Assignment 2 - Transformations

This assignment lets you practice transformations in computer graphics. For your in-depth understanding of this area, it also covers a few related topics that might new to you. You should use any resources (e.g., books, search engines, calculators, and etc.) that may help you accomplish it.

Task 1: Affine Transformation

Let $P' = (p'_1, p'_2, p'_3)^\top$ be transformed from P by a scaling $\mathbf{T}_s$ with the factors s_x, s_y, and s_z; then followed by a translation $\mathbf{T}_t$ in the direction of vector $\mathbf{t} = (t_x, t_y, t_z)^\top$.

a) Write the homogeneous matrices $\mathbf{T}_s$ and $\mathbf{T}_t$ for these two operations.

b) Why do we need homogeneous coordinates? Name three reasons.

As we can see, the geometric meaning of a matrix is a transformation that maps one point to another. With $\mathbf{T}_t$ and $\mathbf{T}_s$, we have:

$$P' = \mathbf{T}_t \mathbf{T}_s P$$

To find the original point P, one can multiply two *inverse matrix* $\mathbf{T}_s^{-1}$ and $\mathbf{T}_t^{-1}$ that comply:

$$\mathbf{T}_s^{-1}\mathbf{T}_t^{-1}P' = \mathbf{T}_s^{-1}\mathbf{T}_t^{-1}\mathbf{T}_t\mathbf{T}_s P = P$$

i.e. $\mathbf{T}_s^{-1}\mathbf{T}_t^{-1}\mathbf{T}_t\mathbf{T}_s = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix.

c) Calculate the matrices $\mathbf{T}_s^{-1}$ and $\mathbf{T}_t^{-1}$ that inverse the transformations you wrote in a).

d) Calculate the coordinates of point P with respect to P'.

e) Can you switch the order of the transformations? Show your evidence by comparing $\mathbf{T}_t\mathbf{T}_s$ and $\mathbf{T}_s\mathbf{T}_t$.

In particular, given $Q = (x_0 + mt, y_0 + nt, z_0 + pt)^\top$ in the direction of $\mathbf{v} = (m, n, p)^\top$ concerning $M_0(x_0, y_0, z_0)$ where $t \in \mathbb{R}$:

f) Calculate $\mathbf{T}_t\mathbf{T}_s Q$

g) What property of affine transformation is shown in question f)? Why?

Include your answers in a Markdown file called “task01.md”.

Task 2: Rotation with Euler Angles

Assume we use a right-handed system. In the lecture, you learned an example to perform 3D rotation, i.e. rotation by θ_x around x-axis. The rotation matrix is:

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}$$

In practice, a rotation in a three-dimensional space is more complicated to describe than a single axis rotation. The rotation angles $(\theta_x, \theta_y, \theta_z)$ around the axes of the global (world) coordinate system are the so-called extrinsic *Euler angles*, and rotation angles $(\theta_X, \theta_Y, \theta_Z)$ around the axes of the local (object) coordinate system (solidary with the moving body) are the so-called intrinsic Euler angles. We use **extrinsic** Euler angles in this task.

a) What are the rotation matrices $\mathbf{R}_y$ and $\mathbf{R}_z$ for the Euler angles θ_y and θ_z?

b) Calculate the rotation matrix $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ that represents a series of rotations around the z-, y- and then the x-axis.

c) Can you switch the order of the transformations? Provide evidence by comparing $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ and $\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x$.

d) Let $\theta_y = \frac{\pi}{2}$, simplify the matrix $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z$ that you calculated in the question b).

e) Let point $L = (l_1, l_2, l_3)^\top$, calculate $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_z L$ where $\theta_y = \frac{\pi}{2}$. What can you conclude from the result? (Hint: Think about where the point is after the transformation.)

Moreover, the rotation sequences regarding three different axes are also called *Tait-Bryan angles*. For instance: $\mathbf{R}_z\mathbf{R}_y\mathbf{R}_x$. However, one can also perform the rotation 3 times with respect to only two axes, which is also called *Proper Euler angles*. For instance: $\mathbf{R}_x\mathbf{R}_y\mathbf{R}_x$.

f) How many possible sequences are there for Tait-Bryan angles?

g) How many possible sequences for Proper Euler angles?

Include your answers in a Markdown file called "task02.md".

Task 3: Rotation with Quaternions

A better and more strict way of doing rotations in a 3D space is to use *quaternions*, which are used more common in practice. A quaternion is similar to a complex number but more abstract. A complex number $z = a + bi \in \mathbb{C}$ where $i^2 = -1$ is the *imaginary unit*, and $a, b \in \mathbb{R}$. The geometric meaning of complex number multiplication embeds a rotation. For instance, given a real number r on the x-axis, the multiplication ri means a counterclockwise rotation of r by $\frac{\pi}{2}$; and the multiplication $rii = ri^2 = -r$ means two counterclockwise rotations, each by $\frac{\pi}{2}$. Intuitively, a complex number is a point on a 2D plane.

Instead of a single imaginary unit, a quaternion $\mathbf{q}$ contains more than one imaginary unit:

$$\mathbf{q} = a + bi + cj + dk \in \mathbb{H} = \text{span}(1, i, j, k)$$

where i, j, k are imaginary units and

$$i^2 = j^2 = k^2 = ijk = -1$$

a) Assume $\mathbf{p} = e + fi + gj + hk$, calculate the quaternion multiplication of $\mathbf{pq}$ arithmetically.

Since a quaternion is represented as a real number and three imaginary parts, one can use a scalar-vector pair to represent a quaternion $\mathbf{q} = (a, \mathbf{v})$ where $\mathbf{v} = (b, c, d)^\top \in \mathbb{R}^3$, and the multiplication of two quaternions can be simplified as:

$$\mathbf{pq} = (e, \mathbf{w})(a, \mathbf{v}) = (ea - \mathbf{w}^\top \cdot \mathbf{v}, e\mathbf{v} + a\mathbf{w} + \mathbf{w} \times \mathbf{v})$$

where $\mathbf{p} = (e, \mathbf{w})$, $\mathbf{w} = (f, g, h)^\top$, $\cdot$ is the dot product, and $\times$ is the cross product. Similar to a conjugate complex number $\bar{z} = a - bi$ with respect to $z = a + bi$, a conjugate quaternion $\bar{\mathbf{q}} = (a, -\mathbf{v})$, and the norm of a quaternion is defined as $\|\mathbf{q}\| = \sqrt{\bar{\mathbf{q}}\mathbf{q}}$.

b) Assume $\mathbf{q} = (\cos\theta, \mathbf{u}\sin\theta), \theta \in \mathbb{R}$, prove $\|\mathbf{q}\| = 1$ if and only if $\|\mathbf{u}\| = 1$.

A rotation by angle θ around a given unit direction $\mathbf{u}$ can be expressed by $\mathbf{q} = (\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2})$. In particular, given the original point $\mathbf{v} \in \mathbb{R}^3$ and the calculation $\mathbf{q}(0, \mathbf{v})\bar{\mathbf{q}}$ results in a quaternion $(s, \mathbf{v}')$ where $\mathbf{v}'$ is the destination resulting from the counterclockwise rotation around $\mathbf{u}$.

c) Let $\mathbf{q}_1 = (0, \mathbf{v}), \mathbf{v} = (0, 2, 0)^\top$, $\mathbf{q}_2 = (\cos\frac{\pi}{4}, \mathbf{u}\sin\frac{\pi}{4}), \mathbf{u} = (1, 0, 0)^\top$. Calculate $\mathbf{q}_2\mathbf{q}_1\bar{\mathbf{q}_2}$, and $\bar{\mathbf{q}_2}\mathbf{q}_1\mathbf{q}_2$.

d) Calculate the quaternions $\mathbf{q}_x, \mathbf{q}_y, \mathbf{q}_z$ that can express the rotations by the angle θ around x-axis, y-axis and z-axis respectively.

Include your answers in a Markdown file called "task03.md".

Task 4: Building A Scene using `Three.js`

`Three.js`[1] is a JavaScript library built on top of WebGL[2] to create and display 3D graphics. As a first step into the graphics world, we use it for high-level constructions. This task helps you get familiar with the general coding scheme in `Three.js`. Your final scene should similar to Figure 1.

Before you start working on building this scene, let's understand how the scene is constructed. In this scene, there are three bunnies above a grid plane, as well as indicators for a right-handed Cartesian coordinate system. More precisely, each bunny consists of a mesh and a material. The coordinate system is shown by three axes, and each axis consists of a cylinder, a cone, and a text label. Moreover,

[1] https://threejs.org
[2] https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

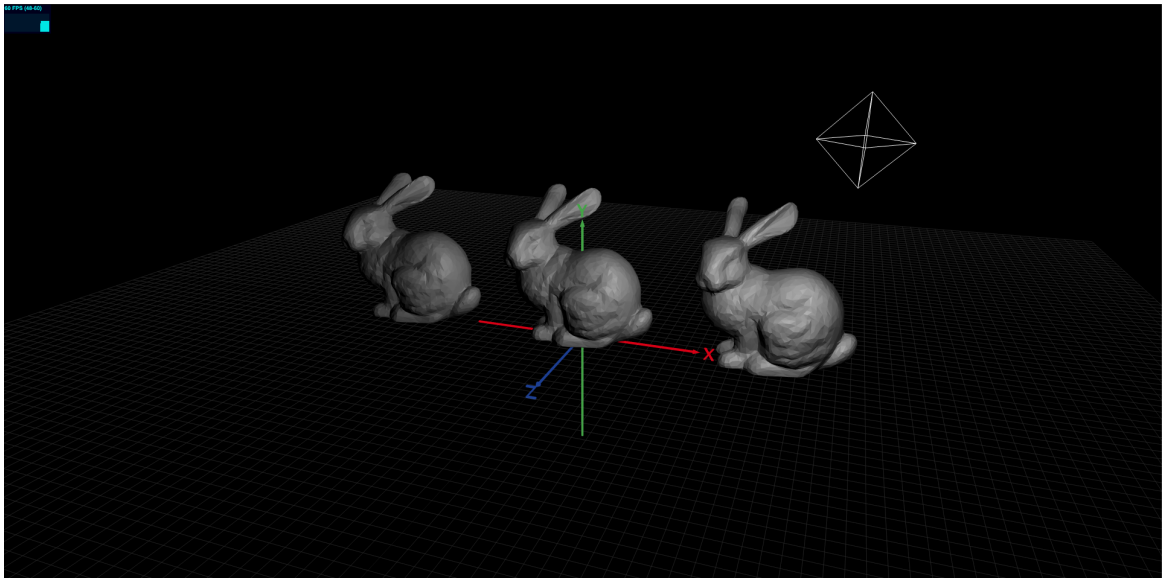

Figure 1: Final scene of the task.

a point light is located in the scene, close to the bunny on the right, and there is a camera that captures the whole scene. You can access an online demo[3] to explore and understand details of this scene more interactively.

a) What is the scene graph of this scene?

b) What nodes in your scene graph contains transformation matrices?

c) How is the position of the bunny on the right calculated given your scene graph? Give the needed parameters and calculate the final homogeneous transformation matrix.

To start coding and to run the code skeleton (https://github.com/mimuc/cg1-ss20), you need **`npm i`** to install all needed dependencies, then use **`npm start`**. The command compiles your current code skeleton and opens a new web page automatically to demonstrate your results. More conveniently, it also automatically refreshes the web page while you are changing your implementation.

In the `src` folder, you should find two JS files. `main.js` boots the whole runtime, creates a `World` object, sets up the scene, and renders it:

```
1 import World from './world'
2 (new World()).setupScene().render()
```

The `World` class includes the following key components:

- `constructor()` that initializes the renderer

[3] http://www.medien.ifi.lmu.de/lehre/ss20/cg1/demo/2-transform/

- `render()` that executes the render loop
- `setupScene()` calls `setupHelpers` and `setupBunnies` that establish the whole scene we are about to create
- `setupHelpers()` includes a helper grid plane, better visualized axes and a point light that lights up the scene
- `setupBunnies` loads and transforms a group of bunnies

See the code below:

```
export default class World {
  constructor() {
    ... // initialize renderer
  }
  render() {
    ... // execute the render loop
  }
  setupScene() {
    this.setupHelpers()
    this.setupBunnies()
    return this
  }
  setupHelpers() {
    // creates many helpers
    this.setupGridPlane()
    this.setupAxes()
    this.setupLight()
  }
  setupBunnies() {
    // creates a group of bunnies
    const rabbits = new Group()
    const loader = new GLTFLoader()
    loader.load('assets/bunny.glb', ...)
    this.scene.add(rabbits)
  }
  ... // more implementations
}
```

d) Look for `// TODO:` in the `world.js`, then implement them to reproduce this scene.

Hint:

- Read the `three.js` documentation and get familiar with these concepts: `Scene` and `PerspectiveCamera`[4], `OrbitControls`[5], `Mesh`[6], `Geometry`[7], `Material`[8], and `PointLight`[9]
- Use the provided constant parameters in the code skeleton for better reproducibility of the scene
- Use debugging tools (e.g. break points) to trace the code logic if you don't fully understand it
- Think about how to test your implementation quickly. For instance, figure out which TODO should be implemented first

After reproducing this scene, you should be able to answer the following questions:

e) What are the two components that determine the final rendered scene in the render loop?

f) What are the two components that create a Mesh?

g) Where does the positive Y-axis point to in `three.js` by default?

h) What type of Euler angles are used in `three.js`?

Answer text questions in the `README.md` of the code skeleton, then include your answers and implementation in a folder called "task04". ***Exclude*** the installed dependencies (folder `node_modules`) in your submission.

[4] https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene
[5] https://threejs.org/docs/index.html#examples/en/controls/OrbitControls
[6] https://threejs.org/docs/index.html#api/en/objects/Mesh
[7] https://threejs.org/docs/index.html#api/en/core/Geometry
[8] https://threejs.org/docs/index.html#api/en/materials/Material
[9] https://threejs.org/docs/index.html#api/en/lights/PointLight

Submission

- Participation in the exercises and submission of the weekly exercise sheets is voluntary and not a prerequisite for participation in the exam. However, participation in an exercise is a good preparation for the exam (the content is the content of the lecture and the exercise).
- For non-coding tasks, write your answers in a Markdown file. Markdown is a simple mark-up language that can be learned within a minute. A recommended the Markdown GUI parser is typora (`https://typora.io/`), it supports parsing embedded formula in a Markdown file. You can find the syntax reference in its Help menu.
- **Please submit your solution as a ZIP file** via Uni2Work (`https://uni2work.ifi.lmu.de/`) before the deadline. We do not accept group submissions.
- Your solution will be corrected before the discussion. Comment your code properly, organize the code well, and make sure your submission is clear because this helps us to provide the best possible feedback.
- If we discover cheating behavior or any kind of fraud in solving the assignments, you will be withdrawn for the entire course! If that happens, you can only rejoin the course next year.
- If you have any questions, please discuss them with your fellow students first. If the problem cannot be resolved, please contact your tutorial tutor or discuss it in our Slack channel.