

06 Math for 3D games

Tvorba a dizajn počítačových hier

Návrh a vývoj počítačových hier

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
- $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{0}$
- Vectors represent direction, not position
- Operations
 - Addition of two vectors:
 - $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}, \mathbf{v} + \mathbf{0} = \mathbf{v}, (\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$
 - For every vector \mathbf{v} exists $-\mathbf{v}$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$
 - $\mathbf{u} - \mathbf{v} = \mathbf{u} + (-\mathbf{v})$
 - Multiplication by a scalar: changes magnitude
 - $(ab)\mathbf{v} = a(b\mathbf{v}), (a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}, a(\mathbf{v} + \mathbf{w}) = a\mathbf{v} + a\mathbf{w}$
- 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 $\mathbf{i}, \mathbf{j}, \mathbf{k}$
 - $\mathbf{v} = (x, y, z) = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$
- Usually $\mathbf{i}, \mathbf{j}, \mathbf{k}$ 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: $\mathbf{u} = (x, y, z) \Rightarrow \|\mathbf{u}\| = d = \sqrt{x^2 + y^2 + z^2}$

- Normalization: $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$

- Dot product: $\mathbf{v} \cdot \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \cos \theta$

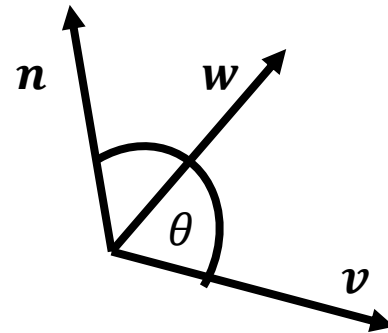
- $\mathbf{v} \cdot \mathbf{w} = v_x w_x + v_y w_y + v_z w_z$

- Cross product: $\mathbf{v} \times \mathbf{w} = \|\mathbf{v}\| \|\mathbf{w}\| \sin \theta \mathbf{n}$

- \mathbf{n} – unit vector ($\|\mathbf{n}\| = 1$), $\mathbf{n} \cdot \mathbf{v} = 0$, $\mathbf{n} \cdot \mathbf{w} = 0$

- Right hand rule

- Real vector space R^3



Points

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

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.scale`
- `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

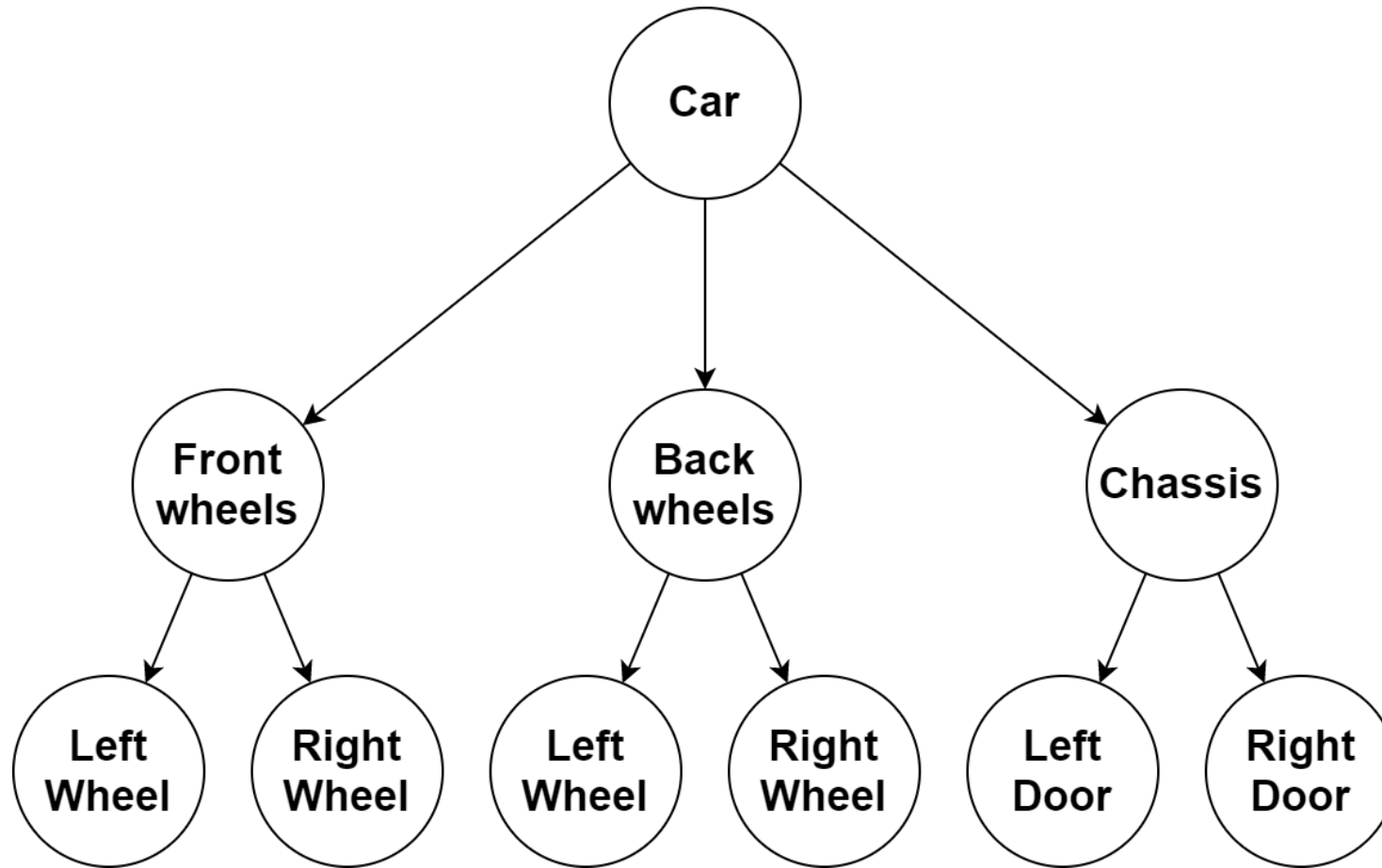
3D Transformation Pipeline

- 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



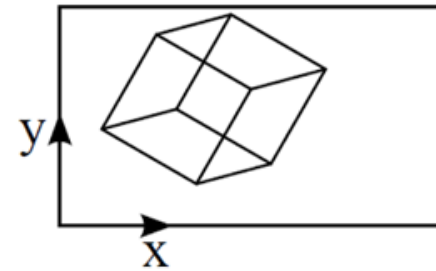
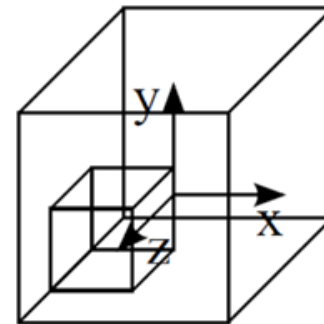
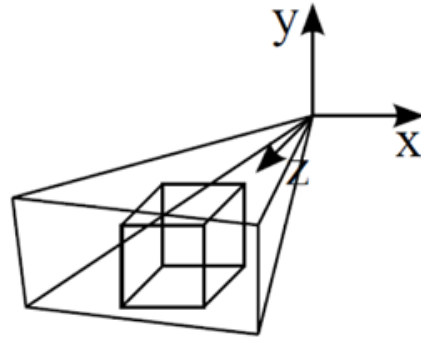
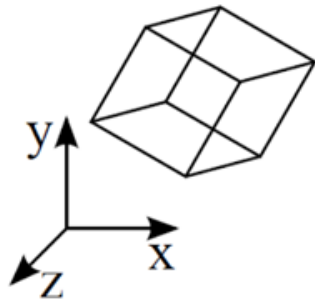
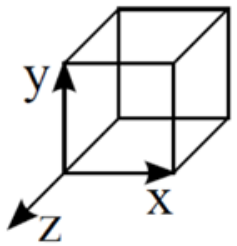
Object transformation

- 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

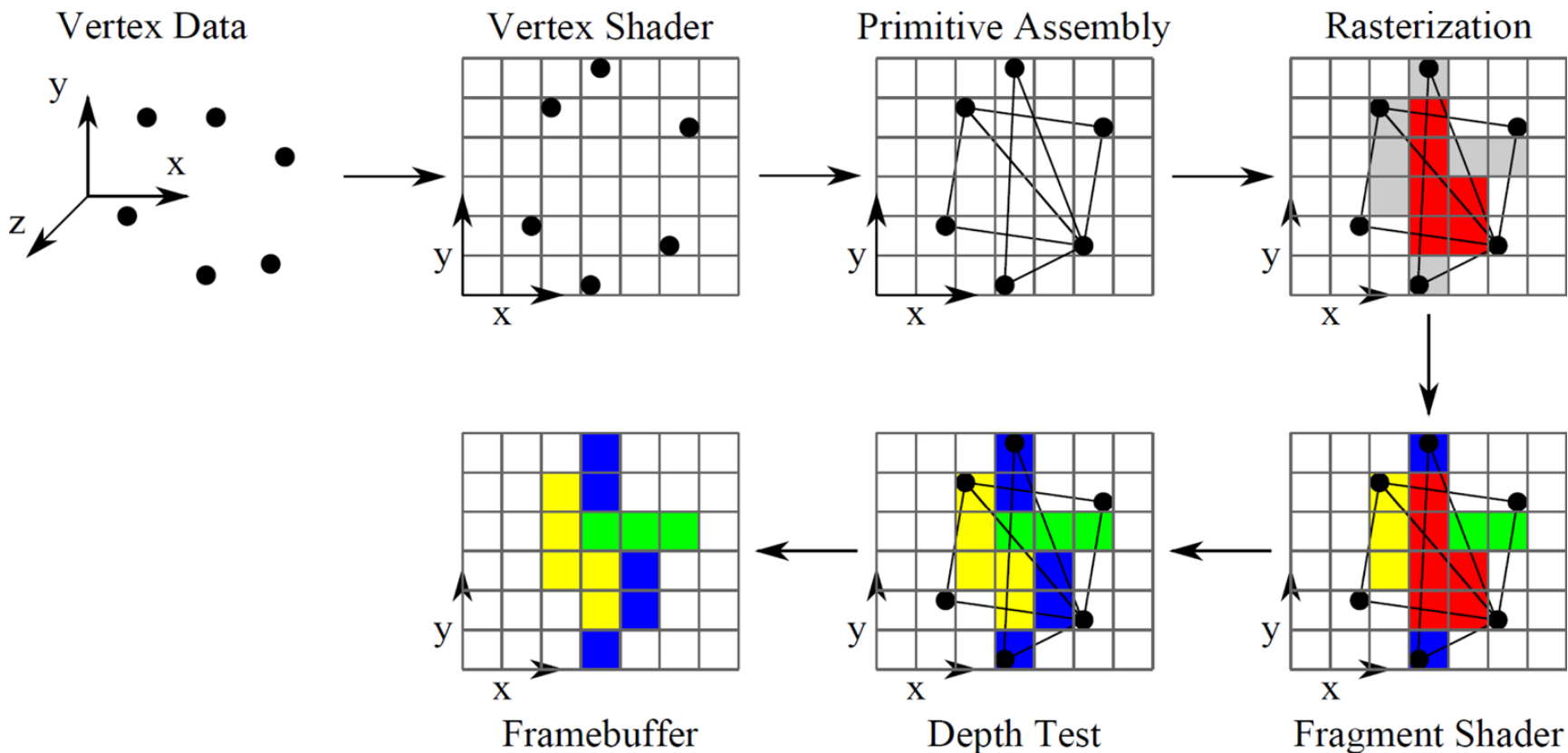
- We determine object positions in the “world” (T_{Model})
 - Created from a hierarchy of transformations – from object through its parents
- By placing the camera in the world, we determine from where (position, direction) we are looking at the world (T_{View})
- The type of camera (orthographic/perspective)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

Object space $\xrightarrow{T_{\text{Model}}}$ World space $\xrightarrow{T_{\text{View}}}$ Eye space $\xrightarrow{T_{\text{Projection}}}$ NDC space $\xrightarrow{T_{\text{Viewport}}}$ Screen space



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