Exercise 2: Projective transformations/ Canonical viewing volume

Computer Graphics 1 | Ugo Finnendahl, Max Kohlbrenner

Space transformations recap

From local to global space e.g. for the leg:

L_{leg}, L_{limb}, L_{body} and L_{body} are given (self constructed)

Three.js stores that in ***`leg.matrix`*** *(`leg` is the Object3d instance)*

L_{leg} transforms a Point p from leg space into limb space:

$$L_{leg} \cdot p$$

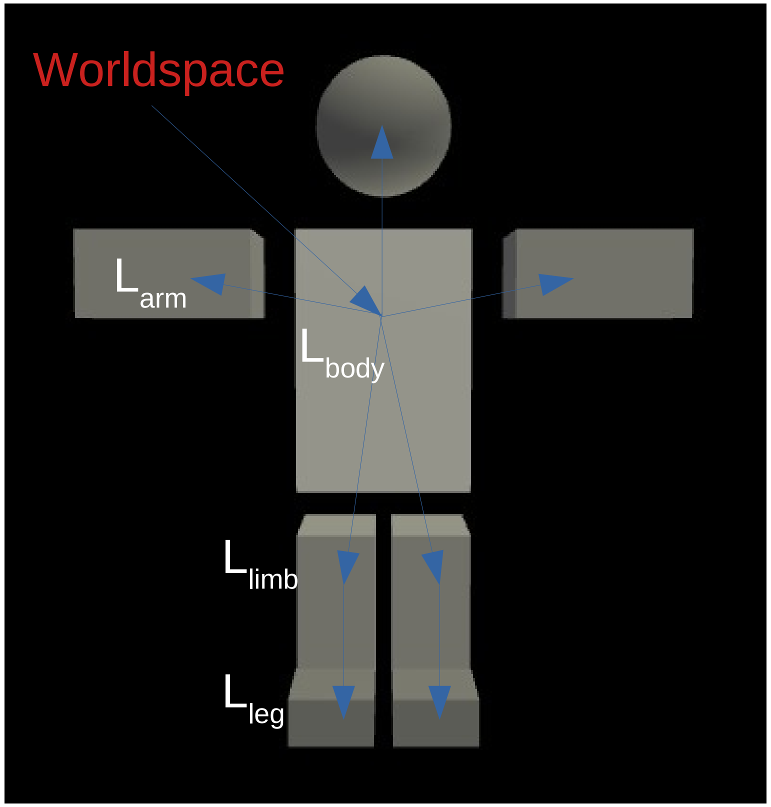
To transform into world space we need to chain these to the root:

$$L_{body} \cdot L_{limb} \cdot L_{leg} \cdot p \quad \text{so} \quad W_{leg} = L_{body} \cdot L_{limb} \cdot L_{leg}$$

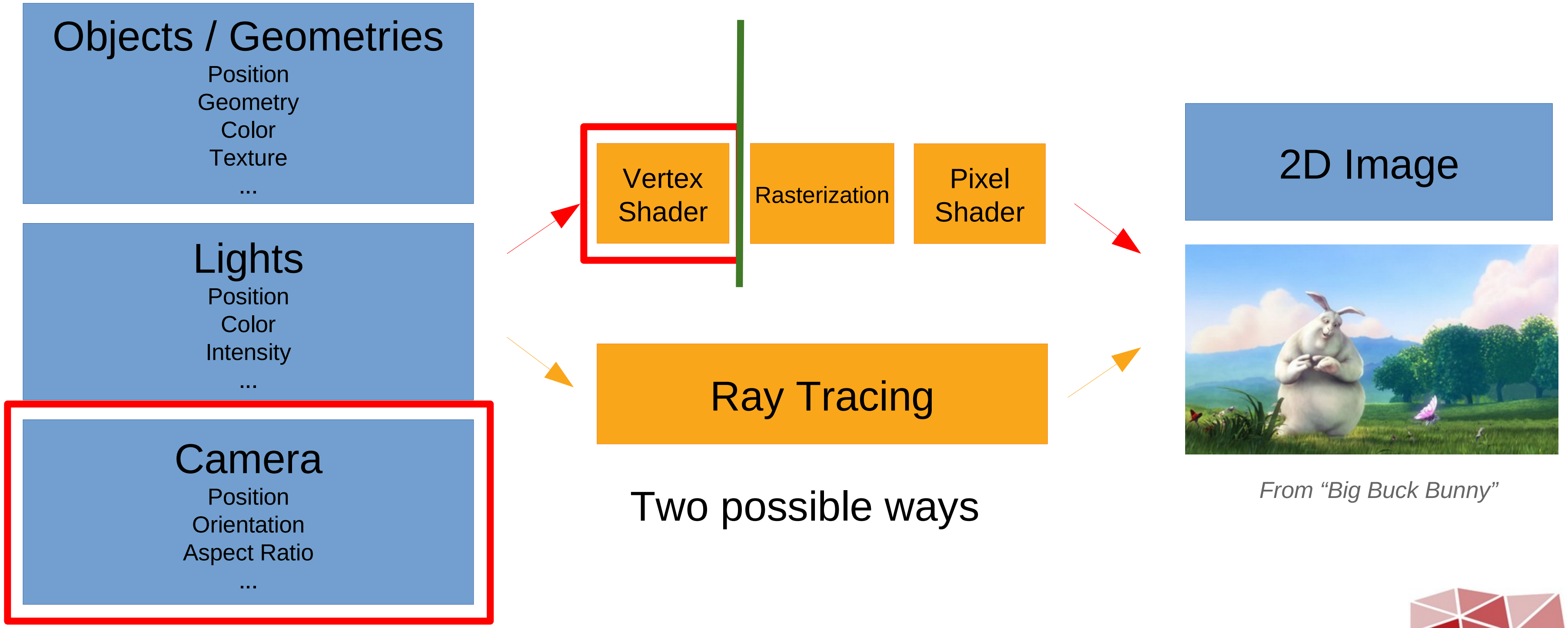
Three.js stores W_{leg} in ***`leg.matrixWorld`*** *this matrix in necessary for the GPU rasterization pipeline.*

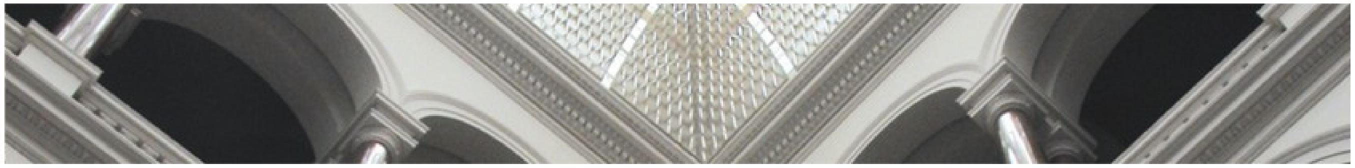
To transform v into arbitray spaces e.g. arm space we need to multiply the inverse:

$$L_{arm}^{-1} \cdot L_{limb} \cdot L_{leg} \cdot p \quad = \quad L_{arm}^{-1} L_{body}^{-1} \cdot L_{body} \cdot L_{limb} \cdot L_{leg} \cdot p \quad = \quad W_{arm}^{-1} \cdot W_{leg} \cdot p$$



Now the new part





Now the new part

We are able to transform between affine spaces. Now we want to do a perspective transformation.

First transform p into camera space:

$$W_{cam}^{-1} \cdot W_{leg} \cdot p$$

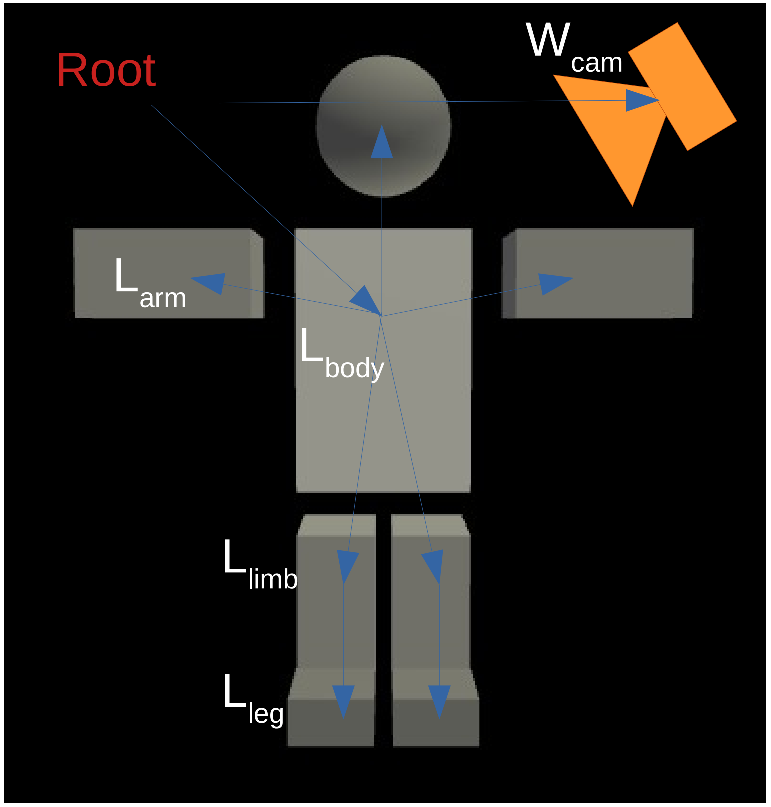
Three.js stores the inverse in ***`camera.matrixWorldInverse`****. It gets updated after calling camera*`.matrixWorldUpdate`*.*

Than apply the perspective transformation P_{cam}:

$$P_{cam} \cdot W_{cam}^{-1} \cdot W_{leg} \cdot p$$

Important: This is not the std. matrix multiplication! It is in homogenous coordiantes.

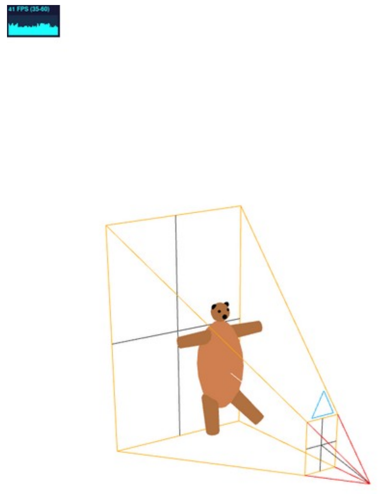
Three.js stores P_{cam} *in* ***`camera.projectionMatrix`****. It gets updated after calling camera*`.updateProjectionMatrix`*.*



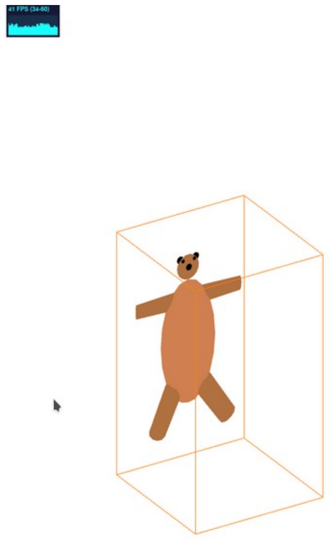
Exercise 2: Projective transformations/ Canonical viewing volume

Live Demonstration (video will be provided on ISIS).

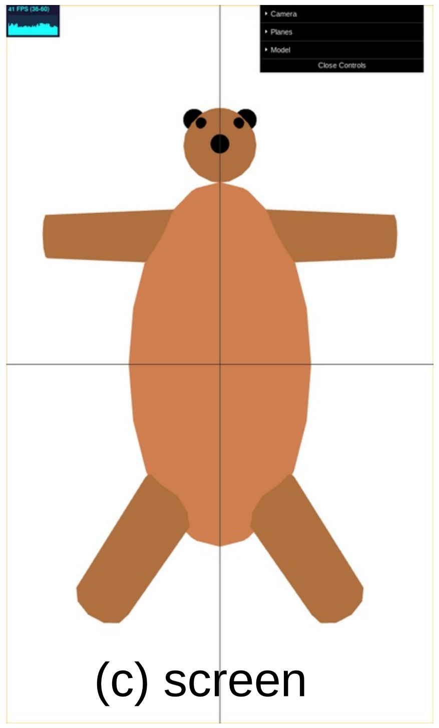
Exercise 02: stages of a perspective camera rendering



(a) world



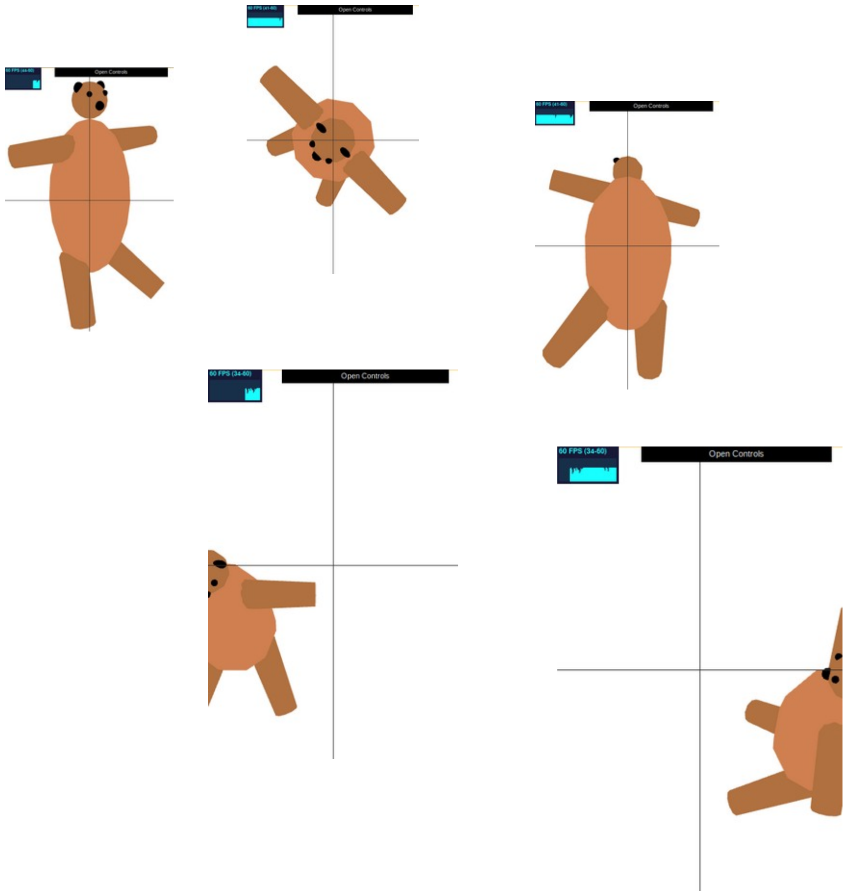
(b) NDC



(c) screen

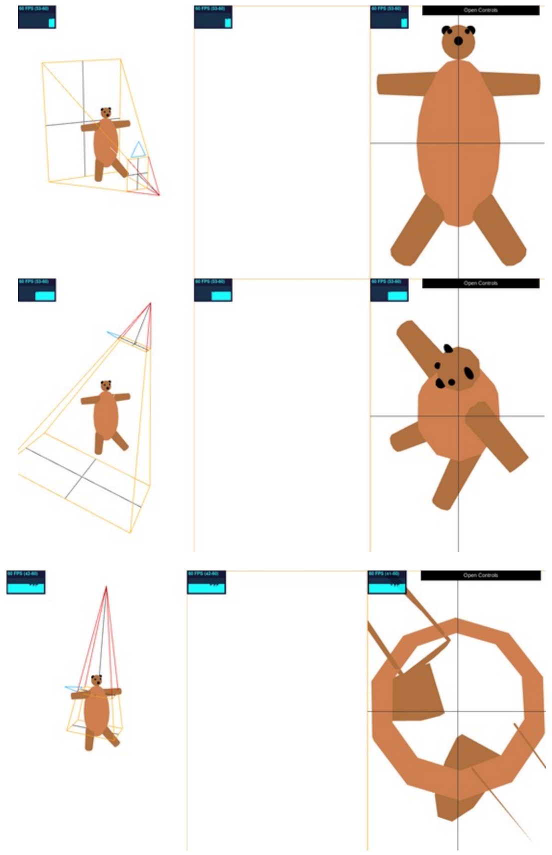
Task 01: Screen Space (right)

- Create a scene with an object
  (createTeddyBear, helper.ts)
- Render it with a perspective camera
  (“screen camera”)
- Add orbitControls to the camera
- Implement GUI functionality to translate
  + rotate the bear



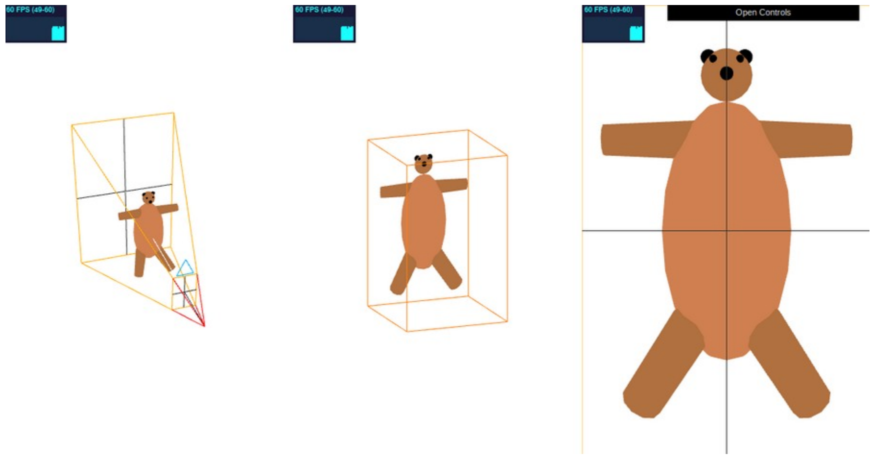
Task 02: World Space (left)

- Render the scene from a different perspective ("world") camera, add OrbitControls
- Update the "screen" camera parameters with the GUI
- Visualize the "screen" camera and its frustum in world space and update the visualization when the "screen" camera is changed by the GUI or its OrbitControls



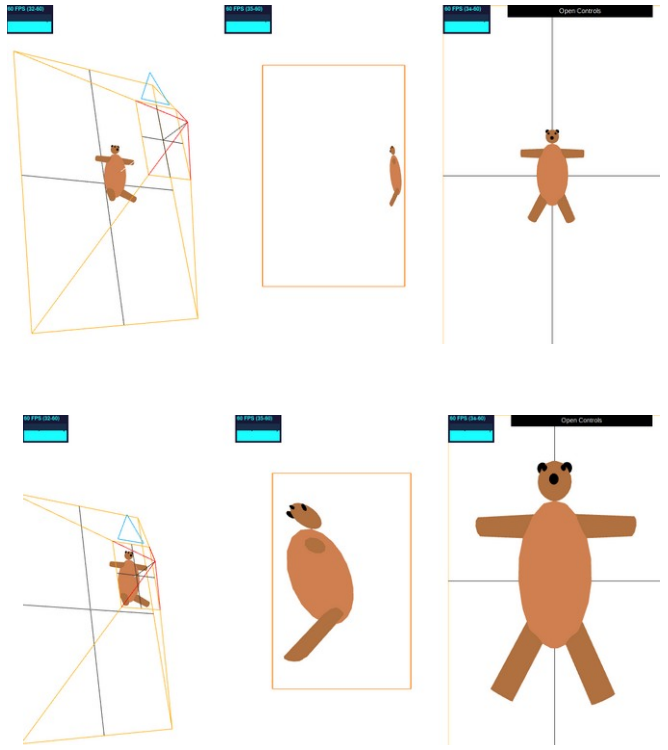
Task 04: Normalized Device Coordinates (NDC, middle)

- Create a new scene with an orthographic camera (middle window) including a cube of length 2 (canonical viewing volume)
- Add a copy of the bear and transform it into the canonical viewing volume
- **Important:** Do NOT use the applyMatrix4 method on either the BufferGeometry or BufferAttribute or the VectorX classes (see following slide)



Task 04 (intuition): NDC Projective Transformation

- Contains the perspective part of the camera transformation
- “Nearer is bigger”
- Transformation to screen space is simple parallel projection
- **NOT** an affine transformation



Task 04: Important Restriction

Work directly on geometry vertices

- You are not allowed to use applyMatrix4 on BufferGeometry BufferAttribute or Vector3/Vector4.
- Write the homogeneous matrix dot vector product by hand.

Hints:

- The geometry vertices are not defined in homogeneous coordinates make sure to handle them as those.
- Be aware that the dot product in homogeneous coordinates is not only a simple matrix multiplication.

Task 04 (hint): Unwanted additional implicit transformations

The visualization of the canonical view space is not straight forward. We try to visualize an intermediate step, that happen while rendering using a shader (see next exercise). The problem is that our perspective transformation step uses the scene graph so after that step we do not need it anymore. But for the visualization we still have a scene graph with all its local coordinate systems. So the transformations are applied **again**.

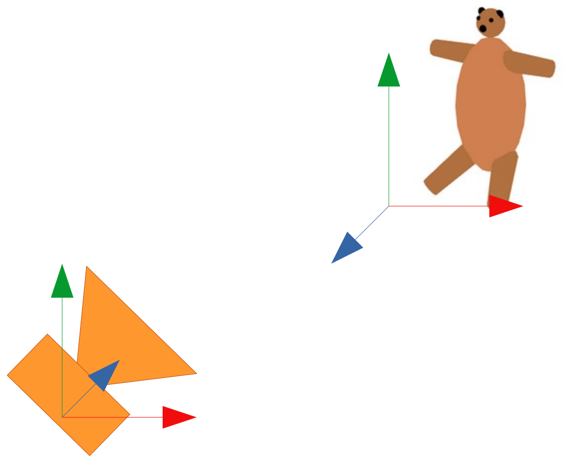
There are two ways to solve the Problem:

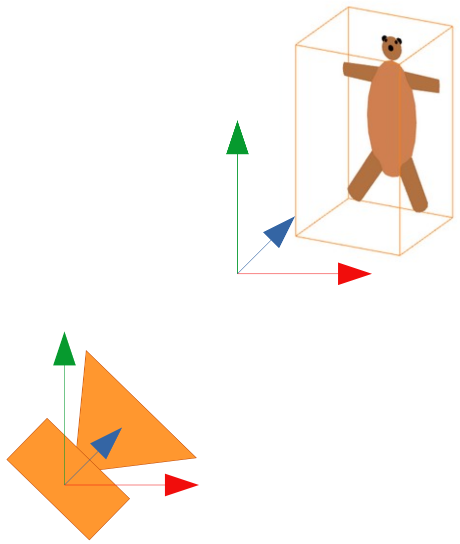
- Make sure every local coordinate system is the same as the world coordinate system (set all to I).
- In the perspective transformation step apply the inverse of the “to world” transformation.

Task 04 (hint): Coordinate System Flip

Convention:

NDC:





Flip Axis for correct visualization!

Exercise 2 – Perspective transformation

- **Hand in through ISIS:** due 24-Nov-2021.
- **Name convention:** {firstname}_{lastname}_cg1_ex{#}.zip (e.g. jane_doe_cg1_ex2.zip)

Important!: Only zip the ***src/*** subfolder.