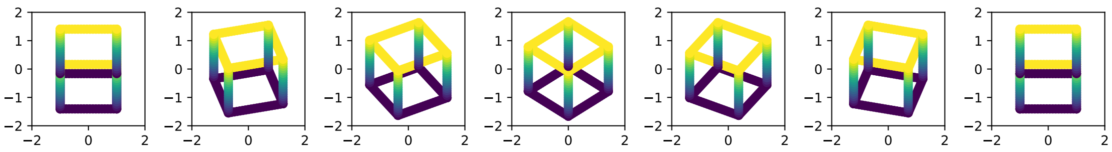
**Visualization**
Prof. Bernhard Schmitzer, Uni Göttingen, summer term 2024

# Problem sheet 4

- *Submission by* ***2024-05-21*** *18:00 via StudIP* ***as a single PDF/ZIP*** *(one day extension due to bank holiday on Monday). Please combine all results into one PDF or archive. If you work in another format (markdown, jupyter notebooks), add a PDF converted version to your submission.*
- *Use Python 3 for the programming tasks as shown in the lecture. If you cannot install Python on your system, the GWDG jupyter server at* `https://jupyter-cloud.gwdg.de/` *might help. Your submission should contain the final images as well as the code that was used to generate them.*
- *Work in groups of up to three. Clearly indicate names and enrollment numbers of all group members at the beginning of the submission.*

**Exercise 4.1: basic 3d projection.**
The file `points.npz` contains 3d point data of the edges of the cube $[-1,1]^3$. Our goal is to visualize this by using 2d plotting functions and our own implementation of mapping the 3d data to 2d data via parallel projection.

1. For angles $\theta \in [0,\pi]$ and $\varphi \in [0,2\pi]$, let
$$n^1 = \begin{pmatrix} \sin(\theta)\cos(\varphi) \\ \sin(\theta)\sin(\varphi) \\ \cos(\theta) \end{pmatrix}$$
be the unit vector associated with $(\theta,\varphi)$ via polar coordinates. Write a function that for given $(\theta,\varphi)$ provides two additional unit vectors $n^2, n^3 \in \mathbb{R}^3$ such that $(n^1,n^2,n^3)$ form an orthonormal basis of $\mathbb{R}^3$ and such that $n^3$ is parallel to the $(z=0)$-plane (i.e. the third component of $n^3$ is zero).
2. Write a function that projects the 3d point data onto the plane spanned by $(n^2,n^3)$ and shows the data as 2d scatter plot. For improved legibility, encode one of the original point coordinates additionally as color.
3. Now create a sequence of plots that illustrate a quarter 'rotation' around the cube, i.e. vary $\phi$ within $[0,\pi/2]$. The result will look best if $\theta$ is chosen between 0 and $\pi/2$. It could look as follows:



4. In the above figures the overlap of different points is not consistent with their distance from the viewer, resulting in points that should be further from the viewer occluding nearer points. This can be fixed by using that `plt.scatter` draws points in the order that they appear in the arguments. For a point $x \in \mathbb{R}^3$, $x^\top n^1$ can be used as a notion of (reverse) depth. The larger this value, the closer a point is to the camera. Compute this value and then sort the points accordingly to fix this depth rendering issue.