COMP220: Graphics & Simulation

2: Getting Started with OpenGL

Learning outcomes

By the end of this week, you should be able to:

- **Implement** a basic scene using OpenGL and SDL
- **Recall** the role of shaders in graphics programming
- **Write** simple shader programs in GLSL

Agenda

Agenda

- Lecture (async):
  - **Introduce** the basic concepts of OpenGL
  - **Review** the roles of the vertex shader and the fragment shader

Agenda

- Lecture (async):
  - **Introduce** the basic concepts of OpenGL
  - **Review** the roles of the vertex shader and the fragment shader
- Workshop (sync):
  - **Implement** a basic scene using OpenGL and SDL
  - **Practice** creating simple shaders in GLSL

FALMOUTH
UNIVERSITY

Introducing OpenGL

What is OpenGL?

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.
- First released in 1992; current version: 4.6.

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.
- First released in 1992; current version: 4.6.
- Major design changes in version 3.3/4.0 (2010):

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.
- First released in 1992; current version: 4.6.
- Major design changes in version 3.3/4.0 (2010):
  - Previously used **fixed function pipeline** with functionality hidden/abstracted: easy to use but inefficient.

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.
- First released in 1992; current version: 4.6.
- Major design changes in version 3.3/4.0 (2010):
  - Previously used **fixed function pipeline** with functionality hidden/abstracted: easy to use but inefficient.
  - Changed to **core-profile** mode: flexible and powerful, but more complex to learn.

What is OpenGL?

- A **cross-language, cross-platform** API **specification** for rendering 2D and 3D vector graphics.
- First released in 1992; current version: 4.6.
- Major design changes in version 3.3/4.0 (2010):
  - Previously used **fixed function pipeline** with functionality hidden/abstracted: easy to use but inefficient.
  - Changed to **core-profile** mode: flexible and powerful, but more complex to learn.
  - Requires a greater understanding of what's actually happening.

OpenGL as a specification

OpenGL as a specification

- Defines what the output of each function should be.

OpenGL as a specification

- Defines what the output of each function should be.
- Libraries are (usually) implemented by the graphics card manufacturers:

OpenGL as a specification

- Defines what the output of each function should be.
- Libraries are (usually) implemented by the graphics card manufacturers:
  - **Core OpenGL (GL)**: functions prefixed with `gl` to model an object view geometric primitives (point, line, polygon).

OpenGL as a specification

- Defines what the output of each function should be.
- Libraries are (usually) implemented by the graphics card manufacturers:
  - **Core OpenGL (GL)**: functions prefixed with `gl` to model an object view geometric primitives (point, line, polygon).
  - **OpenGL Utility Library (GLU)**: functions prefixed with `glu` to extend the core library.

OpenGL as a specification

- Defines what the output of each function should be.
- Libraries are (usually) implemented by the graphics card manufacturers:
  - **Core OpenGL (GL)**: functions prefixed with `gl` to model an object view geometric primitives (point, line, polygon).
  - **OpenGL Utility Library (GLU)**: functions prefixed with `glu` to extend the core library.
  - **OpenGL Utilities Toolkit (GLUT)**: functions prefixed with `glut` to interact with the operating system (e.g. handling input).

OpenGL as a specification

- Defines what the output of each function should be.
- Libraries are (usually) implemented by the graphics card manufacturers:
  - **Core OpenGL (GL)**: functions prefixed with `gl` to model an object view geometric primitives (point, line, polygon).
  - **OpenGL Utility Library (GLU)**: functions prefixed with `glu` to extend the core library.
  - **OpenGL Utilities Toolkit (GLUT)**: functions prefixed with `glut` to interact with the operating system (e.g. handling input).
  - **OpenGL Extension Wrangler Library (GLEW)**: cross-platform C/C++ extension loading library.

OpenGL as a state machine

OpenGL as a state machine

- OpenGL itself **does not** retain information about objects.

OpenGL as a state machine

- OpenGL itself **does not** retain information about objects.
- Uses the **context**, which stores a collection of variables to define how to render triangles.

OpenGL as a state machine

- OpenGL itself **does not** retain information about objects.
- Uses the **context**, which stores a collection of variables to define how to render triangles.
- To change a portion of the image by default requires **clearing** and **redrawing** the whole screen.

OpenGL as a state machine

- OpenGL itself **does not** retain information about objects.
- Uses the **context**, which stores a collection of variables to define how to render triangles.
- To change a portion of the image by default requires **clearing** and **redrawing** the whole screen.
- Change the state by **setting options** (e.g. draw mode) and **manipulating buffers**.

SDL and OpenGL

SDL and OpenGL

- OpenGL only handles rendering of graphics.

SDL and OpenGL

- OpenGL only handles rendering of graphics.
- We need something else to handle windows, events, audio etc.

SDL and OpenGL

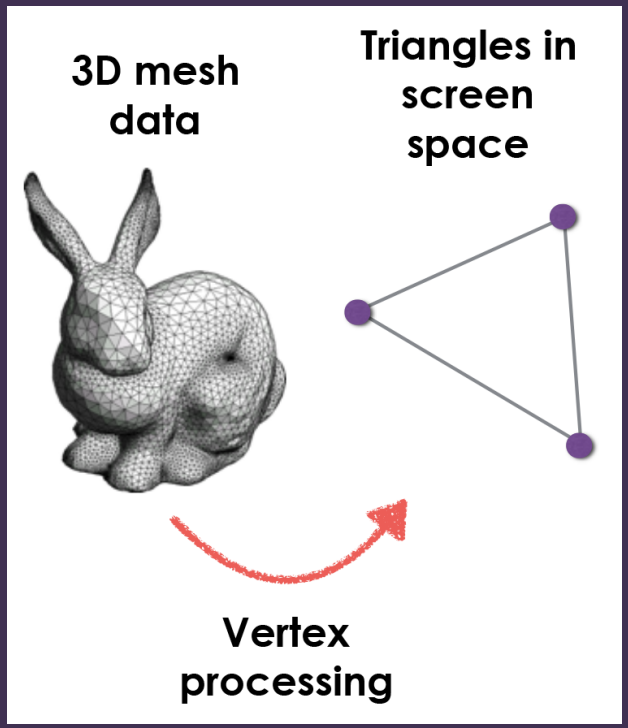
- OpenGL only handles rendering of graphics.
- We need something else to handle windows, events, audio etc.
- We will use **SDL** (which you have used before, including in COMP270).

FALMOUTH
UNIVERSITY

Shaders

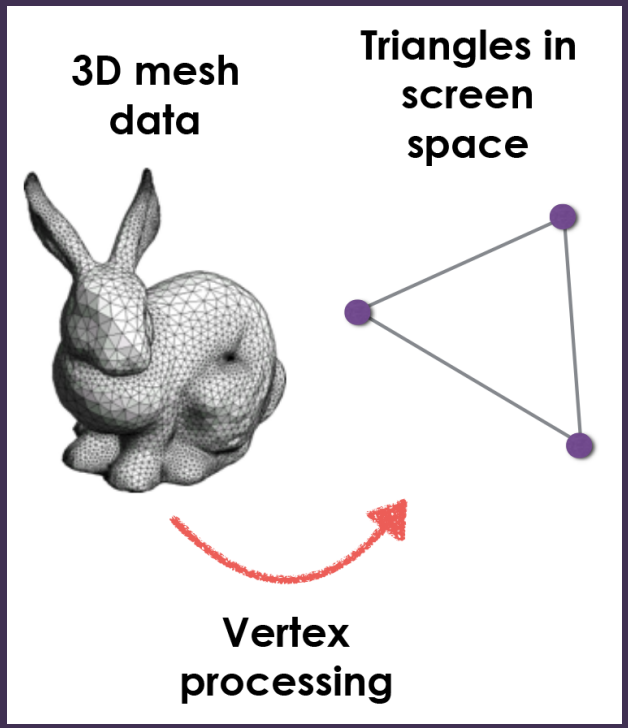
Recap: processing pixels

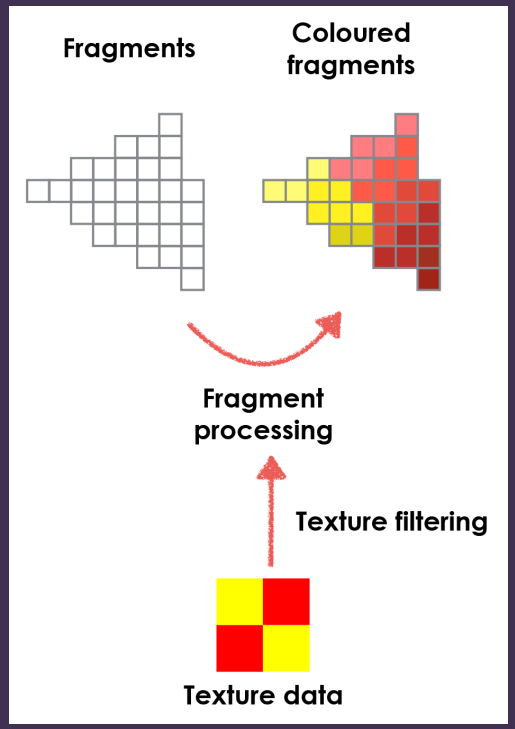
Recap: processing pixels



Vertex processor **transforms** vertices and **projects** them into 2D screen space.

Recap: processing pixels





Vertex processor **transforms** vertices and **projects** them into 2D screen space.

Fragment processor determines the **colour** of each fragment covered by the triangle.

Vertex and fragment shaders

Vertex and fragment shaders

- The vertex processor and fragment processor are **programmable**

Vertex and fragment shaders

- The vertex processor and fragment processor are **programmable**
- Programs for these units are called **shaders**

Vertex and fragment shaders

- The vertex processor and fragment processor are **programmable**
- Programs for these units are called **shaders**
- **Vertex shader**: responsible for geometric transformations, deformations, and projection

Vertex and fragment shaders

- The vertex processor and fragment processor are **programmable**
- Programs for these units are called **shaders**
- **Vertex shader**: responsible for geometric transformations, deformations, and projection
- **Fragment shader**: responsible for the visual appearance of the surface

Vertex and fragment shaders

- The vertex processor and fragment processor are **programmable**
- Programs for these units are called **shaders**
- **Vertex shader**: responsible for geometric transformations, deformations, and projection
- **Fragment shader**: responsible for the visual appearance of the surface
- Vertex shader and fragment shader are separate programs, but the vertex shader can pass arbitrary values through to the fragment shader

Interpolation

Interpolation

- The vertex shader sets a value for each vertex

Interpolation

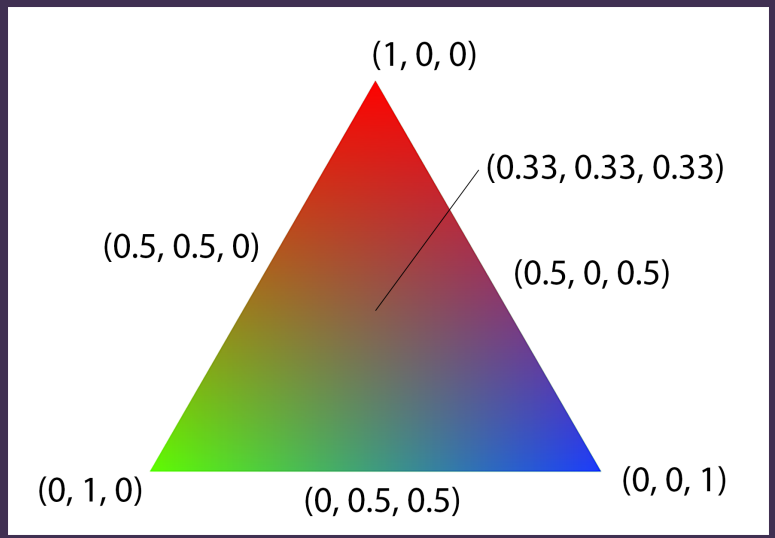
- The vertex shader sets a value for each vertex
- So what is the value in the middle of the triangle?

Interpolation

- The vertex shader sets a value for each vertex
- So what is the value in the middle of the triangle?
- The GPU **interpolates** the value across the triangle

Interpolation

- The vertex shader sets a value for each vertex
- So what is the value in the middle of the triangle?
- The GPU **interpolates** the value across the triangle



Next steps

- **Review** the additional asynchronous material for more background on OpenGL and shaders/GLSL
- **Attend** the workshop to see how to use OpenGL with SDL to set up a scene and render it with shaders in GLSL