# Aufgabe 1: Dreiecksnetz zeichnen

1. Die Klasse TriangleMeshGL repräsentiert ein Dreiecksnetz bestehend aus einem Element Array Buffer und einer Menge von Attributen pro Vertex, wie z.B. Position und Farben. Implementieren Sie den Konstruktor constructor(gl, simpleMeshIO). Dieser erzeugt ein
   * 1. Vertex Array Object,
     2. einen Vertex Buffer für die Positionen,
     3. einen Vertex Buffer für die Farben (falls vorhanden)
     4. und einen Element Array Buffer.

01Mesh2D.js lädt bereits einen Datensatz. Sie können also davon ausgehen, dass in simpleMeshIO alle nötigen Daten vorhanden sind. Lesen Sie dazu den constructor von SimpleMeshModelIO.

* + Belegen Sie nun die Member-Variablen vao und nTriangleIndices mit geeigneten Werten!

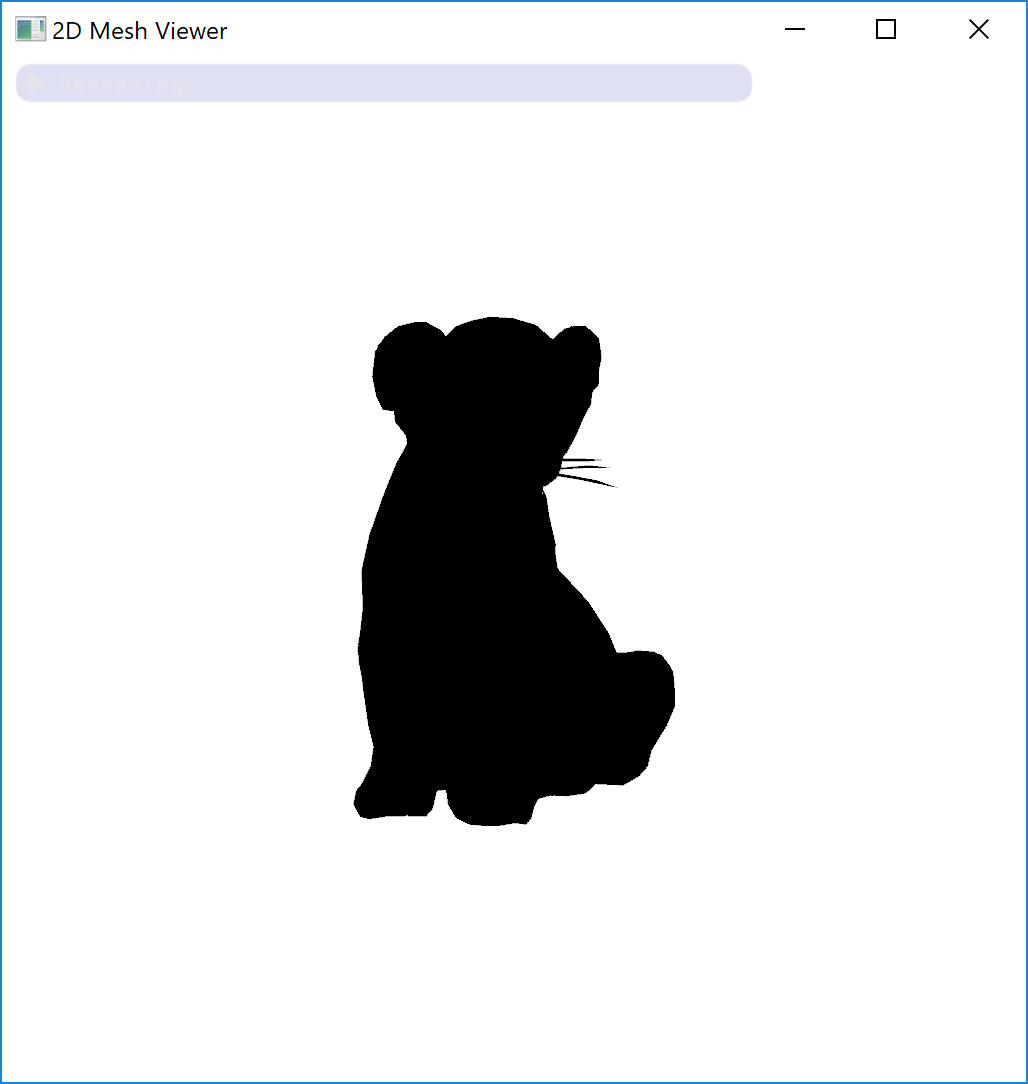


Abbildung 1 Ergebnis von 1b

Verwenden Sie als Binding Positions der Vertex Buffers innerhalb des Vertex Array Objects die Werte von positionAttributeLocation und colorAttributeLocation.

1. 01Mesh2D.js lädt einen Datensatz in mesh und erzeugt daraus ein triangleMesh. Erweitern Sie die Klassen nun so, dass das Dreiecksnetz gezeichnet wird (vgl. Abbildung 1).
2. In dem UI Widget kann der Benutzer eine Hintergrundfarbe wählen. Diese wird jedoch im Moment noch ignoriert. Ändern Sie deshalb die Hintergrundfarbe, so dass diese mit der Eingabe des Benutzers übereinstimmt!

# Aufgabe 2: Farbiges Dreiecksnetzzeichnen

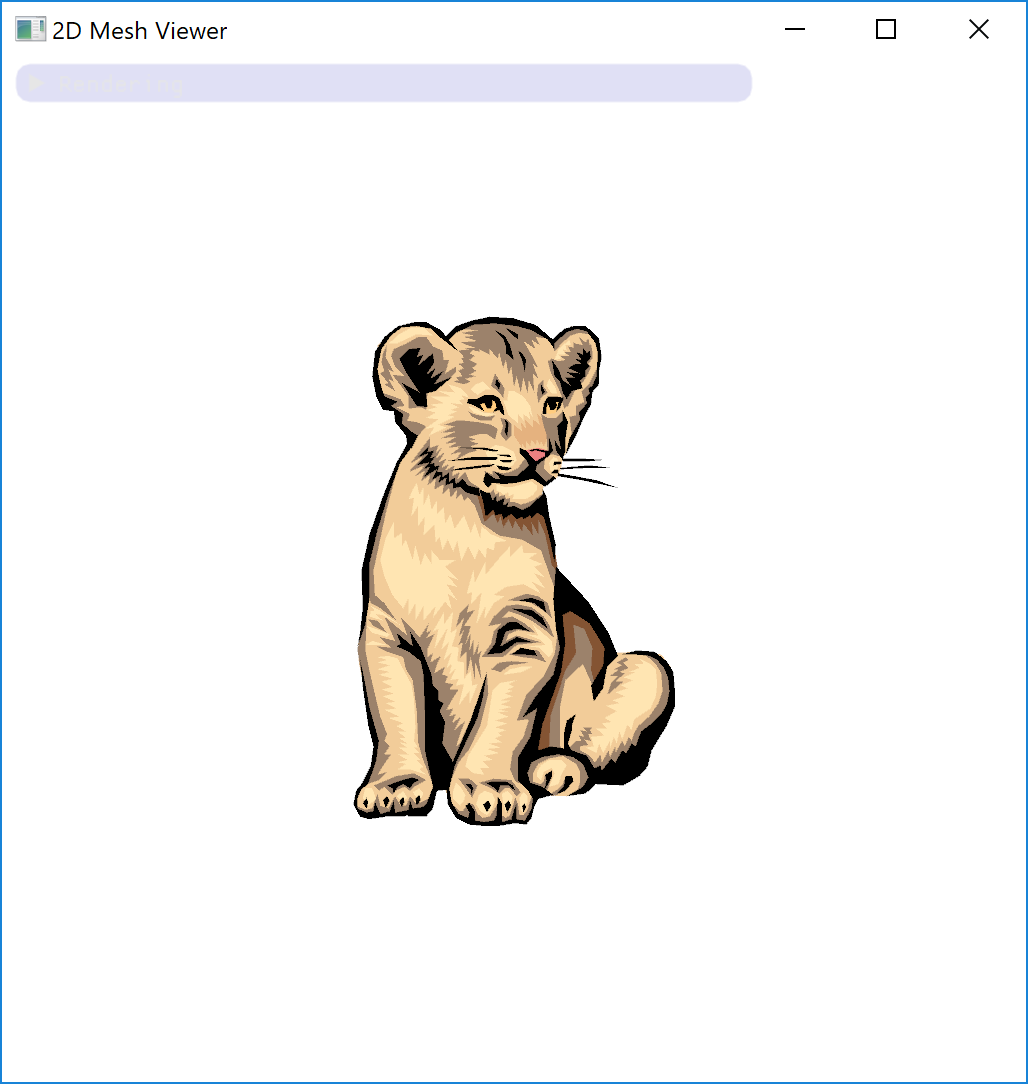


Abbildung 2 Ergebnis von 0

[3 P] Erweitern Sie die Shader und 01Mesh2D.js so, dass das Netz farbig gezeichnet wird (vgl. Abbildung 2)

# Aufgabe 3: Transformationen

Implementieren Sie die Funktionen translation, rotation, scaling, multiply entsprechend Ihrer Spezifikation im Kommentar!

1. Nutzen Sie diese Funktionen nun um das Dreiecksnetz zu verschieben, zu skalieren und zu rotieren.



Abbildung 3 Ergebnis von 3b

* + Parametrisieren Sie die Matrizen entsprechend der Benutzereingaben.
  + Übergeben Sie an den Shadern genau *eine* Matrix, die all diese Transformationen vereint. Nutzen Sie diese Transformation um die Vertex-Positionen im Vertex-Shader passend zu verändern!
  + Dabei soll die Rotation und Skalierung um den Koordinatenursprung des Modellkoordinatensystems erfolgen! Stellen Sie durch geeignet Multiplikationsreihenfolge der Matrizen dies sicher.

1. Wenn der Benutzer die Fenstergröße nun so verändern, dass diese nicht quadratisch ist, werden Sie sehen, dass das Model verzerrt dargestellt wird. Implementieren Sie die Methode aspect, welche diese Verzerrung berichtigt und multiplizieren Sie die resultierende Matrix geeignet an die Matrix, welche die Vertex-Positionen transformiert!