scene Graph

Relativity

Deep Science (not really that deep)

- There is no absolute position of anything
- The best we can do is relate things to other things
  - CSE is at grid reference K17 in UNSW
  - UNSW is on Anzac Pde in Kensington, NSW
  - NSW is an Eastern State in Australia
  - Australia is at 25.2744° S, 133.7751° E on the Earth
  - The Earth is the 3rd planet orbiting the Sun
  - The Sun is in the Orion Arm of the Milky Way
  - etc etc etc

A hierarchy of relative positions

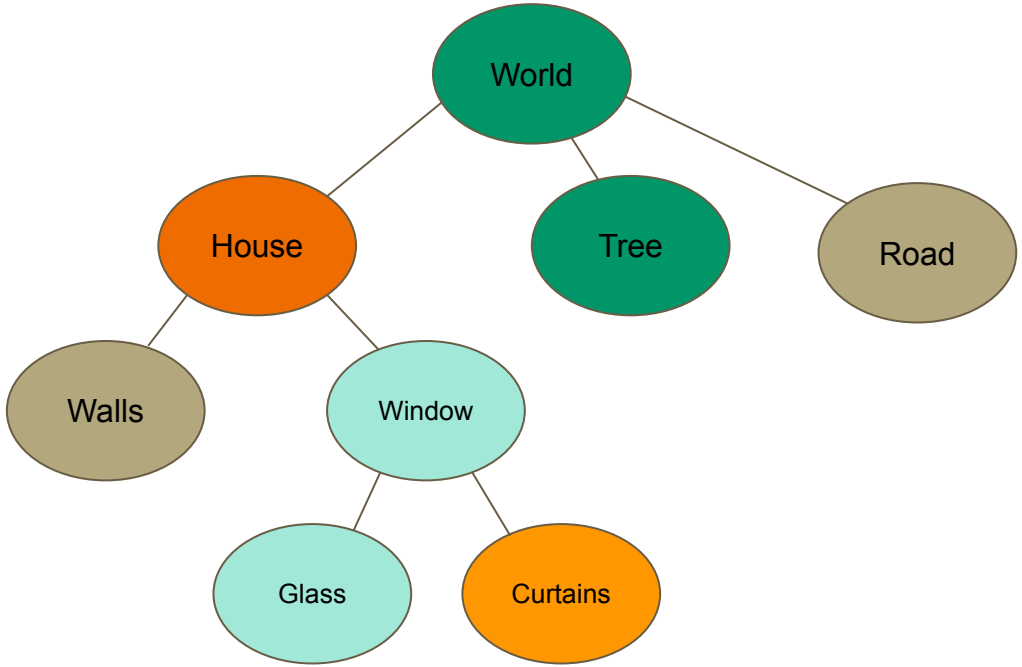
The previous example is actually very useful to us!

- A transform is just a relationship between two coordinate spaces
- Very much like an address of a building in a city
- We can use this to organise our 3D Scene!

A Scene Graph

Imagine a 3D scene with a simple rendering of a house

- The World might have the identity matrix transform
- Each node has a transform relative to its parent node
- If we change the House's location, all its children move with it!
- Generally only leaf nodes will have actual geometry



Why use Scene Graphs?

Simplifies locations of individual vertices

- Compose multiple hierarchical matrices for positions into a single transform
- Keep compound objects together
- Allow objects to be created as separate reusable pieces
- Allow objects to be attached or detached from other objects
- Simplify any movement or animations

Another Example

More Complex Scene Graphs can do interesting things . . .

Image credit: Square Enix

This isn't a character with a camera attached, but it's good to know you can attach objects that aren't geometry

Different Transforms in a Graph

A Character in a Graph

- We can specify certain sets of transforms that are poses
- These can be a specific local transform at each node
- We could then swap between different poses by choosing different transforms
    - Even moving the sword from the hand to the torso when it's sheathed
- If you moved between poses fast enough, you might even be able to do believable animation!
- We could also constrain transforms to limit certain joint movement etc

Code for Scene Graphs

These are handled before we get to OpenGL

- We will usually implement these ourselves
- You're all familiar with trees, right?
- Basic idea of node structures with pointers or references to children and parent nodes
- Each node will also contain a local transform

Depth Testing

Seeing things that are in front of other things

In the real world . . .

- This is something we already know
- Light does not pass through most objects
- So we only see objects that are closer to us than others

In Polygon Rendering

Vision passing through objects?

- There is no notion of light moving
- There is no idea of view being obscured by something else
- All we did was mathematically project objects into a view space
- We never checked if something was obscuring something else
- So in OpenGL, we need something more!

Options for Rendering Depth

Ordering our Triangles Back to Front

- Simple and potentially clean
- Render things at the back first
- Render things at the front afterwards
- Whatever's rendered later replaces the earlier objects
- Are we going to waste a lot of time sorting our triangles?

Time to sort?

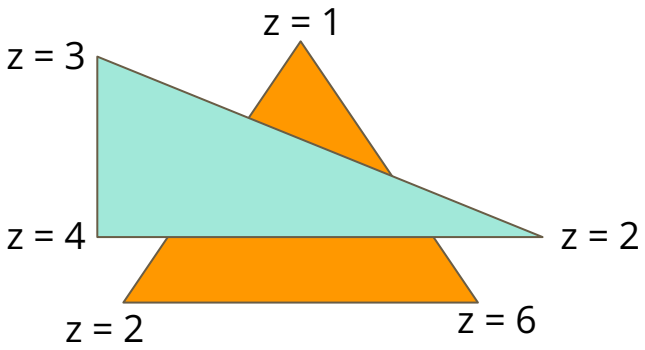
Let's do some simple analysis

- How much effort would it be to sort the triangles in a scene
  - Probably O(nlogn) and associated memory accesses
  - Modern AAA games might have 100,000 visible triangles in a single frame
- Then reorder the index buffer
- Then pass the new order of triangles to the vertex shader
- This feels time consuming . . .
- And we haven't even started rendering!

Complications

What if ordering triangles is logically impossible?

- Something is in front of something else if its Z coordinate is higher
- But a triangle has 3 Z coordinates!
- Two triangles can overlap so that they're both in front and behind each other
- Which of these two is in front?



We can't actually depth sort triangles

Using Depth at the Fragment level

- If we have triangles (or part of) in front of each other
- The rasterizer will create a fragment for each triangle
- This means more than one fragment per pixel!
- We just need to decide which fragment should be visible

Break Time

Art Styles in Games and Film

Image credit: Warner Bros

Image credit: Disney Pixar

- Realism vs Stylised
- In films
  - The Lego Movie: modelled entirely from bricks, scratches and fingerprints, depth of field
  - vs Pixar films: Fantastical world with exaggerated proportions and physics
- In games
  - Red Dead Redemption 2: Realistic lighting and surfacing, replicates real world effects
  - vs Genshin Impact: Anime style, consistent to its own world and fantastical

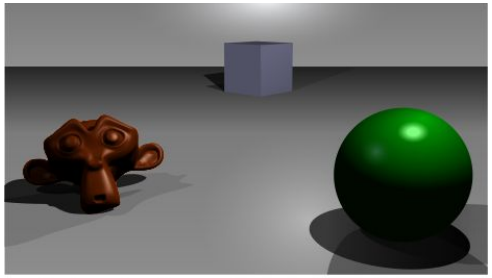
Image credit: Rockstar Games

Image credit: Mihoyo

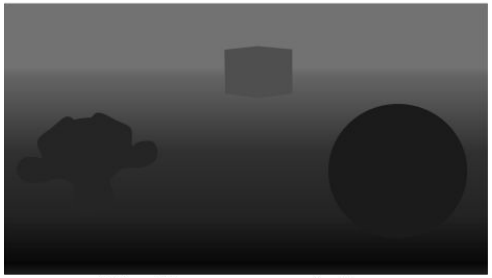
The Depth Buffer (also known as the Z-Buffer)

A screen sized buffer

- Like the frame buffer stores colours
- The Z-buffer stores depths
- Every fragment that has been processed will have its colour data and Z depth stored in these buffers



A simple three-dimensional scene



Z-buffer representation

Image credit: Wikipedia User -Zeus-

Checking the Z-buffer

If multiple fragments are trying to apply colour to a pixel

- The first fragment stores its colour and depth information in the buffers
- The next fragment tests its depth information against the Z-buffer
- If it's closer to the camera, its information replaces what's in the buffers
- If it's not closer, it is discarded

Z Fighting

This is not a Dragonball reference

- The Z buffer doesn't have perfect precision
- Fragments close to each other can "fight" for whichever is closest
- This ugly effect is called Z Fighting
- How do we fix this?
  - At an art level: Be careful how close we place objects
  - At a Z buffer level: Increase precision the closer (and more noticeable) objects are to us

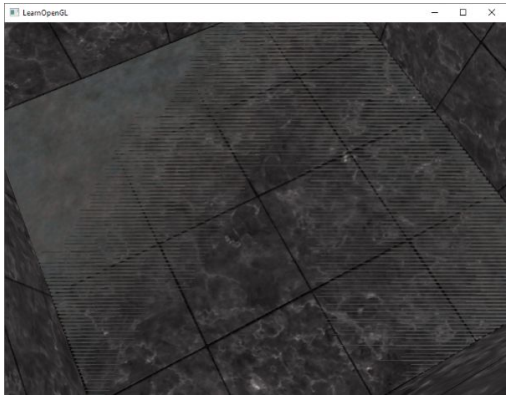


Image credit: learnopengl.com

In OpenGL

OpenGL handles this for us!

- We won't be implementing a depth buffer
- OpenGL will do this by default
- It is possible to enable GL_DEPTH_TEST to play around or write custom depth code

Blending

Transparency in OpenGL

Blending colours from different fragments

- If we have the visible fragment
- And the fragment behind it
- We can combine their colours to show transparency

Image credit: learnopengl.com

Alpha

The fourth component of our colour coordinates

- Red, Green, Blue, Alpha
- A measure of transparency
- 1.0 is opaque
- 0.0 is invisible

Discarding Fragments

Fragments below a visibility threshold won't appear

- We can discard a fragment in the frag shader
- This stops the fragment from rendering
- We use this in cases where we have a polygon like a rectangle . . .
- and it's only partially filled with visible material
- Eg: Grass and other foliage or sprites in 2D (also used in particle systems)
- Alpha is 1.0 for the grass, but 0.0 in the gaps

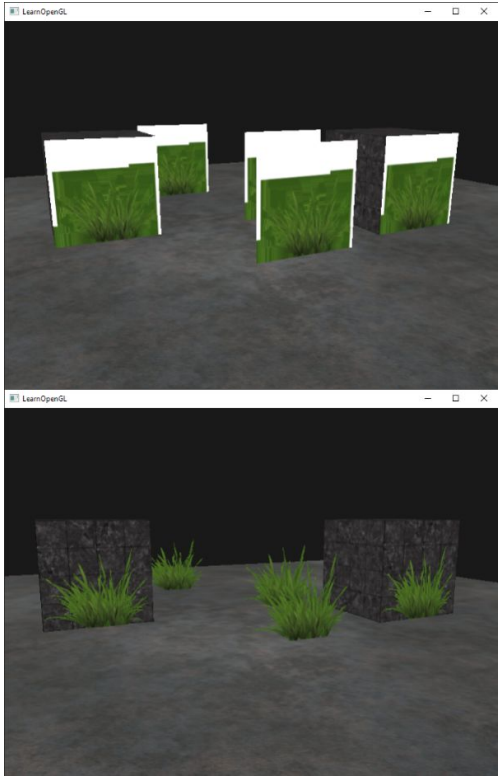


Image credit: learnopengl.com

Blending

If we can't discard a fragment, we'll need to blend

- Two fragments in one pixel
- If the one at the front has alpha < 1.0
- That means it's transparent and should show some of what's behind
- We can do this in a pretty simple way:
  - `front_colour * front_alpha + back_colour * (1.0 - front_alpha)`
  - There are also other options in OpenGL
- This mixes the two colours based on how transparent the front one is

Ordering issues?

We can blend, but do we know this works?

- What happens when the transparent object in front is rendered first?
- It won't have any information on what's behind it!
- Order is important
- All non-transparent objects need to be rendered first
- After this transparent objects can be rendered

Ordering Issues Fixed? (maybe not!)

Everything ok in this picture?

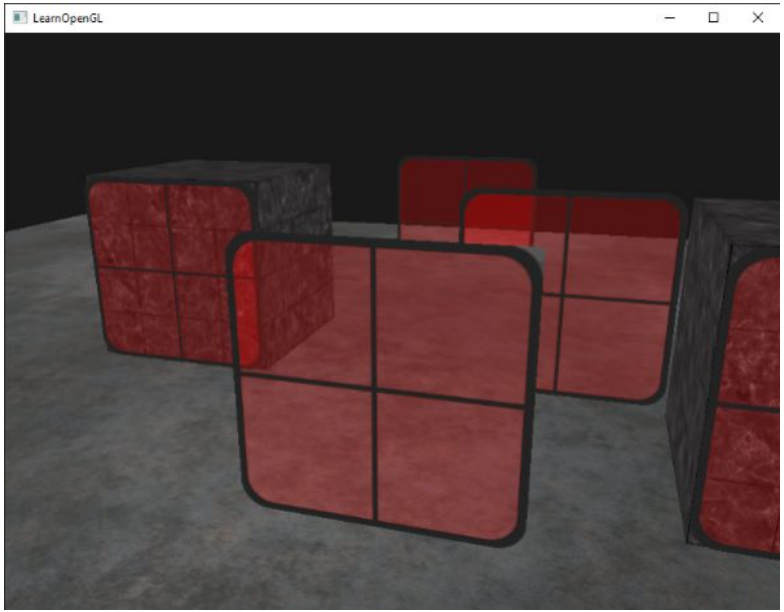


Image credit: learnopengl.com

Ordering issues

We need to sort the transparent objects from back to front

- Each transparent object will see the ones behind it before rendering
- Did we talk earlier about the cost of depth sorting?
- Yes, transparency is very costly!
- There are some tricks, but there's nothing easy

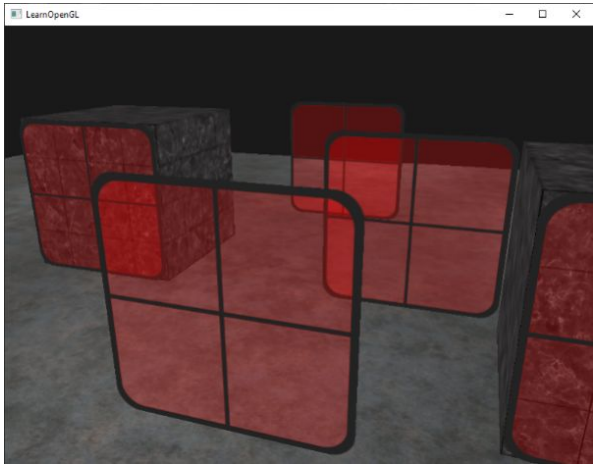


Image credit: learnopengl.com

Transparency in OpenGL

Blending in code

- OpenGL helps us out with blending
- We can enable GL_BLEND and give a reasonable blend function
- OpenGL does not help us with rendering order!
- If we have transparent objects, we must render them AFTER the rest of the scene
- We also have to render them in sorted order

What did we learn today?

3D gives us more, but also makes things more complicated

- Scene Graph
  - Organising many objects
  - Or organising a complex multi-part object
  - Hierarchy of transforms
- Depth Testing
- Blending and Transparency