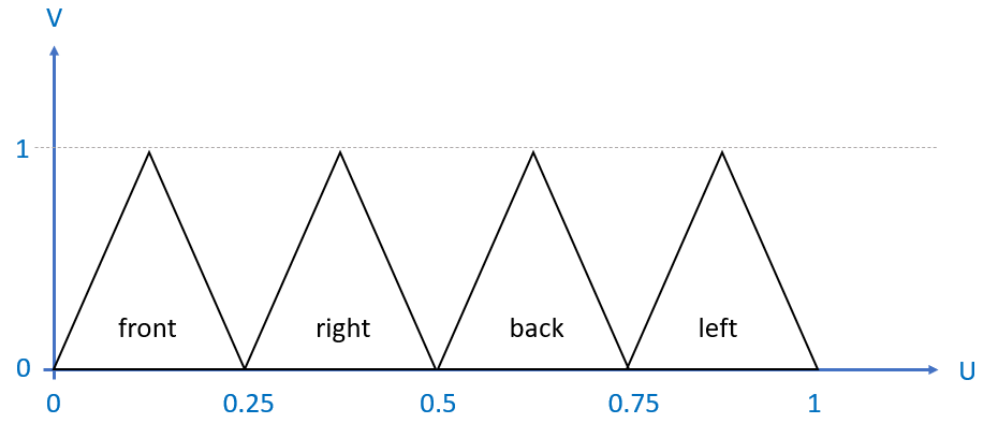
Assignment 3A – Deadline 19.12.2023 23:59

Nehmen Sie aus der 3ten Übung „intro_done“ als Ausgangspunkt – es handelt sich um das Ergebnis aus der OpenGL-Einführung, mit einem Unterschied: Anstatt die Shader im Code zu definieren, werden sie mithilfe einer Wrapper-Klasse aus Files geladen. Sie finden die Shader im Ordner „shaders“ und können Sie ganz normal, wie Textfiles bearbeiten.

Sie sollen jetzt der Pyramide ein weiteres Vertex-Attribut hinzufügen: UVs. Das trifft sich gut, weil wir uns vor Weihnachten noch Texture Mapping ansehen werden.

UVs sind im Wesentlichen ein frei wählbares Vertex Attribut – das soll heißen, es gibt keine richtige Art, UVs zu definieren; Man muss sich immer überlegen, was man für eine Texture hat, und wie man sie aufs Objekt mappen will. Wir möchten die Pyramide wie folgt UV-mappen:



Das stellt Sie nun vor folgendes Problem: In der ursprünglichen Definition der Pyramide sind 5 Vertices, 1x die Pyramidenspitze und 4 Vertices an der Grundfläche. In der Grafik sind aber eindeutig 9 unterschiedliche UV-Koordinaten, u.A. deshalb, weil die Pyramidenspitze mit 4 unterschiedlichen UVs aufscheint; Der Vertex vorne links unten wird ebenfalls doppelt belegt.

Es gibt in diesem Fall keinen Weg, sich das Duplizieren von Daten zu ersparen: wenn sie einen Vertex mit einer eindeutigen Position/Color, aber mit 4 unterschiedlichen UVs haben wollen, dann müssen Sie das wie 4 unterschiedliche Vertices behandeln, und folglich 4-mal in den Vertex-Daten ablegen. (Anders ausgedrückt, 2 Vertices sind nur dann gleich, wenn wirklich **alle** Attribute gleich sind.)

Bauen Sie die Vertex- und Indexdaten entsprechend um. Sie müssen dafür nur das `struct VertexData`, `std::vector<VertexData> vertexData` und `std::vector<unsigned> indexData` abändern, der OpenGL Code kommt mit der Änderung der Daten klar, weil wir überall mit `std::vector::size()`, `sizeof` und `offsetof` gearbeitet haben. Nach der Änderung sollte das Programm immer noch ausführbar sein, und die Pyramide immer noch aussehen wie vorher – die UVs sind jetzt zwar in den Vertex-Daten gespeichert, werden aber nicht verwendet.

Fügen Sie in den Vertex Shader nun ein neues `vec2` Attribut für die UVs ein, und reichen Sie die UVs von dort an den Fragment Shader weiter (im Wesentlichen wie die color). Der Fragment Shader soll die UVs als Farbwert ausgeben (Blau Kanal = 0 → `fragmentColor = vec4(uv.x, uv.y, 0, 1);`). Passen Sie dann die Konfiguration des Vertex Array Objects entsprechend an – mit `glVertexAttribPointer` und `glEnableVertexAttribArray`.