Computer Graphics
The Vertex and Fragment Shader

Raimond Tunnel
The Standard Graphics Pipeline

- **Vertex Transformations**
  - Construct geometry
  - Define transformations
  - Assign material properties

- **Culling & Clipping**
  - Determine front-facing triangles
  - Determine which vertices are visible

- **Rasterization**
  - Fill the triangle with fragments

- **Fragment Shading**
  - Calculate correct color values

- **Visibility Tests**
  - Is the fragment visible?

- **Blending**
  - Blend together multiple fragments

**Vertex Shader**
Object's local space → viewport space

**Fragment Shader**
Calculate correct color values
WebGL

• Based on OpenGL ES 2.0
• Used in the browsers for accessing the pipeline
• Has a shader programming language GLSL

• In newer OpenGL the syntax is different, but the ideas are the same...
Shaders

- First we will have a triangle
  - All meshes are made up of triangles
- Triangle will have 3 vertices
Shaders

- **The Vertex Shader** will be ran on the 3 vertices
- **Purpose:** transform positions from *local space* to clip space (and later *screen space*)
• Rasterization will create fragments (pixels)
• On those the **Fragment Shader** will be ran
• **Purpose:** color the pixels
• **Uniform variables** – global values accessible from all shaders
• **Attribute variables** – values associated with each vertex
- **Varying variables** – values assigned in the vertex shader and **interpolated to fragments**

![Diagram showing vertex and fragment shaders with varying and uniform variables](image-url)
Three.js

• **JavaScript** library on top of **WebGL**
• Makes life **easier**
  • OOP
  • Encapsulates lower level WebGL stuff
  • Provides out of the box working graphics algorithms

https://threejs.org/
Task 1 – Coloring a Sphere

- Download the base files
- Open: 1 – *Coloring a Sphere.html*
  - In Notepad++
  - In SublimeText
  - In your favourite code editor
- Let's **look at the code**...
Lambertian Reflectance Model

- We assume that our material reflects light equally in all directions
- The material is an *ideal matte*
Lambertian Reflectance Model

- In which case the surface point emits more light?
Lambertian Reflectance Model

- With simple trigonometry it is easy to see that the light reaching one surface unit is proportional to the cosine between the surface normal and the vector towards the light source.
Lambertian Reflectance Model

- Greater the angle, less light reaches one point

\[ \cos(10^\circ) = 0.98 \]

\[ \cos(70^\circ) = 0.34 \]
Lambertian Reflectance Model

- Oh my...
Lambertian Reflectance Model

- When the cosine is negative, we make it 0.
- The **dot product** (*skalaarkorrutis*):
  \[ v \cdot u = |v| |u| \cdot \cos(\text{angle}(u, v)) \]
  Geometric definition
  \[ v \cdot u = v_1 u_1 + v_2 u_2 + v_3 u_3 \]
  Algebraic definition
- When the vectors are **normalized**, we get:
  \[ v \cdot u = v_1 u_1 + v_2 u_2 + v_3 u_3 = \cos(\text{angle}(u, v)) \]
Lambertian Reflectance Model

- Intensity of the reflected light also depends on:
  - The intensity of the light source
  - The reflectivity of the material
- In computer graphics we store the intensities of the red, green and blue channel separately.

\[ I_{RGB} = L_{RGB} \cdot M_{RGB} \cdot \max(0, vectorTowardsLight \cdot normal) \]
Let's make it!

What happens when you have errors?
Ambient Light

• In reality the light does not only come from the light source

• Light bounces around and comes from all directions – that light is called ambient light

• Simplest way to take that into account is to just add a small value to the model

\[ I = L_A \cdot M_A + L_L \cdot M_L \cdot \max(0, \text{vectorTowardsLight} \cdot \text{normal}) \]

• Often the ambient material property is the same

\[ I = M \cdot (L_A + L_L \cdot \max(0, \text{vectorTowardsLight} \cdot \text{normal})) \]
Add ambient light to the model
Toon Shading

- Aka cel shading
- Toon shading **discretizes** the colors

- At times an outline of objects is also drawn, but that is a bit more complicated to do...
Toon Shading

- Open: 2 – *Discrete Sphere.html*
- Follow the instructions in the fragment shader to discretize the colors
- Feel free to experiment yourself...
Wobbly Sphere

- Open: 3 – *Wobbly Sphere.html*
- Follow the instructions in the vertex shader and on the CPU side
- Make the vertices of the sphere move
- Feel free to go wild...
Texturing

- Texturing is mapping a 2D image to a 3D surface
- This is done by specifying 2D texture coordinates (called **UV coordinates**) for each vertex
- The mapping done in a 3D modelling software
Texturing

- Open: 4 – *Hut.html*
- As you can guess, if we **interpolate the UV coordinates**, the corresponding fragments will get the correct interpolated UV coordinate.
- If we **sample the base color** from those coordinates, the object will be textured.
Thanks!

- You now have good shader programming skills!