Computer Graphics

COMP3421/9415

2021 Term 3 Lecture 18

What did we learn last lecture?

Shadow Mapping

- Rendering depth from the light's perspective
- Determine whether light can reach a particular fragment
- Also a lot of sampling issues and how to fix them

Deferred Rendering

- Lighting in post processing
- Rendering lights as volumes (geometry)
- Big efficiency benefits

What are we covering today?

Optimisation

- What are our goals?
- How has optimisation driven graphics?
- Culling non-visible parts of the scene

Wrapping up the course

- Where are we now?
- Where can we go?

Graphics Goals

Perfect Graphics?

Can we recreate reality?

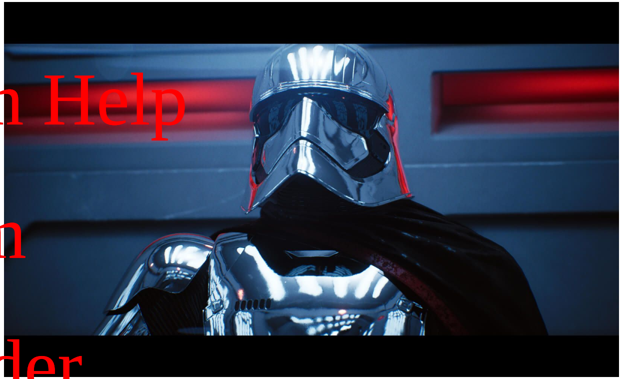
- A Turing Test for Graphics?
- If we believe, is that enough?
- Will we stop developing Graphics once we've tricked enough people?
- Example of Luke Skywalker Deepfake: https://youtu.be/wrHXA2cSpNU

Image credit: Lucasfilm/Disney

Ray Tracing

An idea that mimics physics in lighting

- Can it be the answer to our question?
- With enough rays, will we match reality?
- How close is it to "perfect"?
- How long does it take to get there?
- Nvidia realtime ray tracing demo: https://youtu.be/NgcYLIvlp_k



Real Time Ray Tracing
Image credit: Unreal Engine

Voxels

Shapes without Polygons

- Polygons are integral to our systems
- Everything we've taught has assumed polygon based shapes
- These are inherently unrealistic!
- Voxels attempt to bring pixelation to 3D spaces
- Nvidia Voxel Tech showcase: https://youtu.be/CnwVtuam-28

Image credit: Nvidia

Realtime

Can any of these techniques function in realtime?

- Human belief in persistency
- We hope for at least 60 frames per second (but will accept 24)
- Framerate dropping breaks us out of believing
- Realistic visual techniques must also maintain framerate
- A constant struggle for algorithm optimisation and hardware development

What have we learnt this term?

Everything is an approximation

- Tricks that take less time than simulation
- Angle calculations instead of real lighting
- Polygons instead of real surfaces
- What do we have to do because our hardware isn't capable?
- How much work goes into maintaining frame rate?

Optimisation

The Balancing Act

Maximum Visual Quality

- Ray Tracing Lighting
- Multiple Lights, different shaders per light/material
- Voxels or extremely high polygon count
- Transparency, Reflections
- High quality motion effects, animations etc

Frames per Second

- Blinn-Phong lighting approximation
- Deferred Rendering
- Low polygon count
- Simple effects or outright tricks so that humans don't expect effects!
- Intelligent removal of non-visible elements

Optimisations in this course

Optimisations hidden in techniques we taught

- Polygons
  - Low poly count approximates curves etc
  - Linear interpolation between vertices instead of real data

- Textures and Maps
  - Surface data instead of genuine geometry

- Depth Buffer
  - Approximation of depth instead of actually measuring visibility
- Key Framed Animation
  - Not genuine movement, interpolating between positions

Optimisations in this course

- Blinn-Phong Lighting
  - Ambient Lighting in Blinn-Phong is just a rough guess
  - Specular Lighting is also a rough estimate
- Lightmapping
  - Attempting to push work into pre-processing
- Reflections
  - Using rendering instead of real reflection calculations
- Shadow Mapping
  - Slightly inaccurate depth mapping instead of tracing real light paths
- Deferred Rendering
  - Careful removal of calculations for non-visible or irrelevant fragments

Realtime Graphics and Optimisation

It's more than Optimisation

- So many of our techniques have been designed specifically
- Efficiency first, quality second!
- We are ruled by the 1/60th of a second limitation
- and the specific optimisations of GPU hardware

Culling

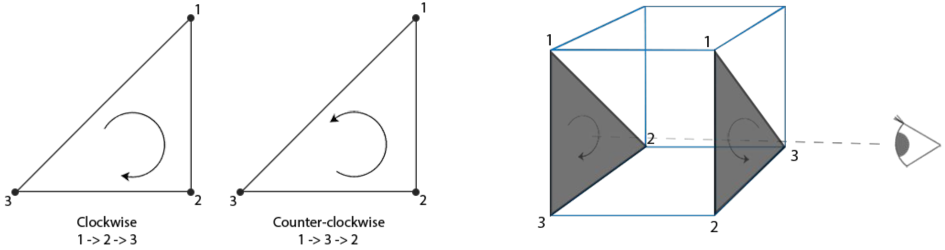
An optimisation we haven't covered!

- Removing polygons from our render path
- Any polygon that isn't visible should not waste processing power
- This is known as "culling"

Back Face Culling

Remember vertex winding?

- Anti clockwise is the front face
- Clockwise is the back of the polygon
- For solid objects, the back face shouldn't exist
- So we only render triangles that appear counter clockwise to the camera
- Removes roughly 50% of polygons from rendering

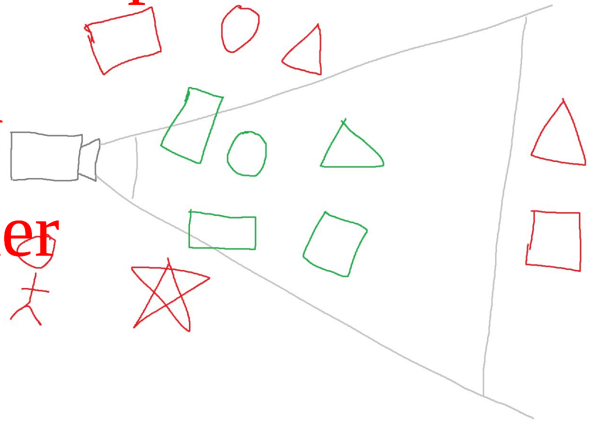


Images credit: learnopengl.com

Frustum Culling

We don't want to render what's outside the camera's view

- The frustum is conveniently made up of planes
- Easy to tell which side of a plane an object is
- Usually use "bounding boxes" for complex objects to simplify calculations
- If something is outside the frustum, we can cull it, making it not render
- Objects on the border can either be fully rendered or "clipped" into visible parts



Objects in red are culled, green are rendered

Break Time

What do you want to make?

- Sometimes it's about the feeling . . .
- Sometimes it's a game

- Sometimes it's CG effects for a short film
- Or this was just so that you could learn what was behind the games/films you love

- No matter what, this course will hopefully have given you a chance to learn something that you can take further . . . if you want.
- Never stop creating!

Where are we now?

What did we learn this term?

An introduction to Computer Graphics

- Approximate simulation of human vision
- Polygon Rendering as a basis
- Maths that supports it
- Blinn Phong Lighting (with maps)
- Graphics as a medium for art

What else did we learn?

The extras that make the graphics pop

- Visual effects like:
  - Reflections
  - Transparency
  - Shadows
  - Post Processing Effects
- An introduction, but not necessarily a full education
- Would take a lot more study to have full mastery over these
- For example, one could do an entire PhD on algorithms for shadowing

Where do we go from here?

Marc's List of Possibilities

Things we haven't covered (this was just off the top of my head):

- Anti Aliasing and Anisotropy
- Geometry Shader
- Particle Systems
- Tesselation
- Physically Based Rendering (PBR)
- Alternative lighting techniques (cel-shading, edge detection effects)
- HUDs and GUIs
- VR - Stereoscopy
- Rendering to non-flat monitors (curved frustums)
- Advanced Transparency
- Advanced Animation Techniques (Inverse Kinematics)
- Physics simulation for realistic animation
- Applying Machine Learning to Graphics Techniques

Any% Speed Run

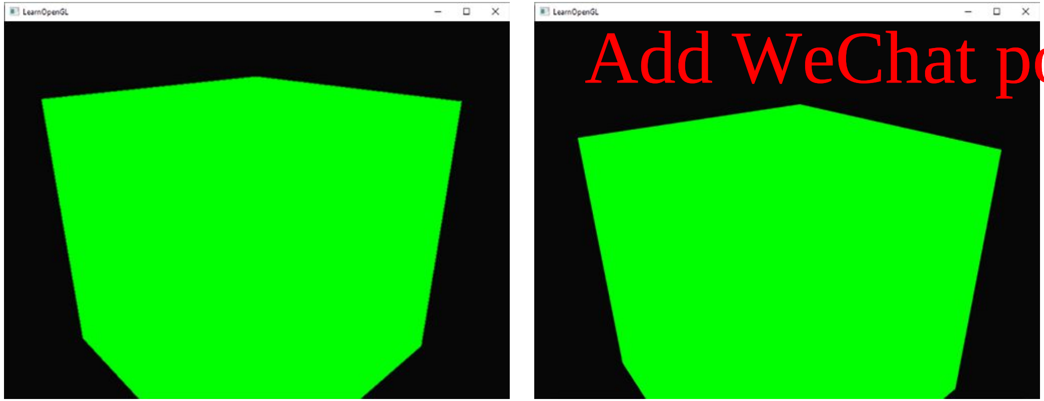
What's next in Graphics?

- Let's try to cover some of the things in Marc's list
- Very quickly!

Anti Aliasing and Anisotropy

Eliminating the "jaggies"

- Jagged edges where diagonal lines are made into pixels (aliasing)
- or awkward sampling of a texture on an angle to the view (anisotropy)
- Both are generally corrected using multi-sampling techniques



Images credit: learnopengl.com

Image credit: Wikipedia users Lampak and THOMAS

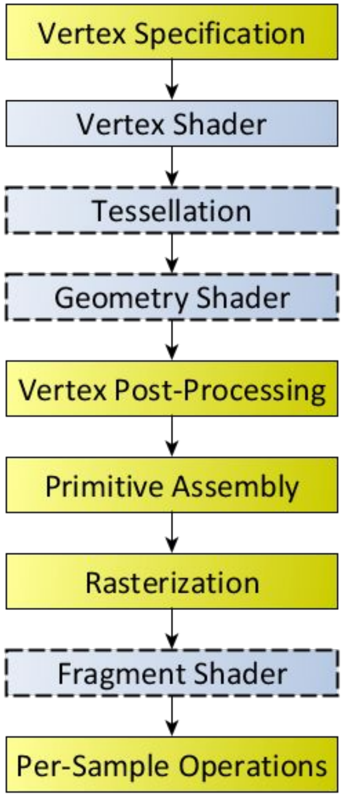
Geometry Shader and Particle Systems

Between the Vertex and Fragment Shaders

- Draws Geometry in the shader
- We can specify vertices, the shader can add extra verts and make shapes

Particle Systems

- Visual effect for things like smoke, fire and other volumetric substances
- Usually made up of hundreds or thousands of rectangles rotated to aim at the camera (billboarding)
- These rectangles can be created in the geometry shader or can be reused geometry



The OpenGL 4.x Shader pipeline
Image credit: Khronos

Tesselation

Adding Geometry data to objects

- A shader that works alongside the vertex shader
- Able to subdivide triangles and create new vertices
- Often used to add detail to curved surfaces
- Also useful for terrain systems

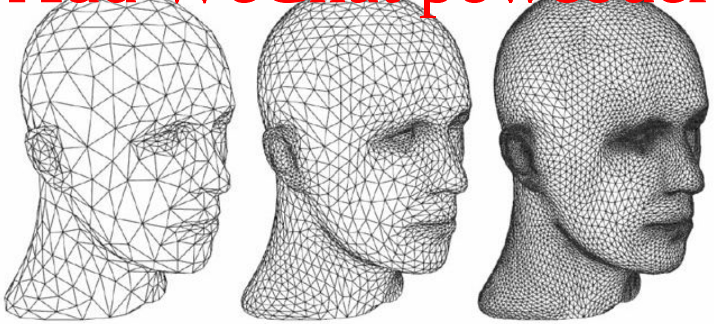


Image credit: Nvidia

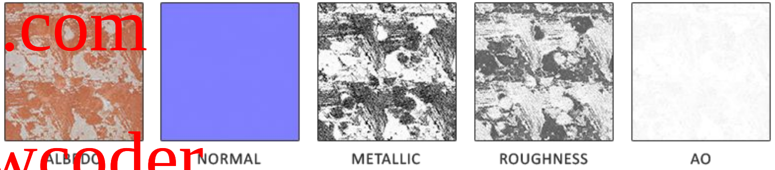
Physically Based Rendering

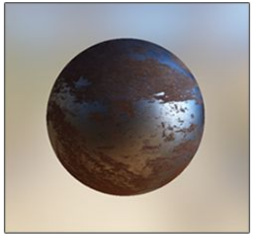
Realism in surface details

- An attempt to render complicated surfaces more realistically

- Uses multiple buffers
- Originally used for metallics

- Surface microfacets (roughness)
- Reflectance and Radiance (techniques for how light reflects)
- Other ideas like fresnel and subsurface scatter





Images credit: learnopengl.com

Stylistic Rendering

Cel Shading, Edge detection etc

- Modification of lighting algorithm to an "on-off" or two tone scheme
- Use of post processing kernels to detect edges of objects and colour them black
- Classic anime or comic feel
- Not limited to a hand drawn feel!
- Many interesting effects are possible!

Image credit: Mihoyo

HUDs and GUIs

2D overlays over the final frame

- The tech is easy! Blend a texture with the final framebuffer

- Difficulty is in design
- Useful information with minimal distraction

In-game GUI elements

- Overlays, GPS paths, 3D highlighting and info
- A lot of interesting possibilities for making UI exist in the game itself!

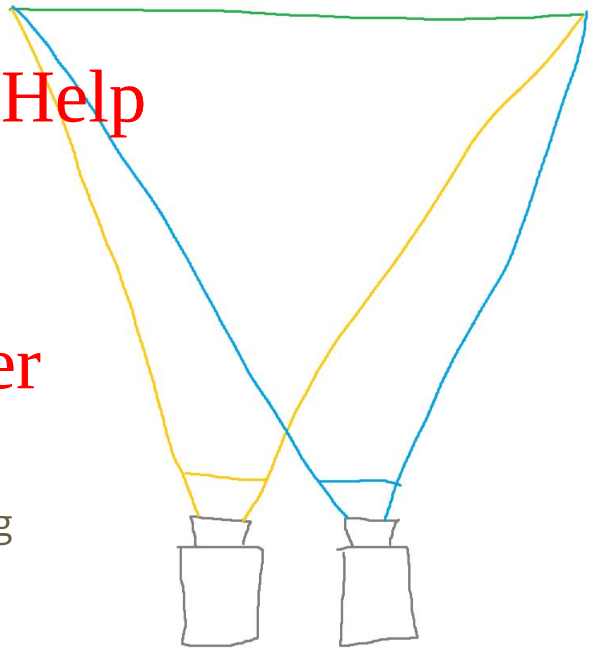
Dead Space (EA 2008)
Image credit: Dino Ignacio (UI Designer)

VR - Stereoscopy

It's all already in 3D right?

- Two eyes mean two screens, possibly on one framebuffer

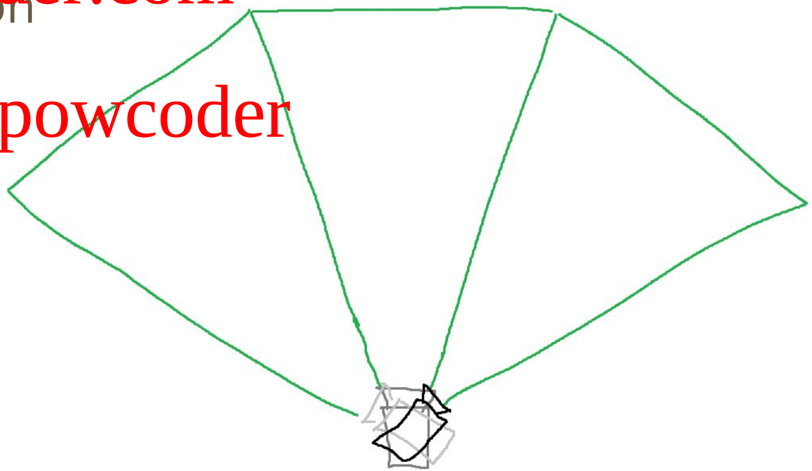
- Simple post processing:
    - Render two different cameras, offset from the centre
    - Render textures write to two different sides of the final framebuffer (or two separate framebuffers depending on your tech)
    - Uses asymmetric frustums to avoid near/far plane clipping issues
    - Some warping of final image to match custom lenses in headsets



Rendering to Curved Monitors

Also used for curved projection screens etc

- Multiple virtual cameras
- Approximate the curve with multiple renders
- Perspective shift issues in the transition between cameras



Advanced Transparency

Order Independent Transparency

- We learnt that transparent objects need to be rendered last (and sorted)
- Our entire pipeline is awkward for transparency

- Some techniques include:
    - Rendering to a 3D framebuffer then blending together afterwards
    - Hardware optimised sorting of depth of objects
    - Depth Peeling (using multiple Z buffers to be able to render at different depths without necessarily discarding objects behind)

Advanced Animation

Inverse Kinematics

- What if our animations aren't pre-baked with keyframes
- But they're reliant on geometry in the scene
- Pressing buttons, opening doors, picking up objects, walking on stairs
- (Lucky us, Robotics research can advance this field for us)
- Inverse Kinematics is:
    - The hand goes here, what do all the joints back to the shoulder need to do?
- Potentially very complex mathematical solutions, compounded by the number of joints

Physics Simulation

Animation based on physical rules

- Particularly liquids
- Also fluid movement for cloth and hair
- Useful for realtime destruction of objects
- Attempt to have realistic simulation of gravity and collisions
- As well as wind and tensile force
- Similar to lighting
    - Very hard to accurately simulate
    - Most techniques are fast approximations

Machine Learning for Graphics

What can we do with the new Deep Learning hotness?

- Realtime application of art styles
- Human-like animation based on learnt movement patterns
- Ray tracing needing less total rays while AI predicts likely colours in gaps
- Deep Learning Super Sampling (DLSS, the new hype word)
    - Applies ray tracing on a lower resolution frame
    - Uses a trained deep learning algorithm to super sample to a higher resolution output
    - Should result in a high quality output while only needing to process a much lower number of pixels

What did we learn this term?

Computer Graphics

- We started with the idea of tricking human vision
- We built up a lot of technology to do this!
- From primitives to multiple renders and post processing
- We learnt how digital art and technology drives our algorithms
- And we developed those algorithms into realtime implementations

Thanks for coming along on the journey!