Praktikum 4: Mesh rendern

Kopieren Sie den Code dieses Praktikums in den des letzten Praktikums.

Aufgabe 1 Mesh Positionen darstellen

In der `setup` Methode wird bereits das Model `lion.smm` geladen. Dieses Modell wird in dem Objekt `model` der Klasse `SampleMeshModelIO` abgelegt.

Machen Sie sich mit den Member-Variablen dieser Klasse vertraut. Die Klasse repräsentiert ein Indexed-Face Set.

1. Faces sind dabei *Dreiecke*, die als `int3` in dem Array indices abgelegt sind.
2. Die *Positionen* sind in dem Array positions gespeichert. Obwohl jede Position ein float3 belegt, werden für das Modell `lion.smm` nur $x, y \in [-1, 1]^2$ verwendet. Jeder Vertex hat also eine Position.
3. Die *Farben* werden auch pro Vertex abgespeichert und zwar in dem Array colors. Jeweils ein float3 repräsentiert eine RGB Farbe. Die Komponenten der Farben liegen jeweils zwischen $[0..1]$.

a) Da sich die Positionen aus der Datei `lion.smm` zwischen $[-1, 1]^2$ befinden, müssen diese auf den Bildschirm $[0..w-1] \times [0..h-1]$ transformiert werden. Implementieren Sie dazu in der Klasse `mat4` die Methode

```
/**
 * Sets this matrix the window transform, that transforms the
 * clip-space points x, y from [-1..1]^2 to [0..w-1]x[0..h-1]
 * and depth from [-1..1] to [0..1].
 * @param w Width of the image plane in pixels.
 * @param h Height the image plane in pixels.
 */
public void setWindowTransform(float w, float h)
{
    this.setIdentity();
    // Implement me
}
```

Die Matrix hat dabei folgende Form:

$$\mathbf{W} = \begin{bmatrix} \frac{w}{2} & & & \frac{w}{2} \\ & -\frac{h}{2} & & -\frac{h}{2} \\ & & \frac{1}{2} & \frac{1}{2} \\ & & & 1 \end{bmatrix}.$$

b) Um eine 3D Position $\vec{p} = [p_x, p_y, p_z]^\top$ von `lion.smm` mit einer 4x4 Matrix wie **W** zu transformieren, brauchen wir eine Methode, welche eine `mat4` mit einem `float3` multipliziert und dabei implizit eine homogene Komponente $p_w = 1$ hinzufügt. Wir brauchen also eine Matrix-Vektor Multiplikation welche

$$\mathbf{W} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

berechnet. Implementieren Sie dazu in der Klasse mat4 die Methode

```
/**
 * Computes and returns the matrix-vector multiplication of
 * this * [rhs.x, rhs.y, rhs.z, 1]^T.,
 * i.e. assuming that the 4th component of rhs is 1.
 * @param rhs float3 the x, y, z component of as right-hand
 *            side of the product.
 * @return float4 The x, y, z and w component matrix-vector
    product
 *         this * (rhs, 1).
 */
public float4 mul_W1(float3 rhs)
{
    // Implement me!
}
```

c) Zeichen Sie nun die Punkte aus `lion.smm`. Iterieren Sie dazu in der `draw`-Methode über die Positionen, die in `model` hinterlegt sind. Transformieren Sie jeden Punkt mit der Matrix **W** und zeichnen Sie ihn als schwarzen Punkt (Farbe ist color(0,0,0) bzw. `#ff000000` in das pixels Array.

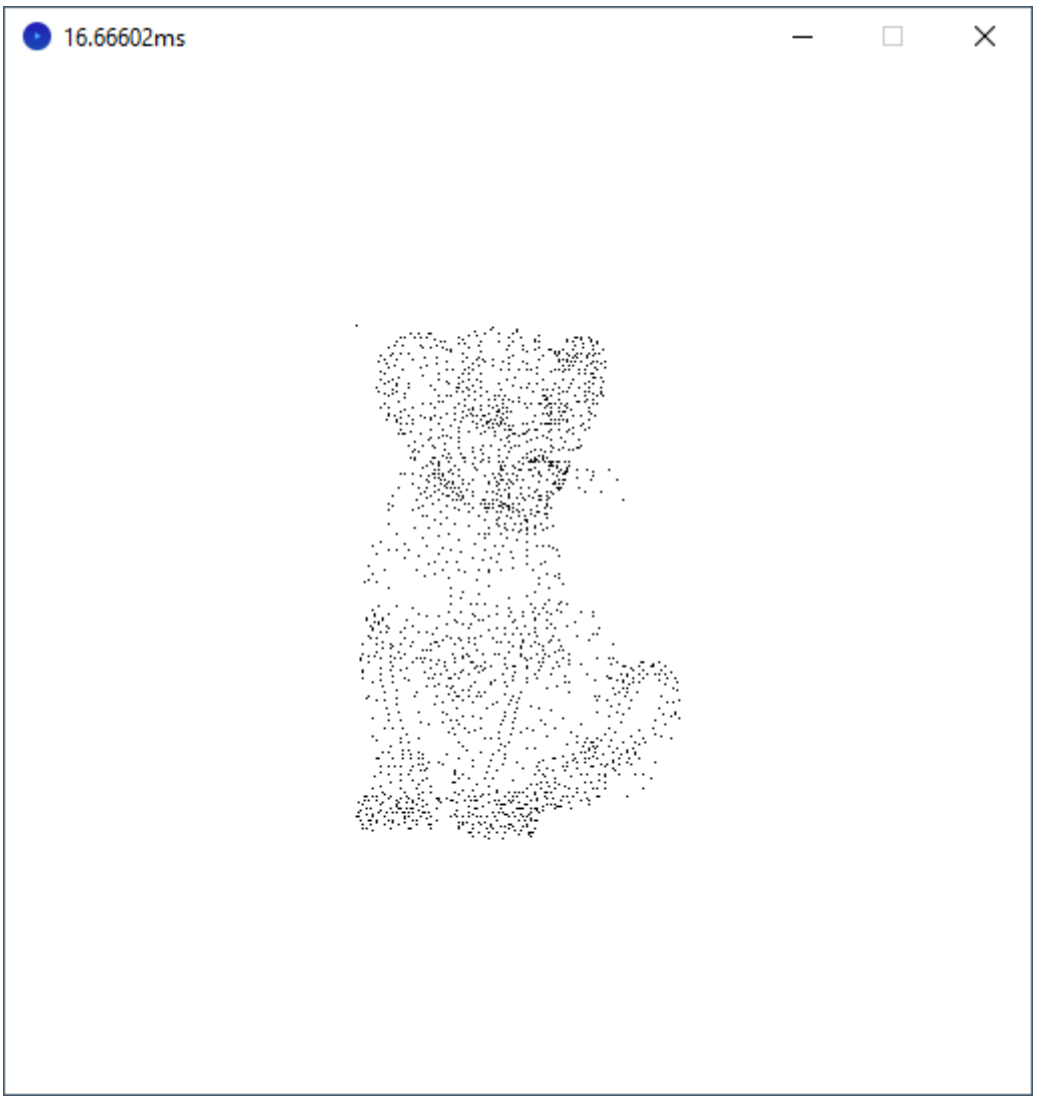

Abbildung 1: Der Tiger als Punktewolke.

Aufgabe 2 Mesh Dreiecke als Linien (Wire-Frame) darstellen

Iterieren Sie nun über die Dreiecke des Modells. Extrahieren Sie für jedes Dreieck die drei Positionen. Transformieren Sie jede dieser Positionen mit der Matrix **W**. Zeichen Sie die Dreiecke mittels Ihres Linien-Raster-Verfahrens.

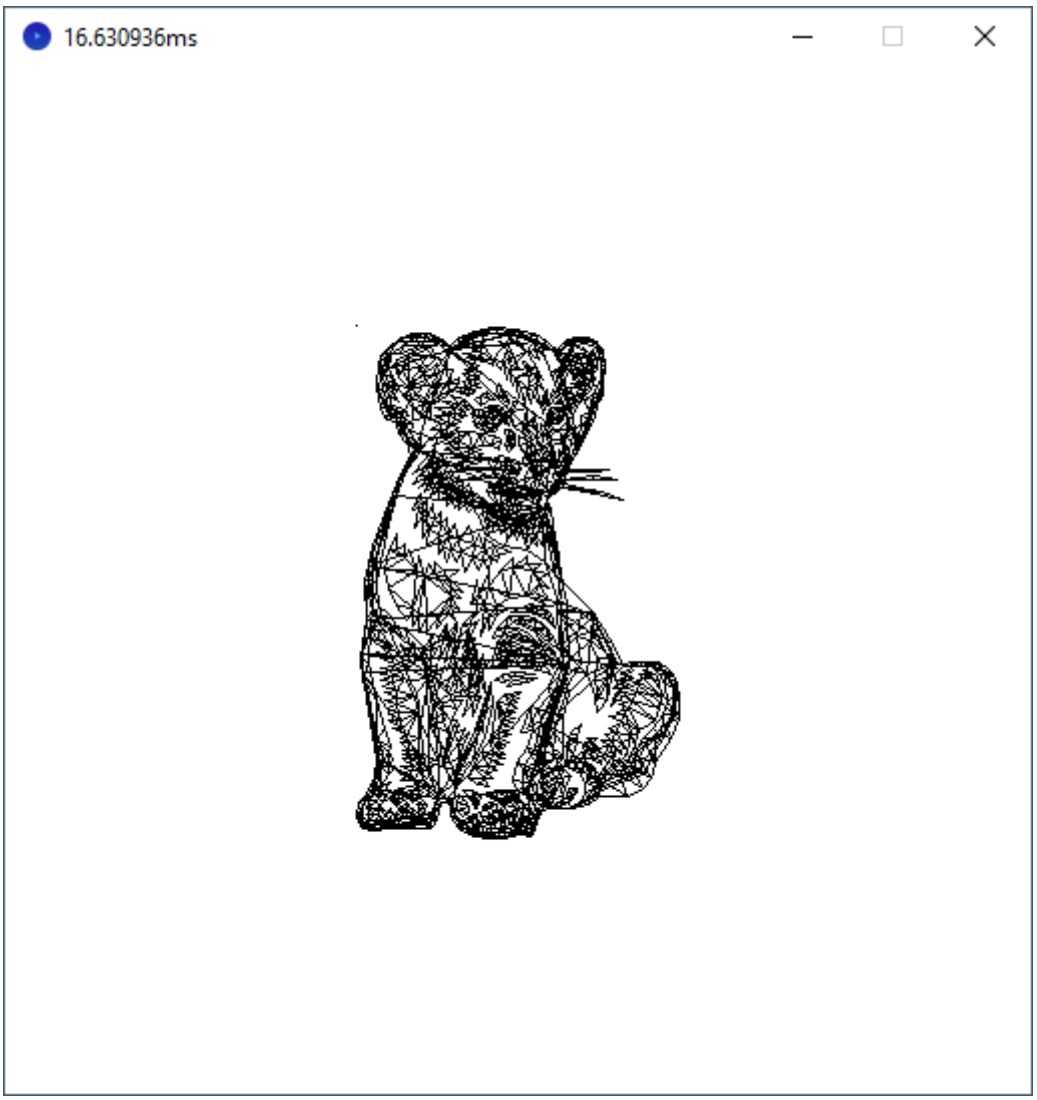

Abbildung 2: Der Tiger als Linen.

Aufgabe 3 Mesh als gefüllte Dreieck darsellen

a) Transformieren Sie nun die Position in den Subpixel-Bereich in dem Sie in der Klasse `SubPixelUtil` eine Methode `public static int2 toSubPixel(float4 regularPixel)` hinzufügen und diese aufrufen. Verwenden Sie die Positionen in Subpixel-Genauigkeit um Ihren Dreiecks-Raster-Algorithmus aufzurufen.

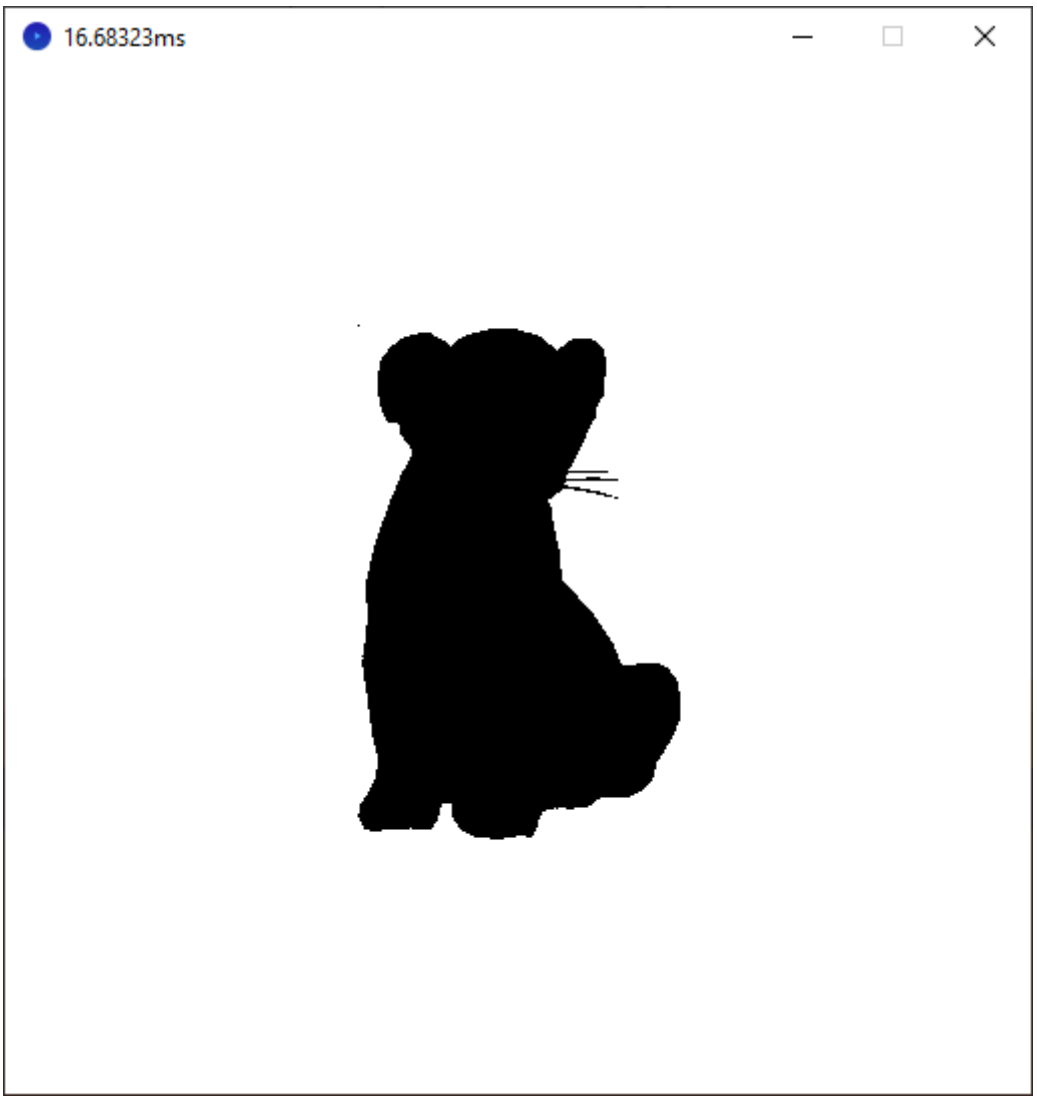

Abbildung 3: Der Tiger mit einer Farbe gefüllt.

b) Jeder Vertex hat eine zugehörige Farbe. Wählen Sie für jedes Dreieck *eine* dieser drei Farben *beliebig* aus (z.B. die erste) und nutzen Sie diese als Farbe für das *gesamte* Dreieck. Achten Sie darauf, dass der Framebuffer Farbkomponenten als Integer-Werte zwischen 0 und 255 erwartet, das Model jedoch Farbkanäle als Float-Werte zwischen 0 und 1 bereitstellt. Wenn Sie die dafür notwendige Konvertierung richtig hinbekommen haben, sollten Sie folgendes Ergebnis bekommen.

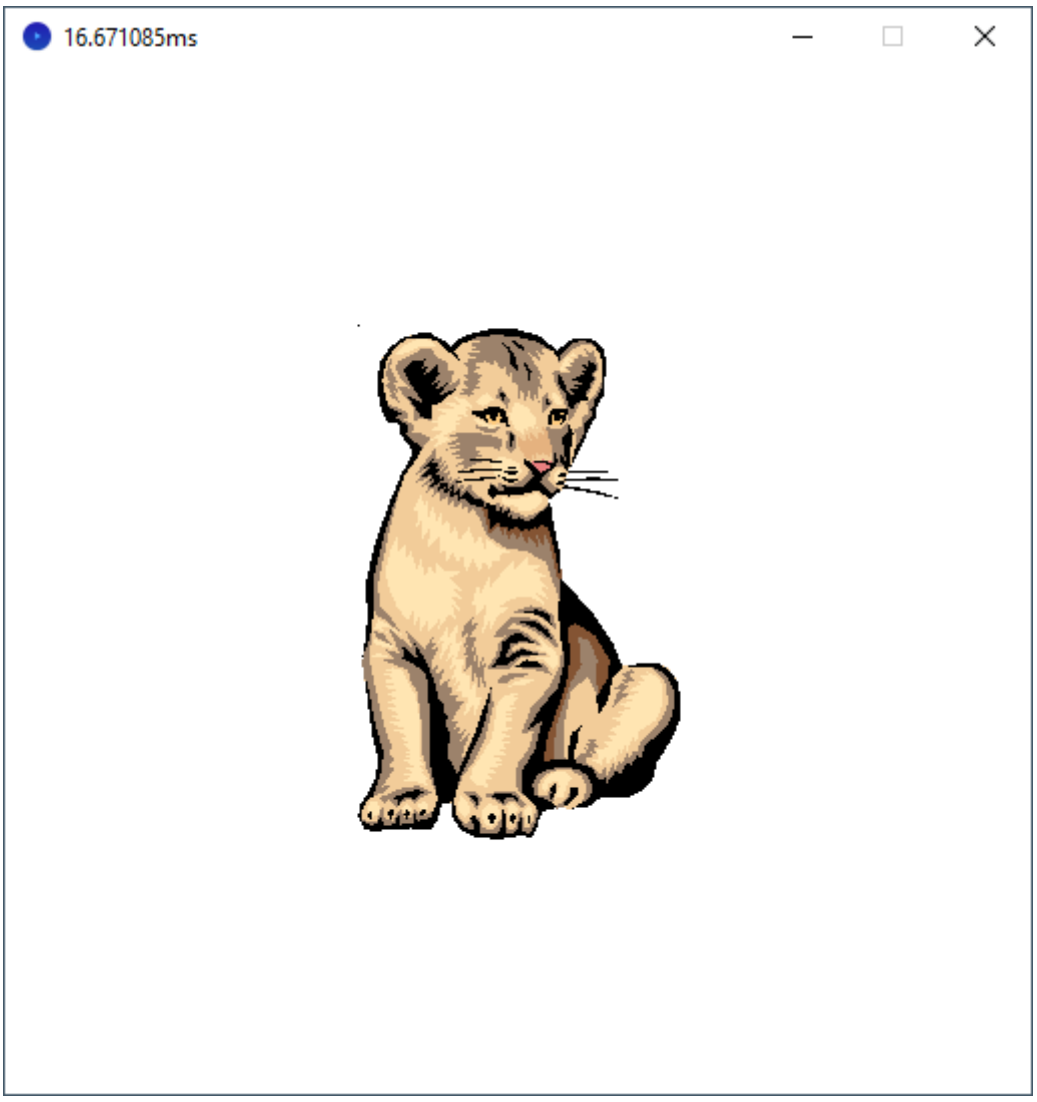

Abbildung 4: Der Tiger mit einfarbig gefüllten Dreiecken.

Aufgabe 4 Rotation und Transformation

a) Implementieren Sie die Methode `setTranslate` in der Klasse `mat4`, welche einen Punkt verschiebt. Zum Testen, rufen Sie

```
1 mat4 translate = new mat4();
2 translate.setTranslate(0.5f * sin(millis()/1000.0f),
3                        0.5f * cos(millis()/2000.0f), 0.0f);
```

in der Draw Methode auf. Wenn Sie die `translate` Matrix geeignet mit den Punkten multiplizieren, sollte das Modelleine Kurve in Achter-Form beschreiben.

b) Zusätzlich zur Translation brauchen wir noch eine Rotation. Implementieren Sie dazu die Methode in der Klasse mat4 und rufen Sie diese mit

```
mat4 rotate = new mat4();
rotate.setRotateZ(millis()/1000.0f);
```

auf. Multiplizieren Sie diese geeignete mit den Punkten.

Abbildung 5: Der Tiger in Aktion.