

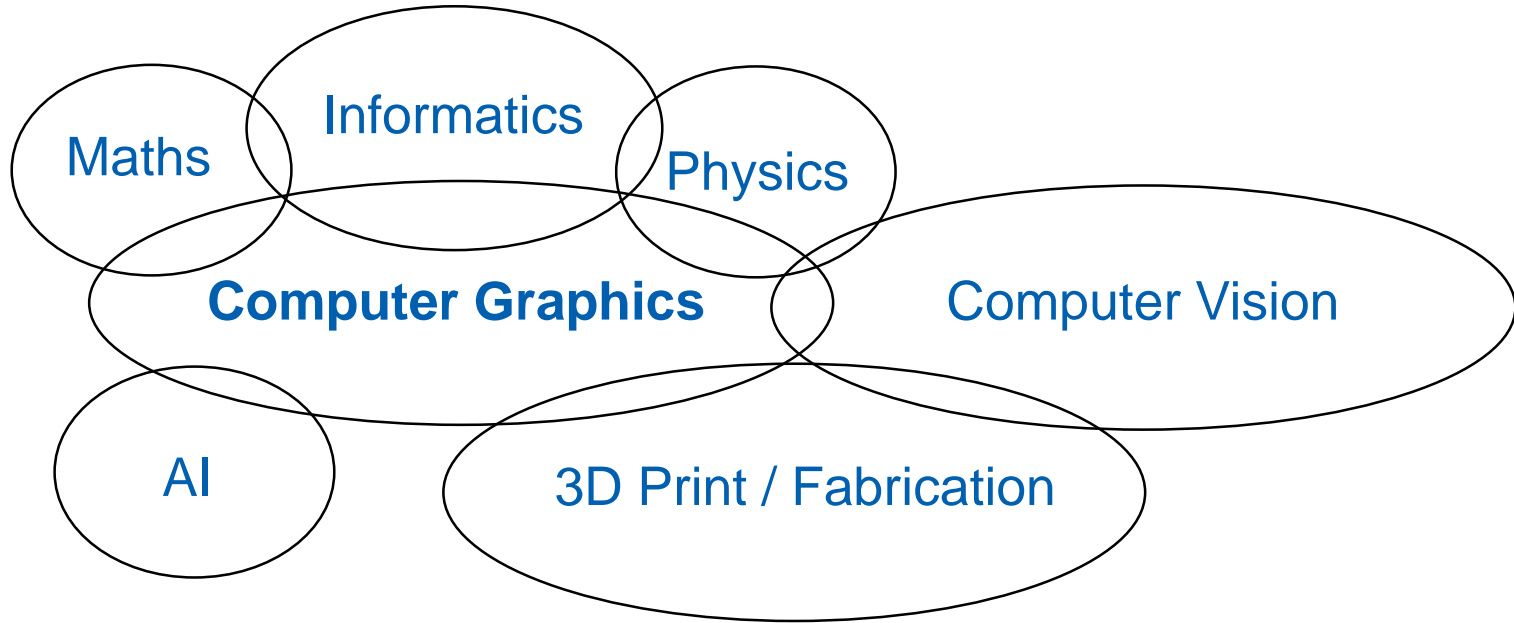
DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

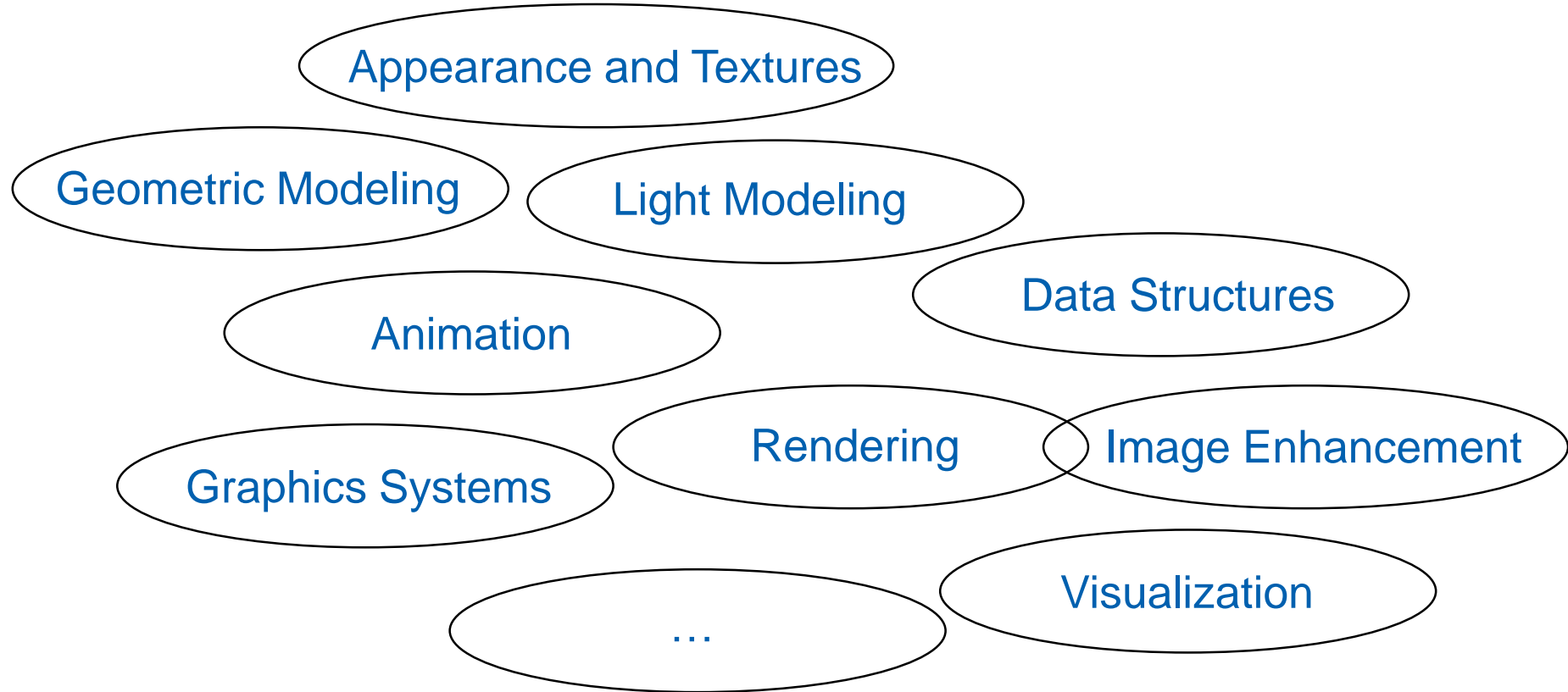
Introduction to 3D geometry

Jiří Bittner

Computer Graphics ?



Computer Graphics Fields



Outline

- Points, Vectors, Transformations MPG – chapter 21
- Camera and Projection MPG – chapter 9
- 3D Scene Representation MPG - chapters 5.11,
5.12, 5.13, 6-8, 14

Points in 3D

- Point is a location in 3D space
- Described by positional vector

- Cartesian coordinates
 - Orthonormal basis

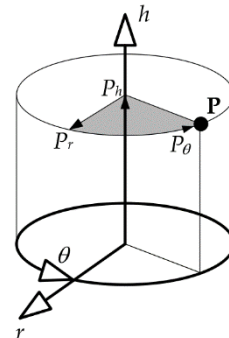
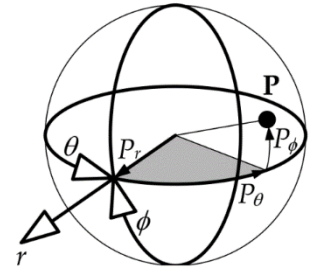
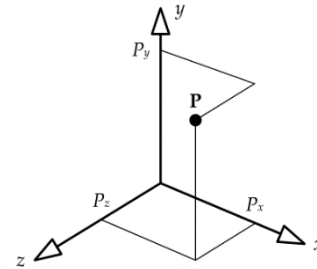
$$P = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

- Spherical coordinates

$$P = \begin{bmatrix} 90^\circ \\ 20^\circ \\ 1 \end{bmatrix}$$

- Cylindrical coordinates

$$P = \begin{bmatrix} 90^\circ \\ 3 \\ 1 \end{bmatrix}$$



Cartesian coordinates

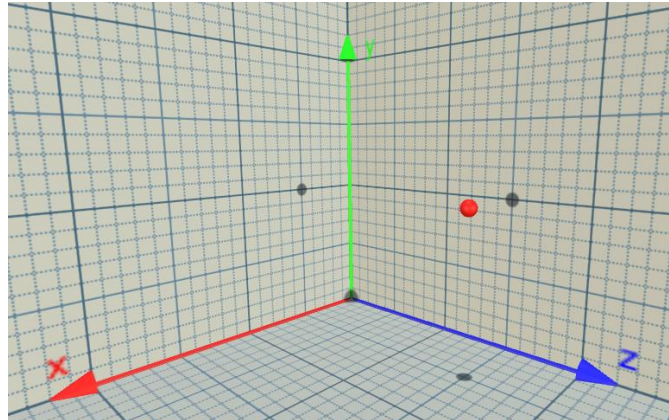
■ Axes

- Orthogonal directions
- Meet at origin
- Uniform scale
- Orthogonal basis

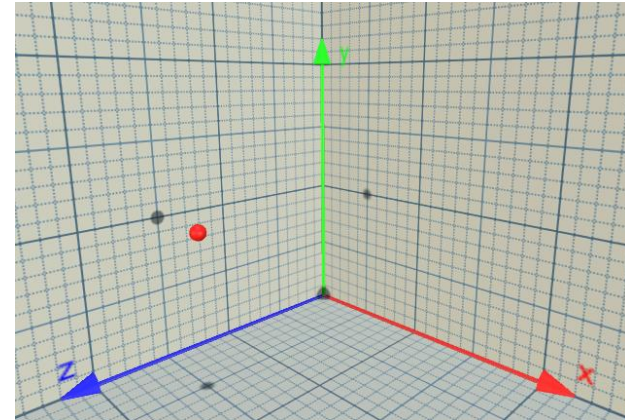
(René Descartes, 1596-1650)

$$A = [5, 10, 15]$$

$$A = \begin{bmatrix} 5 \\ 10 \\ 15 \end{bmatrix}$$



LHS
Direct3D, Unity, ...



RHS
OpenGL

Example: Spherical / Cartesian Conversion

$$r = \sqrt{x^2 + y^2 + z^2}$$

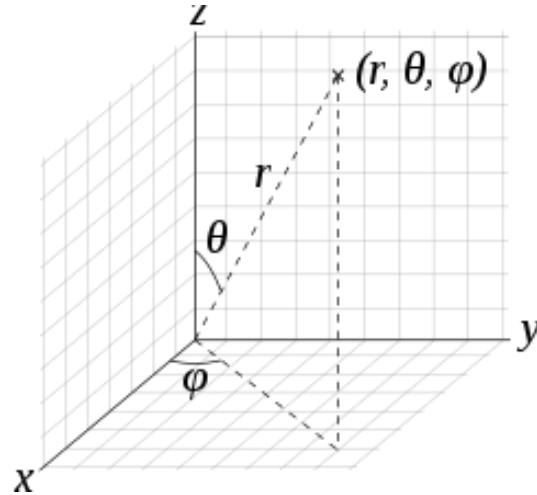
$$\theta = \arccos \frac{z}{r}$$

$$\phi = \arctan \frac{y}{x}$$

$$x = r \cos \phi \sin \theta$$

$$y = r \sin \phi \sin \theta$$

$$z = r \cos \theta$$



https://en.wikipedia.org/wiki/Spherical_coordinate_system

Advanced: Cartesian-Geodesic transformation

<https://www.degruyter.com/document/doi/10.2478/jogs-2013-0012/html>

Linear Transformations

- Scale (Uniform + Non-uniform)
- Mirror
- Shear
- Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 3 \times 3 \\ \text{transformation} \\ \text{matrix} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

- Need also: translation, perspective projection
- Add 4-th coordinate

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \approx \begin{bmatrix} x_h \\ y_h \\ z_h \\ w \end{bmatrix} \quad x = \frac{x_h}{w}, y = \frac{y_h}{w}, z = \frac{z_h}{w}$$

- For directions (points in infinity) $w = 0$

Transformation in homogeneous coordinates

- Finally normalize (divide by w)
 - Perspective division

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 4x4 \\ \text{transformation} \\ \text{matrix} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} \frac{x'}{w'} \\ \frac{y'}{w'} \\ \frac{z'}{w'} \end{bmatrix}$$

Composing transformations

- Matrix multiplication

$$M = R \cdot T \cdot S \dots$$

- Associative

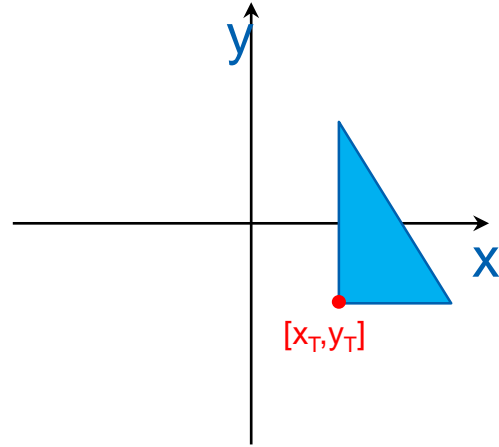
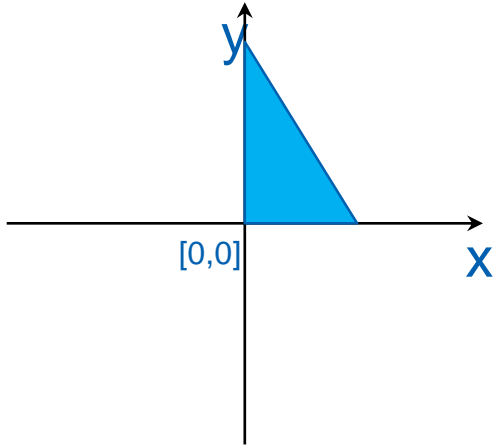
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

- Non-commutative: Transformation order matters!

$$A \cdot B \neq B \cdot A$$

- Transformations applied from right to left

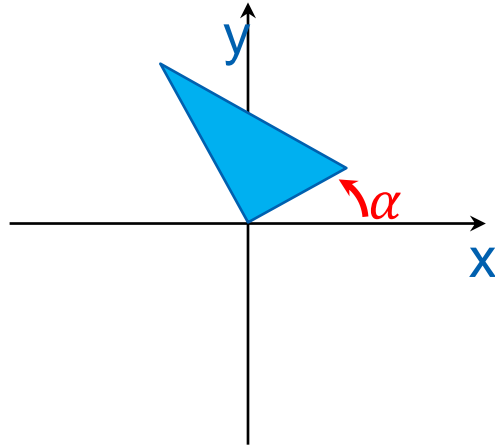
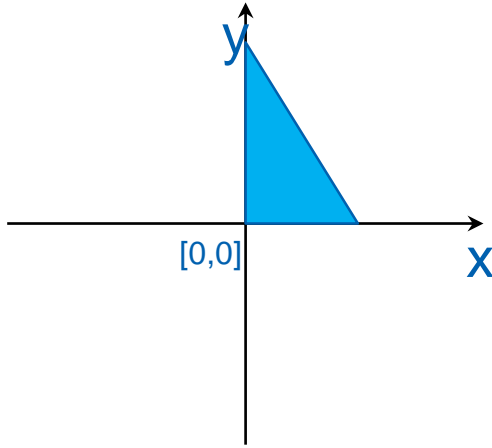
Translation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

$$M_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_{R_z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

$$M_{R_z} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euler angles

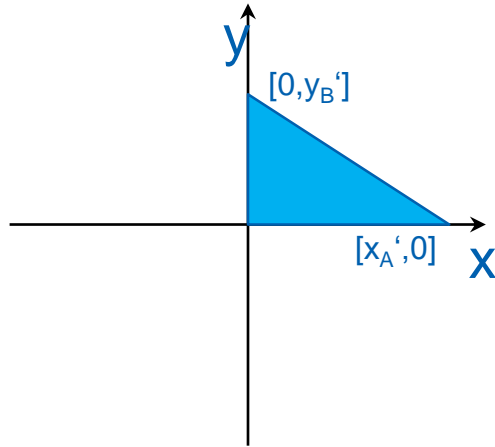
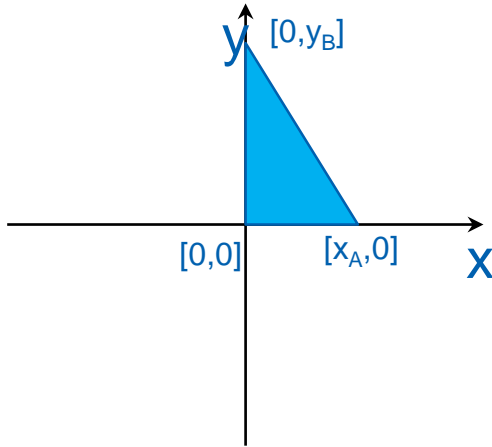
$$M = M_{R_y} M_{R_x} M_{R_z} \text{ (order in Unity)}$$

Euler Angles Rotation

$$M = M_{R_y} M_{R_x} M_{R_z} \text{ (order in Unity)}$$

$$M = \begin{bmatrix} \cos \alpha_y & 0 & \sin \alpha_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha_y & 0 & \cos \alpha_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_x & -\sin \alpha_x & 0 \\ 0 & \sin \alpha_x & \cos \alpha_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \alpha_z & -\sin \alpha_z & 0 & 0 \\ \sin \alpha_z & \cos \alpha_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

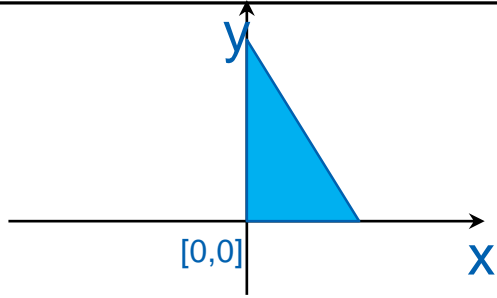
Scaling



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_S \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

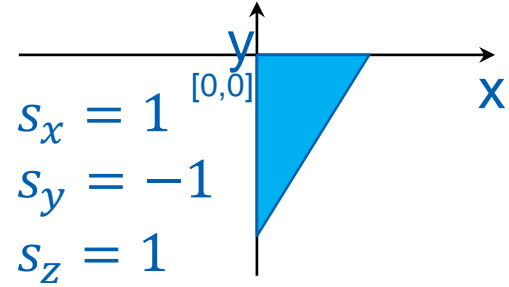
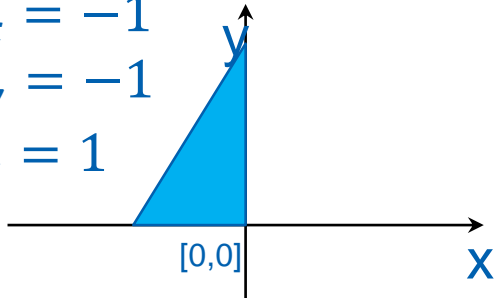
$$M_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Symetry



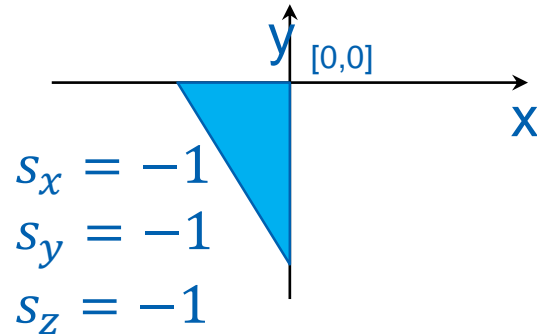
$$M_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} s_x &= -1 \\ s_y &= -1 \\ s_z &= 1 \end{aligned}$$



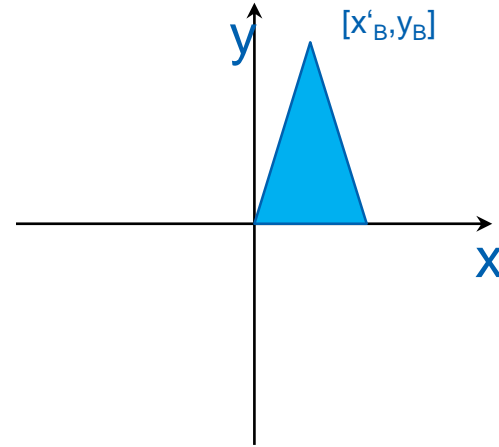
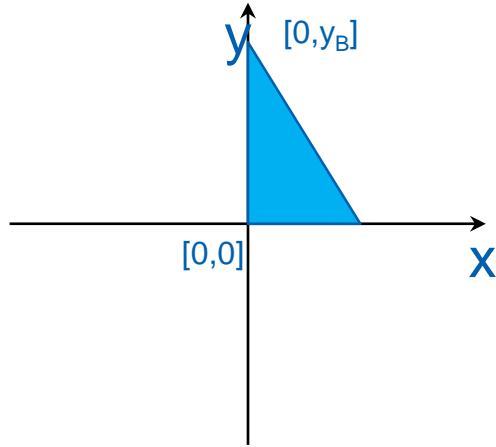
$$\begin{aligned} s_x &= 1 \\ s_y &= -1 \\ s_z &= 1 \end{aligned}$$

Try to avoid: Odd number of -1 flips polygon orientations!



$$\begin{aligned} s_x &= -1 \\ s_y &= -1 \\ s_z &= -1 \end{aligned}$$

Shear

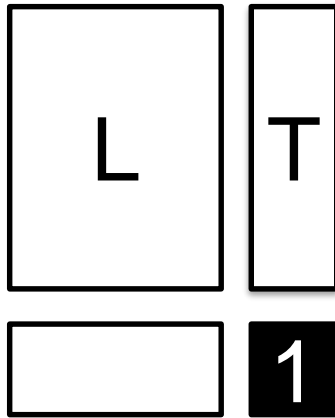


$$M_{SH_x} = \begin{bmatrix} 1 & SH_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SH_x = \frac{x'_B}{y_B}$$

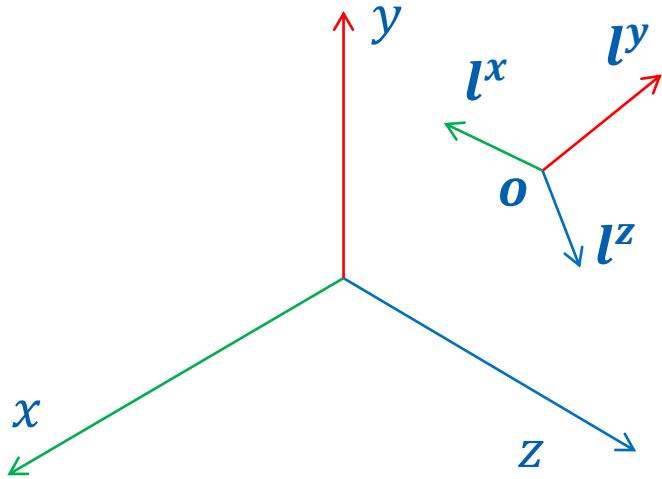
Transformation matrix

- L: linear transformation
- T: translation
- Last row (0, 0, 0, 1) for all affine transformations



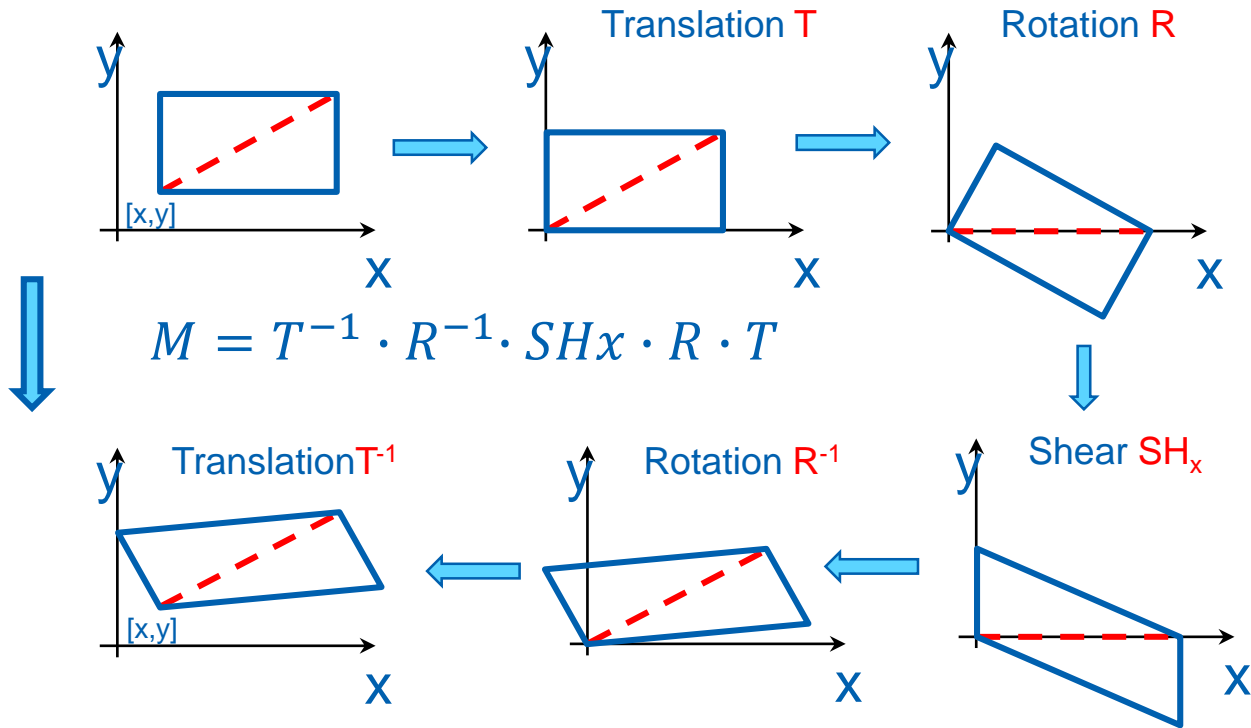
Local Coordinate Frame

- Local Coordinate Basis + Translation



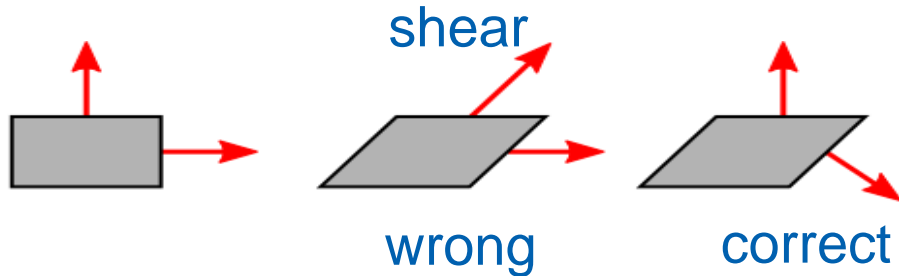
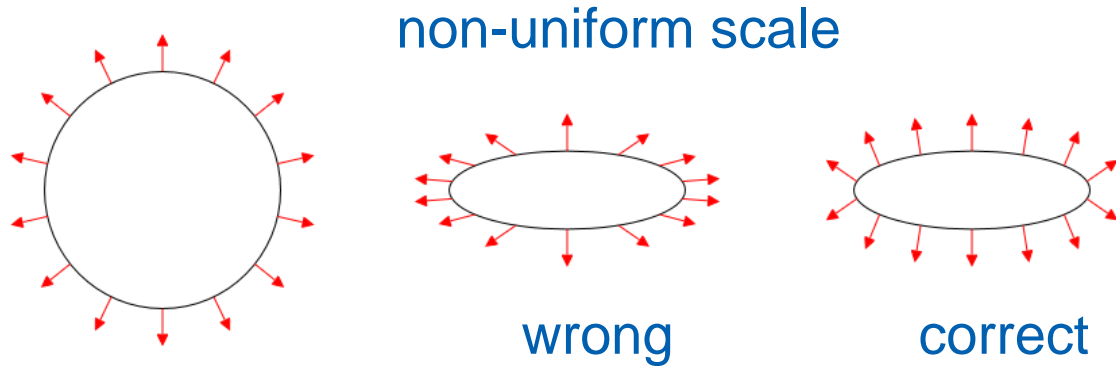
$$M_{local \rightarrow world} = \begin{bmatrix} l_x^x & l_x^y & ? l_x^z & ? \\ l_y^x & l_y^y & ? l_y^z & ? \\ l_z^x & l_z^y & ? l_z^z & ? \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} o_x \\ o_y \\ o_z \\ 1 \end{bmatrix}$$

Example – shear along a diagonal

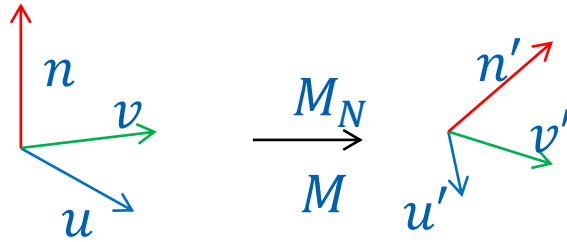


Transforming normals

- Normal – vector perpendicular to the surface



Transforming normals



$$\begin{bmatrix} n_x' \\ n_y' \\ n_z' \\ 0 \end{bmatrix} = M_N \begin{bmatrix} n_x \\ n_y \\ n_z \\ 0 \end{bmatrix}$$

$$n^T u = 0$$

$$n'^T u' = 0$$

$$n'^T M u = 0$$

$$n'^T M u = n^T u$$

$$n'^T M = n^T$$

$$M^T n' = n$$

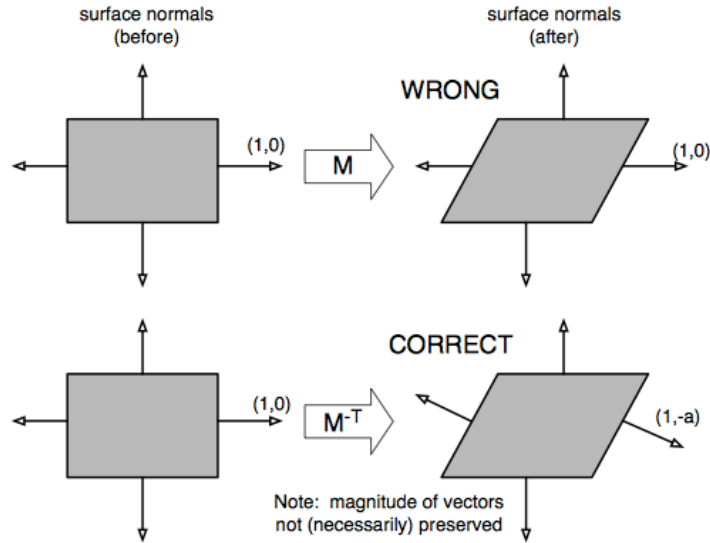
$$n' = M^{T^{-1}} n$$

$$n' = M^{-1T} n$$

Can use just 3x3 submatrix of M !

Transforming normals - example

$$M = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \quad M^{-T} = \begin{bmatrix} 1 & 0 \\ -a & 1 \end{bmatrix}$$

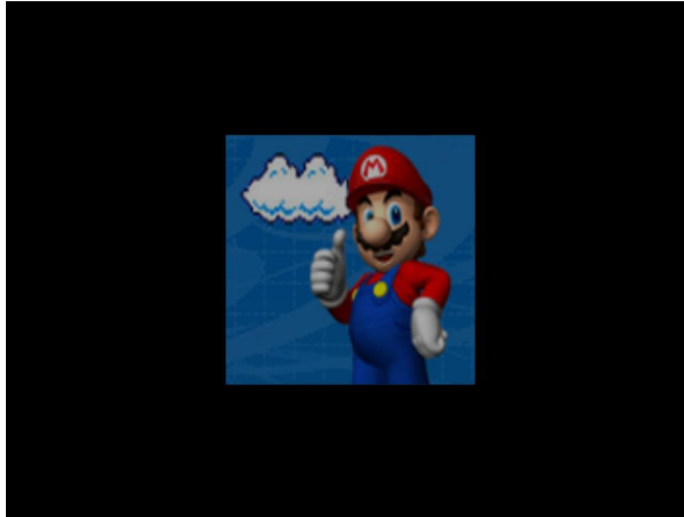


Source: Stack Overflow

DEMO

<https://cent.felk.cvut.cz/predmety/39PHA/demos/transformations.html>

Transformation example



Model matrix

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Reset

View matrix (read only)

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Reset

Quaternions

- Alternative rotation representation
- Generalization of complex numbers
 - three basis elements i, j, k
 - $i^2 = j^2 = k^2 = i j k = -1, ij = -ji = k, jk = -kj = i, ki = -ik = j$
- Quaternion is a 4-tuple

William Rowan Hamilton 1843



$$\mathbf{q} = [x, y, z, w]$$

$$\mathbf{q} = i x + j y + k z + w = [\mathbf{v}, w]$$

$$\mathbf{v} = [x, y, z] = i x + j y + k z$$

Quaternions and Rotation

- Unit quaternion ($|q| = 1$) represents rotation in 3D

$$q = \left[a \sin \frac{\alpha}{2}, \cos \frac{\alpha}{2} \right]$$

3D rotation about axis a by angle α

Quaternion Operations

- Sum $q_1 + q_2 = [v_1 + v_2, w_1 + w_2]$

- Dot product $q_1 \cdot q_2 = v_1 \cdot v_2 + w_1 \cdot w_2$

- **Multiplication** (Hamilton product)

$$q_1 * q_2 = [v_1, w_1] * [v_2, w_2] = [w_1 v_2 + w_2 v_1 + v_1 \times v_2, w_1 w_2 - v_1 \cdot v_2]$$

- **Composition of rotations** (associative, non-commutative)

- **Conjugate**

$$q^* = [-v, w]$$

- Inverse rotation

Transformation with quaternion

- Express vector as quaternion

$$\mathbf{u} = (x, y, z, 0)$$

- Rotation of \mathbf{u} using q

$$\mathbf{u}' = (x', y', z', 0) = q * \mathbf{u} * q^*$$

- Two quaternion multiplications + conjugate

Quaternion to Rotation Matrix

- Quaternion $\mathbf{q} = [x, y, z, w]$ corresponds to rotation matrix

$$\mathbf{R} = \begin{pmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{pmatrix}$$

- Rotation composition faster with quaternions
- Vector transformation faster with matrix

Rotation Interpolation

- Matrix interpolation
 - Breaks orthonormality - artefacts
- Quaternion interpolation
 - Linear interpolation (LERP)
 - Spherical linear interpolation (SLERP) - constant angular step

LERP and SLERP

$$q = \frac{w_A q_A + w_B q_B}{|w_A q_A + w_B q_B|}$$

LERP

$$w_A = 1 - \beta$$

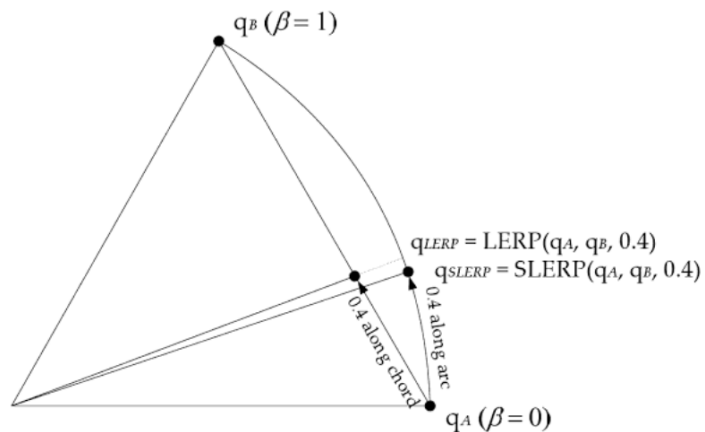
$$w_B = \beta$$

SLERP

$$w_A = \frac{\sin(1 - \beta)\theta}{\sin \theta}$$

$$w_B = \frac{\sin \beta\theta}{\sin \theta}$$

$$\theta = \arccos q_A q_B$$



J. Gregory, Game Engine Architecture

Transformation Representation - SQT

- SQT (SRT)
 - Scale, Quaternion, Translation
- Uniform scale: $1+4+3 = 8$ scalars
 - Sequence of SQT can be composed to SQT
- Non-uniform scale: $3+4+3=10$ scalars
- Correct interpolation of rotation, scale, and translation !
- Compact representation
- Fast composition of transformations
- Slower application of transformation

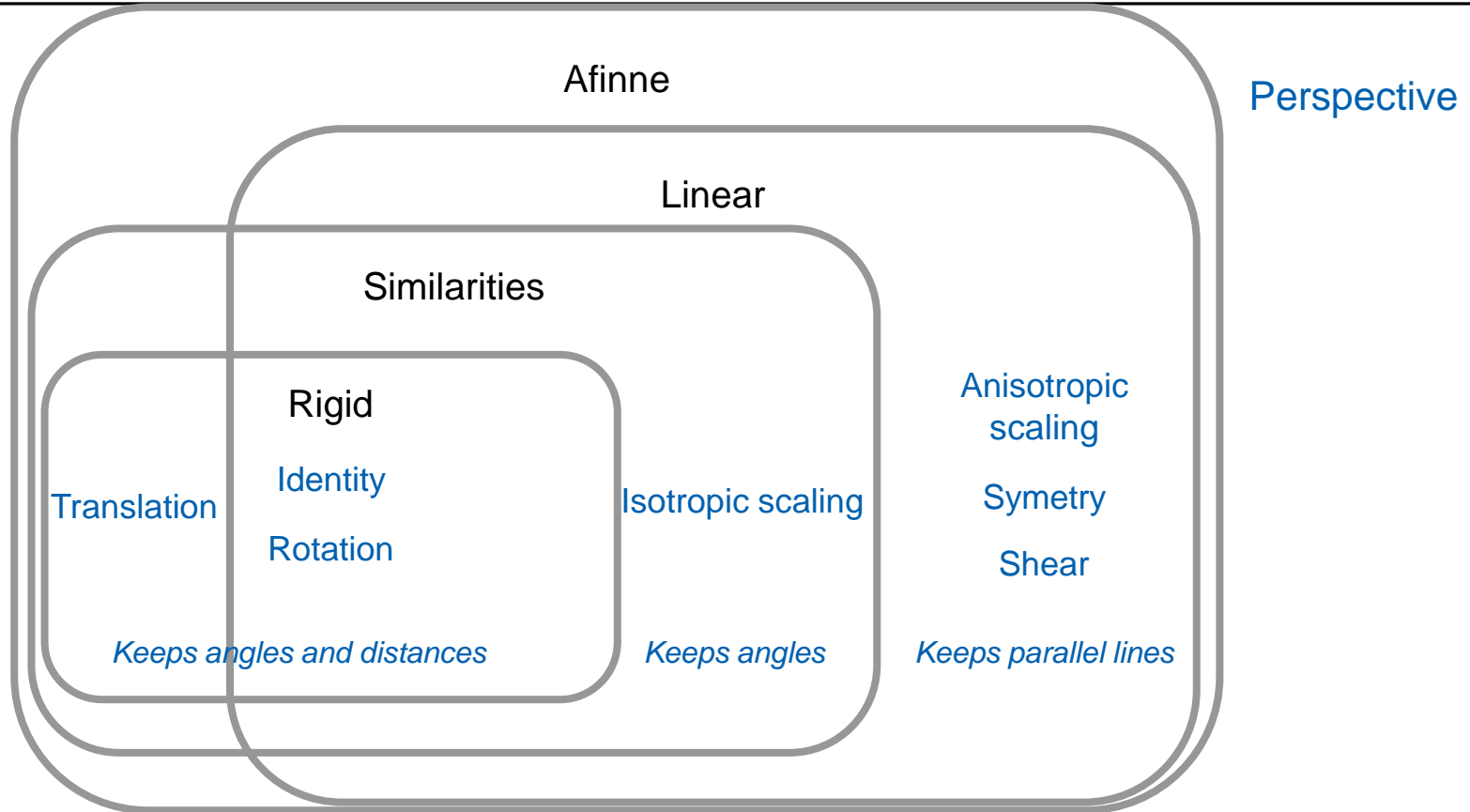
Transformation Representation - Matrix

- Matrix 4x4
- General affine transformation + perspective
- Simple concatenation (matrix multiplication)
- Fast application of transformation

Transformation – Summary

- Interpolation and composition of rotations – use quaternions (animations)
- Applying (many) transformations – use matrices (rendering)
- Conversions between representations

Transformations

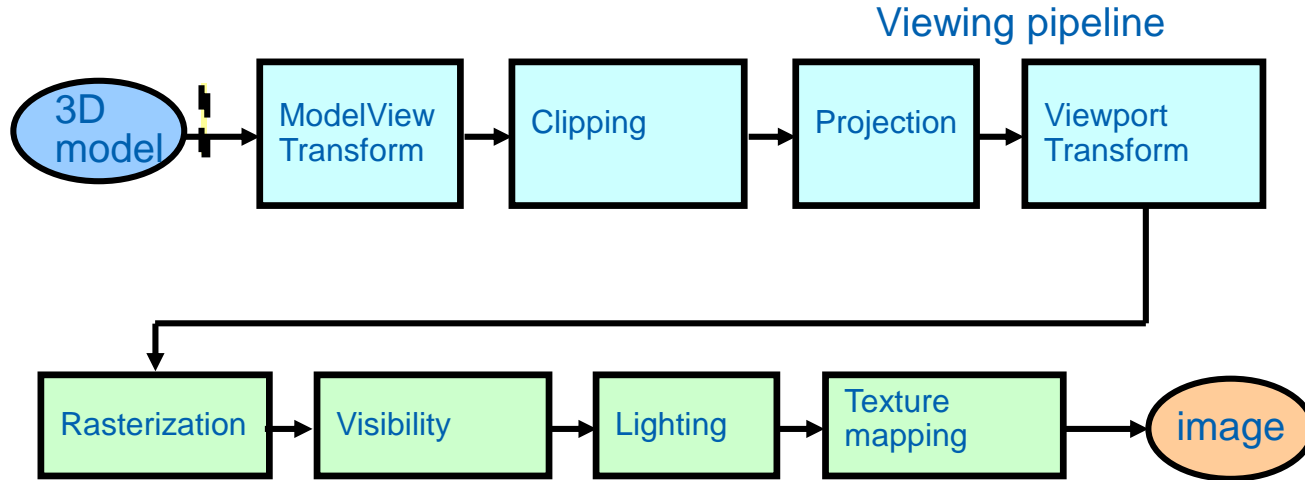


Outline

- Points, Vectors, Transformations MPG – chapter 21
- Camera and Projection MPG – chapter 9
- 3D Scene Representation MPG - chapters 5.11,
5.12, 5.13, 6-8, 14

Rendering Pipeline

- 1. part – transformations (*viewing pipeline*)
- 2. part – further operations



Coordinate systems overview

- Object / Modeling / Local coordinates
 - Relative to object origin
- World coordinates
 - Global scene coordinates
- Camera / Eye / View coordinates
 - Camera in the origin, looks along $-z$
- Clip coordinates
 - After multiplication by projection matrix
- Normalized device coordinates
 - Cuboid after perspective division $[-1,-1,-1] - [1,1,1]$
- Screen / Window coordinates
 - x, y pixel position, z in $0..1$ range

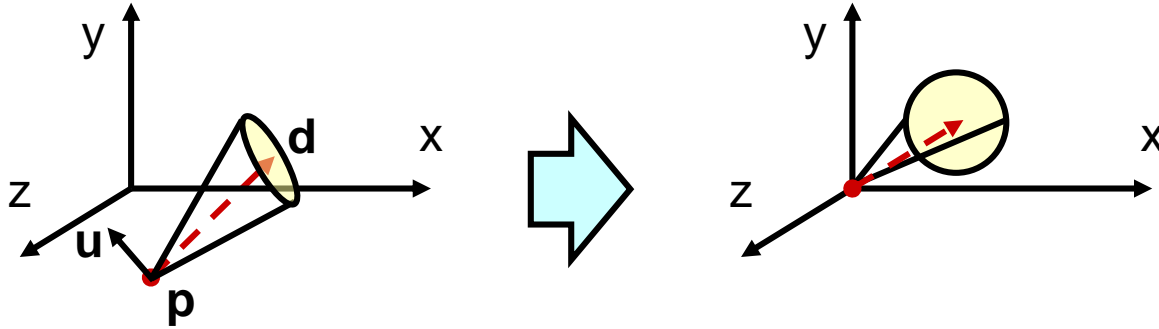


Camera

- Idealized camera (pin-hole)
 - Idealized geometric optics
 - Realistic effects (often) as post process
- Camera description
 - Explicit parameters (position, orientation)
 - Node in a scene graph
 - Other parameters – viewing angle, viewport, rendering setup, ...
- Series of transformations
 - Viewing transformation (camera position / orientation)
 - Projection transformation (viewing volume)
 - Viewport transformation (viewport on the screen)
 - Composed with the modeling transformation

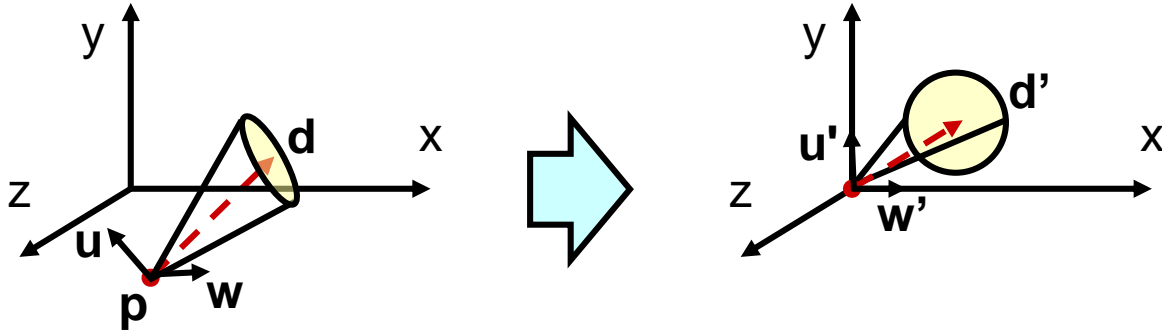
View transformation

- Transformation of scene to unified position



- Camera position p** to $[0,0,0]$... translation
- View direction d** // with z axis ... rotation
- Up vector u** // with y axis ... rotation around z axis
- Camera matrix M : Viewing transformation = M^{-1}

View transformation matrix

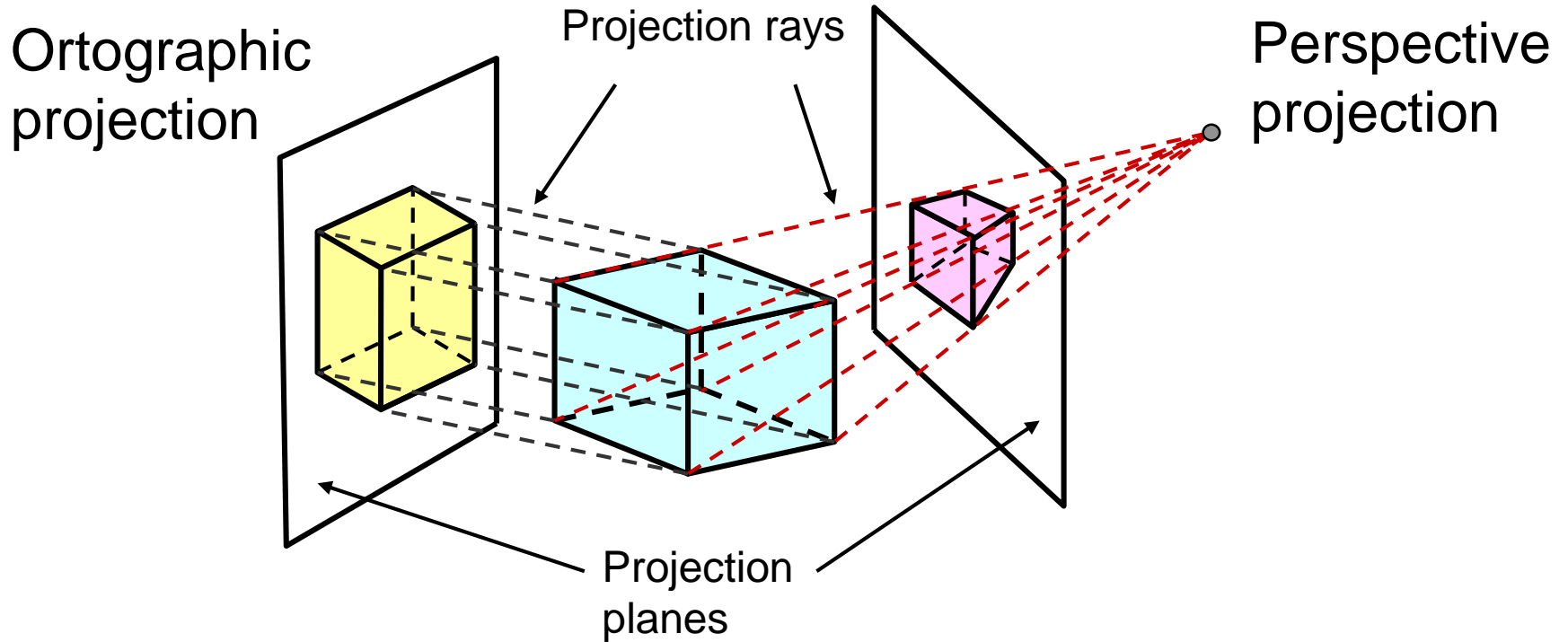


- $M_V = M_C^{-1}$ (M_C camera matrix)

$$M_V = \begin{bmatrix} w_x & u_x & d_x & p_x \\ w_y & u_y & d_y & p_y \\ w_z & u_z & d_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

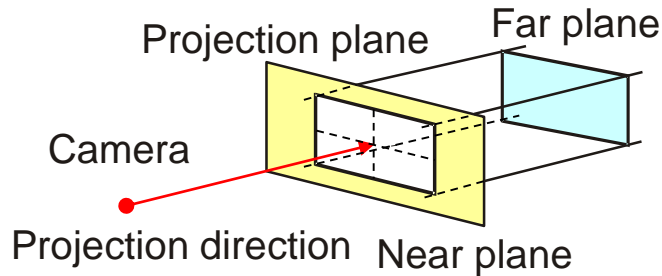
$$w = d \times u$$

Orthographic and perspective projection

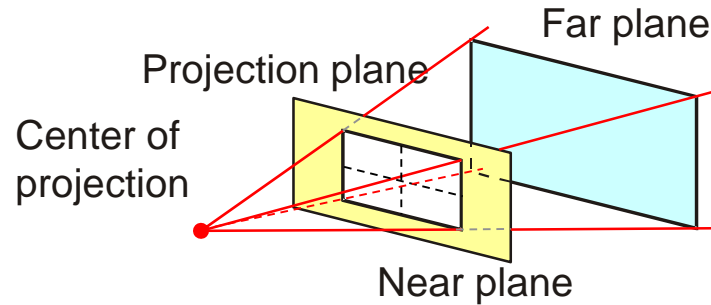


Camera – projection transformation

- Transformation from 3D space to 2D projection plane
- *Viewing volume / frustum (záběr)*



Orthographic projection
view volume = cuboid

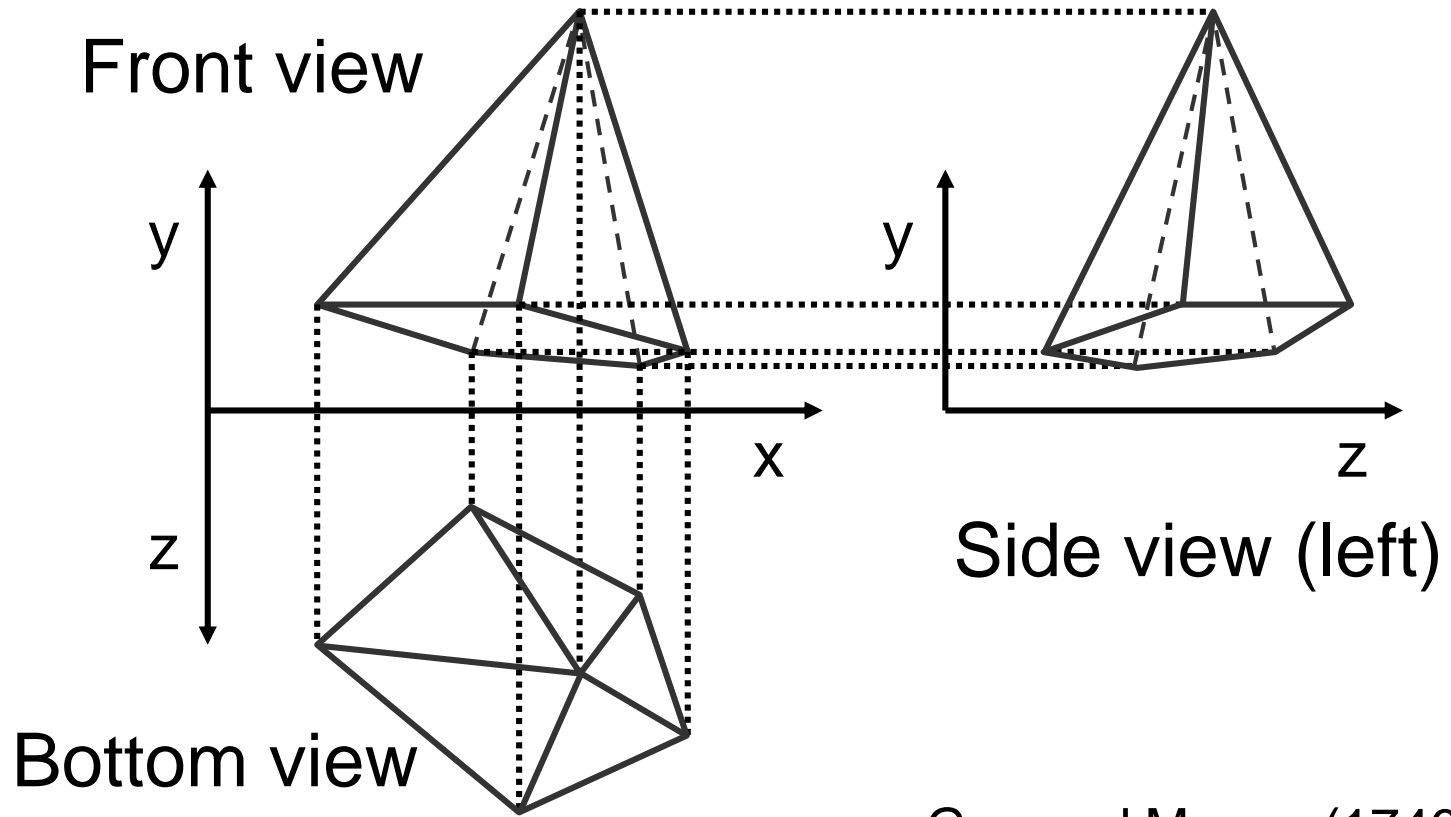


Perspective projection
view volume = pyramid frustum

Orthographic Projection

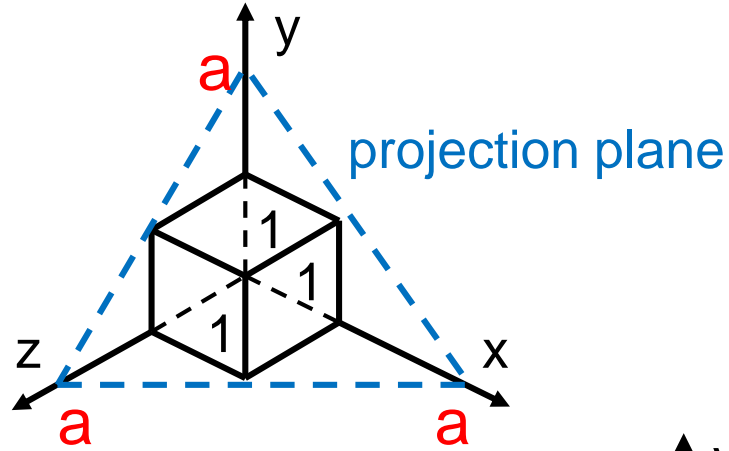
- All rays parallel!
- Rays **orthogonal** to projection plane
 - Monge's projection: top, front, side
 - Axonometry (arbitrary projection plane)
- Rays **non-orthogonal** to projection plane (oblique projection)
 - Cavalier projection (the same scale on axes)
 - Cabinet projection (z axis scale = 1/2)

Monge's projection

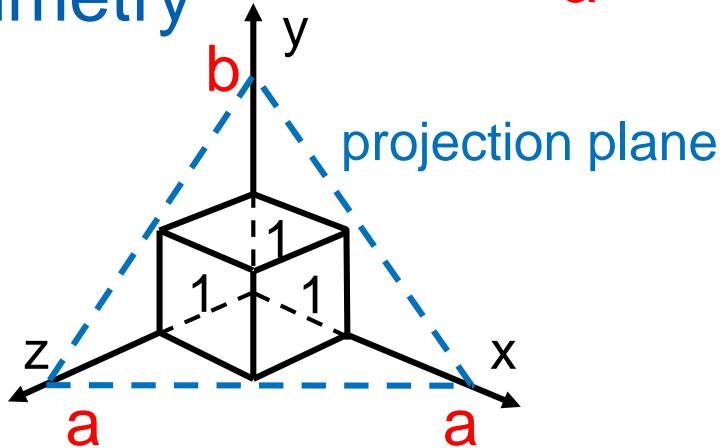


Axonometry

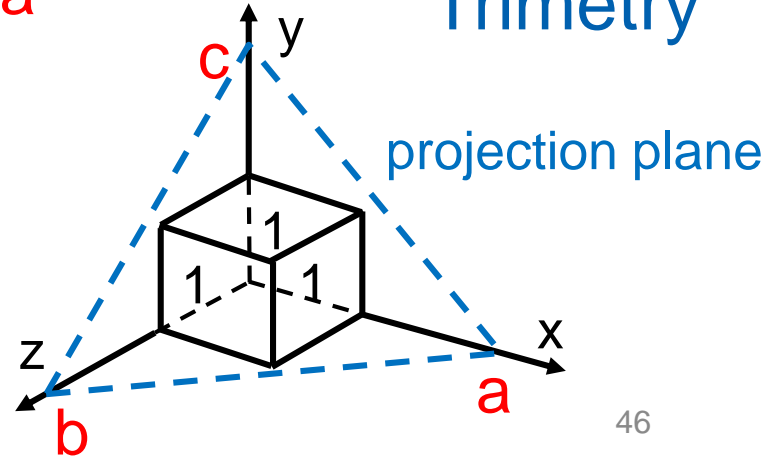
Isometry



Dimetry

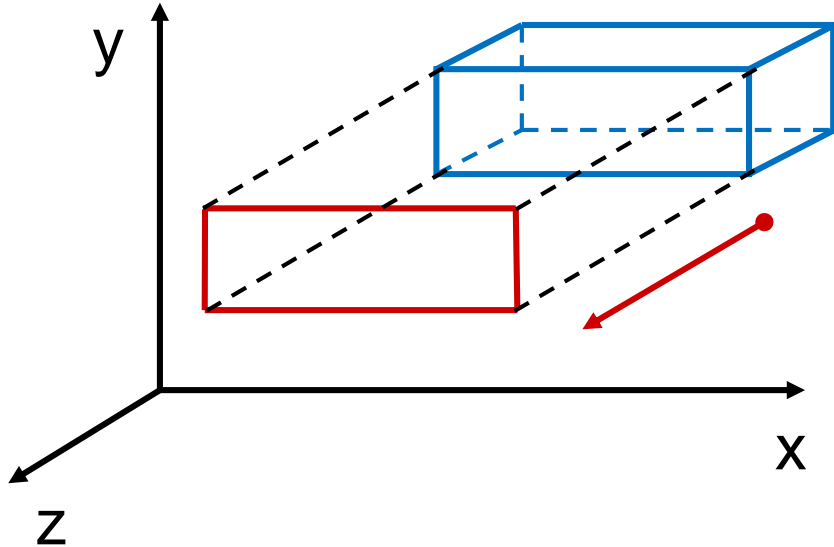


Trimetry



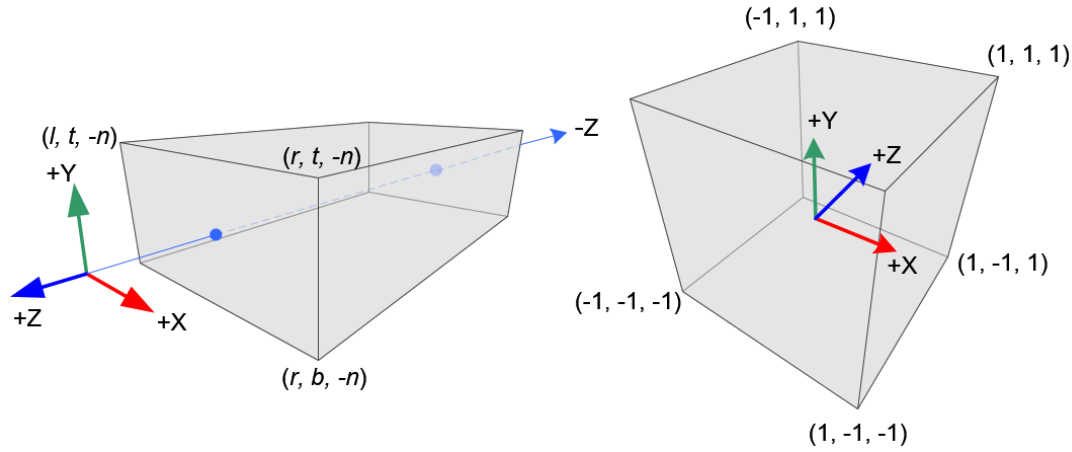
Projection: Matrix Form

- Align projection direction to z axis (rotation)
- Projection plane = xy



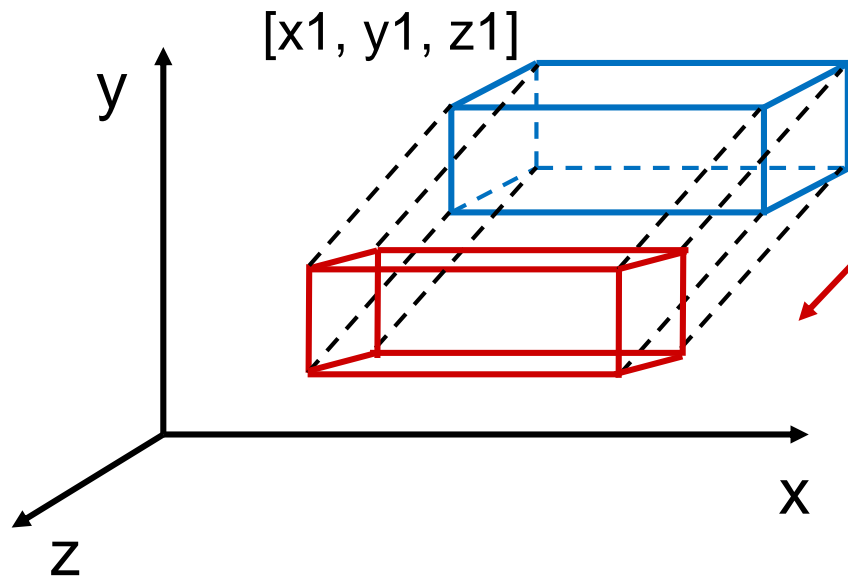
$$M_{//} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic Projection (OpenGL)



$$P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Oblique Projection



$$\begin{aligned}x &= x_1 + x_p \cdot t \\y &= y_1 + y_p \cdot t \\z &= z_1 + z_p \cdot t\end{aligned}$$

$$z = 0 \Rightarrow t = -z_1 / z_p$$

Projection direction $[x_p, y_p, z_p]$

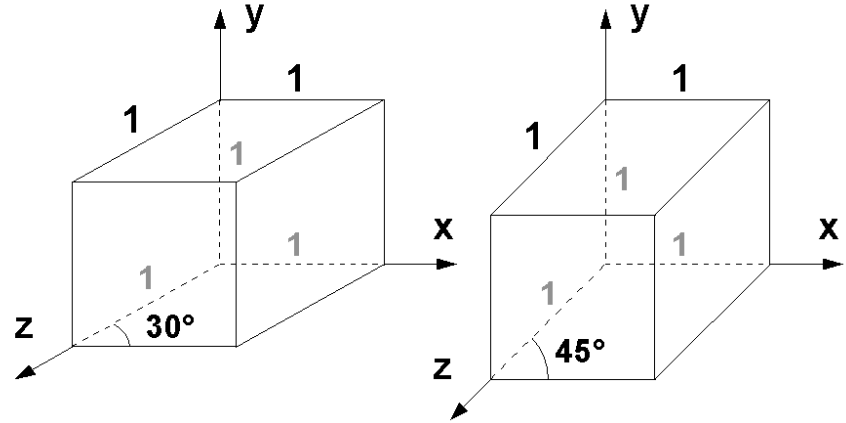
$$M = \begin{bmatrix} 1 & 0 & -\frac{x_p}{z_p} & 0 \\ 0 & 1 & -\frac{y_p}{z_p} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = M_{//} \cdot M_{zk}$$

Oblique Projection

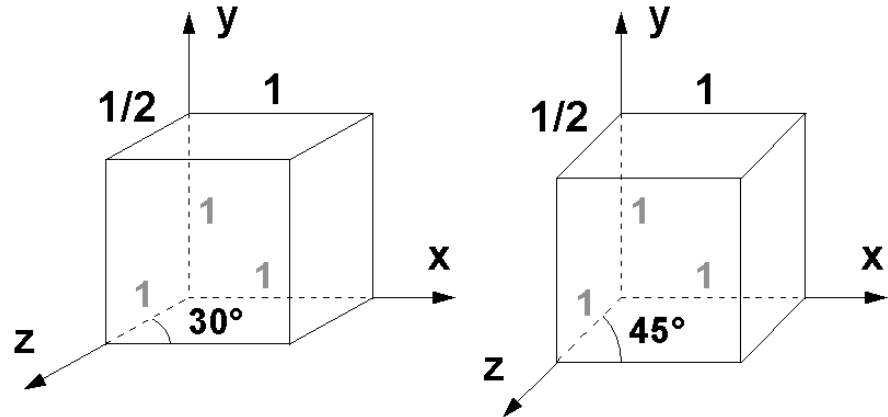
Cavalier

$$M = \begin{bmatrix} 1 & 0 & -\cos \beta & 0 \\ 0 & 1 & -\sin \beta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

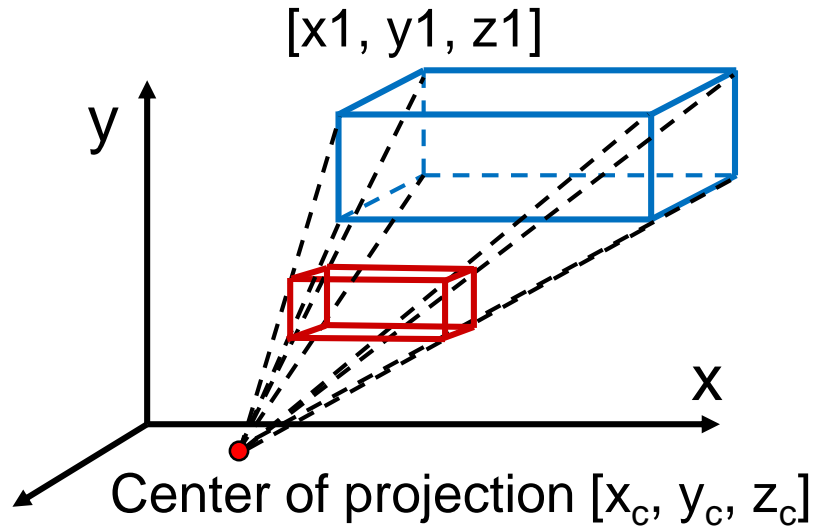


Cabinet

$$M = \begin{bmatrix} 1 & 0 & \frac{-\cos \beta}{2} & 0 \\ 0 & 1 & \frac{-\sin \beta}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Perspective Projection

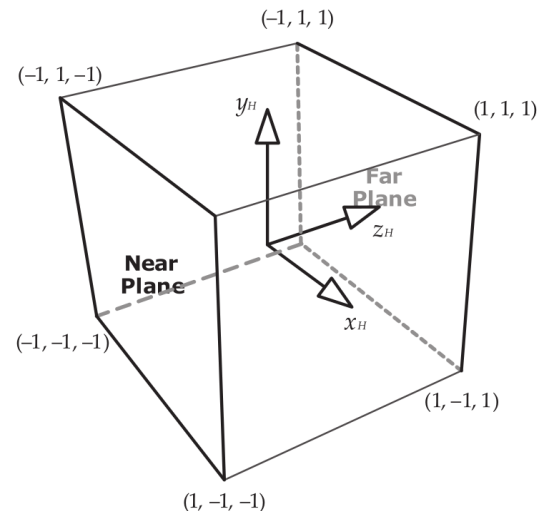
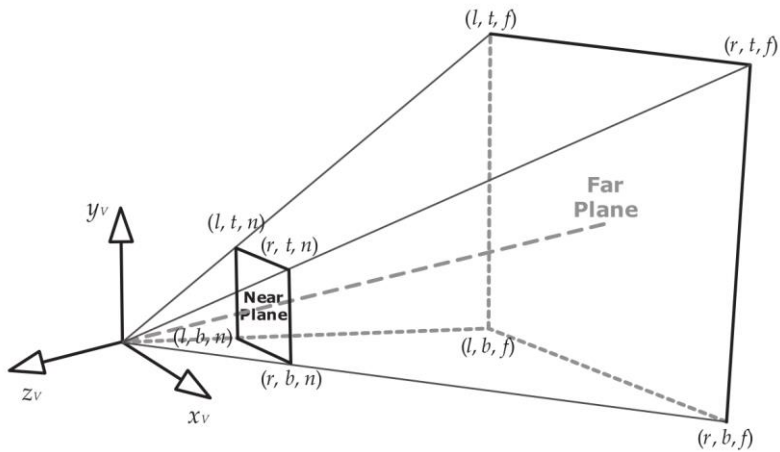


$$\begin{aligned}x &= x_c + (x_1 - x_c) \cdot t \\y &= y_c + (y_1 - y_c) \cdot t \\z &= z_c + (z_1 - z_c) \cdot t\end{aligned}$$

$$z = 0 \Rightarrow t = z_c / (z_c - z_1)$$

$$M = \begin{bmatrix} 1 & 0 & -\frac{x_c}{z_c} & 0 \\ 0 & 1 & -\frac{y_c}{z_c} & 0 \\ 0 & 0 & \frac{z_c}{z_c} & 0 \\ 0 & 0 & -\frac{1}{z_c} & 1 \end{bmatrix}$$

Perspective Projection (OpenGL)

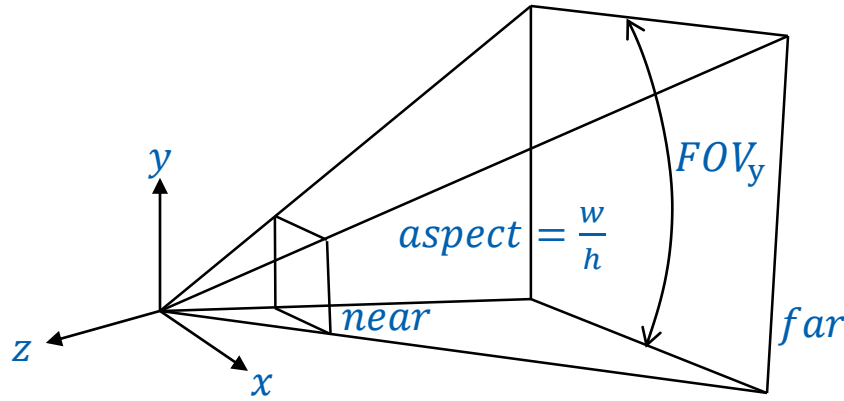


camera / eye space

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

NDC / clip space

Symmetrical Perspective Projection



$$P = \begin{bmatrix} \frac{\cotg \frac{FOV_y}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cotg \frac{FOV_y}{2} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Camera – Viewport Transformation

- Size (W, H) and position (X, Y) of the viewport

$$x' = (x_{\text{NDC}} + 1) \frac{W}{2} + X$$

$$y' = (y_{\text{NDC}} + 1) \frac{H}{2} + Y$$

$$z' = (z_{\text{NDC}} + 1) \frac{1}{2}$$

- x_{NDC} y_{NDC} z_{NDC} result of previous transf. (range -1..1)

Coordinate systems overview

- Object / Modeling / Local coordinates
 - Relative to object origin
- World coordinates
 - Global scene coordinates
- Camera / Eye / View coordinates
 - Camera in the origin, looks along $-z$
- Clip coordinates
 - After multiplication by projection matrix
- Normalized device coordinates
 - Cuboid after perspective division $[-1,-1,-1] - [1,1,1]$
- Screen / Window coordinates
 - x, y pixel position, z in $0..1$ range

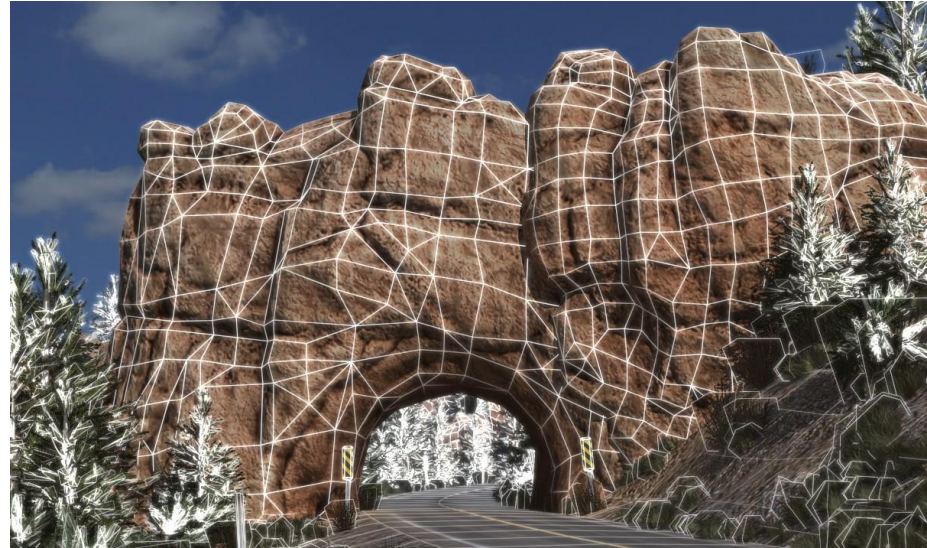


Outline

- Points, Vectors, Transformations MPG – chapter 21
- Camera and Projection MPG – chapter 9
- 3D Scene Representation MPG - chapters
5.11, 5.12, 5.13, 6-8, 14

Introduction to 3D geometry

- Scene = mathematical model of *the world* in computer
 - Rendering
 - Animation
 - Colisions
 - ...
- Geometry (3D models)
- Materials
- Lights
- Camera
- ...



3D Models

- Boundary representation (B-rep)
- Volumetric representation
- Constructive solid geometry (CSG)
- Implicit surfaces
- Point clouds
- ...

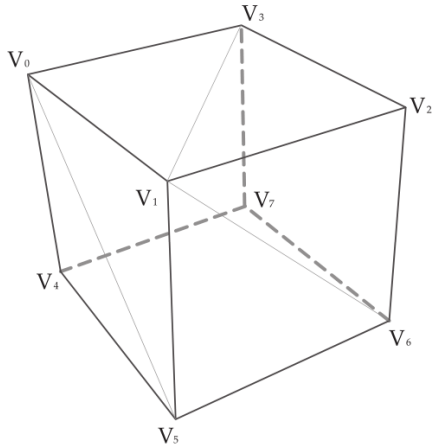
Polygonal Mesh

- Classical boundary representation
- Describes object surface (vertices, edges, faces)
- List of polygons defining object boundary (surface)
 - Better convex polygons
 - Even better just triangles (triangulation)
- Different representations
 - Sequence of vertices (separator)
 - Vertex array + index array
 - Triangle strips / fans
 - Polyhedra / Mesh graphs

Triangle Mesh

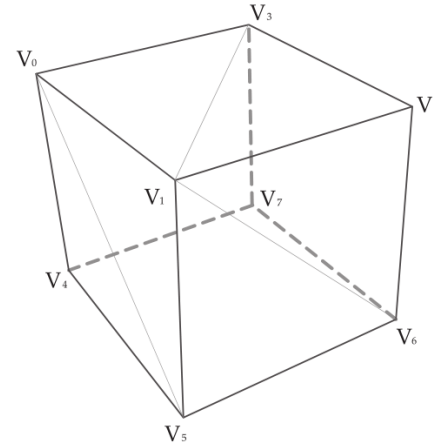
- Just triangles
 - HW friendly !
 - Simplified rendering, clipping, collisions, ...

Triangle Mesh



V ₀	V ₁	V ₃	V ₁	V ₂	V ₃	V ₀	V ₅	V ₁	...	V ₅	V ₇	V ₆
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-----	----------------	----------------	----------------

triangle list



Vertices

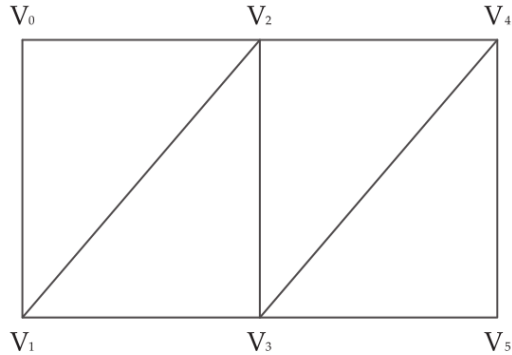
V ₀	V ₁	V ₂	V ₃	V ₄	V ₅	V ₆	V ₇
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Indices

0	1	3	1	2	3	0	5	1	...	5	7	6
---	---	---	---	---	---	---	---	---	-----	---	---	---

indexed triangle list

Triangle Mesh – Compact Representation



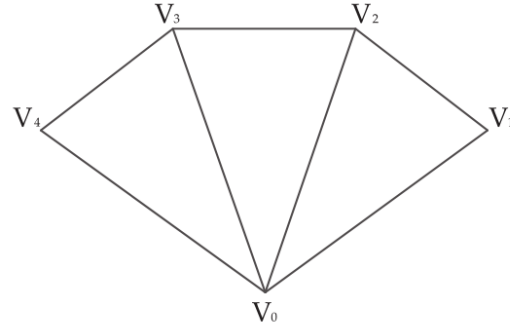
Vertices

V ₀	V ₁	V ₂	V ₃	V ₄	V ₅
----------------	----------------	----------------	----------------	----------------	----------------

Interpreted as triangles:

0 1 2	1 3 2	2 3 4	3 5 4
-------	-------	-------	-------

triangle strip



Vertices

V ₀	V ₁	V ₂	V ₃	V ₄
----------------	----------------	----------------	----------------	----------------

Interpreted as triangles:

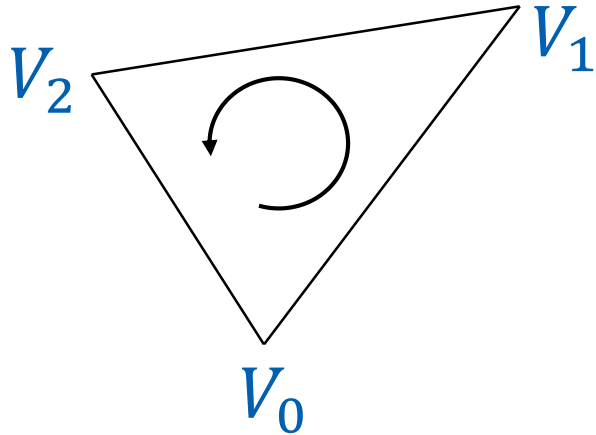
0 1 2	0 2 3	0 3 4
-------	-------	-------

triangle fan

No indices – saves memory!

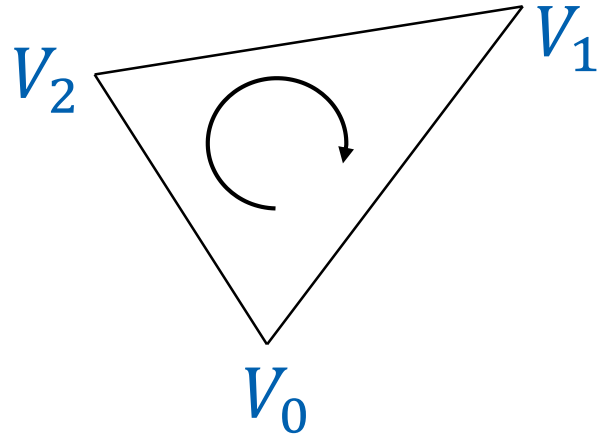
Triangle Mesh – Winding Order

- Defining front and back faces



$V_0V_1V_2$

CCW (counter clock wise)



$V_0V_2V_1$

CW (clock wise)

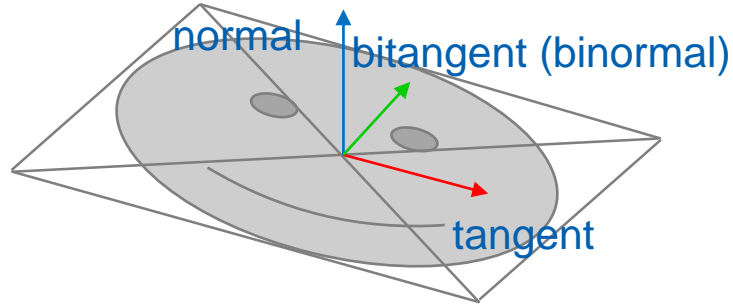
Storing Other Information

- Vertices
 - Position
 - Normal, tangent, bitangent (binormal)
 - Texture coordinates, (Color)
 - Skinning: joint id, weights, ...
- Edges
 - sharp, auxiliary
- Faces
 - normal, material
- Solids
 - material, texture

Triangle Mesh

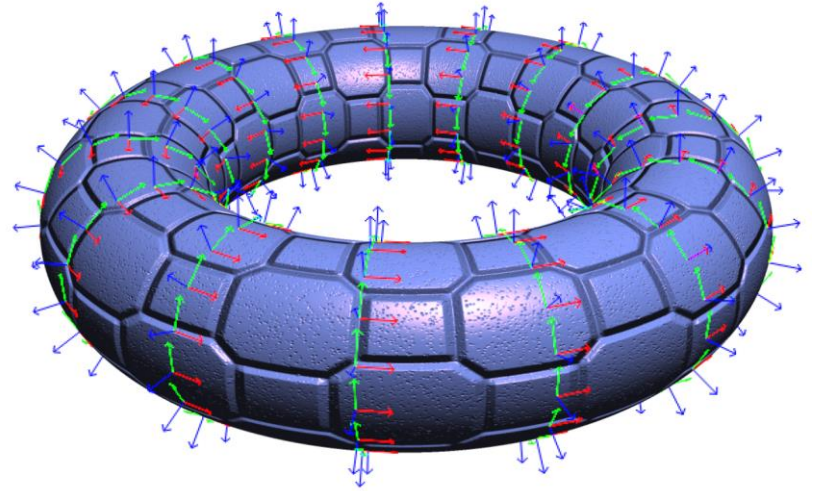
- Mesh processing / optimization
 - Computing normal, tangents, bitangents
 - Computing texture params (mesh parametrization, unwrap)
 - Sharing vertices (welding)
 - Optimizing vertex order (GPU transform cache utility)
 - Mesh compression (lossy coding of positions, normals, uvs)
 - Unity - low: 20/8/16 bits, high: 10/6/8

Triangle Mesh – Tangent Space



$$\mathbf{t} \sim \frac{\delta u}{\delta \mathbf{p}} = \begin{bmatrix} \frac{\delta u}{\delta x} & \frac{\delta u}{\delta y} & \frac{\delta u}{\delta z} \end{bmatrix}$$

$$\mathbf{b} \sim \frac{\delta v}{\delta \mathbf{p}} = \begin{bmatrix} \frac{\delta v}{\delta x} & \frac{\delta v}{\delta y} & \frac{\delta v}{\delta z} \end{bmatrix}$$



Source: Mikkelsen 2008, Simulation of Wrinkled Surfaces Revisited

Triangle Mesh – Tangent Space

$$\Delta \mathbf{p}_1 = \mathbf{p}_1 - \mathbf{p}_0$$

$$\Delta \mathbf{p}_2 = \mathbf{p}_2 - \mathbf{p}_0$$

$$\Delta \mathbf{u}_1 = \mathbf{u}_1 - \mathbf{u}_0$$

$$\Delta \mathbf{u}_2 = \mathbf{u}_2 - \mathbf{u}_0$$

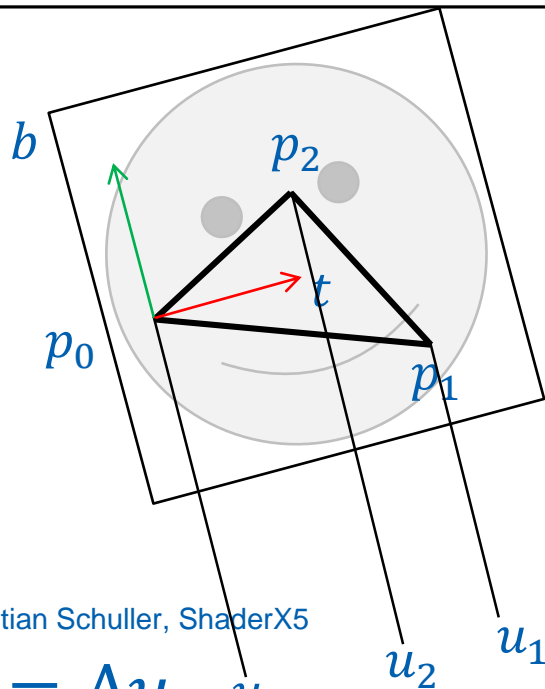
$$\mathbf{t} = \begin{bmatrix} \Delta \mathbf{p}_1 \\ \Delta \mathbf{p}_2 \\ \Delta \mathbf{p}_1 \times \Delta \mathbf{p}_2 \end{bmatrix}^{-1} \begin{bmatrix} \Delta \mathbf{u}_1 \\ \Delta \mathbf{u}_2 \\ 0 \end{bmatrix}$$

Source: Christian Schuller, ShaderX5

$$\Delta \mathbf{p}_1 \mathbf{t} = \Delta \mathbf{u}_1$$

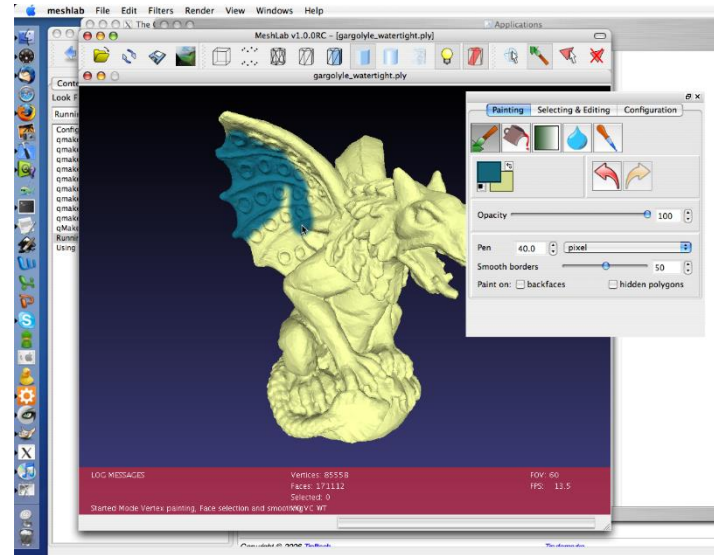
$$\Delta \mathbf{p}_2 \mathbf{t} = \Delta \mathbf{u}_2$$

$$(\Delta \mathbf{p}_1 \times \Delta \mathbf{p}_2) \mathbf{t} = 0$$

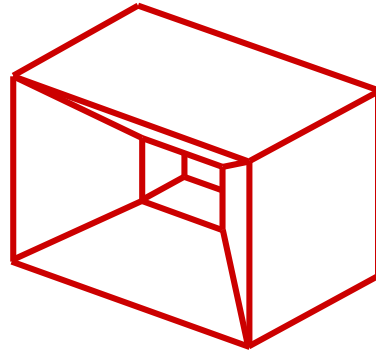
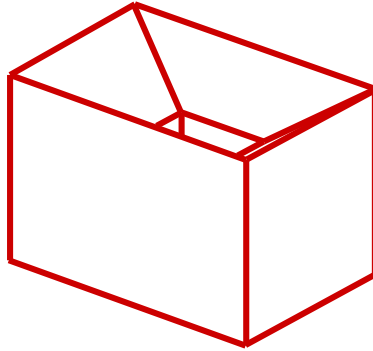
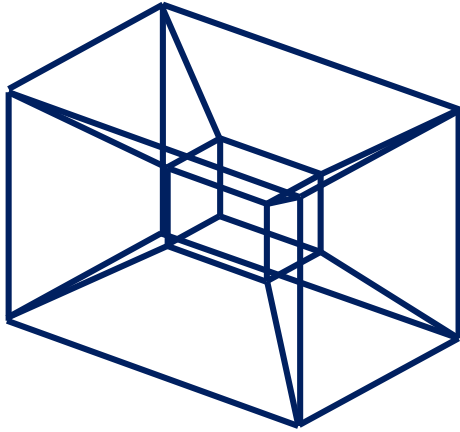


Triangle Mesh

- Modeling Tools
 - Maya, 3DS Max, Blender, Cinema
- Processing / Optimization Tools
 - MeshLab
 - NvTriStrip
 - ...



Wireframe Model

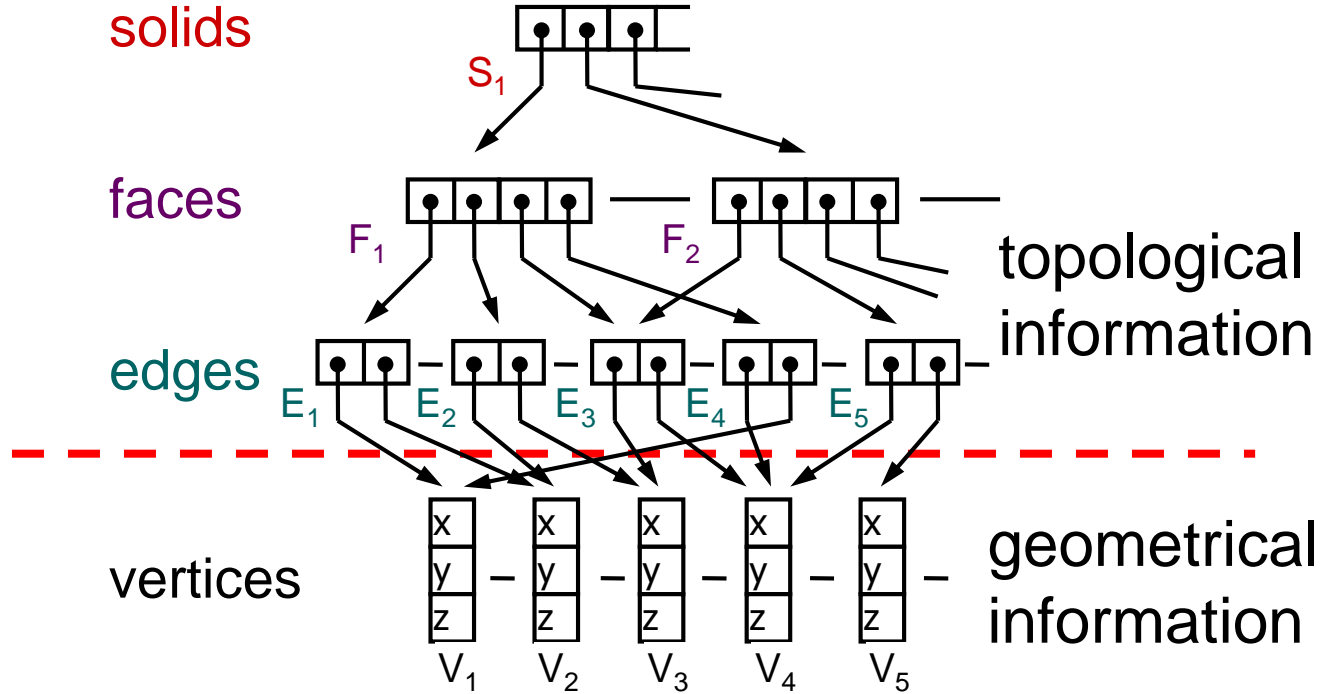


But very useful
for debugging!

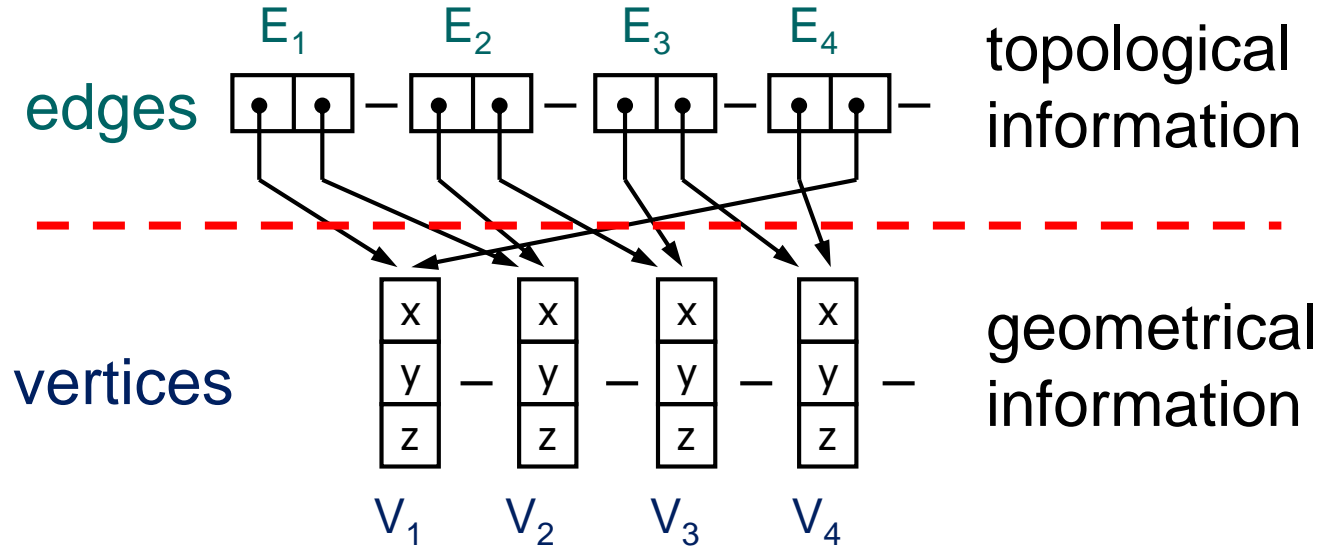
Ambiguous interpretation

Mesh Graph

Hierarchical representation



Wireframe Model

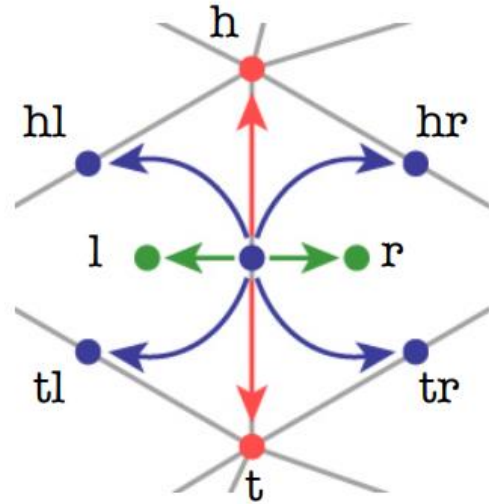


Face Types

- Planar vs Curved
 - Planar: polygon mesh
 - Curved: Bezier patches, NURBS
- Single sided vs Double sided
 - Face orientation based on vertex order: ccw / cw
- Faces with holes
 - One outer boundary + several inner boundaries
 - Auxiliary edges connecting the boundaries

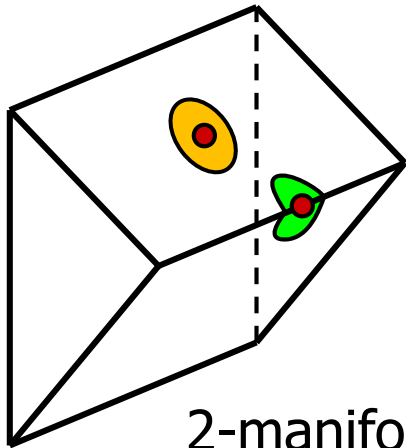
Winged-Edge: Finding edges of a face

```
EdgesOfFace(f) {  
  e = f.e;  
  do {  
    if (e.l == f) {  
      e = e.hl;  
    } else {  
      e = e.tr;  
    } while (e != f.e);  
  }  
}
```

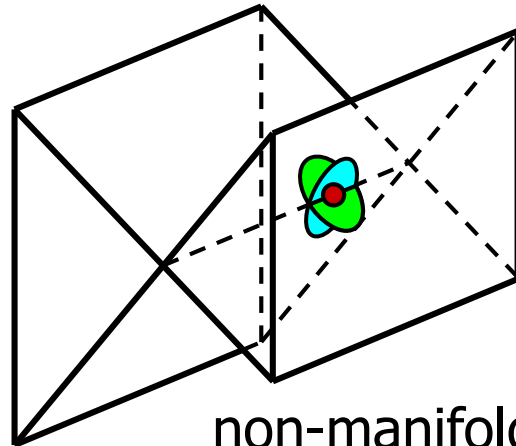


Manifolds

- 2-manifold: for every surface point there is a neighborhood topologically equivalent with plane
- Important for manufacturing, CAD/CAM



2-manifold

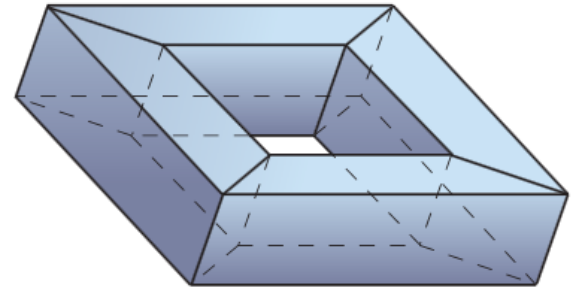


non-manifold

Euler-Poincare Formula for Manifolds

$$V - E + F - R = 2(S - H)$$

- V #vertices
- E #edges
- F #faces
- R #rings (holes in faces)
- H #holes (holes through object)
- S #shells (separate objects)



$$V=16$$

$$E=32$$

$$F=16$$

$$R=0$$

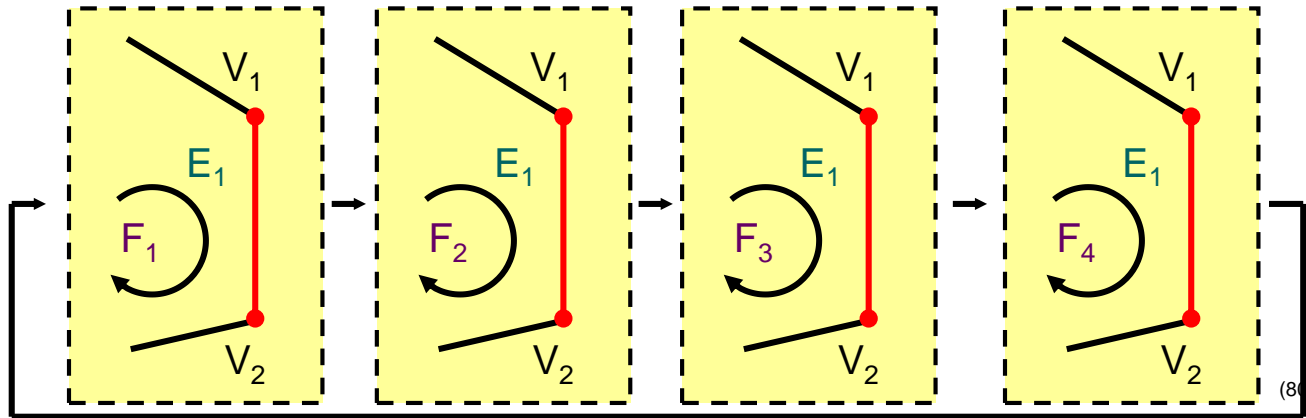
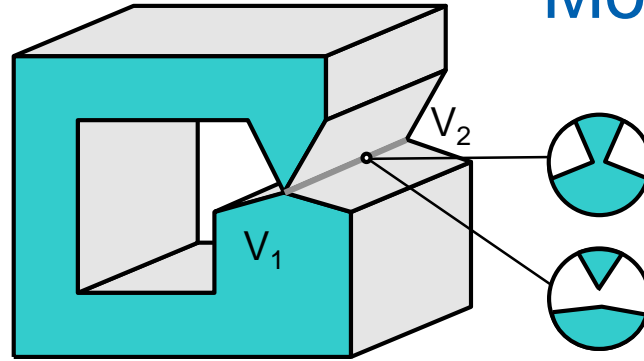
$$H=1$$

$$S=1$$

Representing non-manifolds

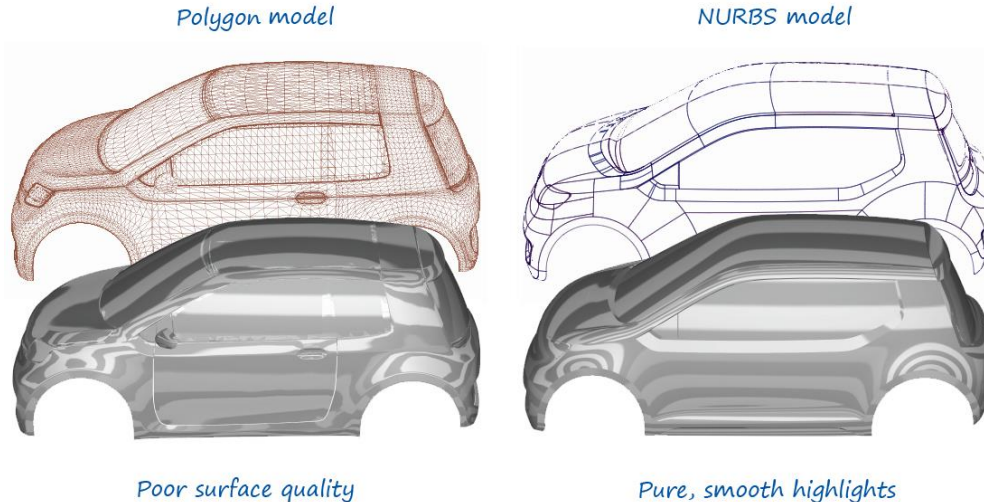
- Winged-edge: only manifolds
- Winged half-edge

More in VGE!



Smooth Surfaces

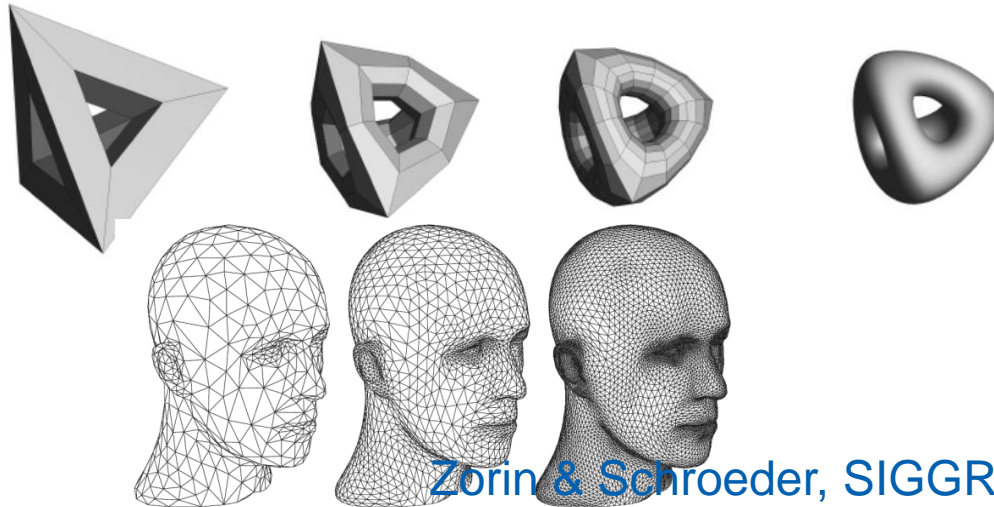
- Parametric surfaces
 - Sphere, Elipsoid, Cylinder
- Piecewise parametric surfaces / patches
 - Bezier patches, NURBS (patch approximation using cubic polynomials)



Source: Autodesk

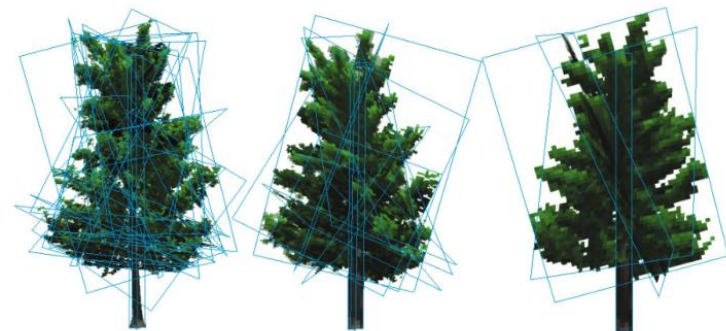
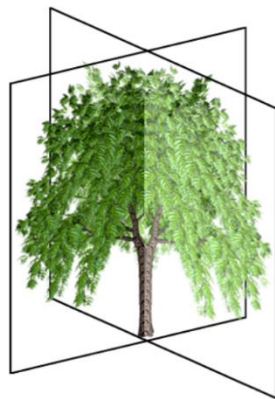
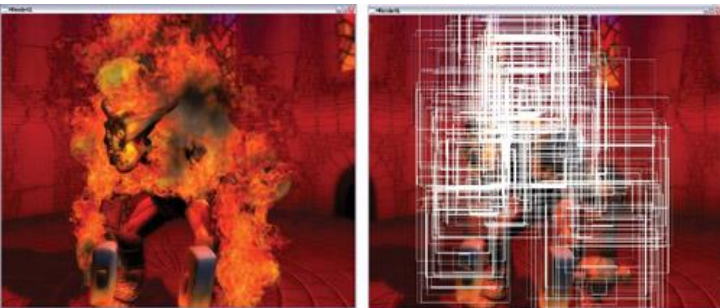
Subdivision Surfaces

- Progressively subdivide coarse mesh
 - Subdivide triangles/quads & reposition vertices based on their neighborhood
- Different subdivision schemes
 - Loop, Catmull-Clark, Doo-Sabin
 - HW support: hull+tessellation shaders



Sprites, Billboards

- Replacing geometry with images – sprites / billboards/ impostors
- Billboard: oriented sprite
 - towards a camera or based on object features ...



Other B-reps

- Height fields
 - Regular / irregular
 - Terrains
- Impostors with depth
- LODs
 - Multiresolution representation

B-rep: Summary

- Easy (GPU) rendering
- Complicated operations on solids
 - No explicit information on what is the interior

Implicit Surfaces

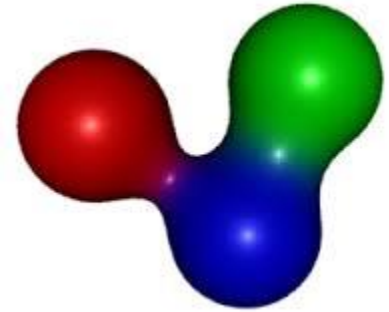
$$f(x, y, z) = f(\mathbf{p}) = 0$$

Examples:

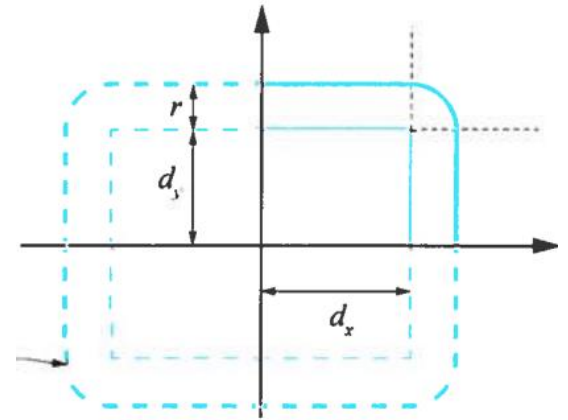
$$f(\mathbf{p}) = \|\mathbf{p} - \mathbf{c}\| - r$$

$$f(\mathbf{p}) = \mathbf{n} \cdot \mathbf{p} + d$$

$$f(\mathbf{p}) = \|\max(|\mathbf{p}| - d, \mathbf{0})\| - r$$



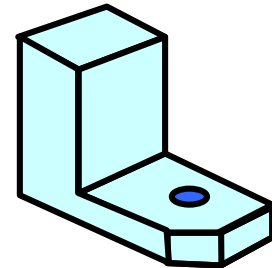
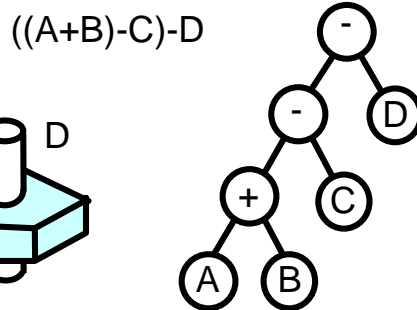
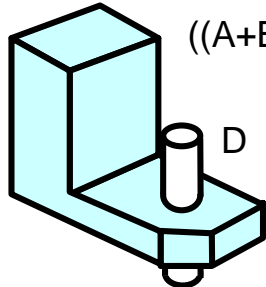
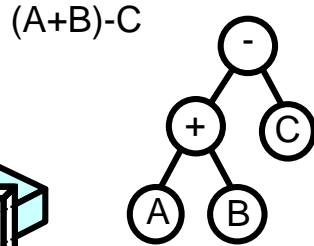
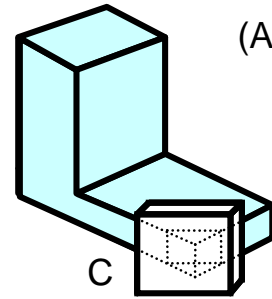
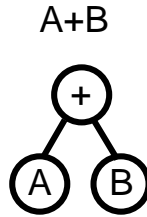
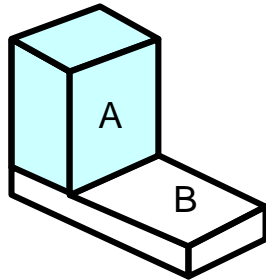
Source: scratchapixel.com



Source: Real-Time Rendering 4th edition

CSG Tree

- Leaves: geometric primitives (often implicit form)
- Inner nodes: set operations, transformations
- Root = model



CSG Tree - Properties

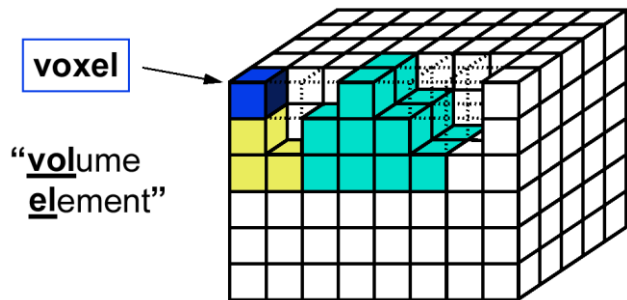
- Easy „point in solid“ test
- Modeling resembles manufacturing
 - Popular for 3D printing!
- Memory compact (primitives defined analytically)
- More complex rendering



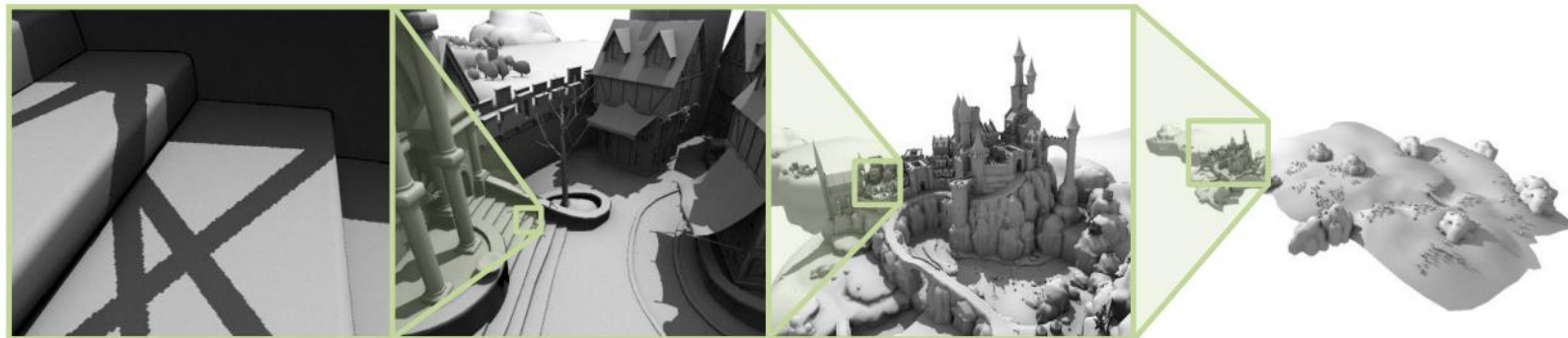
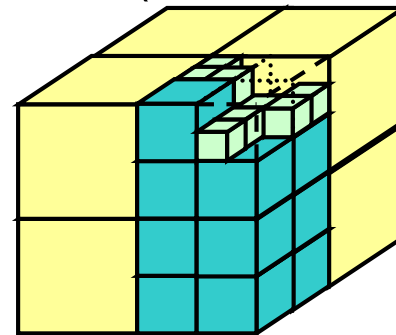
DP: Markéta Karaffová 2016

Volumetric representation

3D grid (regular structure)



Octree (hierarchical structure)



Volumetric rep.- properties

- Easy „point in solid“ test
- More difficult rendering (ray casting)



Source: Ikits et al. GPU Gems 2

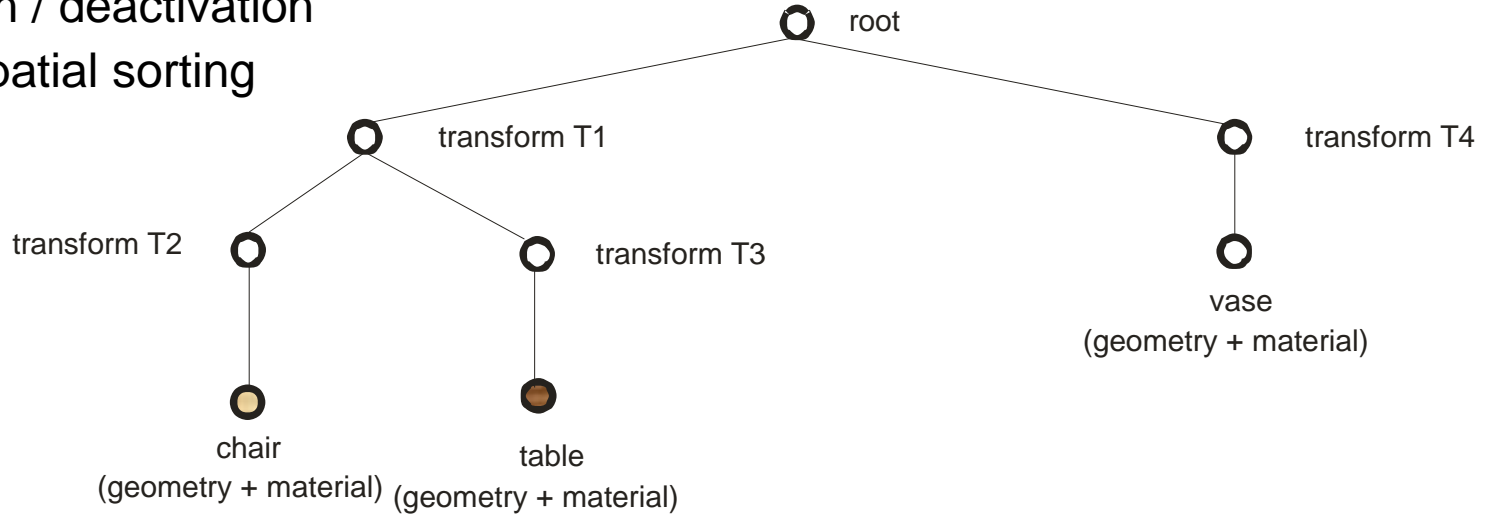


DP: M. Benatsky 2011

Scene Graph

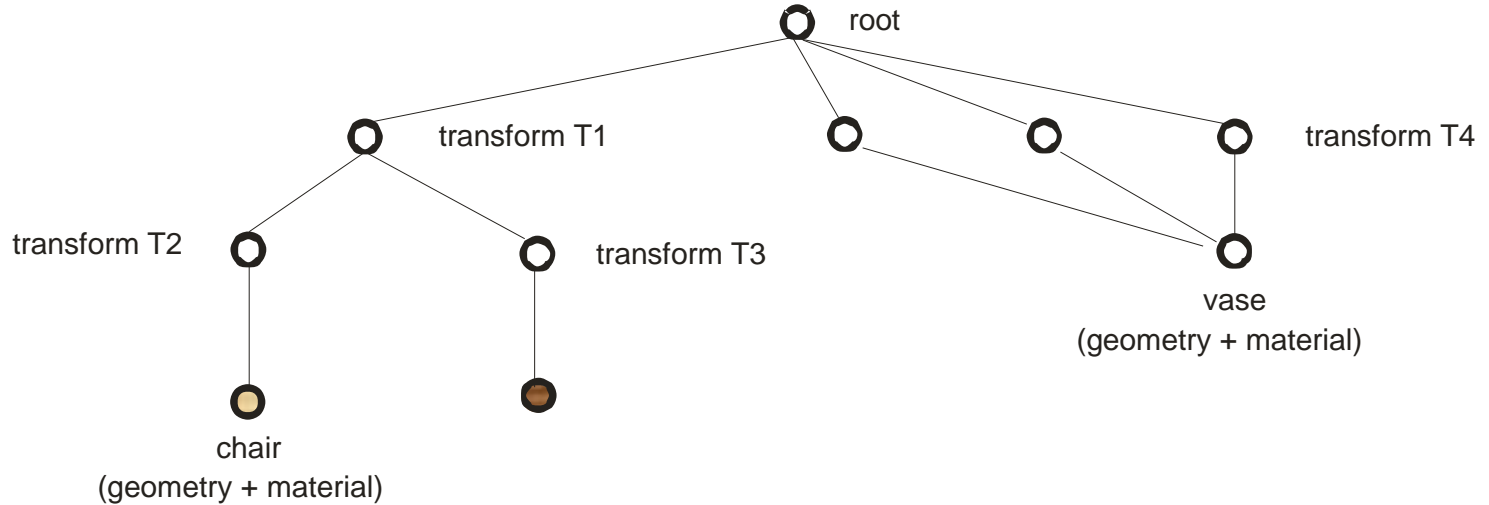
- Logical / Semantical grouping

- Transformation composition
- Naming
- Activation / deactivation
- Partial spatial sorting



Scene graph - Instancing

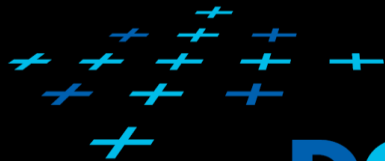
- One template / many copies



- Saving memory, propagating changes
- Not a tree anymore: DAG!
 - Implications for a renderer

Summary

- Points, Vectors, Transformations MPG – chapter 21
- Camera and Projection MPG – chapter 9
- 3D Scene Representation MPG - chapters 5.11,
5.12, 5.13, 6-8, 14



DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Questions?