# 07 Math for 3D games

Tvorba a dizajn počítačových hier

Návrh a vývoj počítačových hier

Michal Ferko

11. 11. 2021

# What we need the math for

- Placing/moving/rotating/scaling objects
  - Creating hierarchies of objects
- Animation
- Rendering
- Physics & simulation

- Not just 3D, also 2D (but it’s simpler)

# Floating point numbers

- IEEE 754 standard
- Single (32-bit) and double precision (64-bit)
  - GPUs almost exclusively single precision
  - Most engines perform all operations as 32-bit floats
- GPU FLOPS
- Half precision sometimes used on GPUs to speed up execution

# Vectors

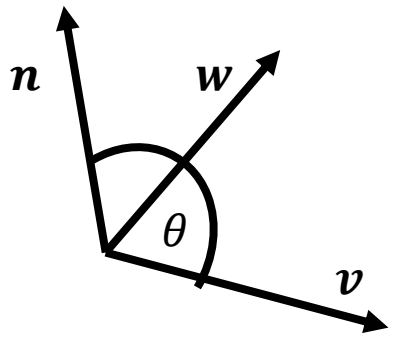
- A geometric vector is an entity with magnitude (length) and direction, represented graphically as a line segment with an arrowhead on one end to indicate its direction
- $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{0}$
- Vectors represent direction, not position
- Operations
  - Addition of two vectors:
    - $\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}, \boldsymbol{v} + \mathbf{0} = \boldsymbol{v}, (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w} = \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w})$
    - For every vector **v** exists −**v** such that **v** + (-**v**) = **0**
    - **u** − **v** = **u** + (-**v**)
  - Multiplication by a scalar: changes magnitude
    - (ab)**v** = a(b**v**), (a + b)**v** = a**v** + b**v**, a(**v** + **w**) = a**v** + a**w**
- Unit vector

# Vectors (2)

- Linear combination
- Linearly dependent, linearly independent, parallel
- No coordinates so far
- Vector representation is derived from a set of linearly independent vectors called a *basis* from which we derive other vectors using linear combinations
- For 3D, we define 3 vectors $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$
  - $\boldsymbol{v} = (x, y, z) = x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$
- Usually $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are unit vectors perpendicular to each other – *standard Euclidean basis*
- Vector addition and scalar multiplication are then performed component-wise

# Vectors (3)

- Length
  - Standard Euclidean norm: $\boldsymbol{u} = (x, y, z) \Rightarrow \|\boldsymbol{u}\| = d = \sqrt{x^2 + y^2 + z^2}$
  - Normalization: $\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$
- Dot product: $\boldsymbol{v} \cdot \boldsymbol{w} = \|\boldsymbol{v}\|\|\boldsymbol{w}\| \cos\theta$
  - $\boldsymbol{v} \cdot \boldsymbol{w} = v_x w_x + v_y w_y + v_z w_z$
- Cross product: $\boldsymbol{v} \times \boldsymbol{w} = \|\boldsymbol{v}\|\|\boldsymbol{w}\| \sin\theta\, \boldsymbol{n}$
  - $\boldsymbol{n}$ – unit vector ($\|\boldsymbol{n}\| = 1$), $\boldsymbol{n} \cdot \boldsymbol{v} = 0, \boldsymbol{n} \cdot \boldsymbol{w} = 0$
  - Right hand rule
- Real vector space $R^3$



# Points

- Represent a location in space
- We measure the location relative to the origin
  - Coordinate systems $< P, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3 >$
  - Origin and vector basis
  - $Q = (x, y, z) = P + x\boldsymbol{e}_1 + y\boldsymbol{e}_2 + z\boldsymbol{e}_3$

# Vectors and Points - Unity

- Vectors and points share classes
  - `Vector2, Vector3, Vector4`
- Used for positions, directions, other **spatial** functionality
- Basic operations included
  - Addition, subtraction, multiplication, magnitude, normalization
  - Angle, Dot, Cross, Reflect…
  - `Mathf` class
- `Transform.position,Transform.lossyScale (+local variants)`
- `Transform.rotation` is a **Quaternion**
- `Transform.forward, Transform.up, Transform.right`

# Transformations

- **Affine Transformations** – talk by Jim Van Verth
- **Orientation Representation** – talk by Jim Van Verth
- Transform Translate/Rotate/Scale
- Look At

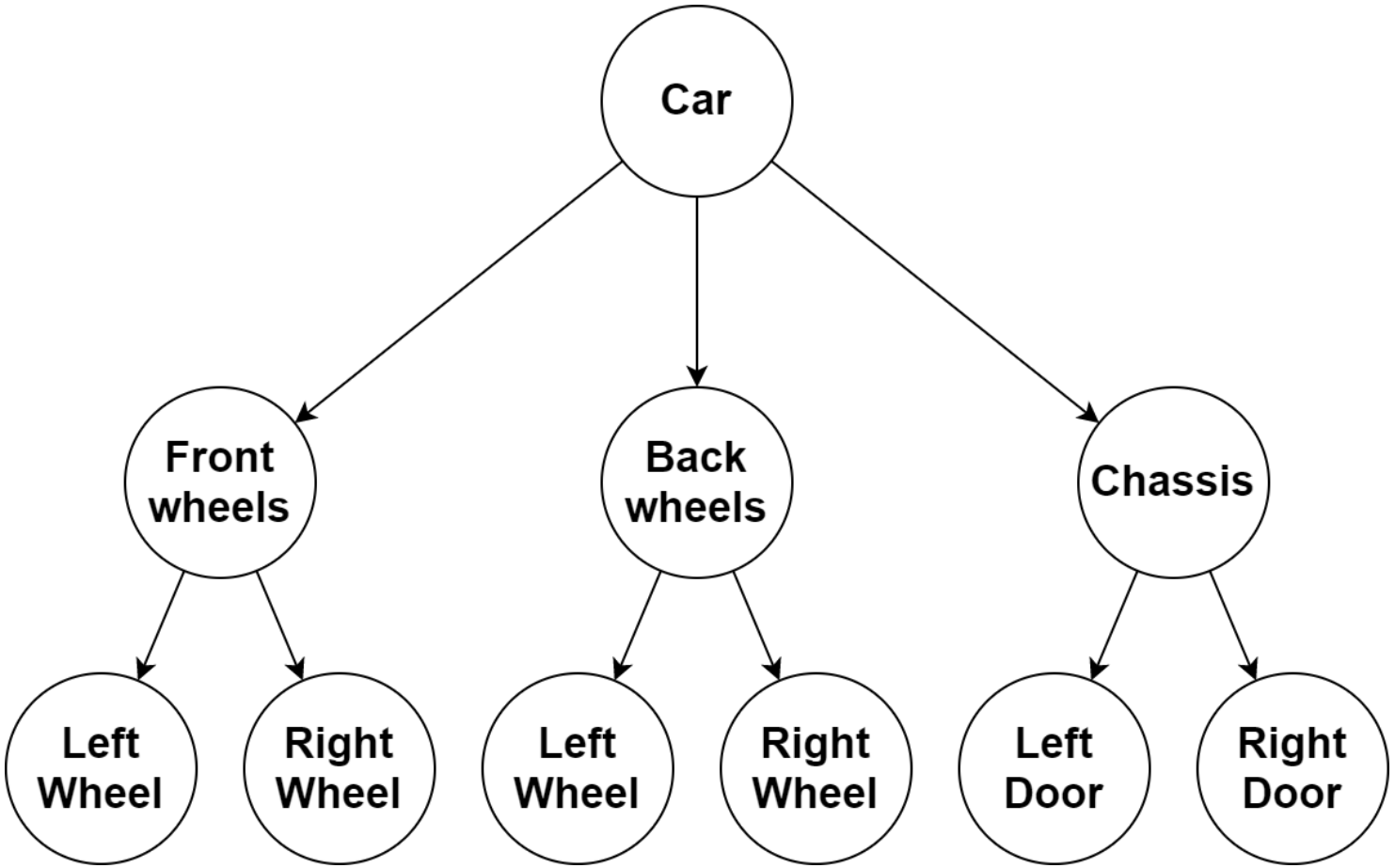
# 3D Transformation Pipeline

- We want to solve 3 problems
  - Construct hierarchies of objects
    - Transformation of object combined with its parents
  - Manipulate camera
    - Viewing transformation
  - Render object to screen
    - Projection/screen transformation

# Scene Graph

- Basic structure for hierarchical scenes
  - Used almost anywhere, even for e.g. video editing
- It is a tree structure – directed acyclic graph
  - Nodes can have any number of children

# Scene Graph


Car
Front wheels
Back wheels
Chassis
Left Wheel
Right Wheel
Left Wheel
Right Wheel
Left Door
Right Door

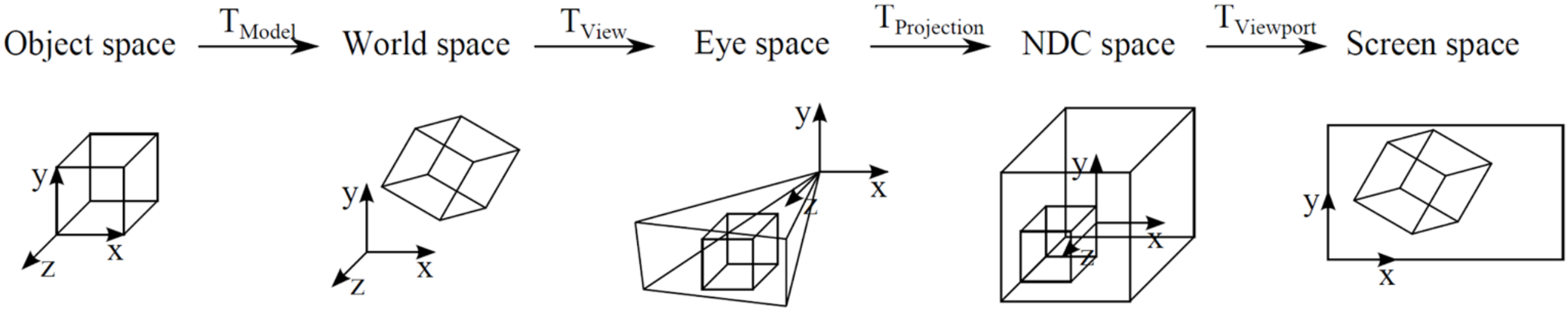
# Object transformation

- Each scene graph node has an affine transformation
- Final object transformation
  - Its own transformation
  - Multiplied by the parent’s transformation
  - Multiplied by the grandparent’s transformation
  - …
  - The result is a multiplication of several 4x4 matrices

    ⇒ one 4x4 transformation matrix

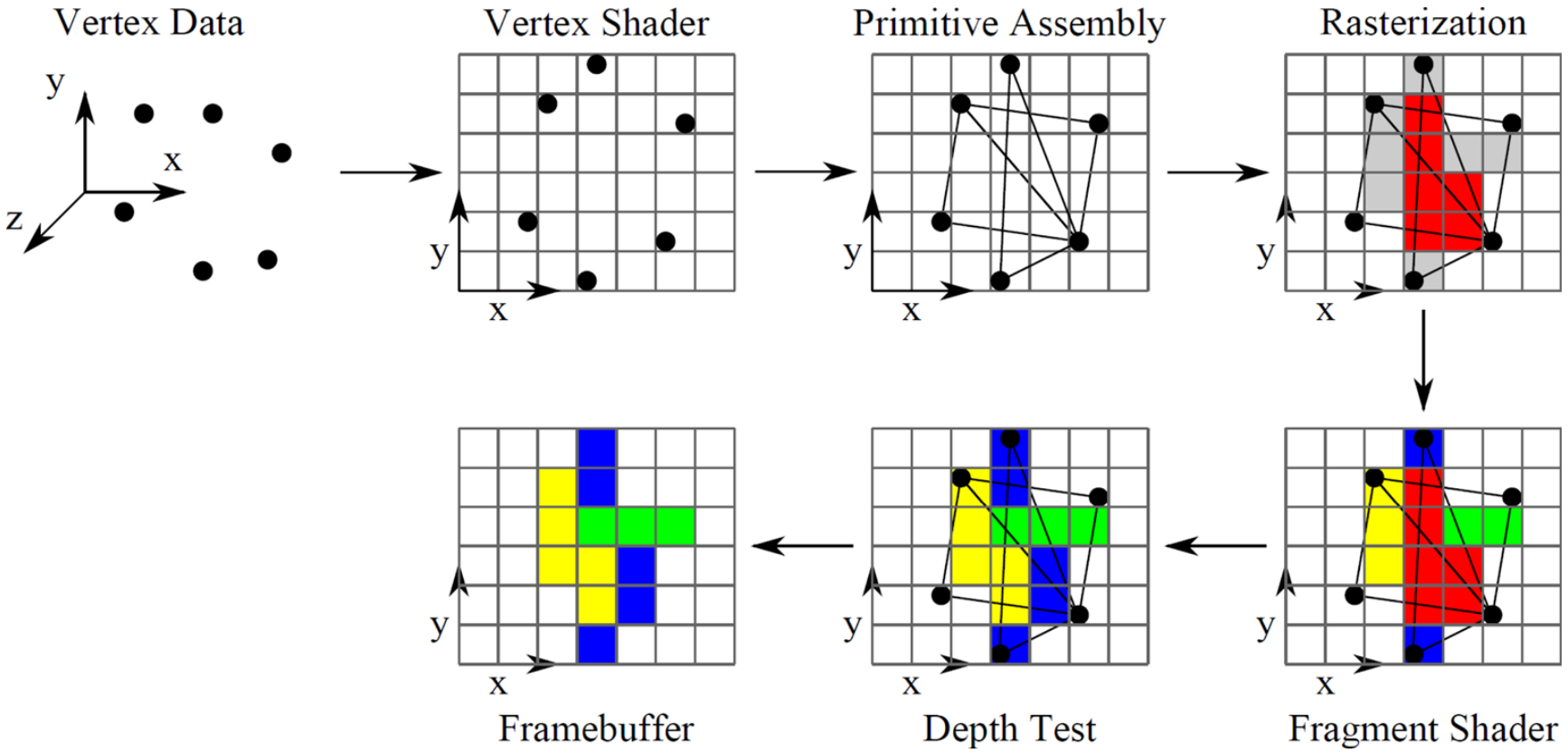
# Different spaces

- We determine object positions in the “world” ($T_{Model}$)
  - Created from a hierarchy of transformations – from object through its parents
- By placing the camera in the world, we determine from where (position, direction) we are looking at the world ($T_{View}$)
- The type of camera (orthographic/perspective) determines our projection 3D => 2D ($T_{Projection}$)
- We select the part of the window we render to ($T_{Viewport}$)



# Real-time Rendering Pipeline

- The GPU renders 3D scenes from triangles
- Offline rendering is very different (ray tracing instead of rasterization)



# References


- Mathematics for 3D Game Programming and Computer Graphics
- Essential Mathematics for Games and Interactive Applications: A Programmer’s Guide