DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Introduction to 3D geometry

Jiří Bittner

The lecture will start at 16:15

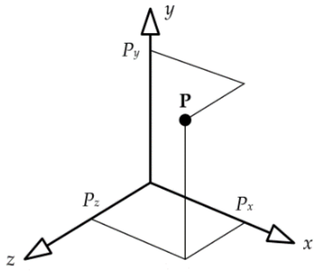
Outline

- 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

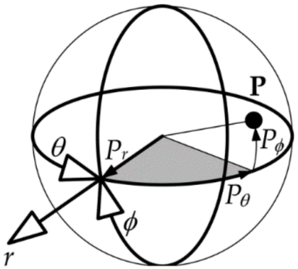
- Point is a location in 3D space
- Cartesian coordinates
  - Orthonormal basis

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



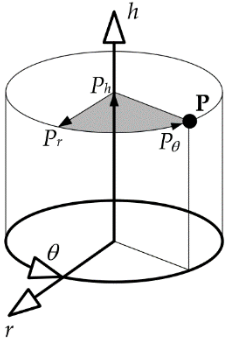
- Spherical coordinates

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



- Cylindrical coordinates

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

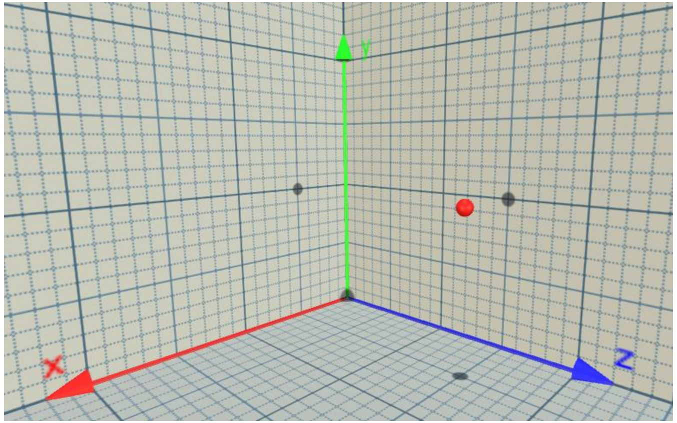


Cartesian coordinates

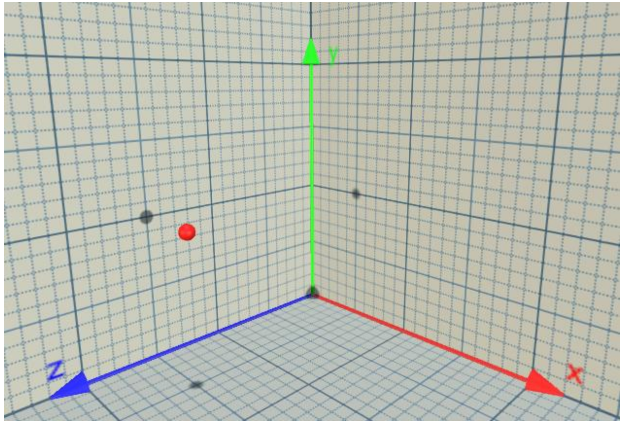
- Axes (René Descartes, 1596-1650)
  - Orthogonal directions
  - Meet at origin
  - Uniform scale
  - Orthogonal basis

$A = [5, 10, 15]$

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



LHS
Direct3D, Unity, …



RHS
OpenGL

Linear Transformations

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

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 3x3 \\ transformation \\ matrix \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Homogeneous coordinates

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

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

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

Transformation in homogeneous coordinates

- Finally normalize (divide by w)
  - Perspective division

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

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

Composing transformations

- Matrix multiplication

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

- Associative

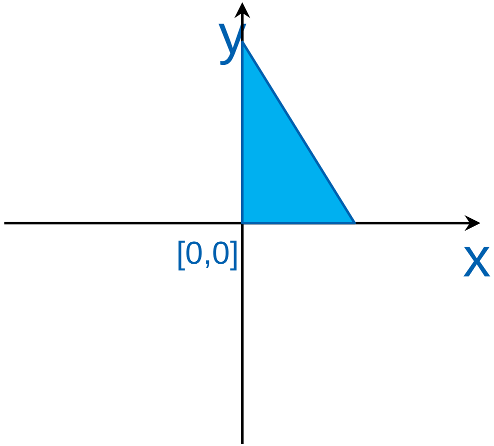
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

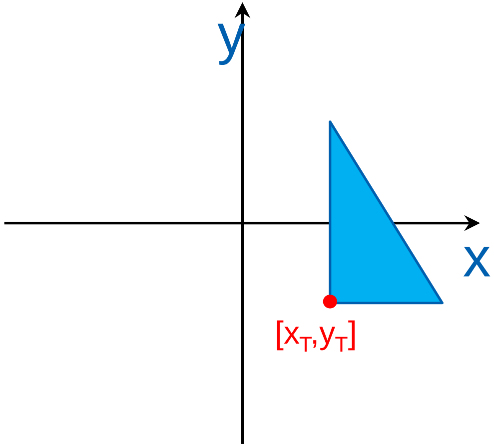
- Non-commutative: Transformation order matters!

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

- Transformations applied from right to left

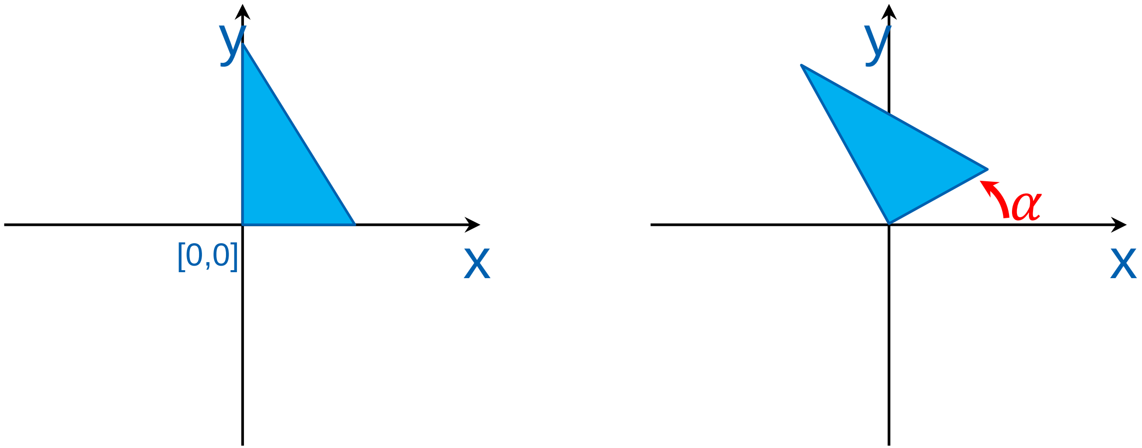
Translation





$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad M_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_{R_z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad M_{R_z} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Euler angles $M = \mathrm{M}_{\mathrm{R_y}}\mathrm{M}_{\mathrm{R_x}}\mathrm{M}_{\mathrm{R_z}}$ (order in Unity)

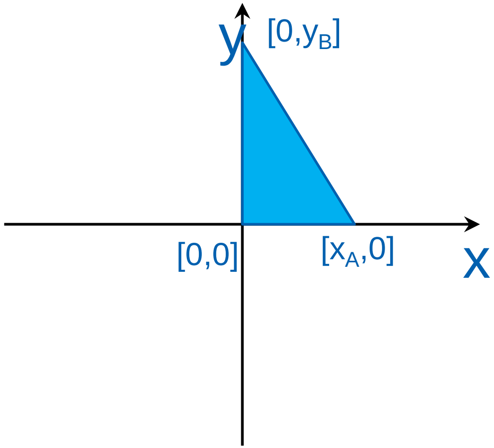
Euler Angles Rotation

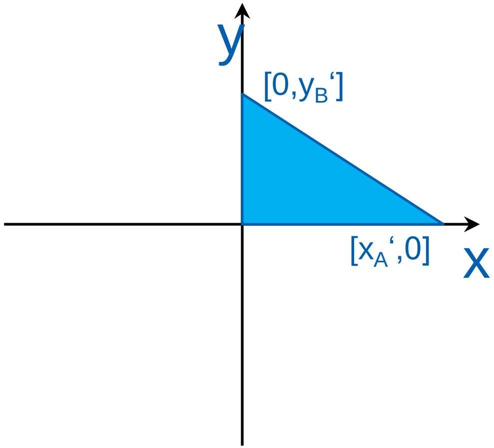
$M = \mathrm{M_{R_y}} \mathrm{M_{R_x}} \mathrm{M_{R_z}}$ (order in Unity)

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

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

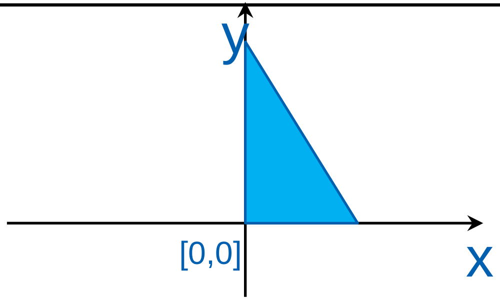
Scaling

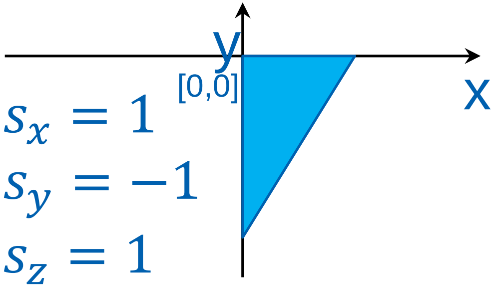




$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = M_S \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \qquad M_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Symetry



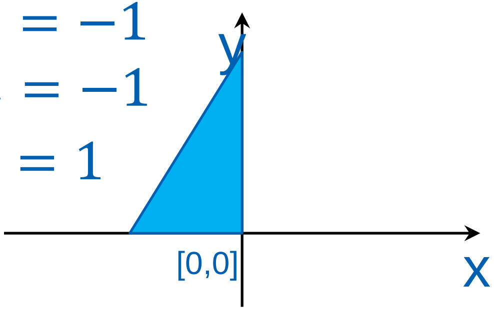


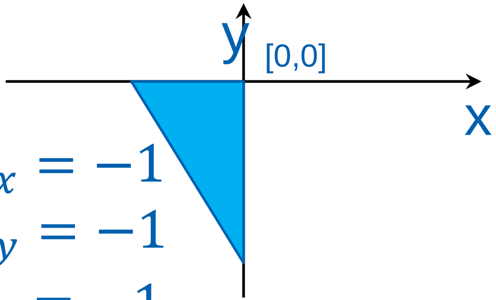
$s_x = 1$
$s_y = -1$
$s_z = 1$

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

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

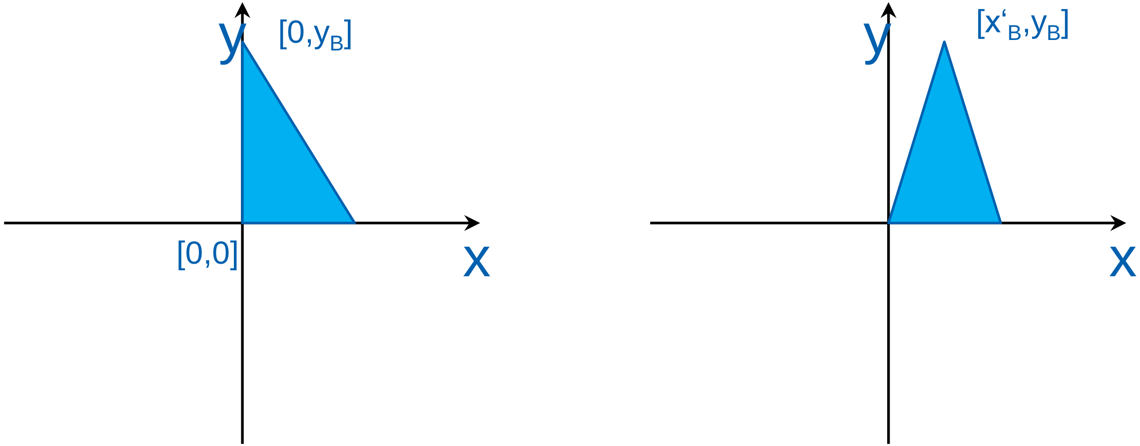
$s_x = -1$
$s_y = -1$
$s_z = 1$





$s_x = -1$
$s_y = -1$
$s_z = -1$

Shear

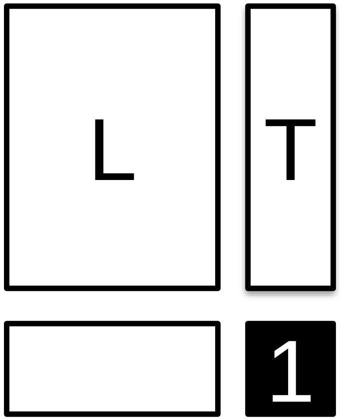


$$M_{SH_x} = \begin{bmatrix} 1 & SH_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

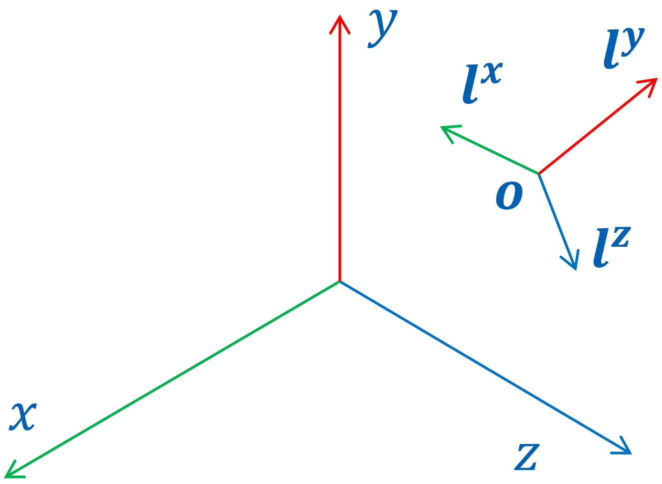
Transformation matrix

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



Local Coordinate Frame

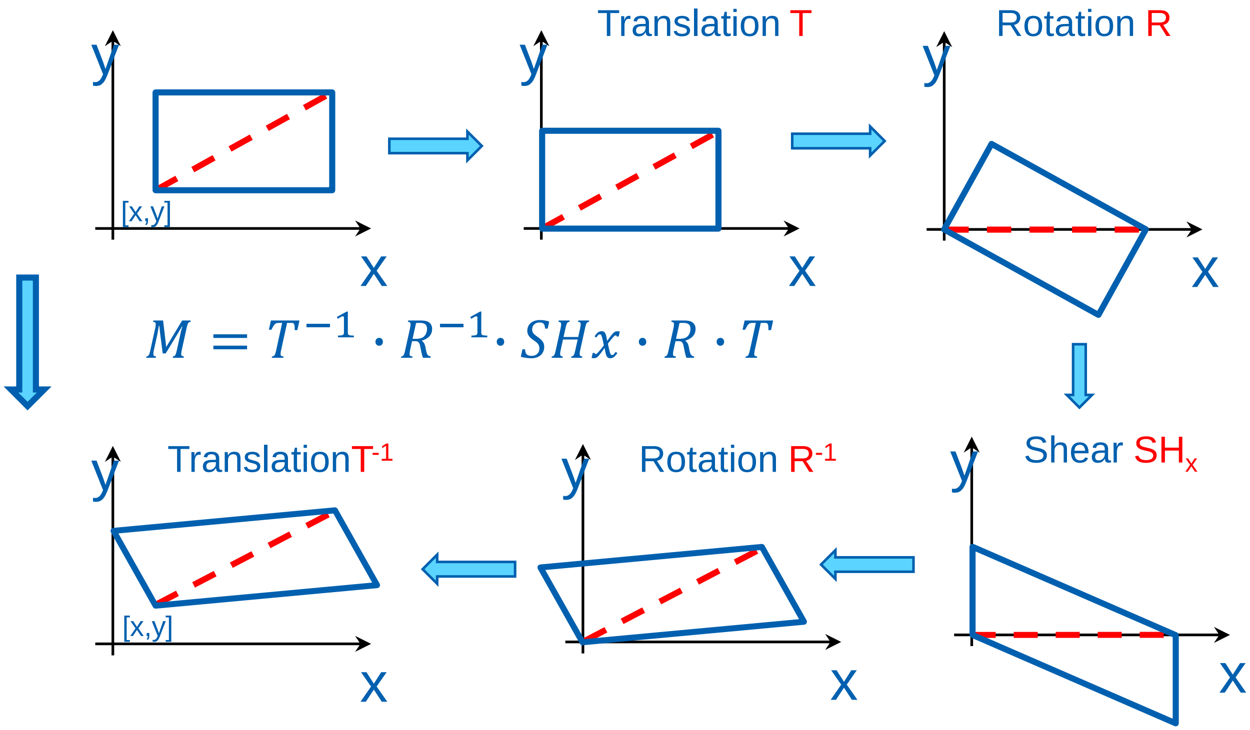
- Local Coordinate Basis + Translation



$$\boldsymbol{M} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{l}^x_x & \boldsymbol{l}^y_x & \boldsymbol{l}^z_x & \boldsymbol{o}_x \\ \boldsymbol{l}^x_y & \boldsymbol{l}^y_y & \boldsymbol{l}^z_y & \boldsymbol{o}_y \\ \boldsymbol{l}^x_z & \boldsymbol{l}^y_z & \boldsymbol{l}^z_z & \boldsymbol{o}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

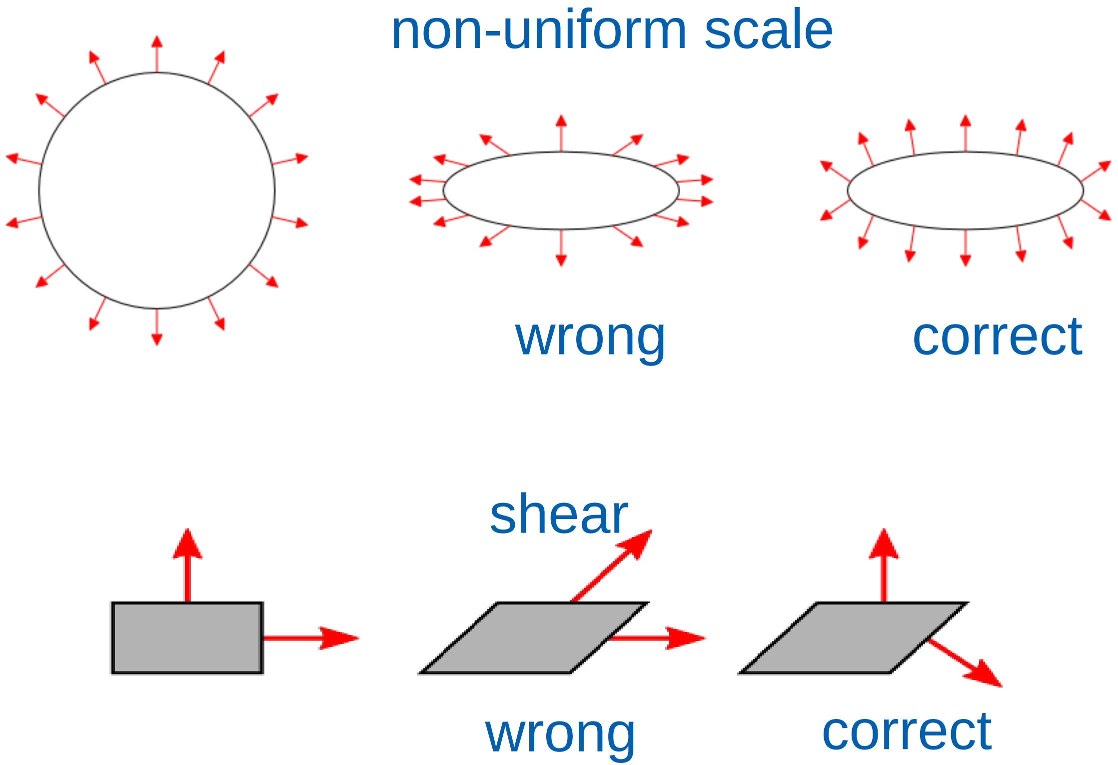
Example – shear along a diagonal



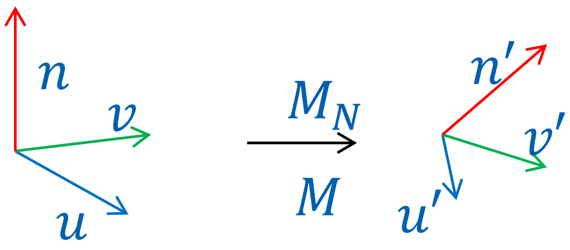
$$M = T^{-1} \cdot R^{-1} \cdot SHx \cdot R \cdot T$$

Transforming normals

- Normal – vector perpendicular to the surface



Transforming normals



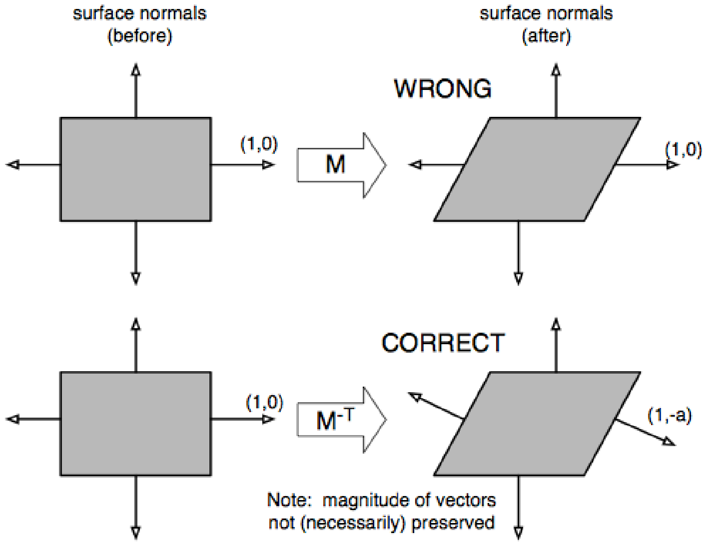
$$\begin{bmatrix} n_x' \\ n_y' \\ n_z' \\ 0 \end{bmatrix} = M_N \begin{bmatrix} n_x \\ n_y \\ n_z \\ 0 \end{bmatrix}$$

$$n^T u = 0$$
$$n'^T u' = 0$$
$$n'^T M u = 0$$
$$n'^T M u = n^T u$$
$$n'^T M = n^T$$
$$M^T n' = n$$
$$n' = {M^T}^{-1} n$$
$$\boldsymbol{n' = {M^{-1}}^T n}$$

Can use just 3x3 submatrix of M !

Transforming normals - example

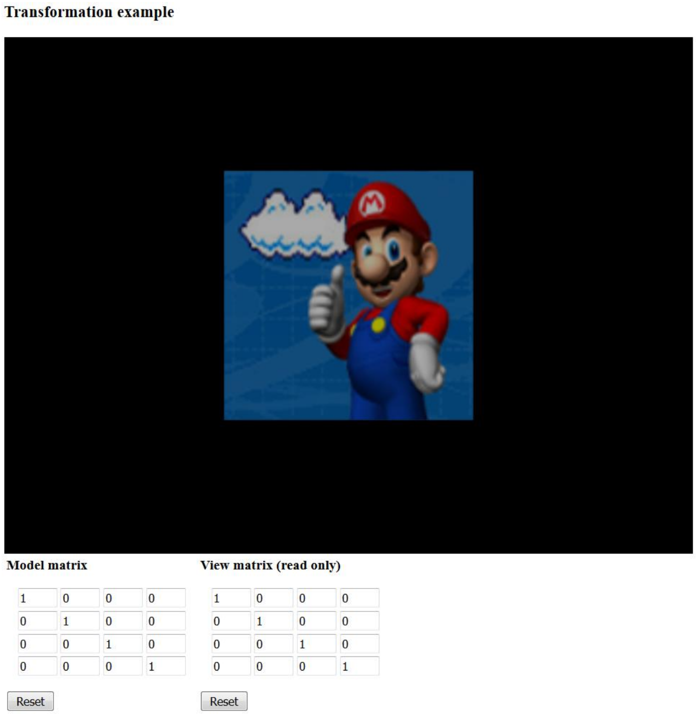
$$M = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \quad M^{-T} = \begin{bmatrix} 1 & 0 \\ -a & 1 \end{bmatrix}$$



Source: Stack Overflow

DEMO

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



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-tupple

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

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

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

William Rowan Hamilton 1843

q = (x, y, z, w) = (**v,** r) , **v** = xi + yj + zk

Quaternions and Rotation

- Unit quaternion ($|\boldsymbol{q}| = 1$) represents rotation in 3D

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

3D rotation about axis $\boldsymbol{a}$ by angle α

Quaternion Operations

- Sum $$\boldsymbol{q}_1 + \boldsymbol{q}_2 = [\boldsymbol{v}_1 + \boldsymbol{v}_2, w_1 + w_2]$$
- Dot product $$\boldsymbol{q}_1 . \boldsymbol{q}_2 = \boldsymbol{v}_1 . \boldsymbol{v}_2 + \ w_1 . w_2$$
- **Multiplication** (Hamilton product)

$$\boldsymbol{q}_1 * \boldsymbol{q}_2 = [\boldsymbol{v}_1, r_1] * [\boldsymbol{v}_2, r_2] = [r_1\boldsymbol{v}_2 + r_2\boldsymbol{v}_1 + \boldsymbol{v}_1 \times \boldsymbol{v}_2, r_1r_2 - \boldsymbol{v}_1\boldsymbol{v}_2]$$

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

$$\boldsymbol{q}^* = [-\boldsymbol{v}, r]$$

  - Inverse rotation

Transformation with quaternion

- Express vector as quaternion

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

- Rotation of $\boldsymbol{u}$ using $\boldsymbol{q}$

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

- Two quaternion multiplications + conjugate

Quaternion to Rotation Matrix

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

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

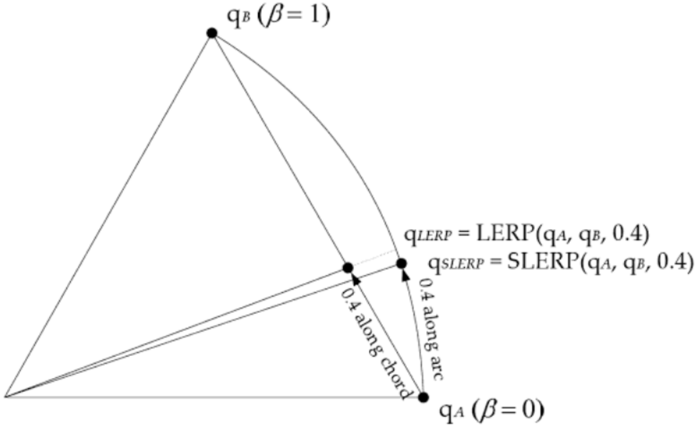
- Rotation composition faster with quaternions
- Vector transformation faster with matrix

Rotation Interpolation

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

LERP and SLERP

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



J. Gregory, Game Engine Architecture

LERP

$$w_A = 1 - \beta$$
$$w_B = \beta$$

SLERP

$$w_A = \frac{\sin{(1-\beta)\theta}}{\sin\theta}$$
$$w_B = \frac{\sin\ \beta\theta}{\sin\theta}$$
$$\theta = \text{acos}\, q_A q_B$$

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

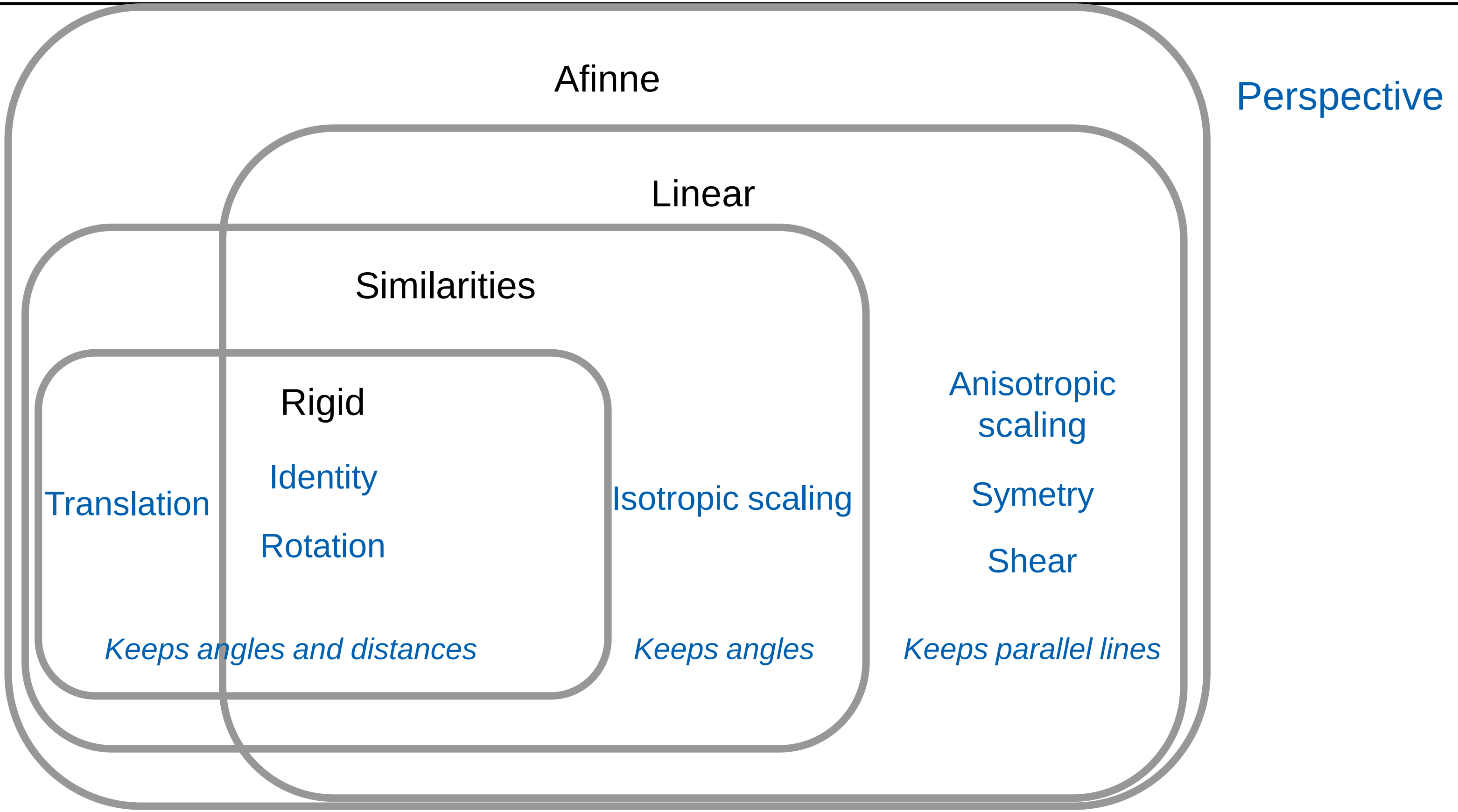
Transformation Representation - Matrix

- 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

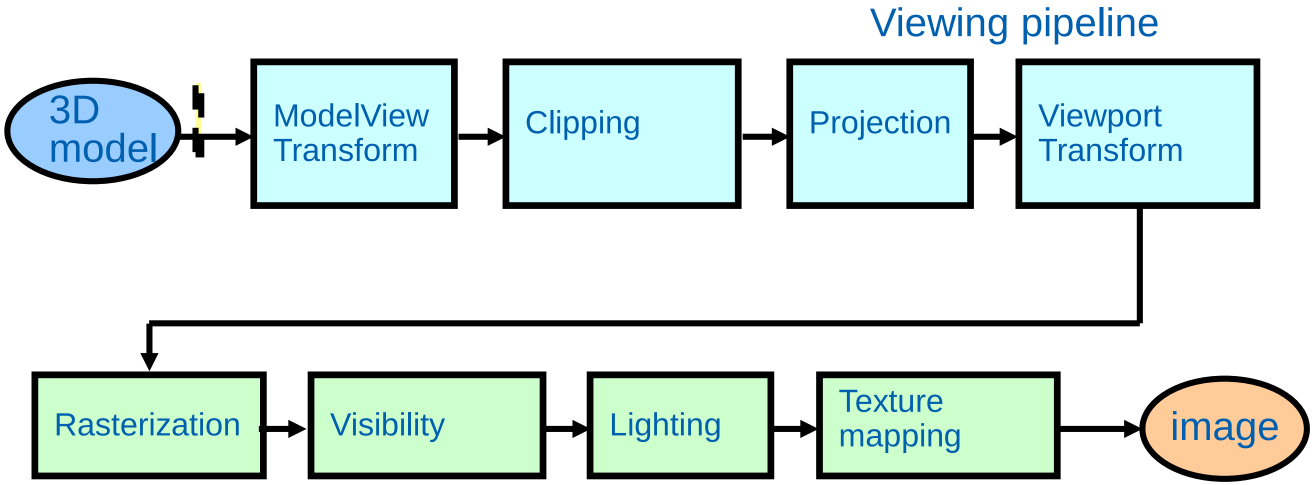


Outline

- 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

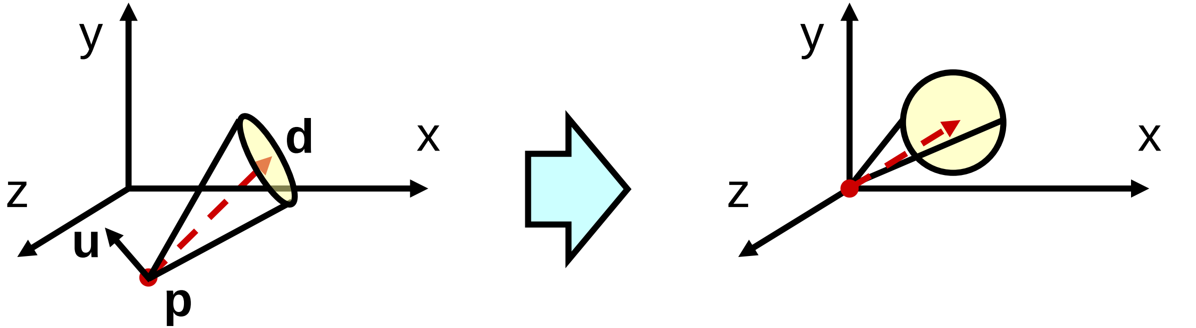
- Object / Modeling / Local coordinates
  - Relative to object origin
- World coordinates
  - Global scene coordinates
- Camera / Eye / View coordinates
  - Camera in the origin, looks along –z
- Clip coordinates
  - After multiplication by projection matrix
- Normalized device coordinates
  - Cuboid 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

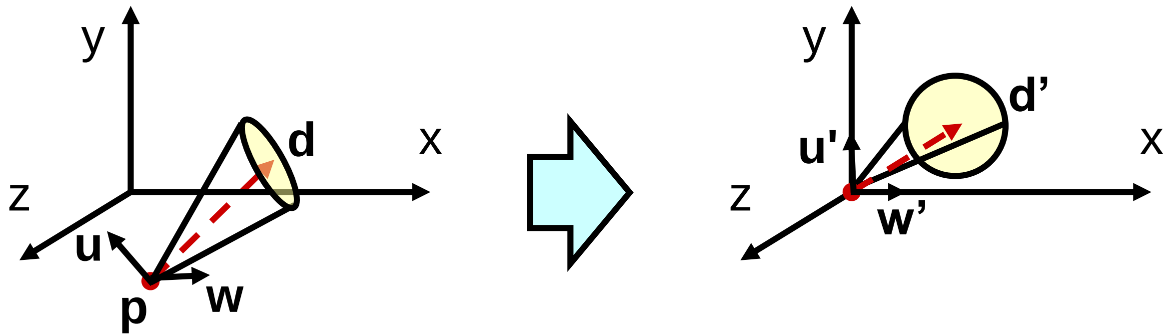
View transformation

- 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

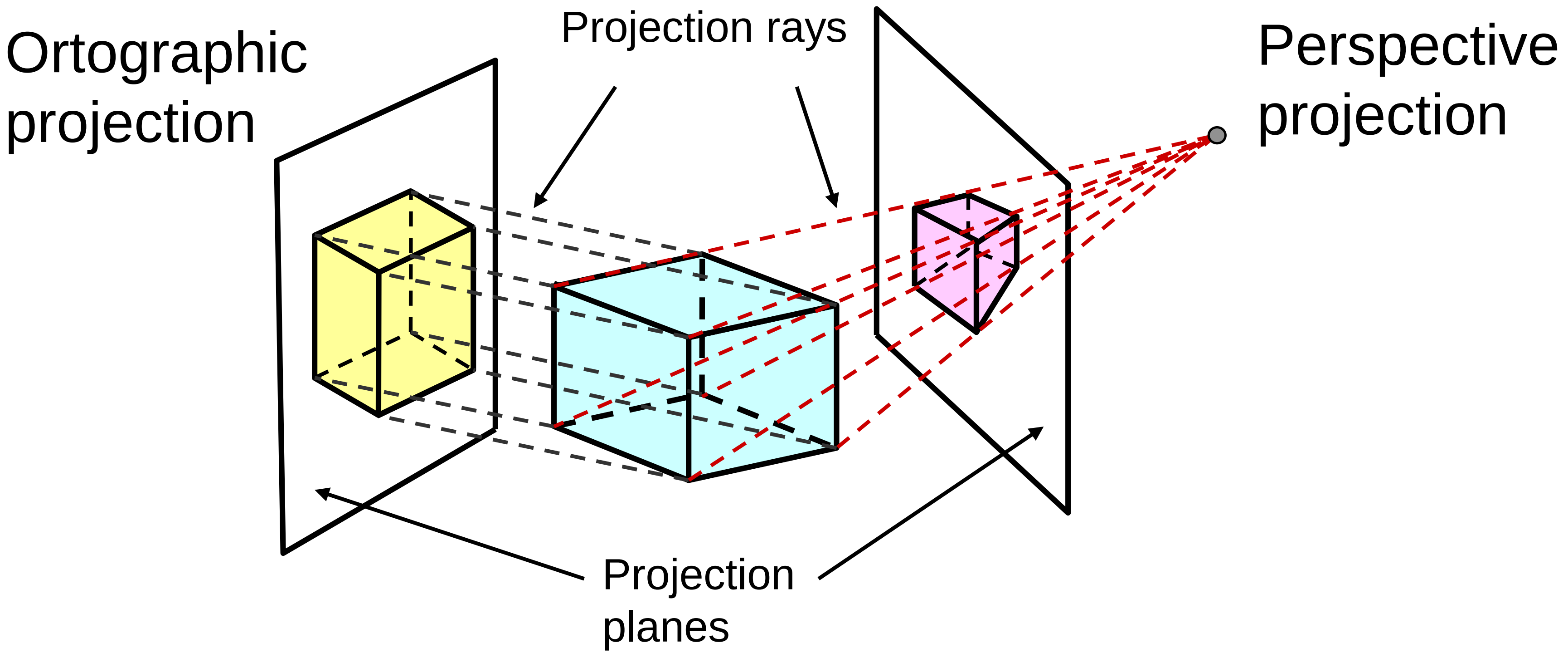


- $M_V = M_C^{-1}$ (M_C camera matrix)

$$w = d \times u$$

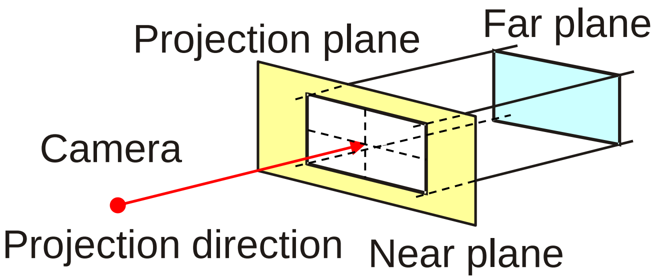
$$M_V = \begin{bmatrix} w_x & u_x & d_x & p_x \\ w_y & u_y & d_y & p_y \\ w_z & u_z & d_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$$

Orthographic and perspective projection

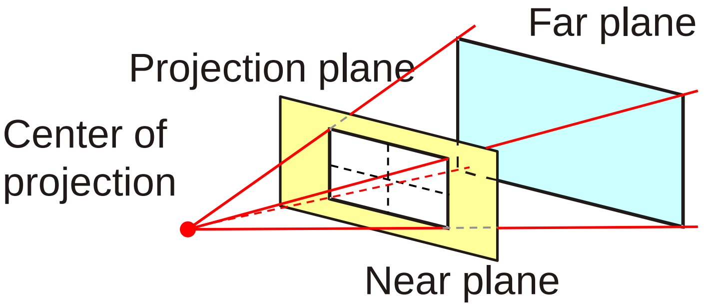


Camera – projection transformation

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



Ortographic projection
view volume = cuboid



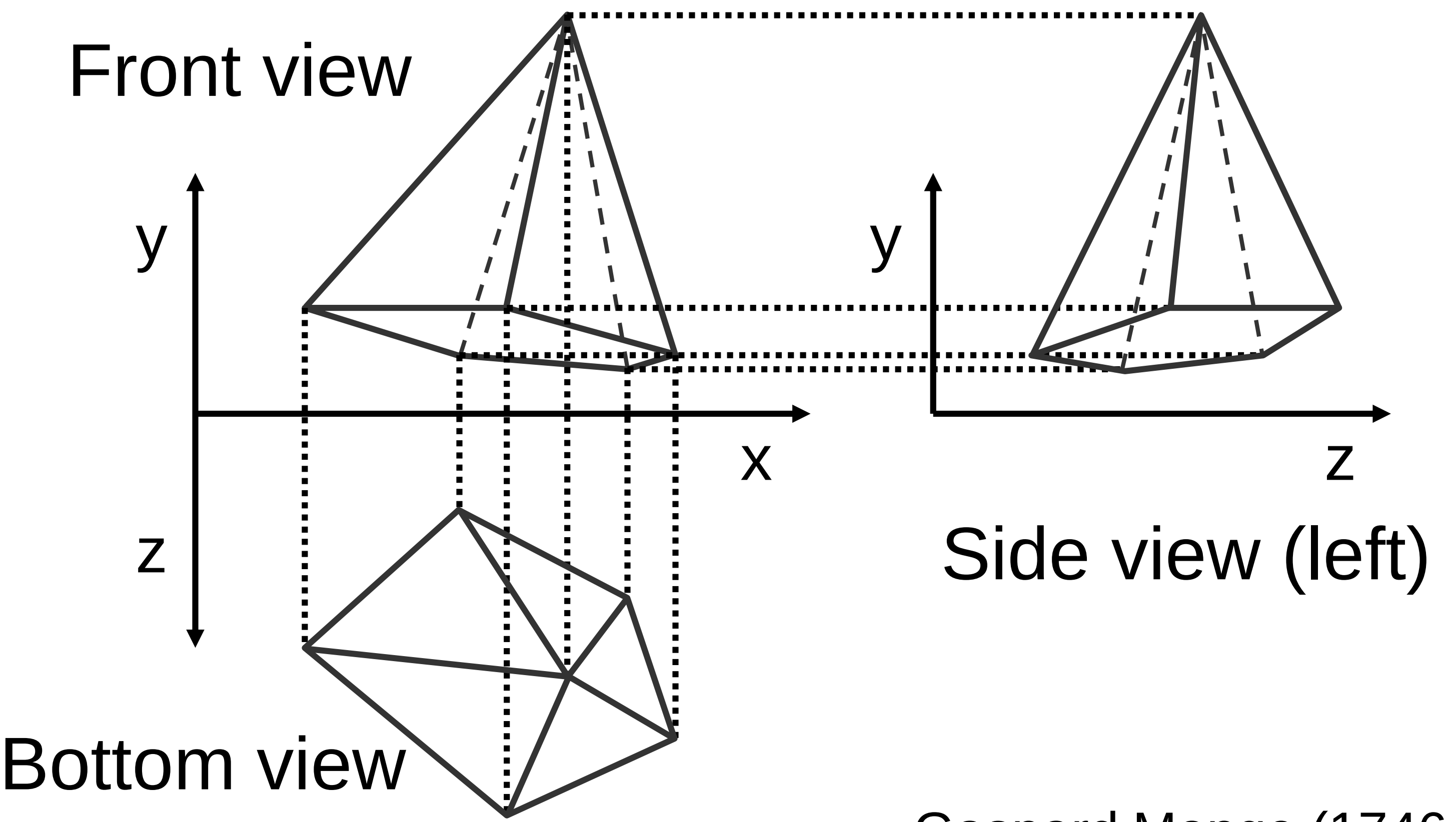
Perspective projection
view volume = pyramid frustum

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)

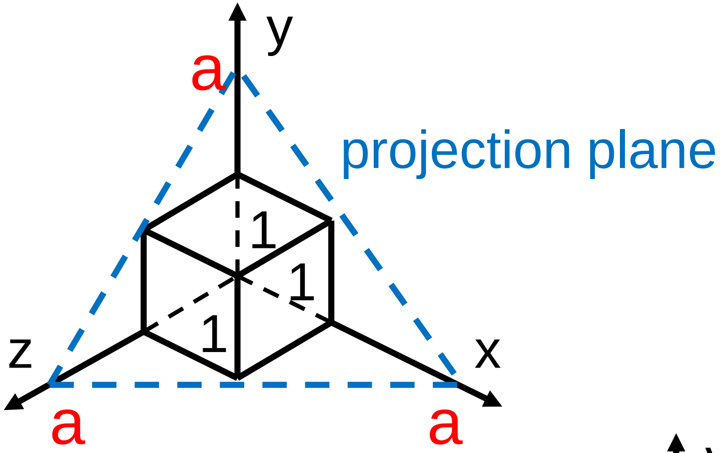
Monge’s projection



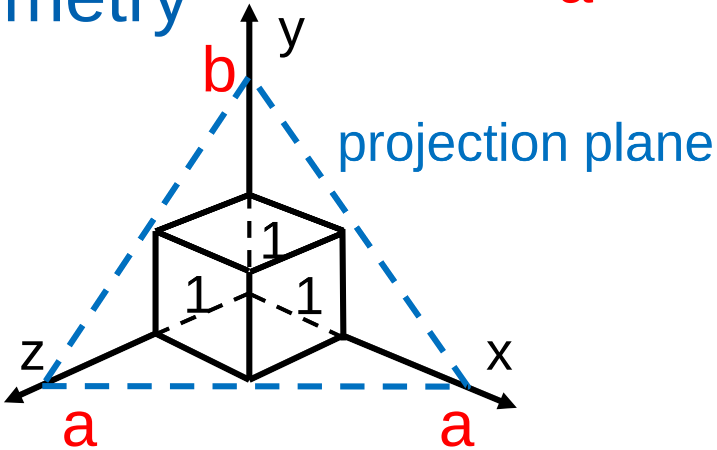
Gaspard Monge (1746 - 1818)

Axonometry

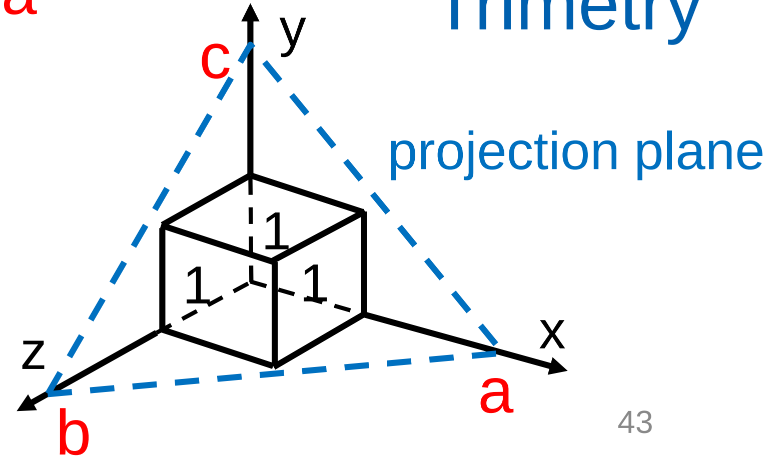
Isometry



Dimetry

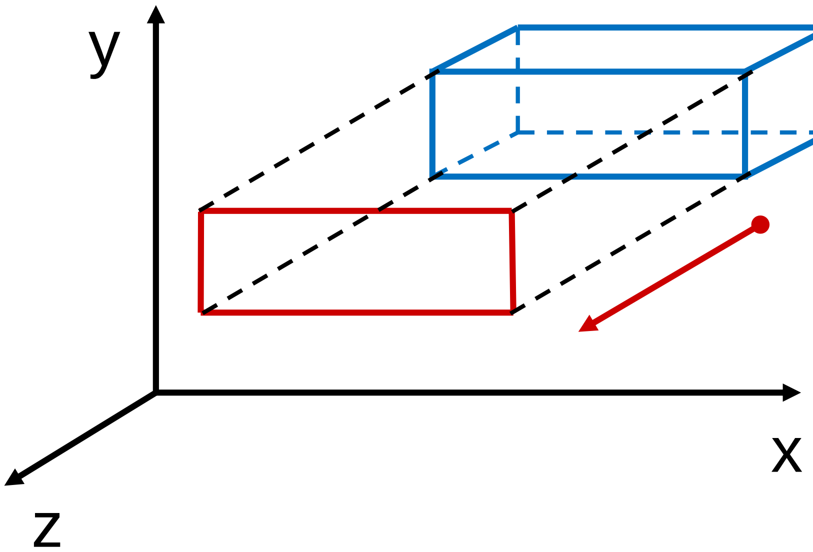


Trimetry



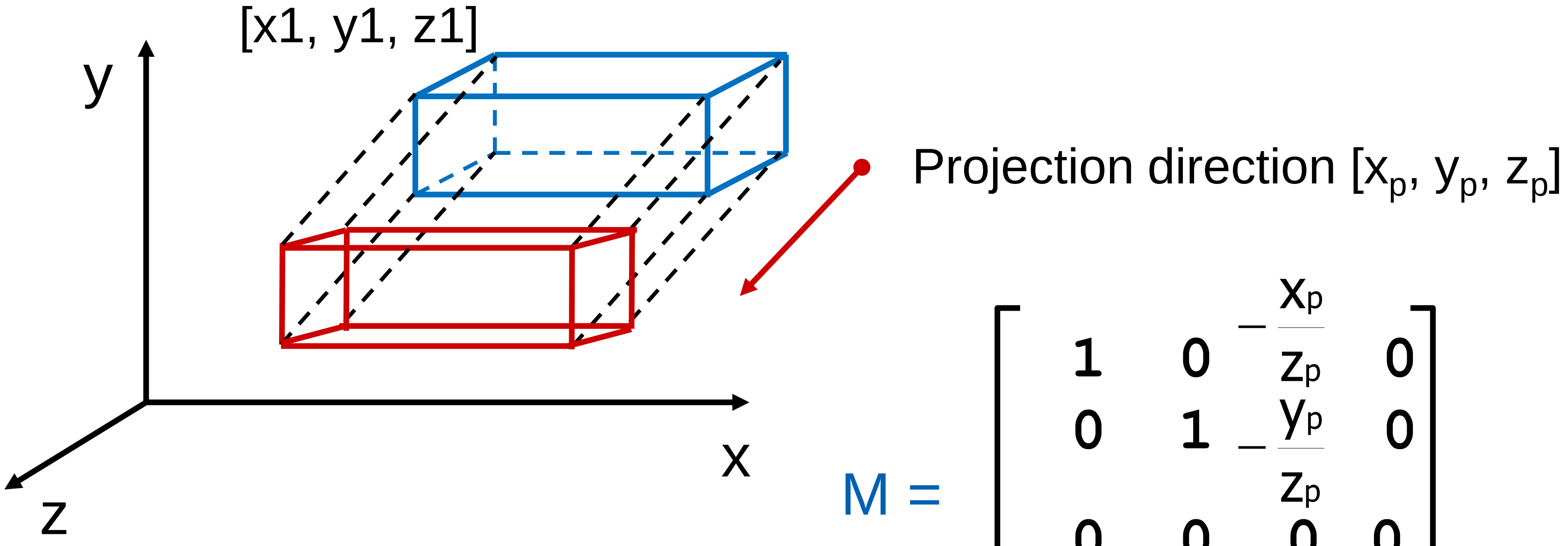
Projection: Matrix Form

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



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

Oblique Projection



$x = x1 + x_p \cdot t$

$y = y1 + y_p \cdot t$

$z = z1 + z_p \cdot t$

$z = 0 \Rightarrow t = -z1/ z_p$

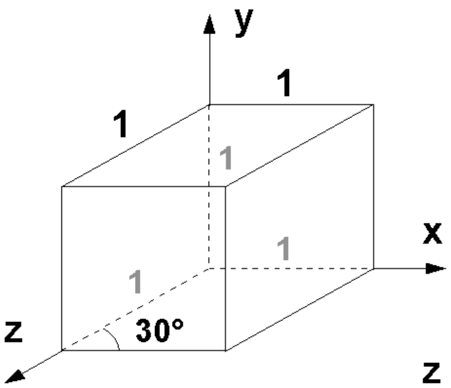
$$M = \begin{bmatrix} 1 & 0 & -\frac{x_p}{z_p} & 0 \\ 0 & 1 & -\frac{y_p}{z_p} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

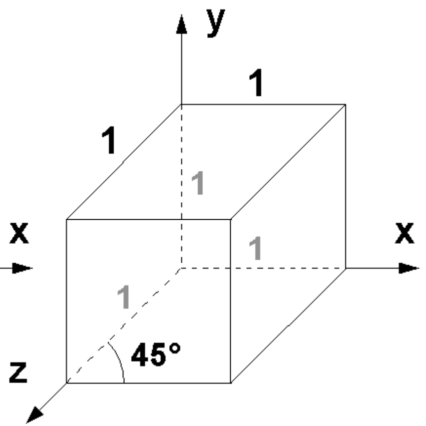
$M = M_{//} \cdot M_{zk}$

Oblique Projection

Cavalier

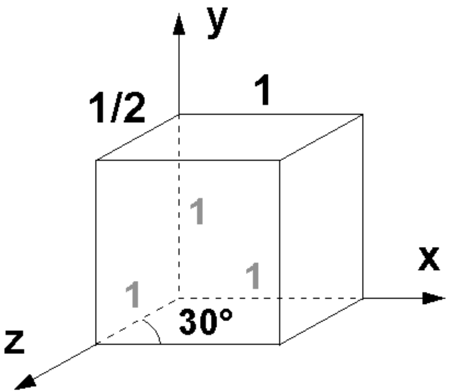
$$M = \begin{bmatrix} 1 & 0 & -\cos\beta & 0 \\ 0 & 1 & -\sin\beta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

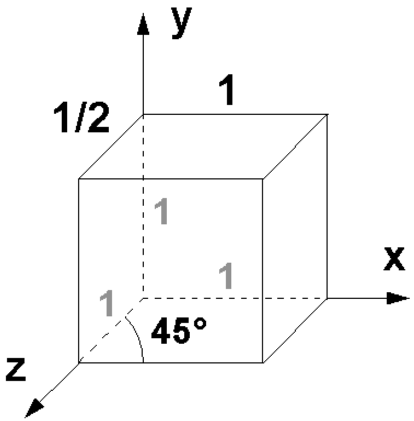




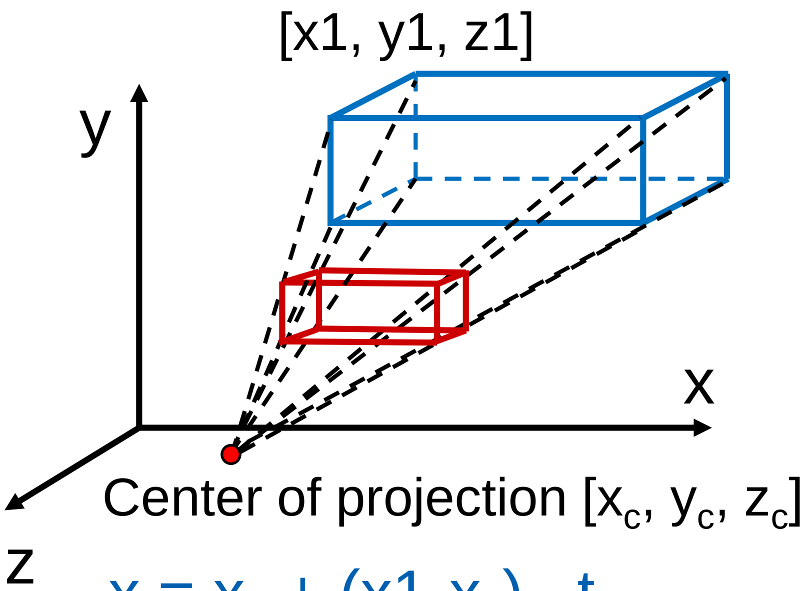
Cabinet

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





Perspective Projection



$x = x_c + (x1 - x_c) \cdot t$

$y = y_c + (y1 - y_c) \cdot t$

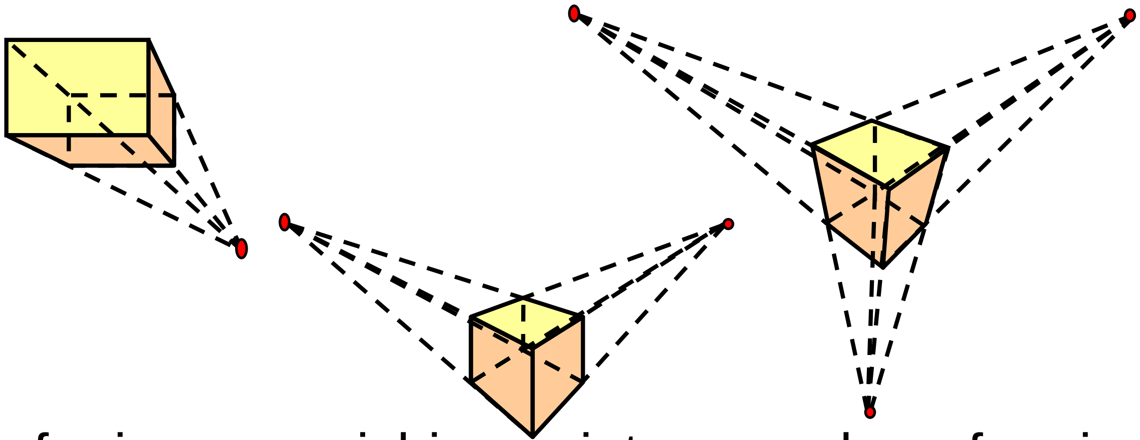
$z = z_c + (z1 - z_c) \cdot t$

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

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

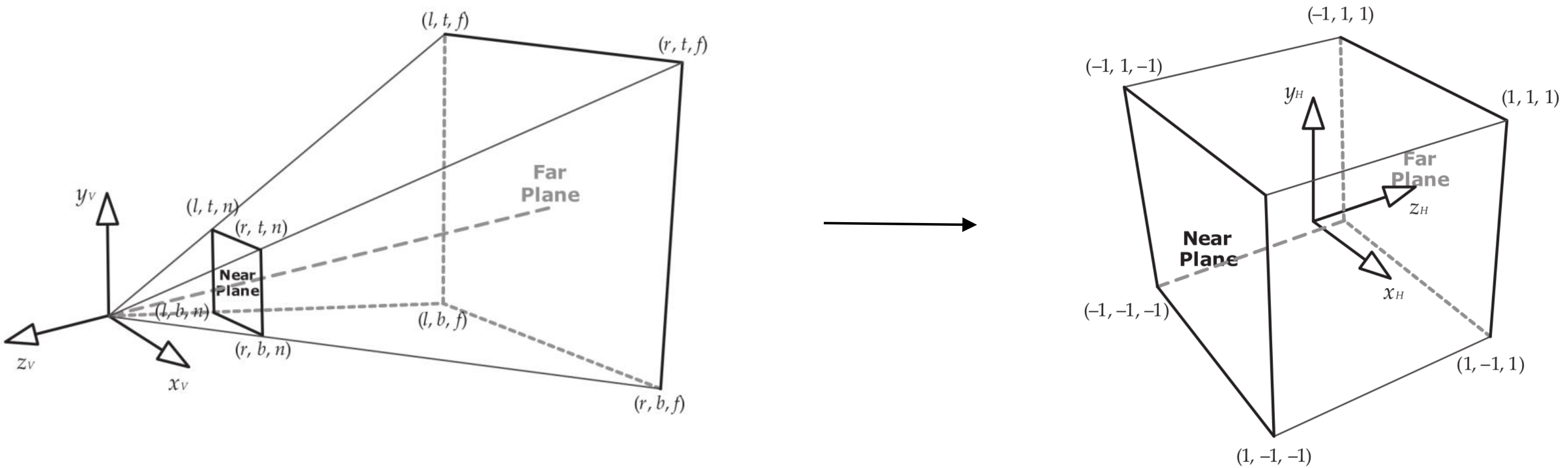
Vanishing Points (Úběžníky)

- Perspective projection does not keep parallelism
- Lines parallel to coordinate axes meet in **primary vanishing points**
  - 1-, 2-, 3-point perspective



- Number of primary vanishing points = number of projection plane intersections with coordinate axes

Perspective Projection (OpenGL)

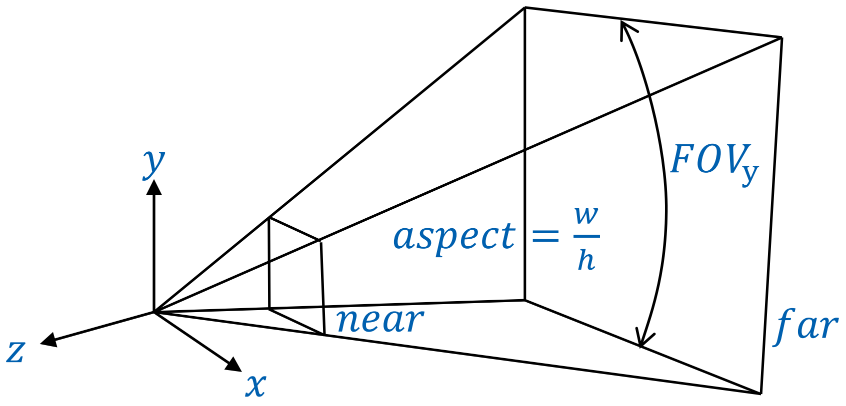


camera / eye space

NDC / clip space

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

Symetrical 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

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

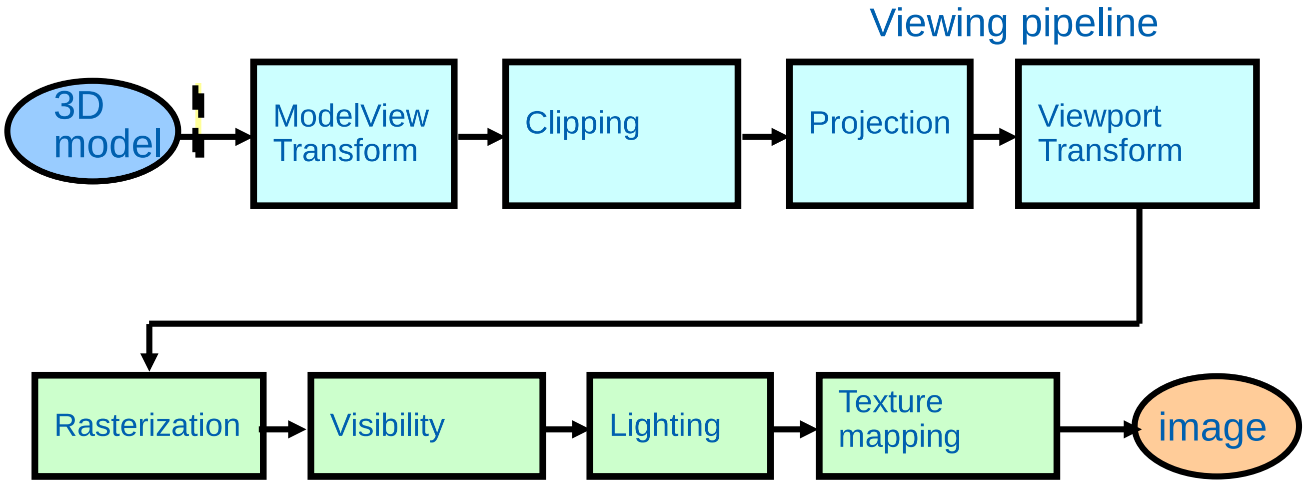
$$y' = (y_{\mathrm{NDC}} + 1)\frac{H}{2} + Y$$

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

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

Rendering Pipeline

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



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
- …

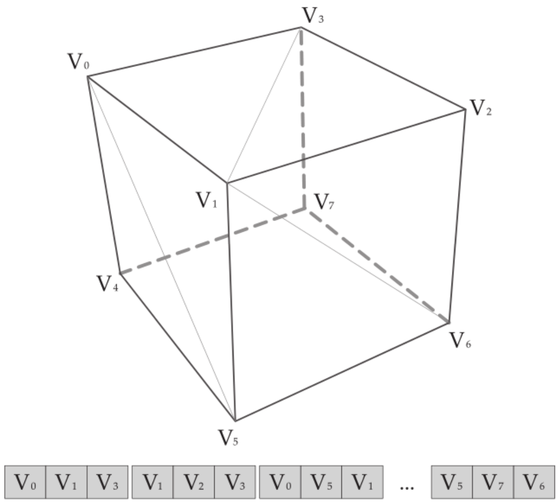
Polygonal Mesh

- 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
  - …

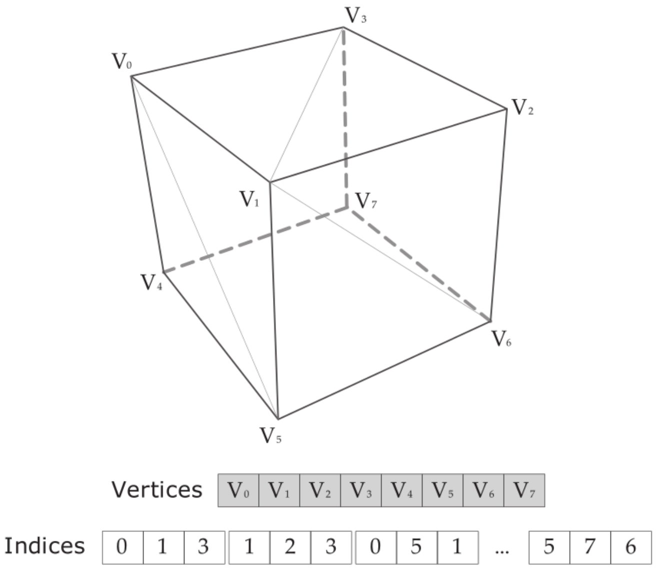
Triangle Mesh

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

Triangle Mesh

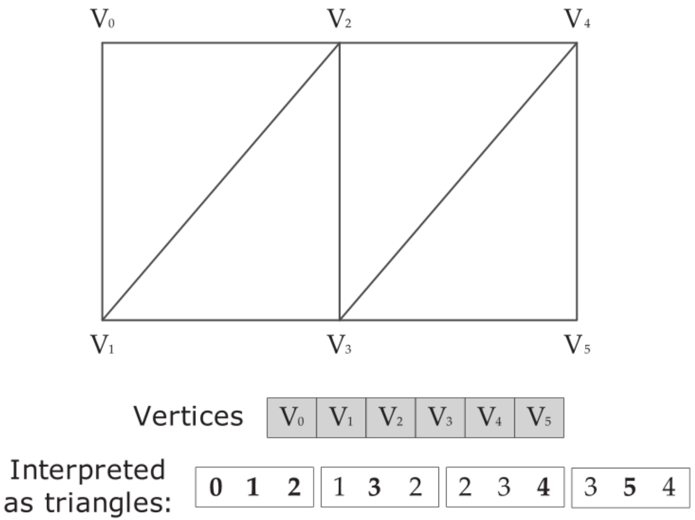


triangle list

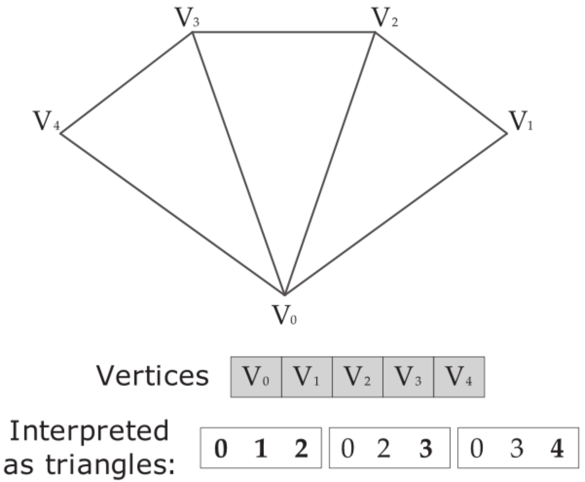


indexed triangle list

Triangle Mesh – Compact Representation



triangle strip

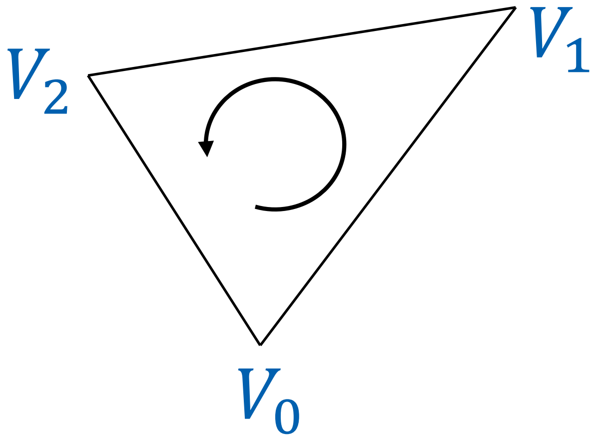


triangle fan

No indices – saves memory!

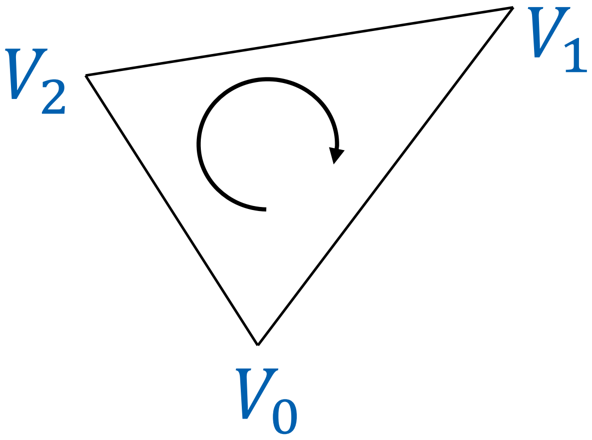
Triangle Mesh – Winding Order

- Defining front and back faces



$V_0 V_1 V_2$

CCW (counter clock wise)



$V_0 V_2 V_1$

CW (clock wise)

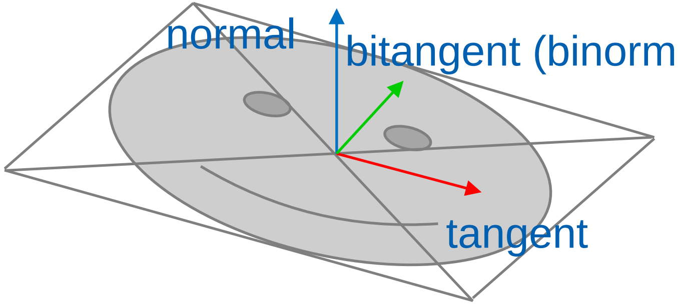
Storing Other Information

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

Triangle Mesh

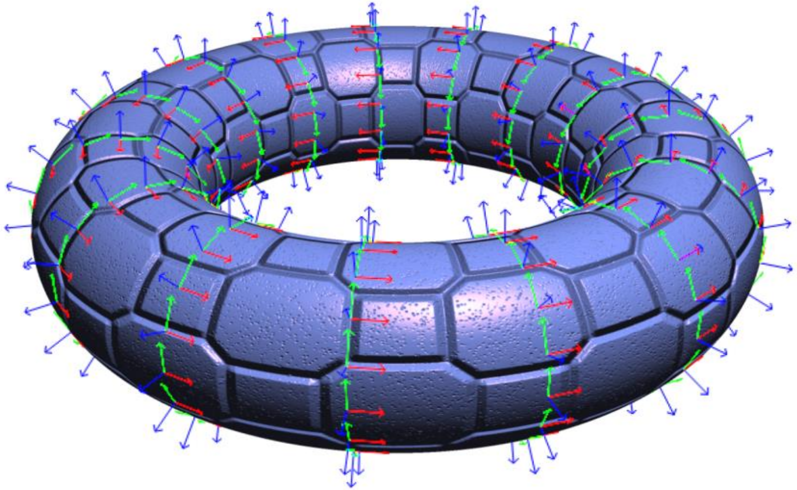
- Mesh processing / optimization
  - Computing normal, tangents, bitangents
  - Computing texture params (mesh parametrization, unwrap)
  - Sharing vertices (welding)
  - Optimizing vertex order (GPU transform cache utility)
  - Mesh compression (lossy coding of positions, normals, uvs)
    - Unity -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

Triangle Mesh – Tangent Space



$$\boldsymbol{t} \sim \frac{\delta u}{\delta \boldsymbol{p}} = [\frac{\delta u}{\delta x} \quad \frac{\delta u}{\delta y} \quad \frac{\delta u}{\delta z}]$$

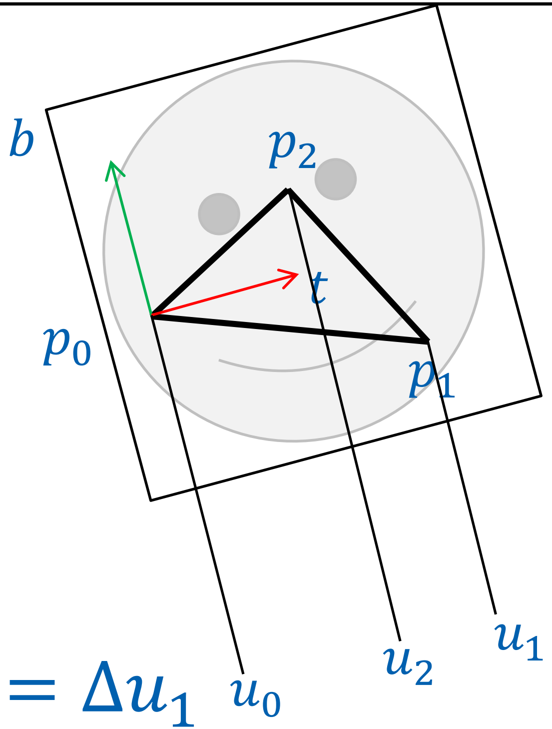
$$\boldsymbol{b} \sim \frac{\delta v}{\delta \boldsymbol{p}} = [\frac{\delta v}{\delta x} \quad \frac{\delta v}{\delta y} \quad \frac{\delta v}{\delta z}]$$



Source: Mikkelsen 2008, Simulation of Wrinkled Surfaces Revisited

Triangle Mesh – Tangent Space

$$\Delta \boldsymbol{p}_1 = \boldsymbol{p}_1 - \boldsymbol{p}_0$$
$$\Delta \boldsymbol{p}_2 = \boldsymbol{p}_2 - \boldsymbol{p}_0$$
$$\Delta u_1 = u_1 - u_0$$
$$\Delta u_2 = u_2 - u_0$$



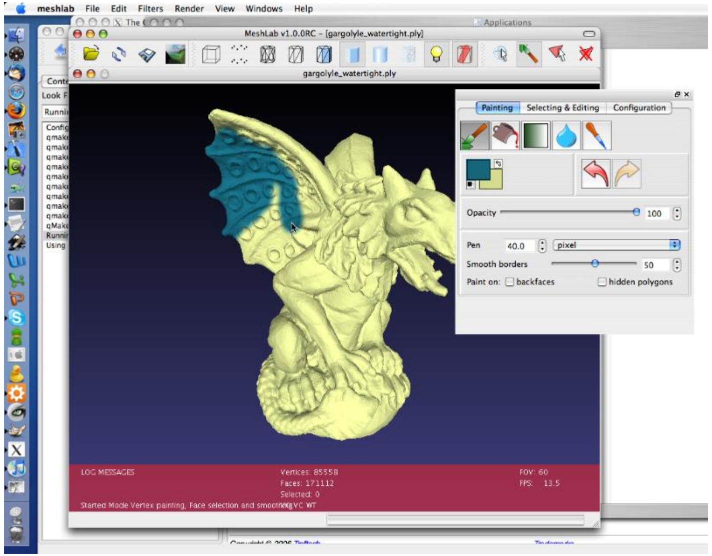
$$\boldsymbol{t} = \begin{bmatrix} \Delta \boldsymbol{p}_1 \\ \Delta \boldsymbol{p}_2 \\ \Delta \boldsymbol{p}_1 \times \Delta \boldsymbol{p}_2 \end{bmatrix}^{-1} \begin{bmatrix} \Delta \mathrm{u}_1 \\ \Delta u_2 \\ 0 \end{bmatrix}$$

$$\Delta \boldsymbol{p}_1 \boldsymbol{t} = \Delta u_1$$
$$\Delta \boldsymbol{p}_2 \boldsymbol{t} = \Delta u_2$$
$$(\Delta \boldsymbol{p}_1 \times \Delta \boldsymbol{p}_2)\boldsymbol{t} = 0$$

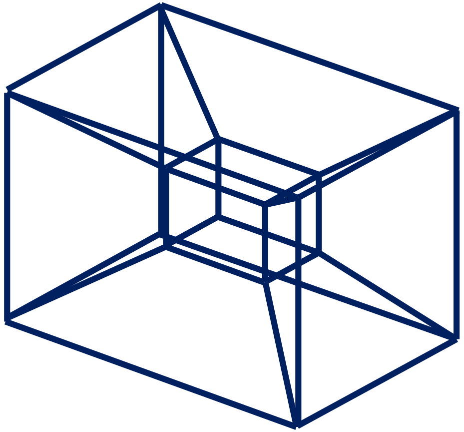
Source: Christian Schuller, ShaderX5

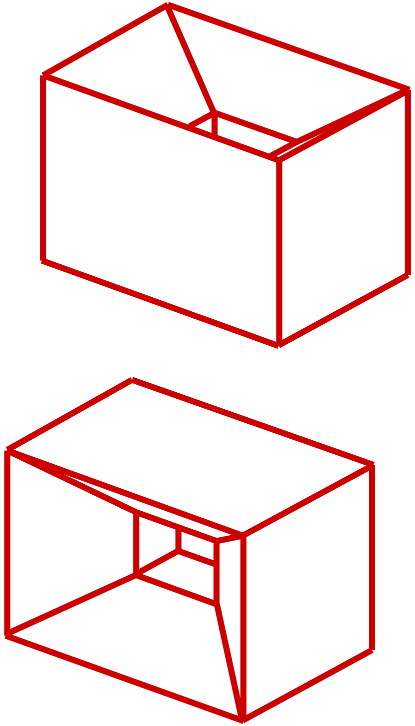
Triangle Mesh

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



Wireframe Model



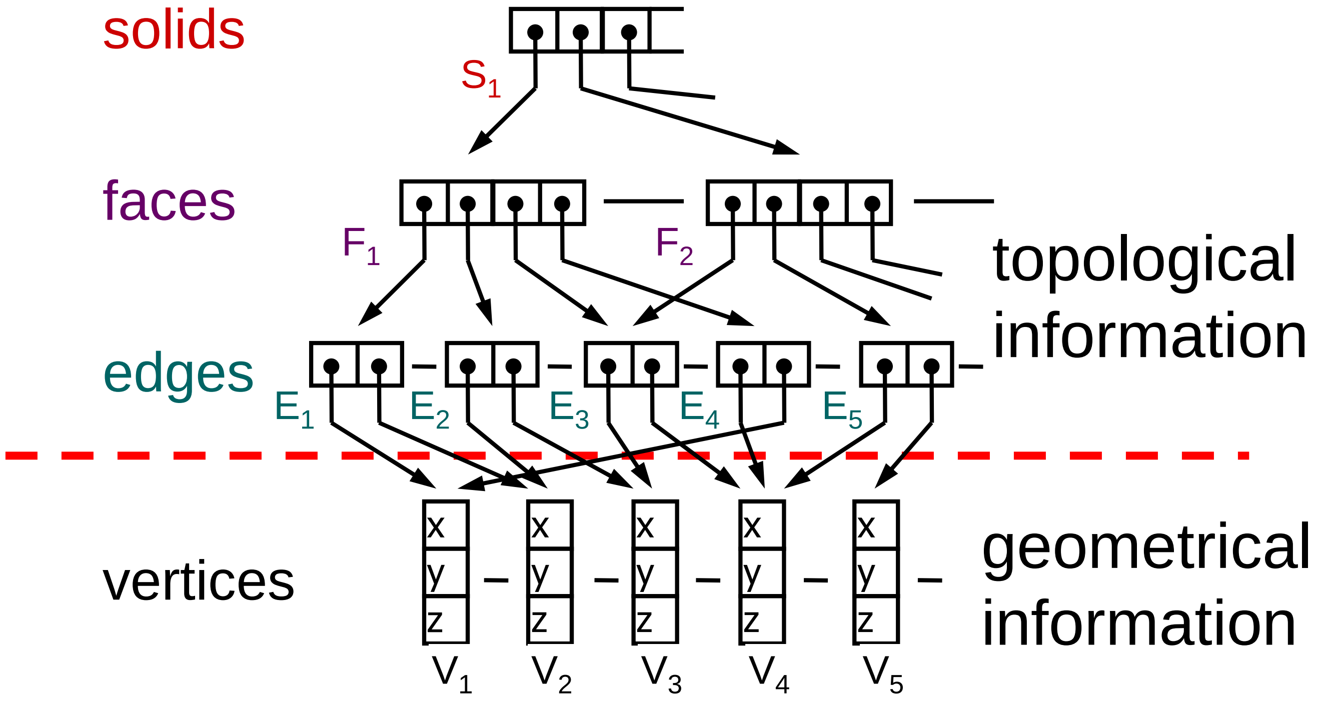


Ambiguous interpretation

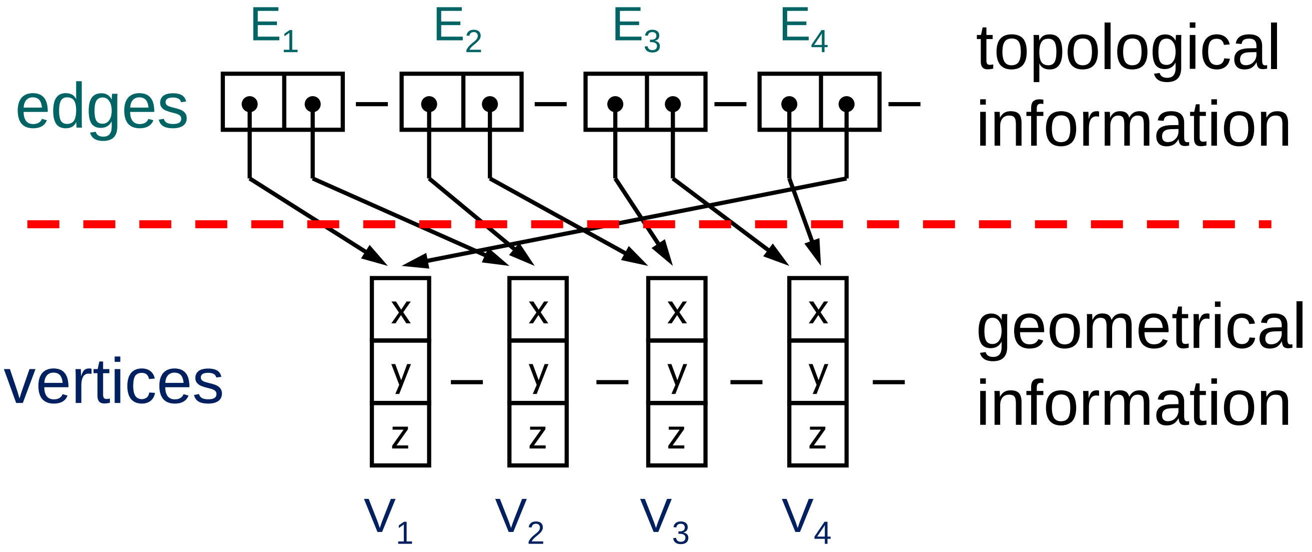
But very useful for debugging!

Mesh Graph

Hierarchical representation

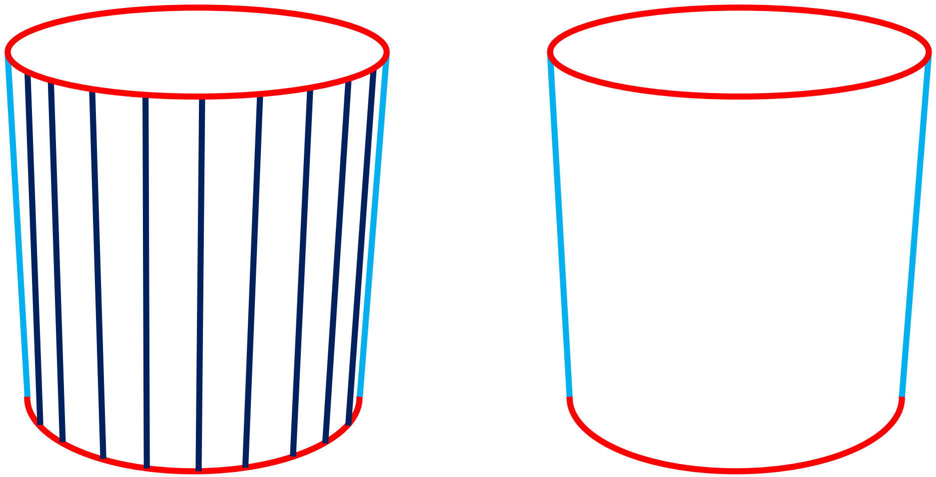


Wireframe Model



Edge Types

- Sharp edge (real)
- Interior edge (auxiliary)
- Silhouette edge (while rendering)



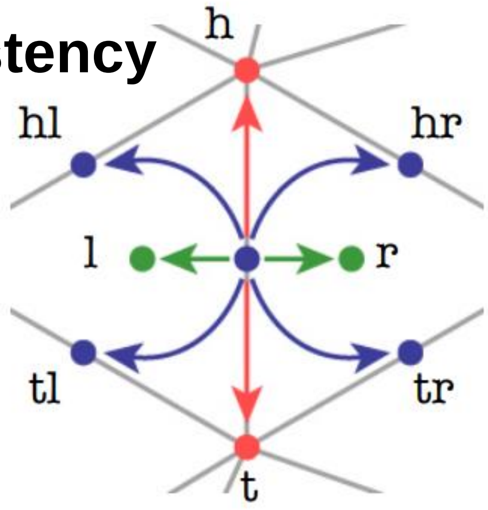
Face Types

- 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

- Information about the neighborhood
- Useful for **editing & maintaining consistency**

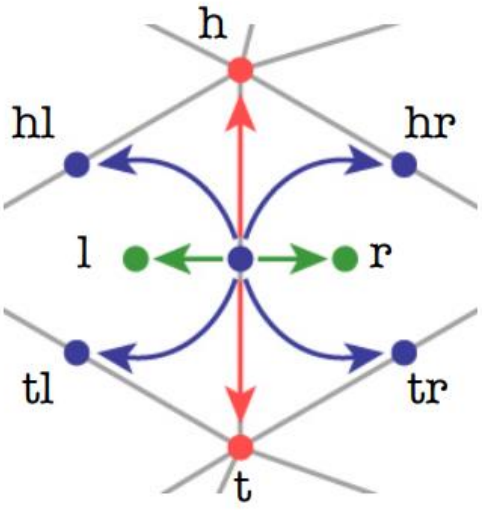
```
Edge {
        Edge* hl, hr, tl, tr;
        Vertex* h, t;
        Face* l, r;
}
Face {
        Edge* e; // any adjacent edge
}
Vertex {
        Vector3 position; // geometrical information
        Edge* e; // any incident edge
}
```



Source: Dinesh Manocha, Stanford

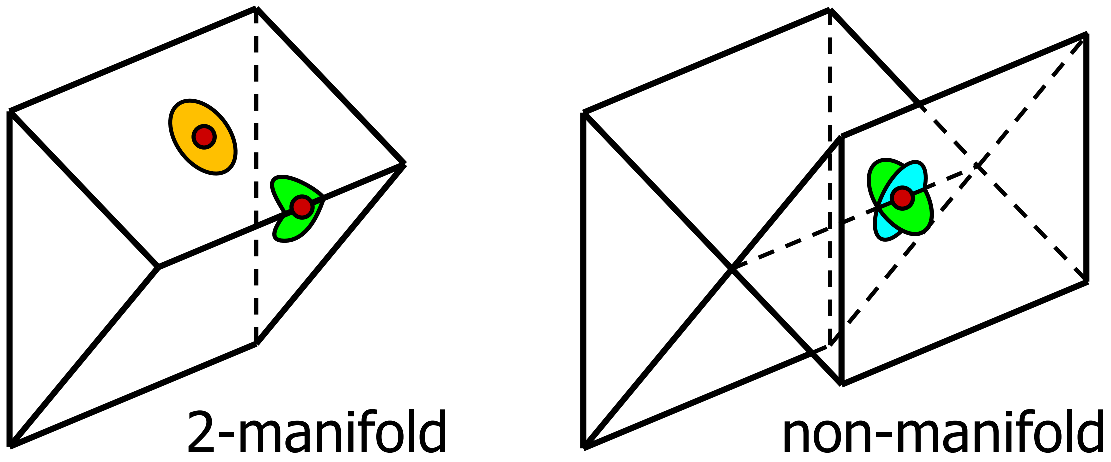
Winged-Edge: Finding edges of a face

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



Manifolds

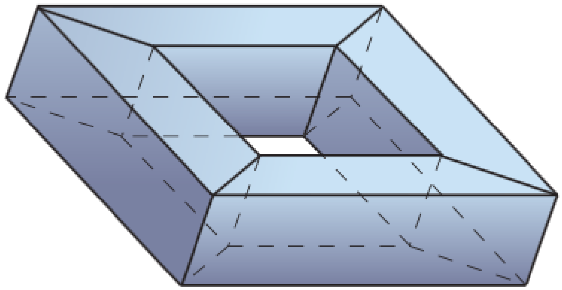
- 2-manifold: for every surface point there is a neighborhood topologically equivalent with plane
- Important for manufacturing, CAD/CAM



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)

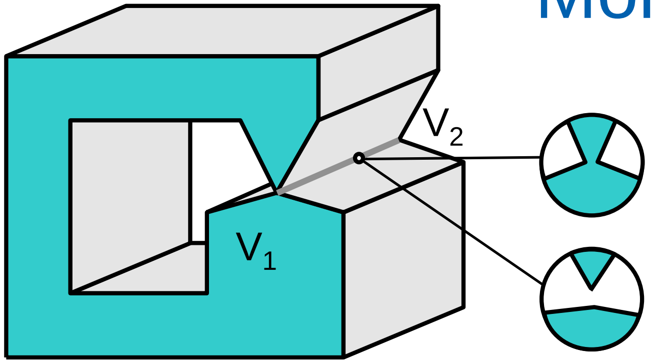


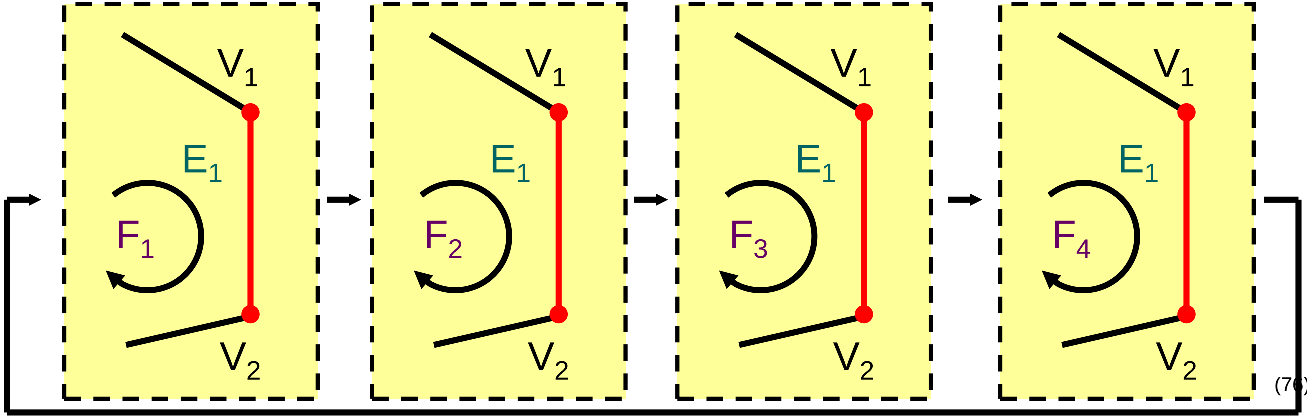
V=16
E=32
F=16
R=0
H=1
S=1

Representing non-manifolds

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

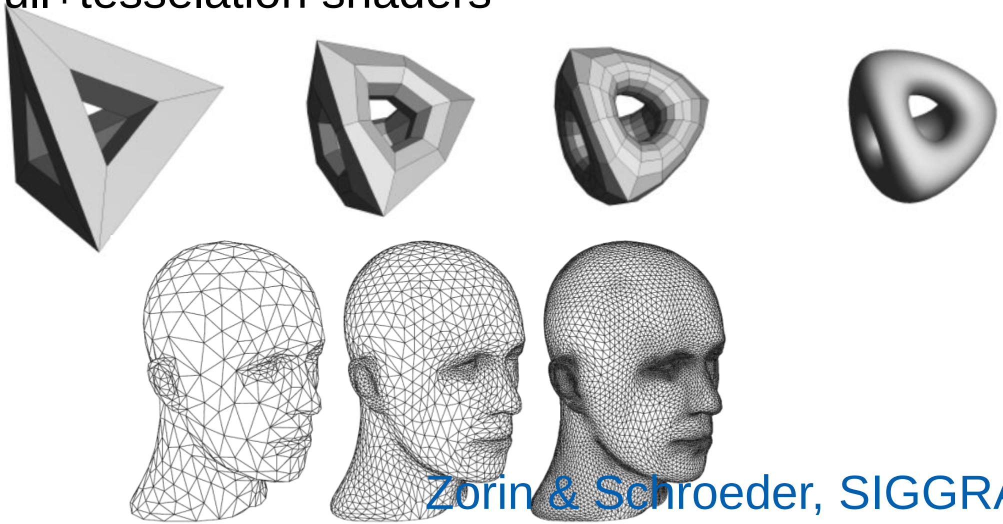
More in VGE!





Subdivision Surfaces

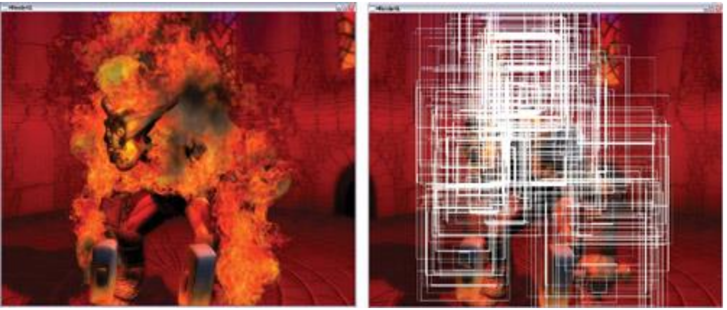
- Progressively subdivide coarse mesh
  - Subdivide triangles/quads & reposition vertices based on their neighborhood
- Different subdivision schemes
  - Loop, Catmull-Clark, Doo-Sabin
  - HW support: hull+tesselation shaders



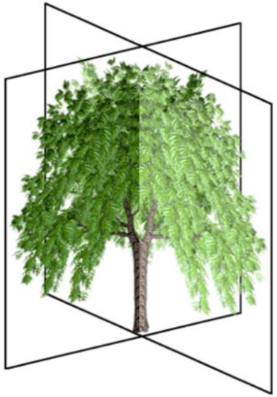
Zorin & Schroeder, SIGGRAPH 99

Sprites, Billboards

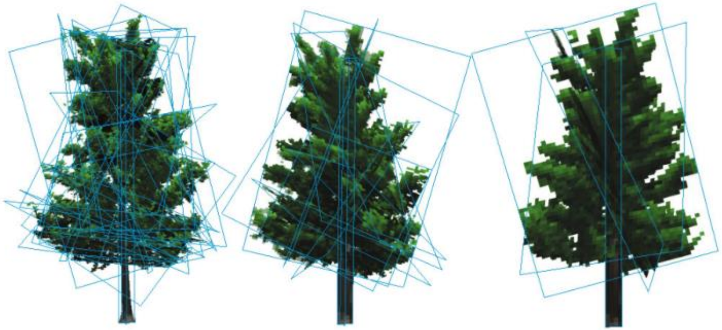
- Replacing geometry with images – sprites / billboards/ impostors
- Billboard: oriented sprite
  - towards a camera or based on object features …



[Nguyen 2004] Fire in the Vulcan Demo. NVIDIA.



[Umlauf 2004] Image-Based Rendering of Forests.



[Fuhrmann 2005] Extreme Model Simplification for Forest Rendering

Other B-reps

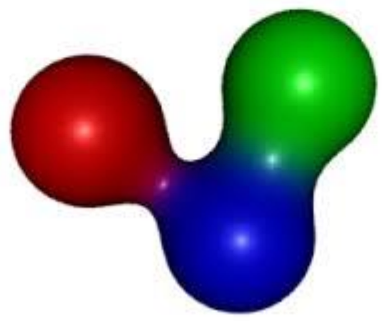
- Parametric surfaces
  - Bezier surface, Coons surface, NURBS
- Height fields
  - Regular / irregular
  - Terrains
- Impostors with depth
- LODs
  - Multiresolution representation

B-rep: Summary

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

Implicit Surfaces

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



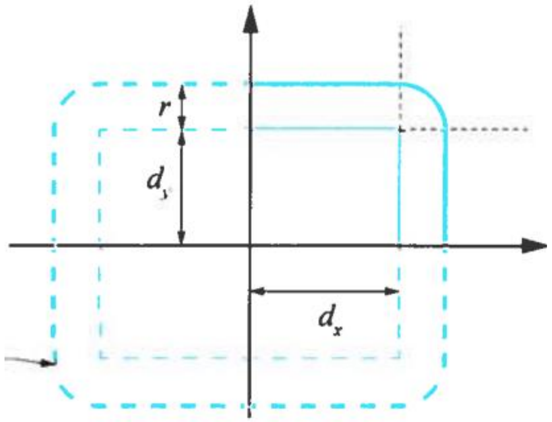
Source: scratchapixel.com

Examples:

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

$$f(\boldsymbol{p}) = \boldsymbol{n}\,\boldsymbol{p} + d$$

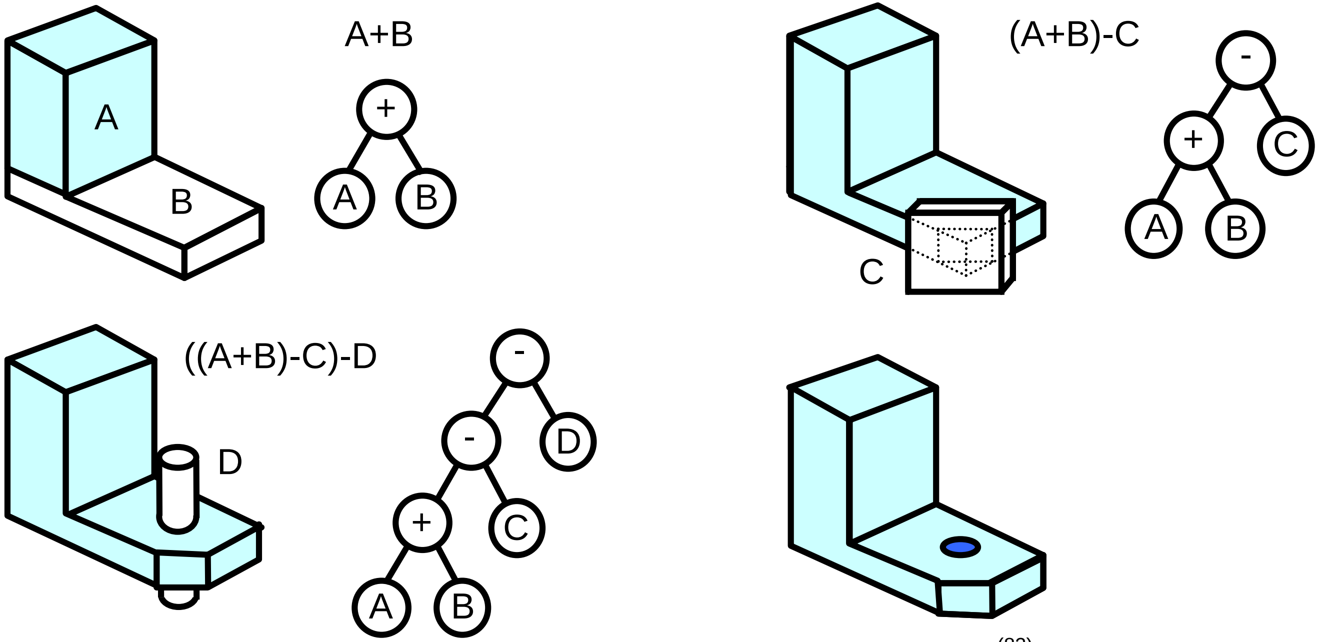
$$f(\boldsymbol{p}) = ||\max(|\boldsymbol{p}| - \boldsymbol{d}, 0)|| - r$$



Source: Real-Time Rendering 4th edition

CSG Tree

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



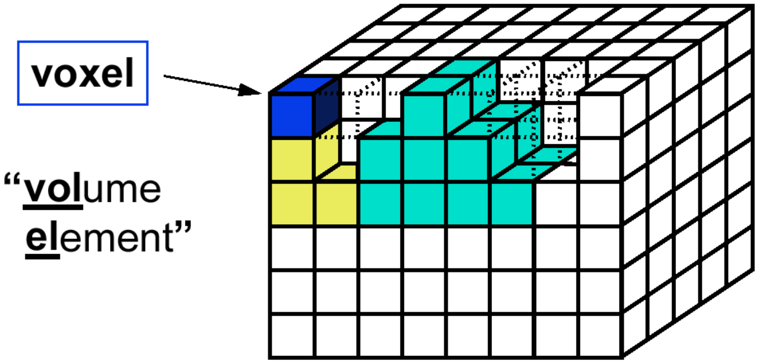
CSG Tree - Properties

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

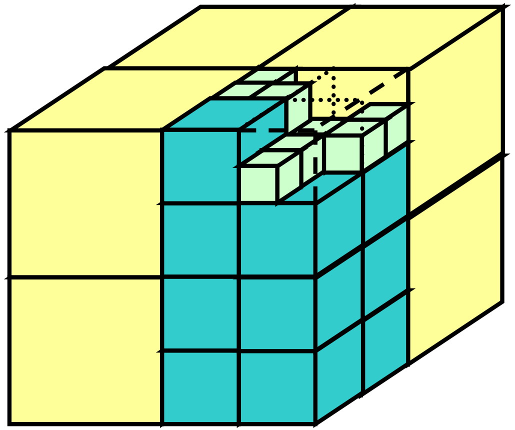
DP: Markéta Karaffová 2016

Volumetric representation

3D grid (regular structure)



Octree (hierarchical structure)



Volumetric rep.- properties

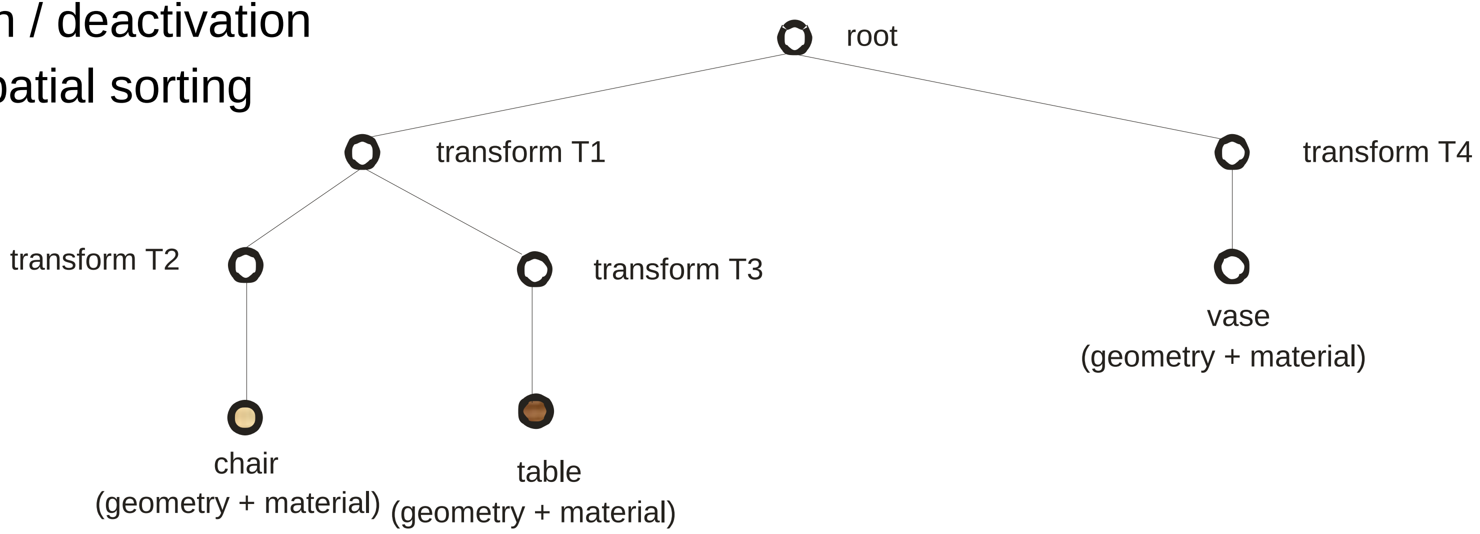
- Easy „point in solid“ test
- More difficult rendering (ray casting)

Source: Ikits et al. GPU Gems 2

DP: M. Benatsky 2011

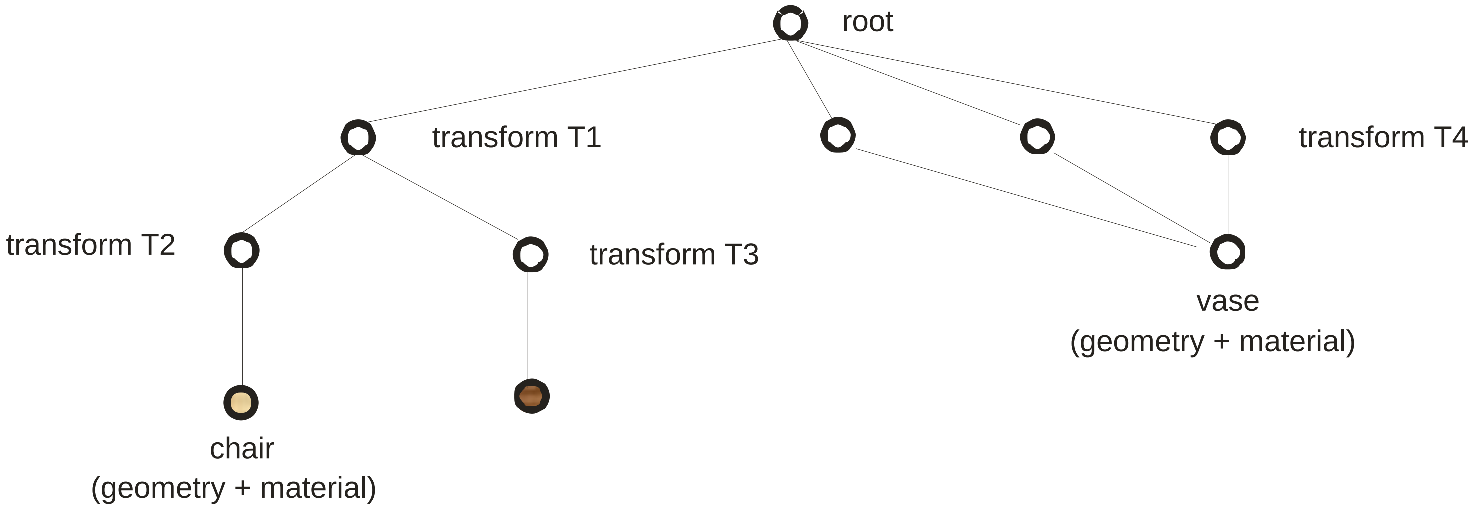
Scene Graph

- 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?