Gajda Krzysztof 78275

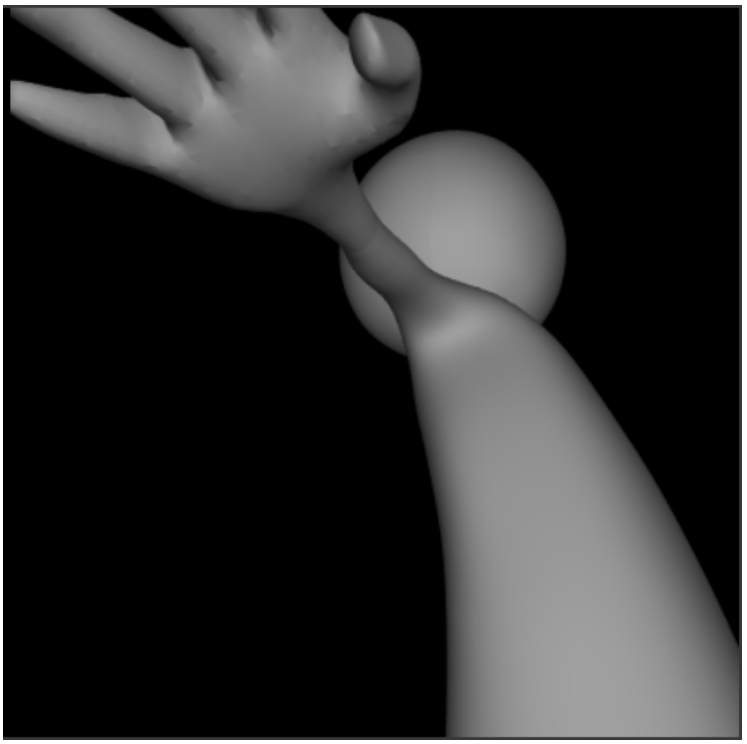
Zad1.

```
!pip install tensorflow-graphics
!pip install trimesh

import numpy as np
import tensorflow as tf
import trimesh

import tensorflow_graphics.geometry.transformation as tfg_transformation
from tensorflow_graphics.notebooks import threejs_visualization

# Download the mesh.
# Courtesy of Keenan Crane www.cs.cmu.edu/~kmcrane/Projects/ModelRepository/.
# !wget -N https://storage.googleapis.com/tensorflow-graphics/notebooks/index/cow.obj
# Load the mesh.
mesh = trimesh.load("stickman.OBJ")
mesh = {"vertices": mesh.vertices, "faces": mesh.faces}
# Visualize the original mesh.
_ = threejs_visualization.triangular_mesh_renderer(mesh, width=400, height=400)
# Set the axis and angle parameters.
axis = np.array((0., 1., 0.))  # y axis.
angle = np.array((np.pi / 4.,))  # 45 degree angle.
# Rotate the mesh.
mesh["vertices"] = tfg_transformation.axis_angle.rotate(mesh["vertices"], axis,
                                                        angle).numpy()
# Visualize the rotated mesh.
_ = threejs_visualization.triangular_mesh_renderer(mesh, width=400, height=400)
```



Zad 2.
Tensorflow.js to biblioteka JavaScript stworzona przez Google w celu umożliwienia użycia uczenia maszynowego w dowolnej aplikacji internetowej.

W celu dodania wizualizacji w przeglądarce, Tensorflow.js korzysta z małej biblioteki o nazwie tfjs-vis. Można jej następnie użyć do utworzenia bocznego panelu, zwanego Visor, na którym można wyświetlić całą zawartość.
Możemy teraz zaprojektować prosty model uczenia maszynowego (createModel), a następnie stworzyć funkcję trenującą go (trianModel).

W tym przypadku skorzystałem z funkcji callback, aby wywołać bibliotekę obsługi wykresów Tensorflow.js o nazwie tfjs-vis, by zobaczyć na wykresie jak zmieniała się strata podczas treningu modelu w czasie rzeczywistym.

```
function createModel() {
  // Create a sequential model
  const model = tf.sequential();

  // Add two hidden layers
  model.add(tf.layers.dense({ inputShape: [1], units: 5, useBias: true }));
  model.add(tf.layers.dense({ units: 10, useBias: true }));
  // Add an output layer
  model.add(tf.layers.dense({ units: 1, useBias: true }));

  return model;
}

async function trainModel(model, inputs, labels) {
  // Prepare the model for training.
  model.compile({
    optimizer: tf.train.adam(),
    loss: tf.losses.meanSquaredError,
    metrics: ["mse"]
  });

  const batchSize = 32;
  const epochs = 50;

  return await model.fit(inputs, labels, {
    batchSize,
    epochs,
    shuffle: true,
    callbacks: tfvis.show.fitCallbacks(
      { name: "Training Progresses" },
      ["loss", "mse"],
      { height: 200, callbacks: ["onEpochEnd"] }
    )
```

Training Progresses

onEpochEnd