Computer Graphics Charles University

3D Scene Representations

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

3D Scene Representations

- **Volumetric representation**
  - Direct information about the internal structure
  - Easy test **„point×solid”** (does the point lie inside?), **display** can be difficult
  - Sometimes used as a **auxiliary data structure** for fast searching
- **Surface representation**
  - Direct information about the surface (edges, faces)
  - Difficult test **„point×solid”** (a „solid“ does not have to have internal volume), relatively easy **display**

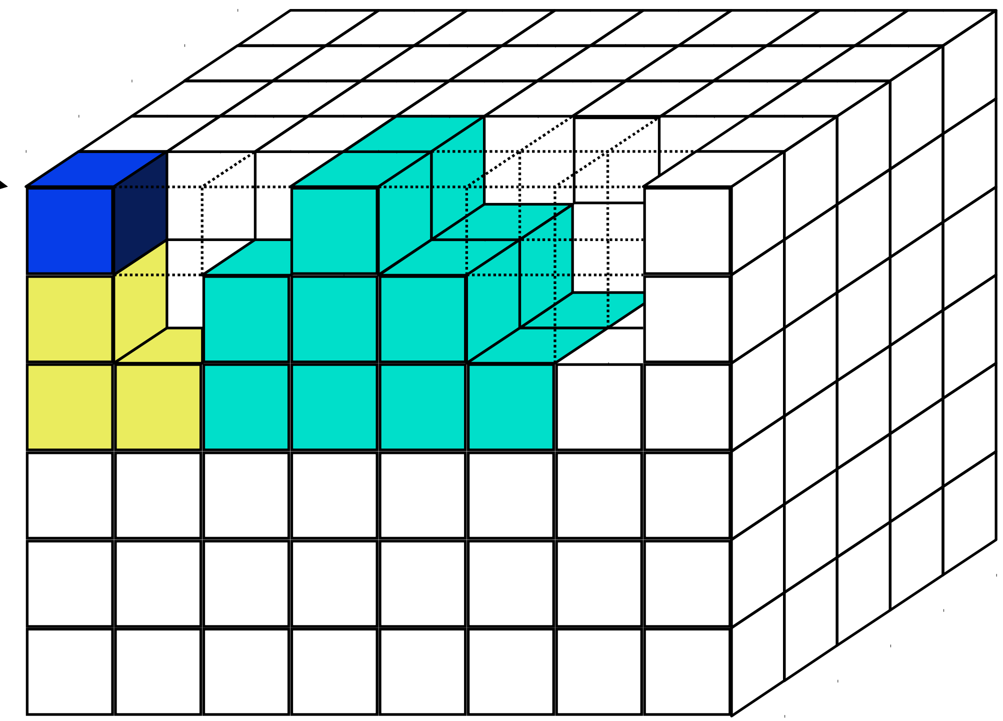
Volume Representation

- ✔ **Numerical representations**
  - – Direct quantification of occupied space (discrete representation – limited precision)
  - – Mainly used as auxiliary data structures for fast searches
  - – Also: medical data! (MRI, CT)
  - – **Voxel data, octree**
- ✔ **CSG representation**
  - – Powerful and accurate method (basic solids, geometric transformations, <u>set theoretic operations</u>)
  - – Difficult **display** (only with ray-based methods)

Numerical Representation

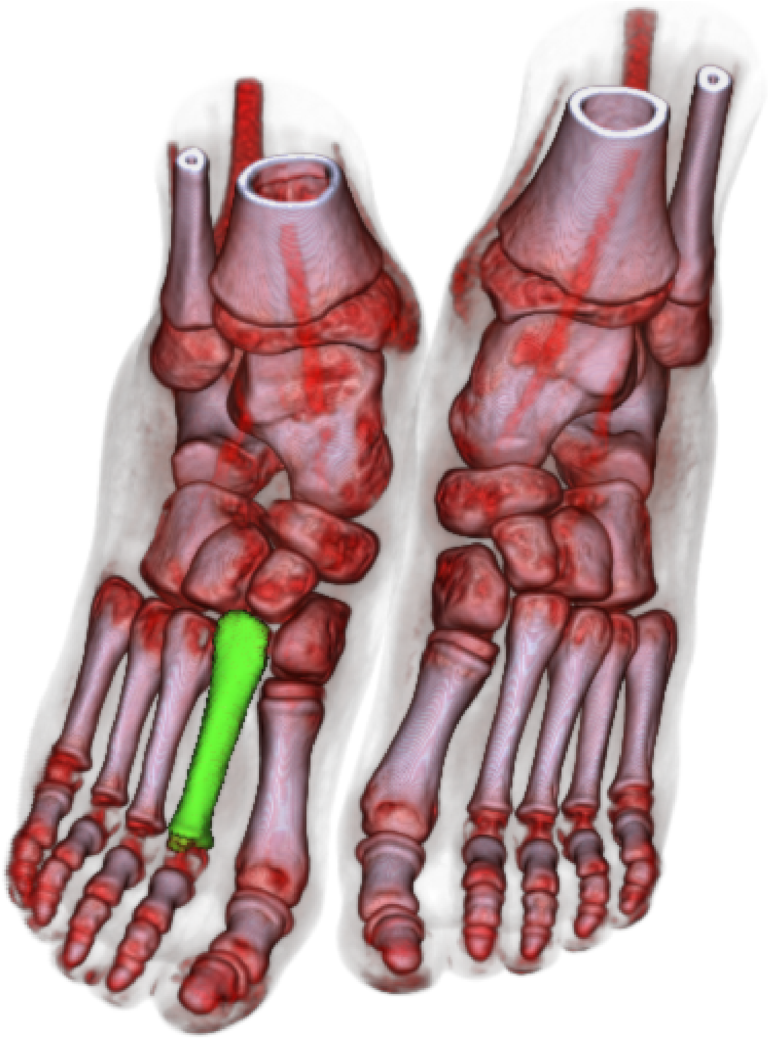
voxel

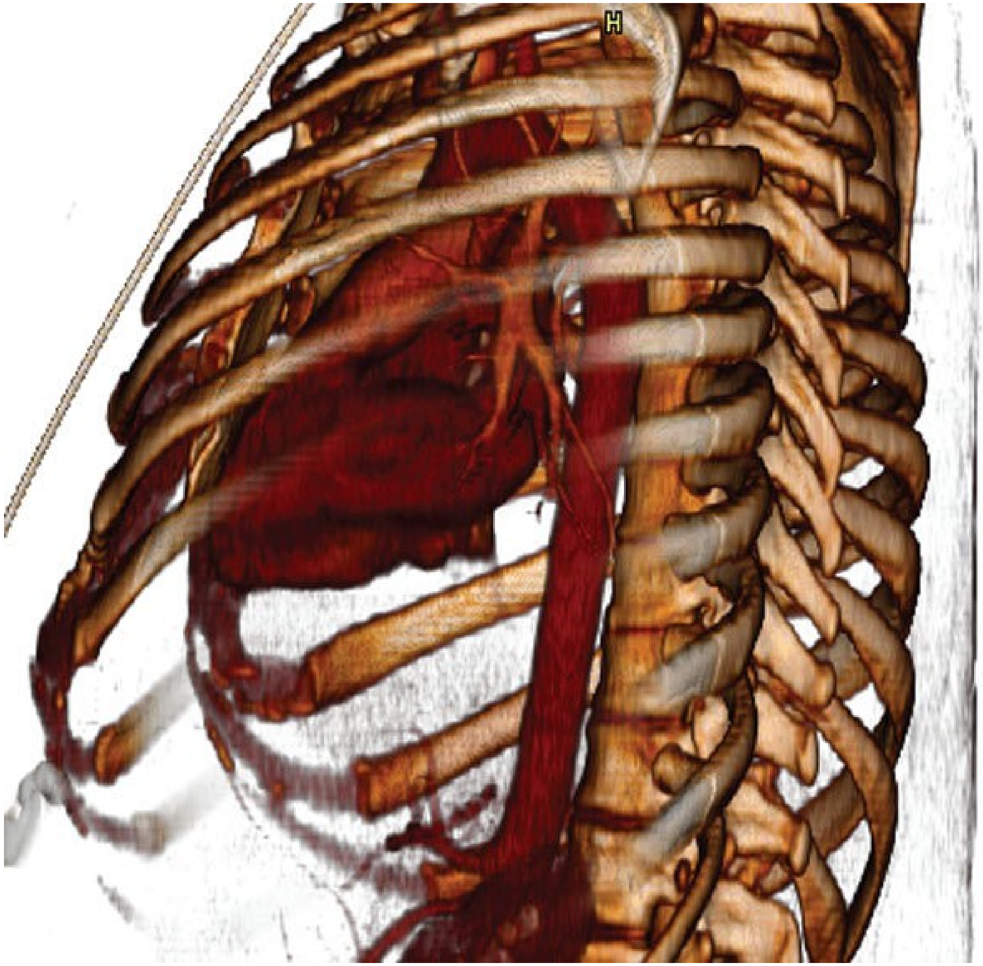
“**vol**ume **el**ement”



Volume of k×l×m voxels

Single bit: 0 – empty, 1 – solid
Multi-bit A: 0 – emtpy, n > 0 – solid with index **n**
Multi-bit B: 0 – empty, n > 0: density value



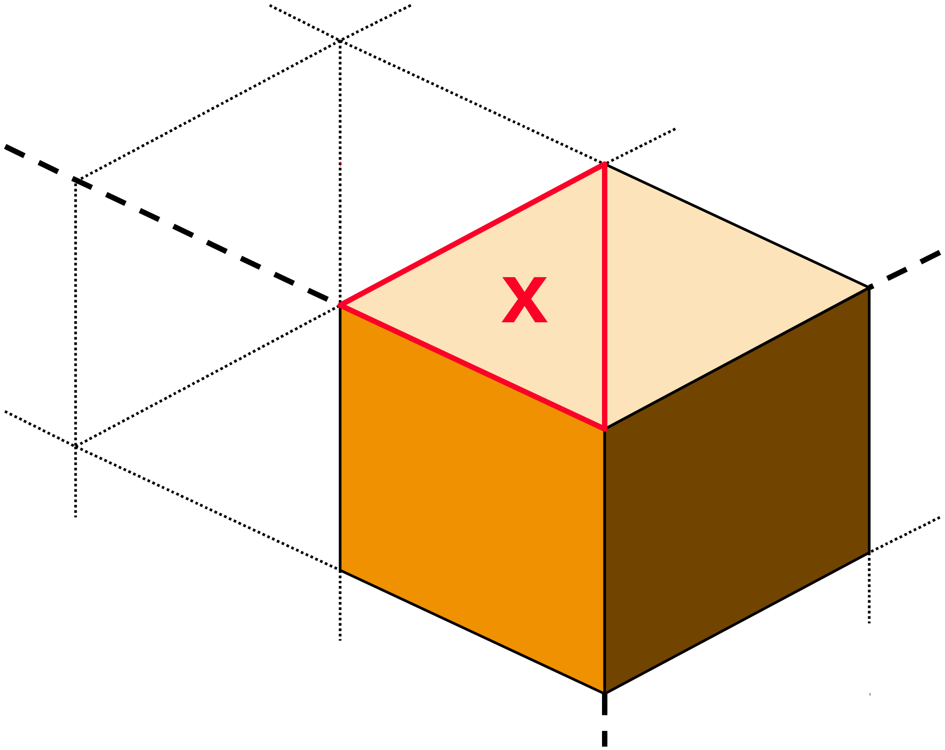


Displaying Numerical Volume Data

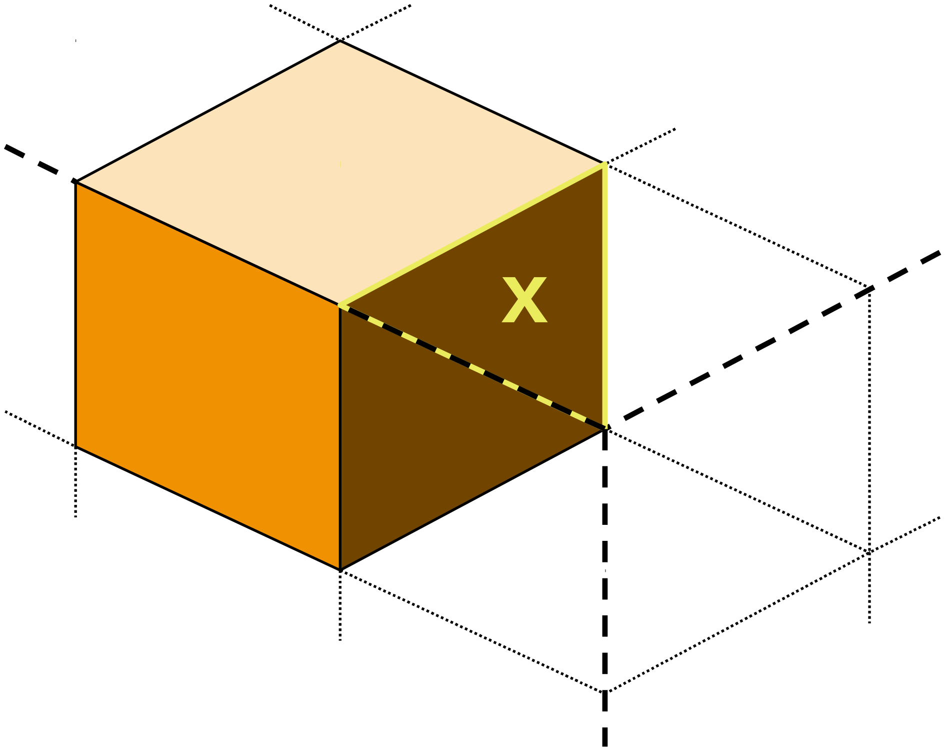
- **Drawing back-to-front**
  - Shows only the front voxel faces
  - pouze stěny na povrchu těles (faces between **0** and **>0**)
  - Multiple re-drawing steps

- **Special projection**
  - Effective algorithm without re-drawing
  - **„Ant-attack”** on ZX-Spectru (128×128×8 voxels)

Special projection

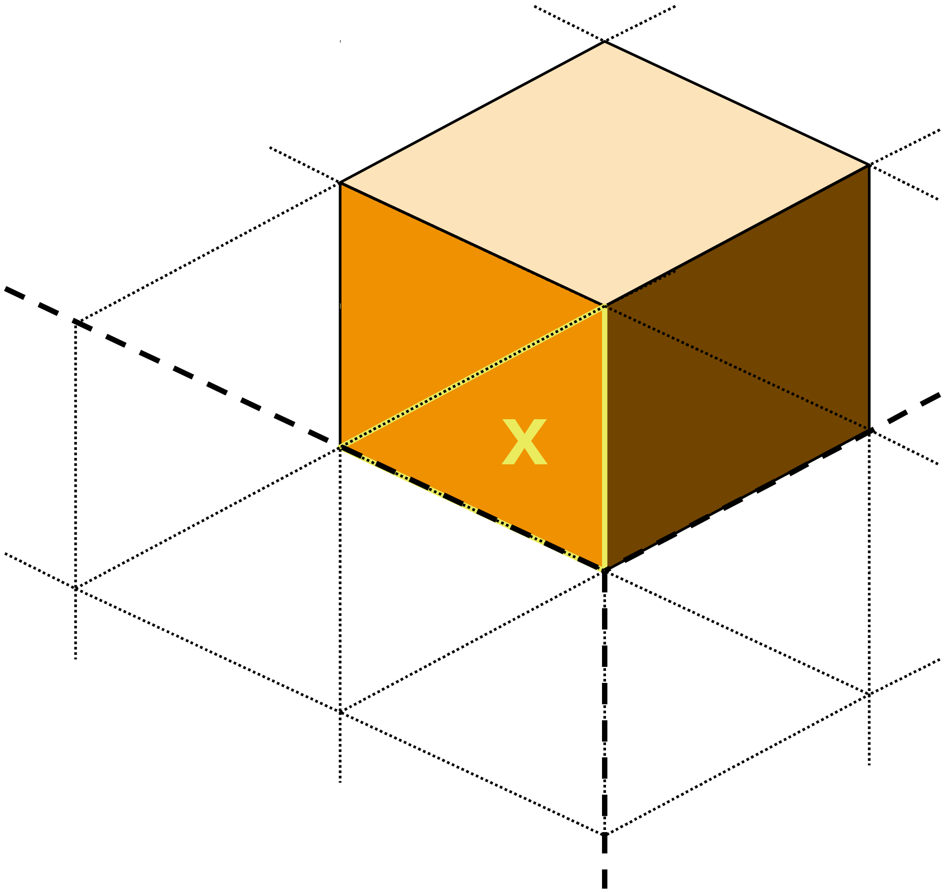


1. top face
[0,0,0]

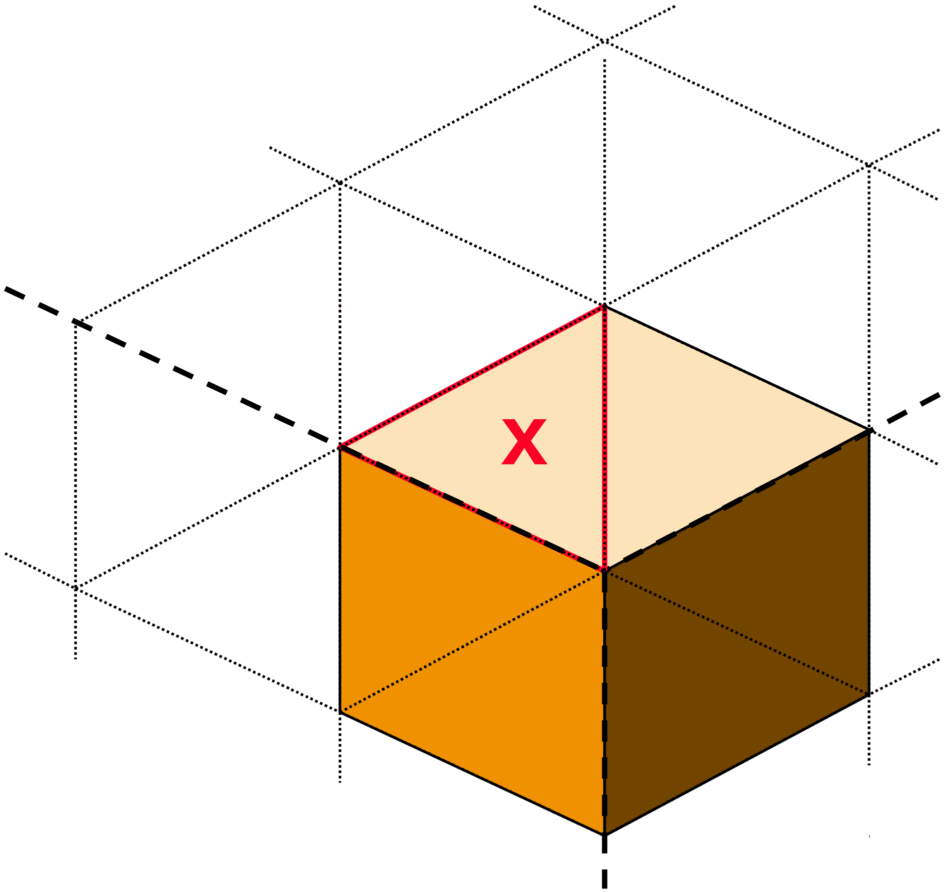


2. right face
[0,1,0]

Special projection

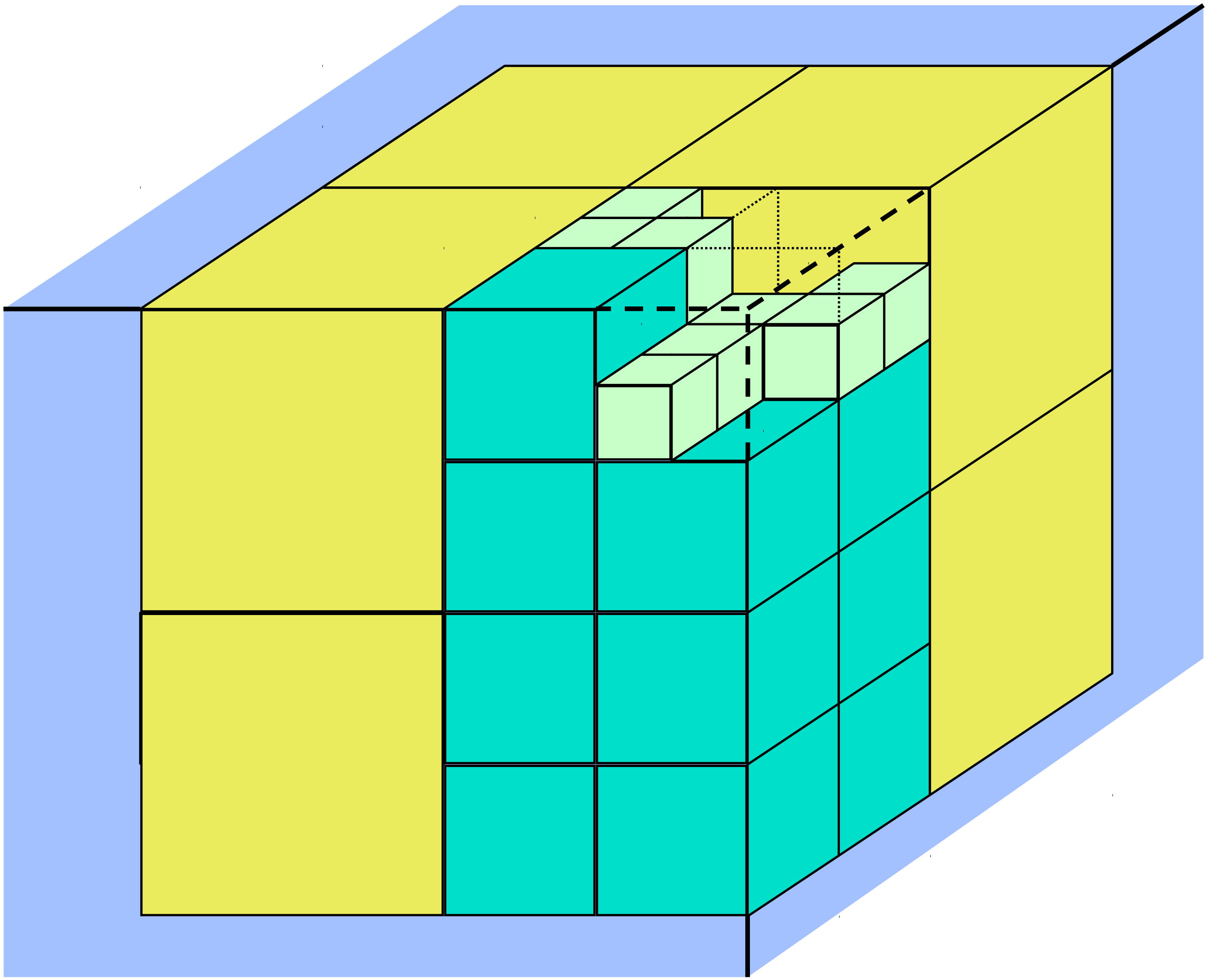


3. left face
[1,1,0]



4. top face
[1,1,1]

Octree

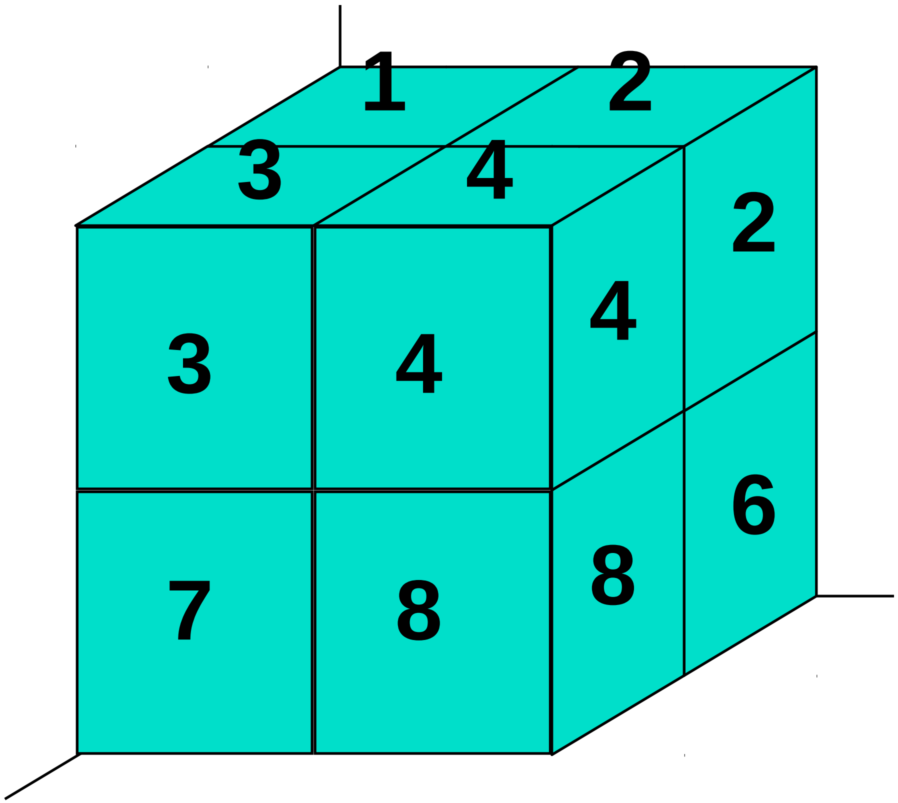


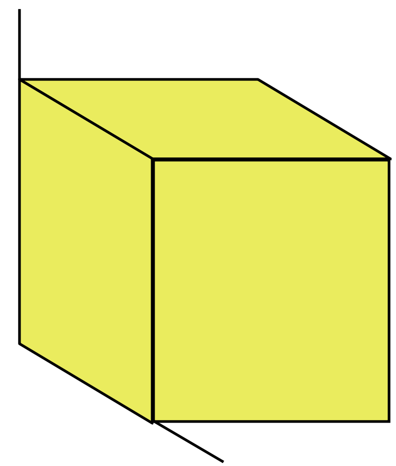
Octree

- **3D analogy of quadtrees**
  - If the interior of a cube is inhomogeneous, it is divided into eight sections (this is done down to voxel level)
  - Can save memory compared to cell model

- **Drawing back-to-front**
  - Only front-facing cube faces
  - Only faces of the cube
  - Multiple re-drawing of some pixels

Back-to-front drawing





sequence:

5-6-1-2-7-8-3-4

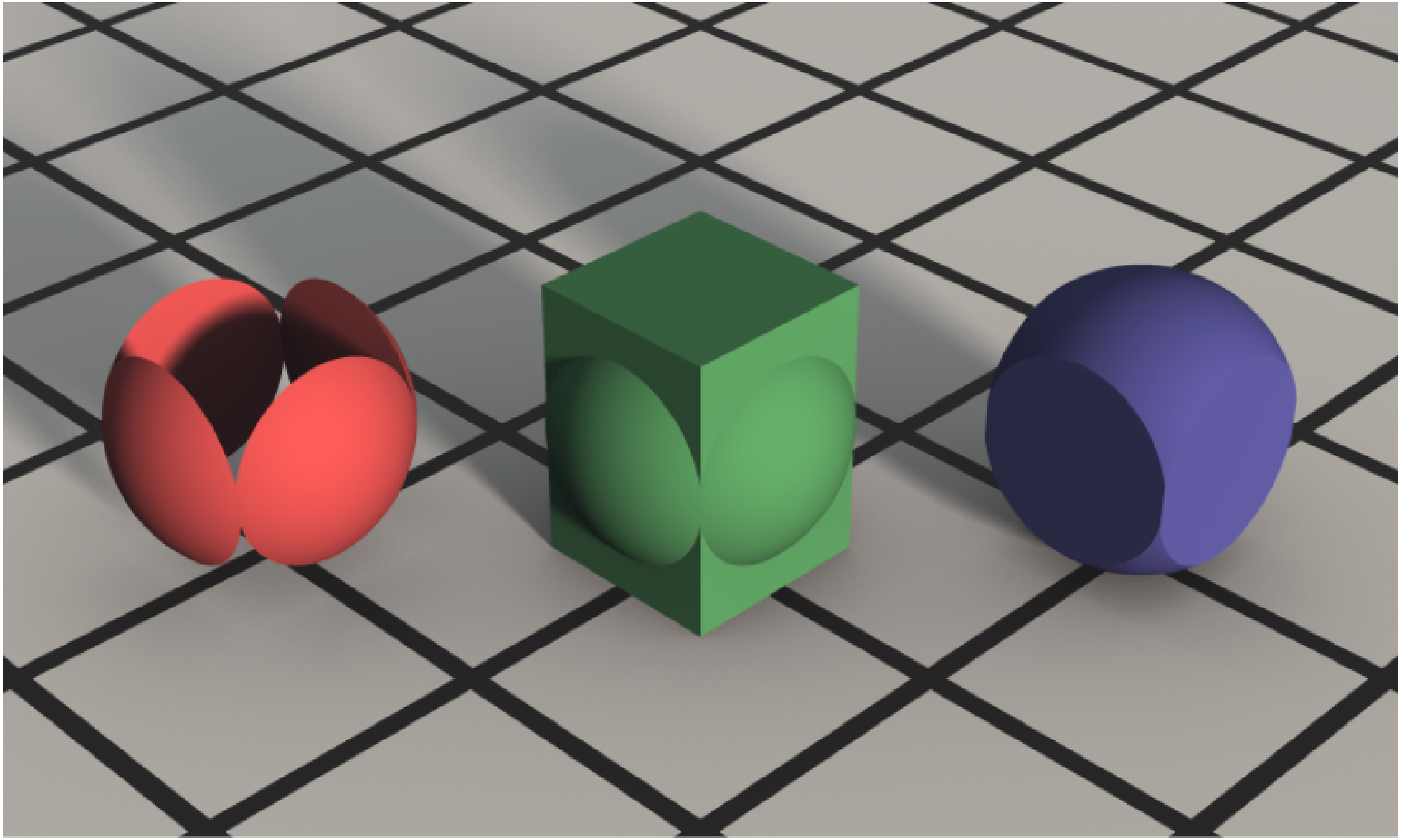
sequence:

6-5-2-1-8-7-4-3

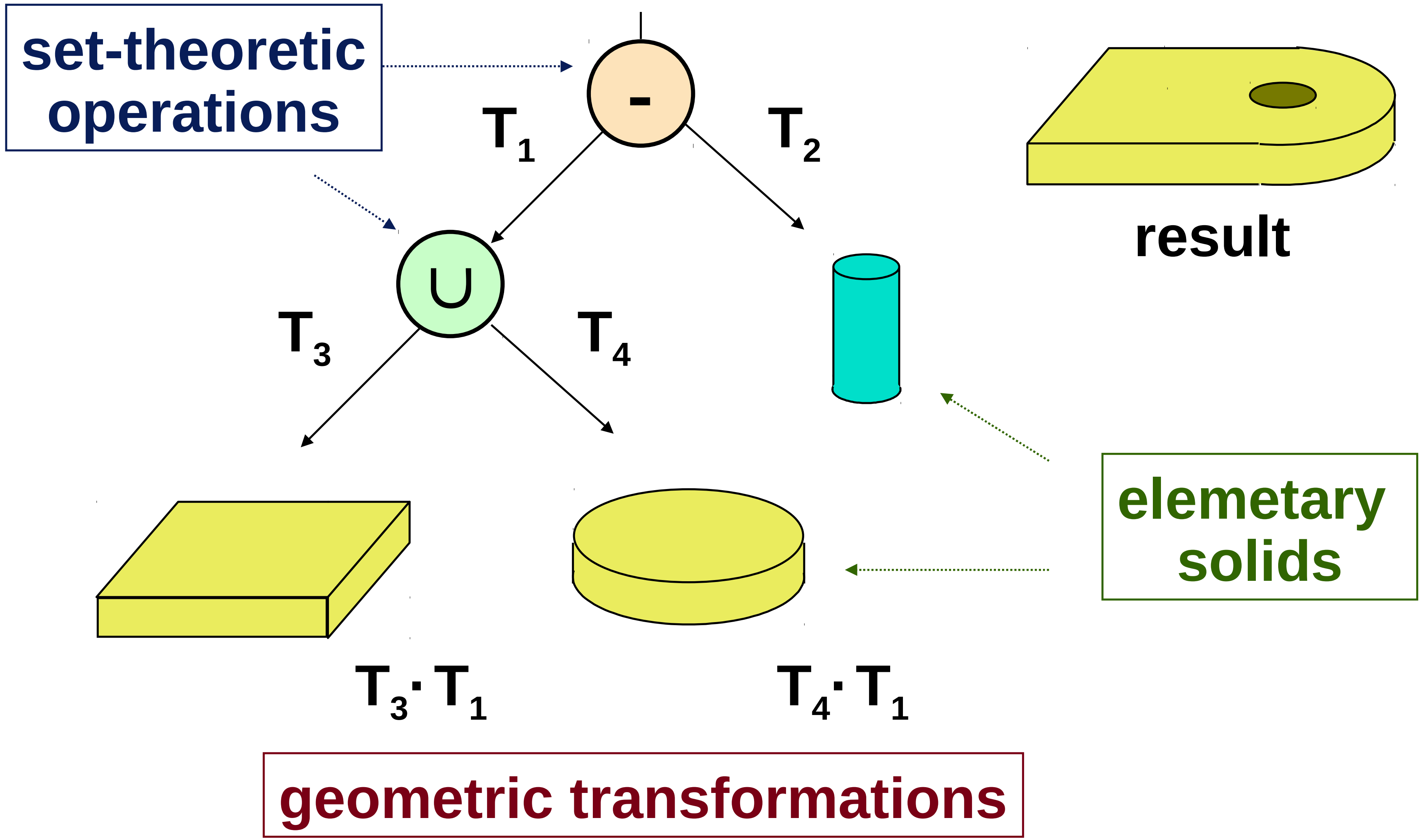
CSG („Constructive Solid Geometry“)

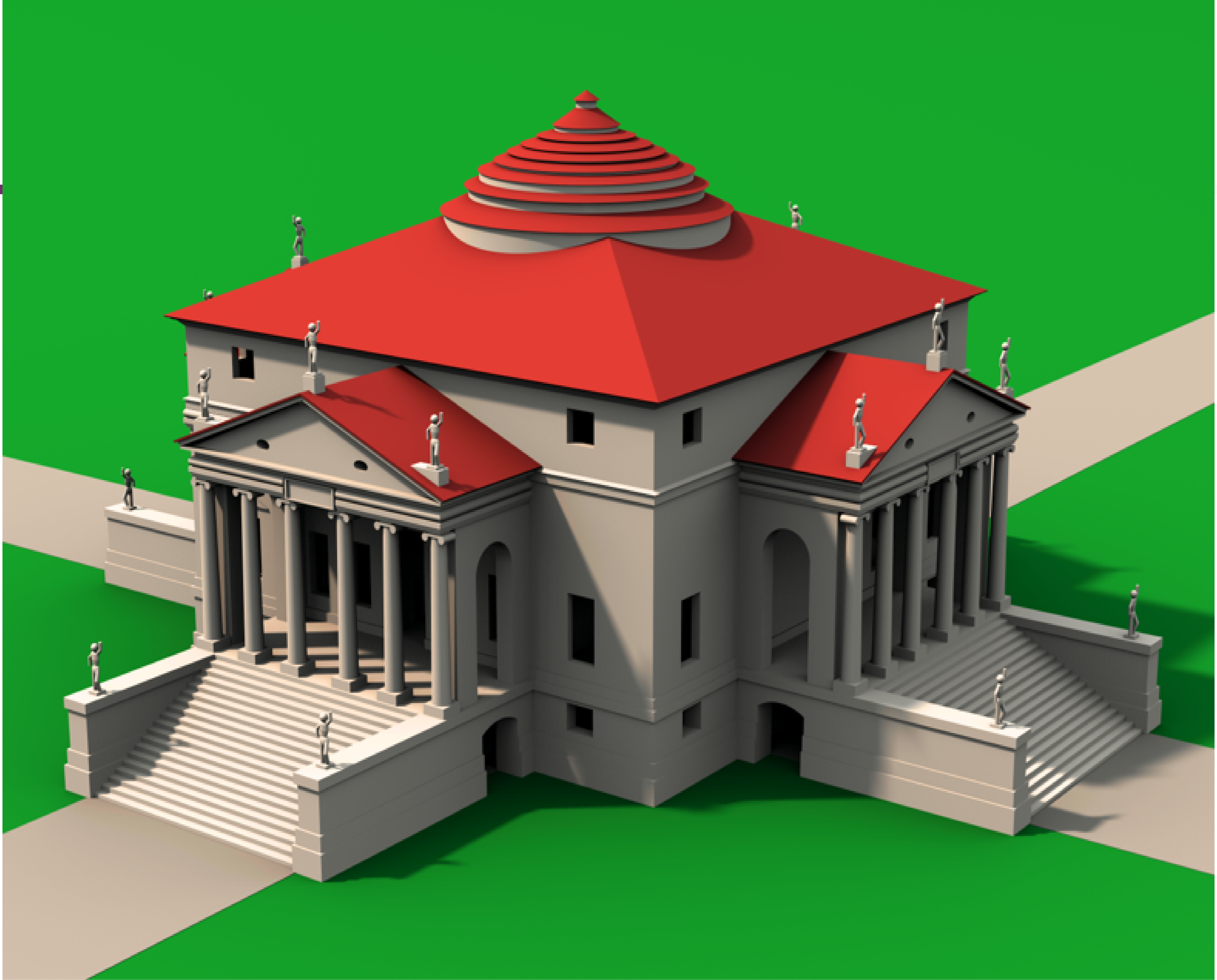
- **Elementary geometric bodies**
  - – Easy to define and evaluate
  - – Cube, sphere, half-space, cylinder, …
- **Set-theoretic operations**
  - – Assembly of compound solids from elemetary solids
  - – OR, AND, SUB
- **Geometric transformations**
  - – Modifications of elementary and compound solids
  - – (homogeneous) matrix transformations

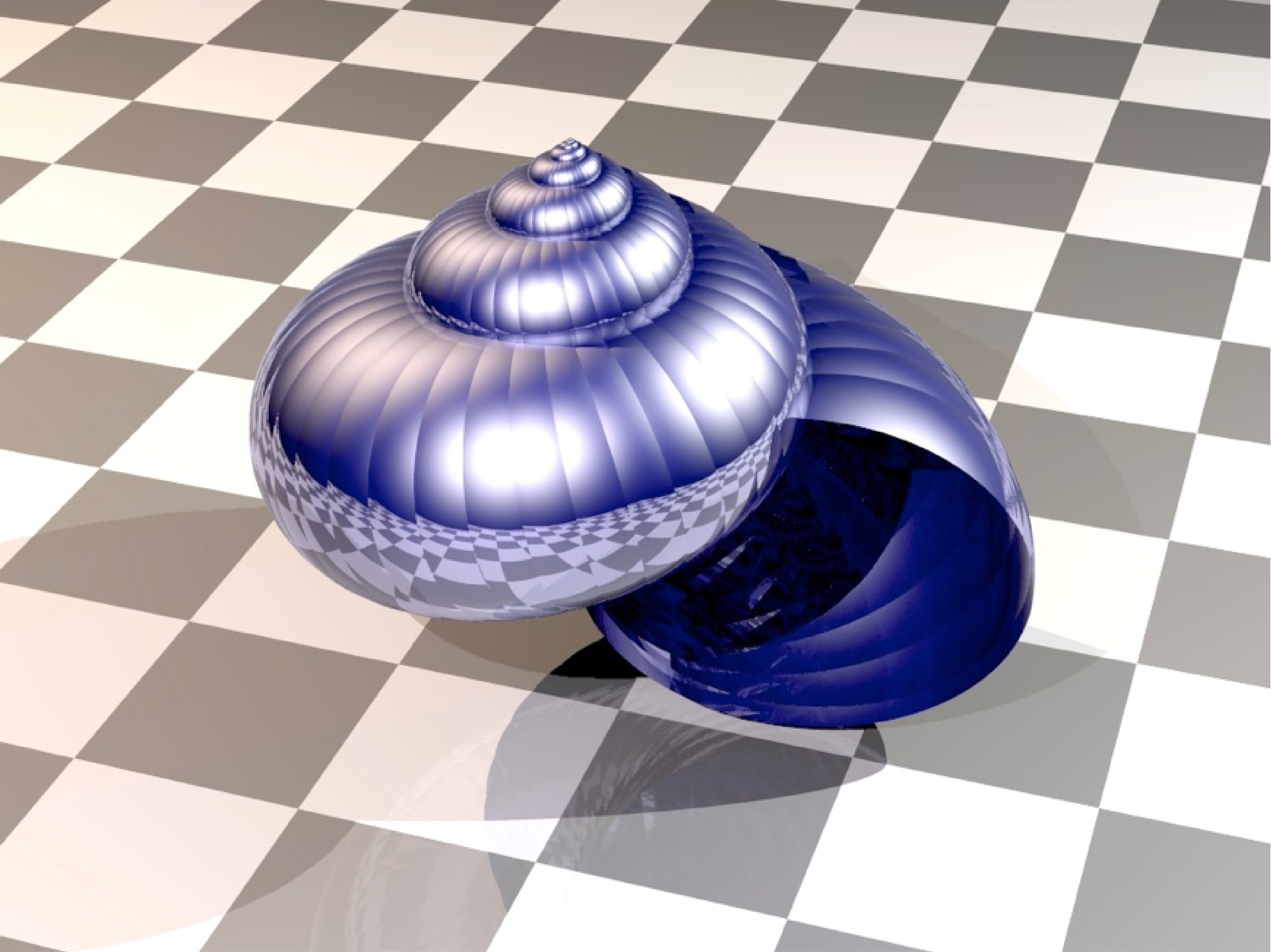
CSG Operators



CSG tree


set-theoretic operations
-
∪
T_1
T_2
T_3
T_4
result
elemetary solids
$T_3 \cdot T_1$
$T_4 \cdot T_1$
geometric transformations





Transformations in a CSG Tree

- **Semantics of** transformations $\mathbf{T}_i$
  - $\mathbf{T}_i$ can be stored at each point in a CSG tree
  - When entering a sub-tree (genuine subtree, elementary solid), there is a coordinate transform to the subtree system
  - Any subtree is transformed by $\mathbf{T}_i$ in addition to what has happened in the tree traversal up to that point
- **Easy transformation** of any subtree
  - Only one matrix has to be changed
- **Inverse transformation $\mathbf{T}_i^{-1}$**
  - For calculations on the tree (test point×CSG, rendering)

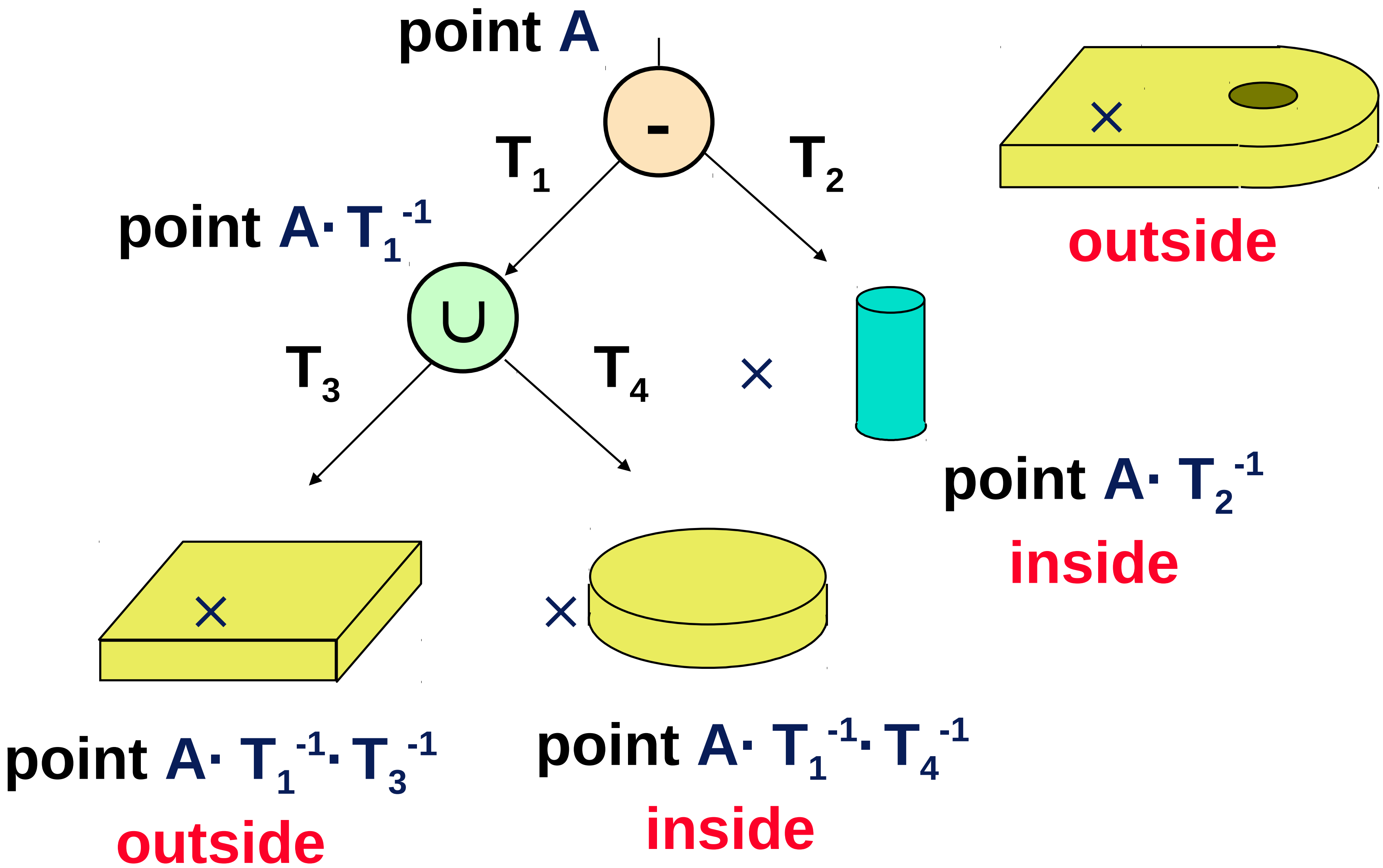
Transformations in a CSG Tree

- **Transformations are stored only in the leaves**
  - **Cumulative totals** (e.g. $\mathbf{T_3 \cdot T_2 \cdot T_1}$, or the inverse $\mathbf{T_1^{-1} \cdot T_2^{-1} \cdot T_3^{-1}}$)
  - Speed-up for calculations on the entire tree (for editing, one retains individual transformations)
- **Efficient storage of elementary solids**
  - Solids are stored in their **normal position**, all changes are done via geometric transformations
  - Cube (unit, one corner in the origin), sphere (radius 1, center in the origin), cylinder (x/y center in the origin, height 1 along the z axis), ...

Test „point×CSG tree”

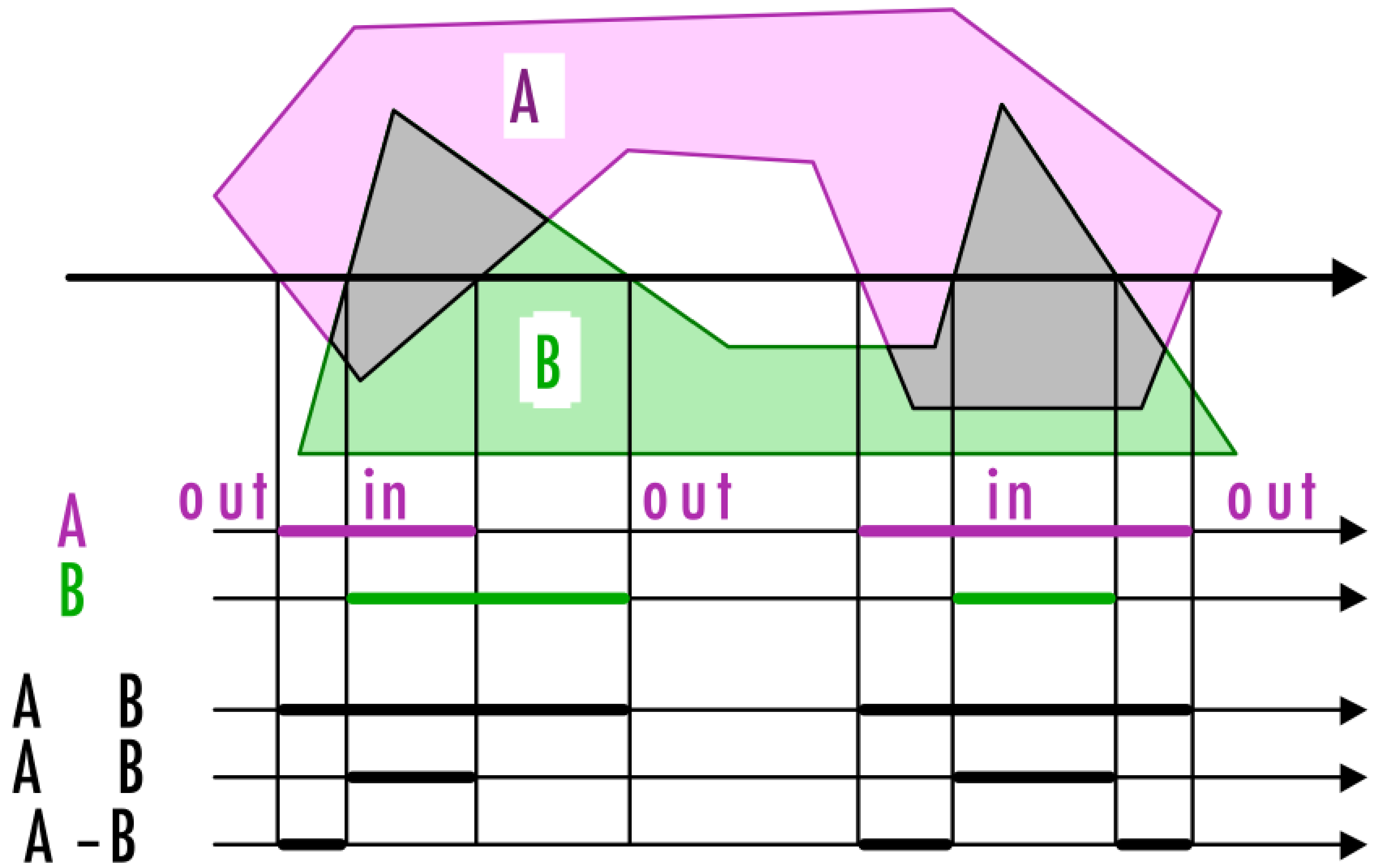
- **Is a given point A inside the solid?**
  - We might also want to know the sub-trees containing **A**
- **Tests „point×elementary solid”** are easy! (especially for standard solids)
- **Traversal of the CSG tree**
  - The coordinates of point **A** are converted to the coordinate systems of the traversed sub-trees (inverse transformation)
  - Instead of set theoretic operations, their **equivalent Boole operations** are used ($\vee$ instead of $\cup$, $\wedge$ instead of $\cap$, ...)

Test „point×CSG tree”



Rendering of CSG Trees

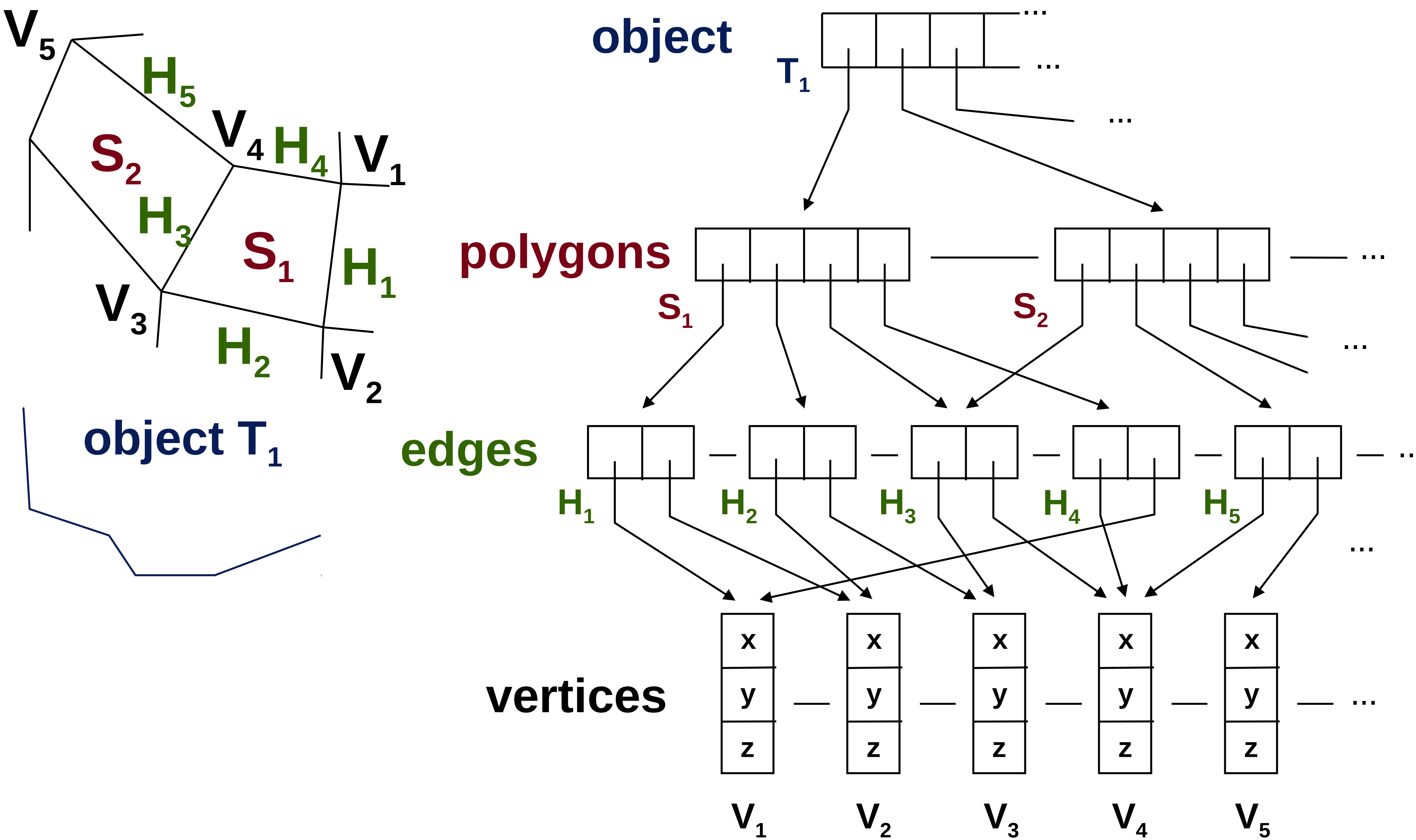
- **Conversion to surface representations**
  - For every **elementary solid**: conversion to a **polygon mesh**
  - **Set theory operations on those meshes** (limited precision: operations done on tesselated geometry!)
- **Ray based techniques** („Ray-casting")
  - Accurate results for pixel-based rendering
  - Computationally demanding method


A
B
out
in
out
in
out
A
B
A B
A B
A - B

Surface Representation

- **Wireframe models**
  - Pseudo-surface representation
  - Only **vertices** and **edges** of solids (visibility cannot be computed)
- **Vertices, edges, polygons, objects (VEPO)**
  - Complete topological information
- Special case of VEPS: **Winged-edge**
  - redundantní informace pro **rychlé vyhledávání** sousedních objektů (hrany incidentní s vrcholem, ..)

VEPO Surface Representation



Face-Vertex Meshes

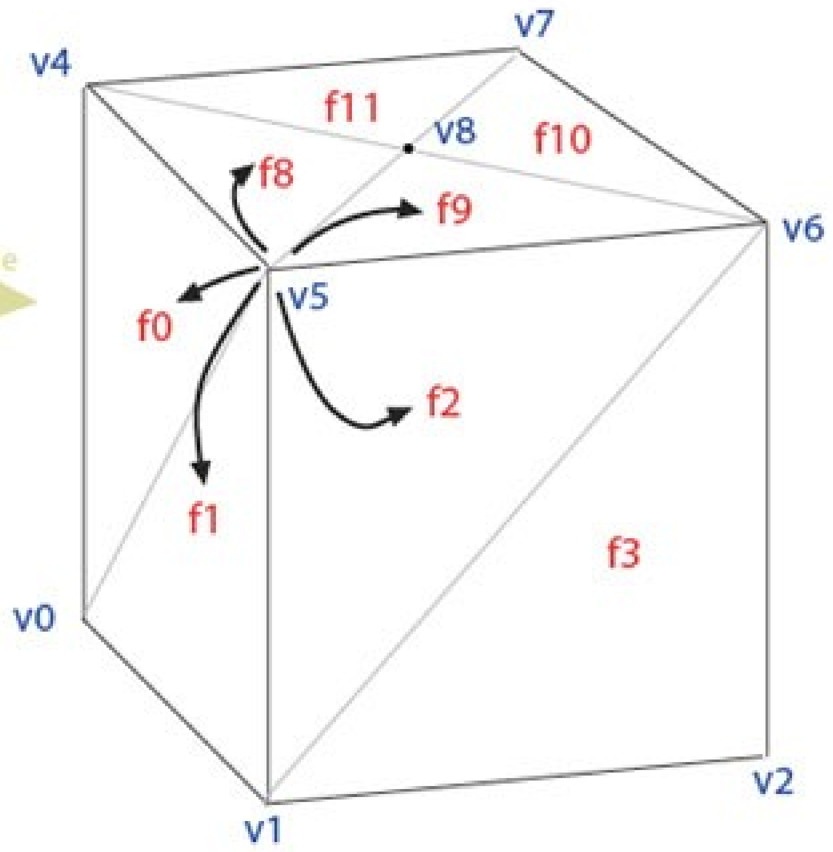
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

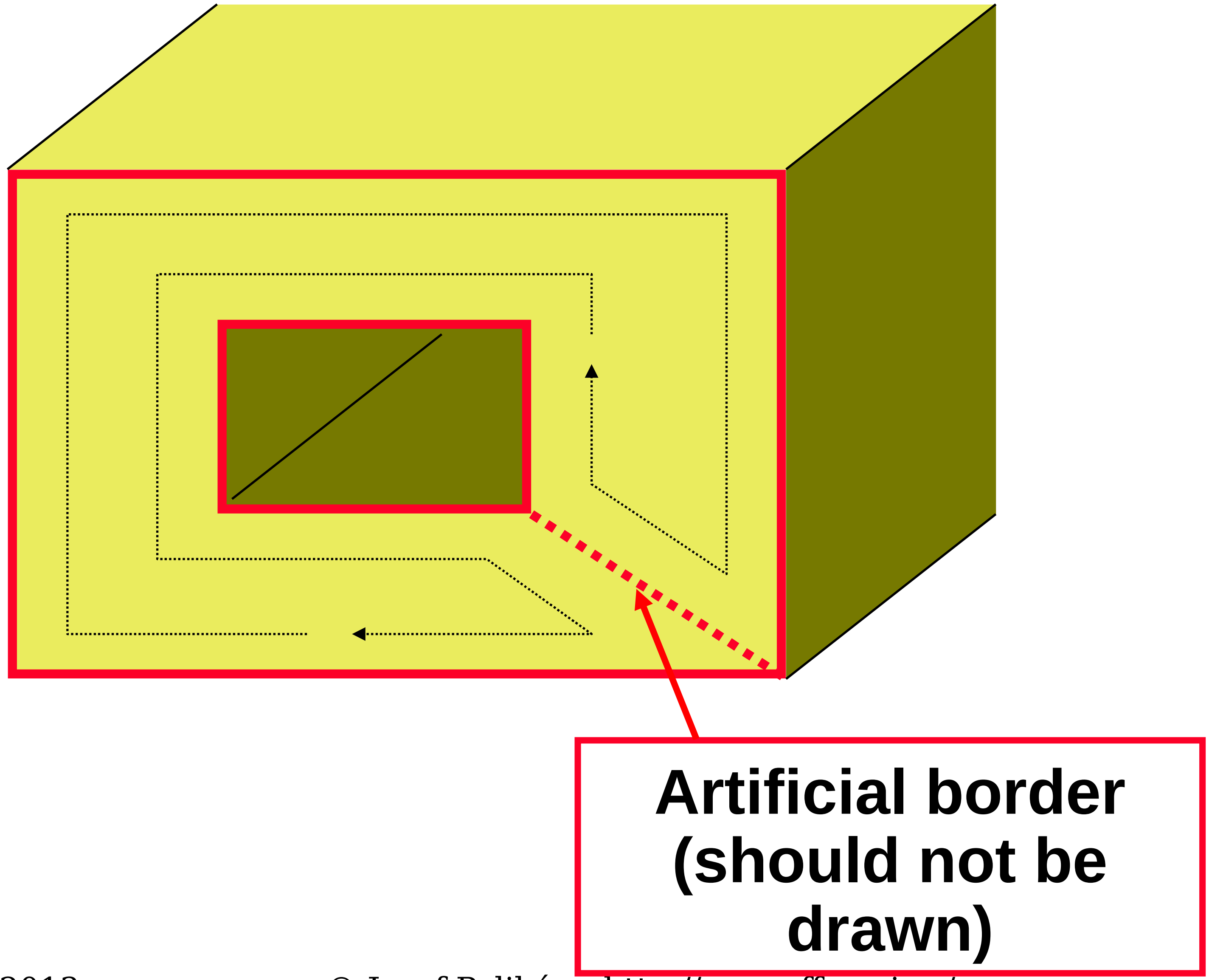
Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15

example

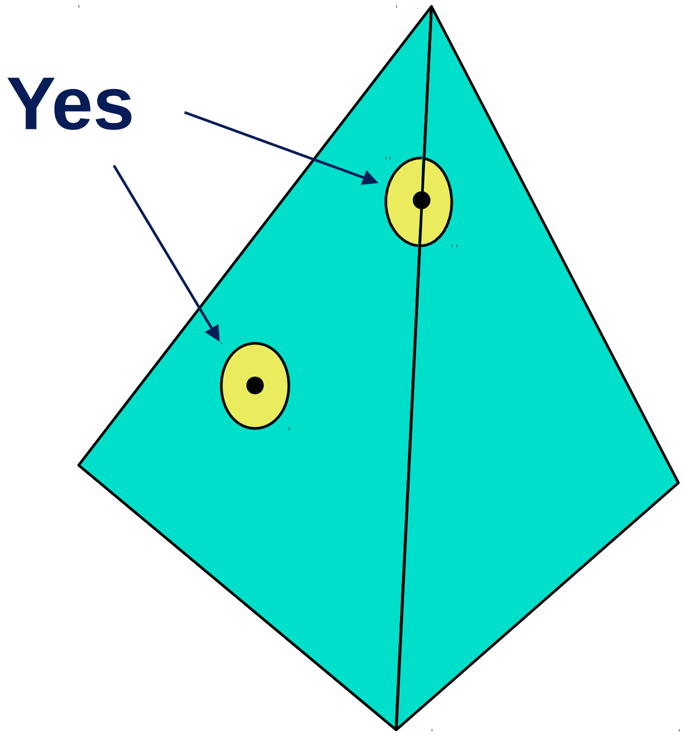


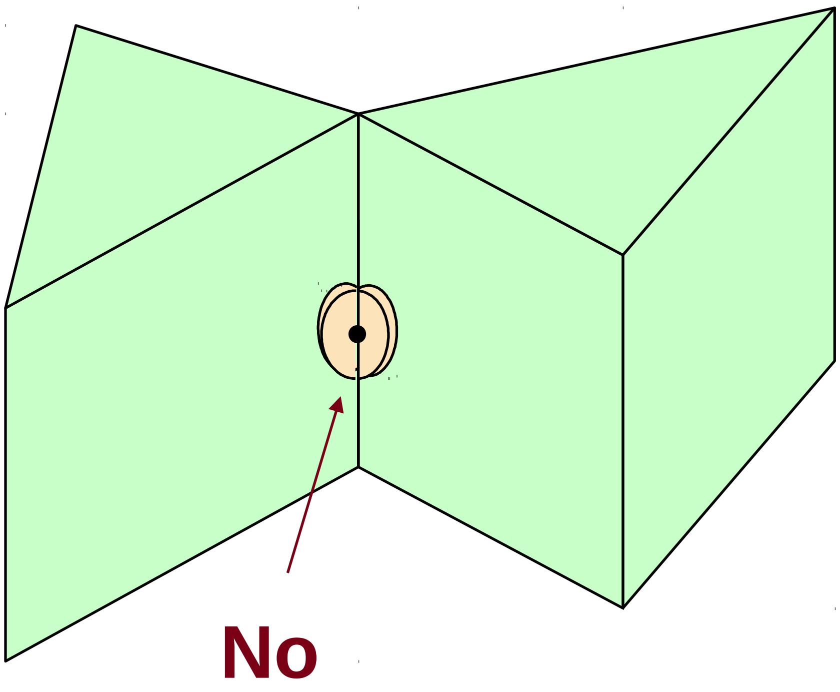
„Leaky” Polygon



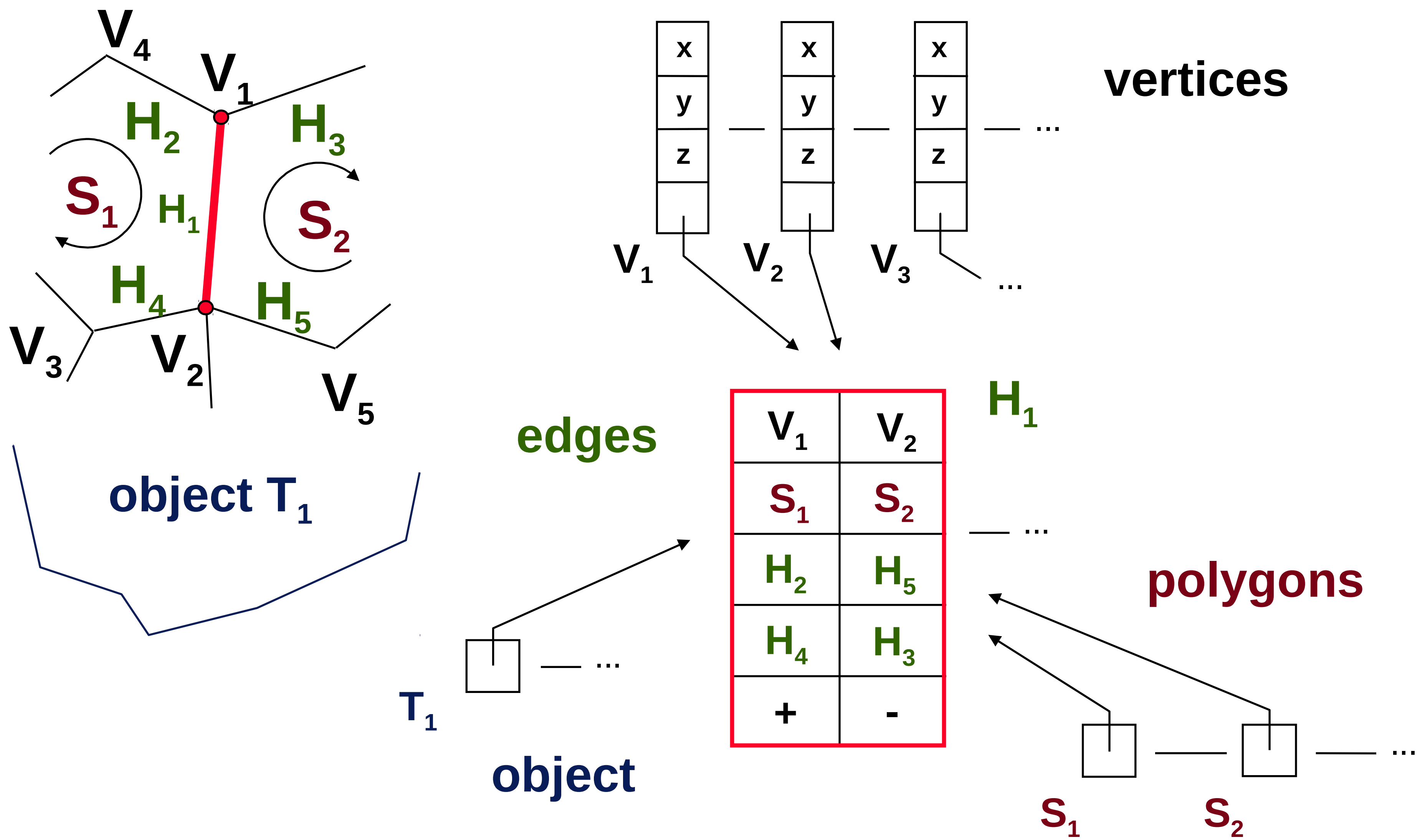
„2-Manifold” (i.e. Surfaces)

Def: for each surface point there exists a surround which is topologically equivalent to the plane





Winged Edge



V_4
V_1
H_2
H_3
S_1
H_1
S_2
H_4
H_5
V_3
V_2
V_5
object T_1
x
y
z
vertices
V_1
V_2
V_3
edges
H_1
V_1 V_2
S_1 S_2
H_2 H_5
H_4 H_3
+ -
polygons
T_1
object
S_1
S_2

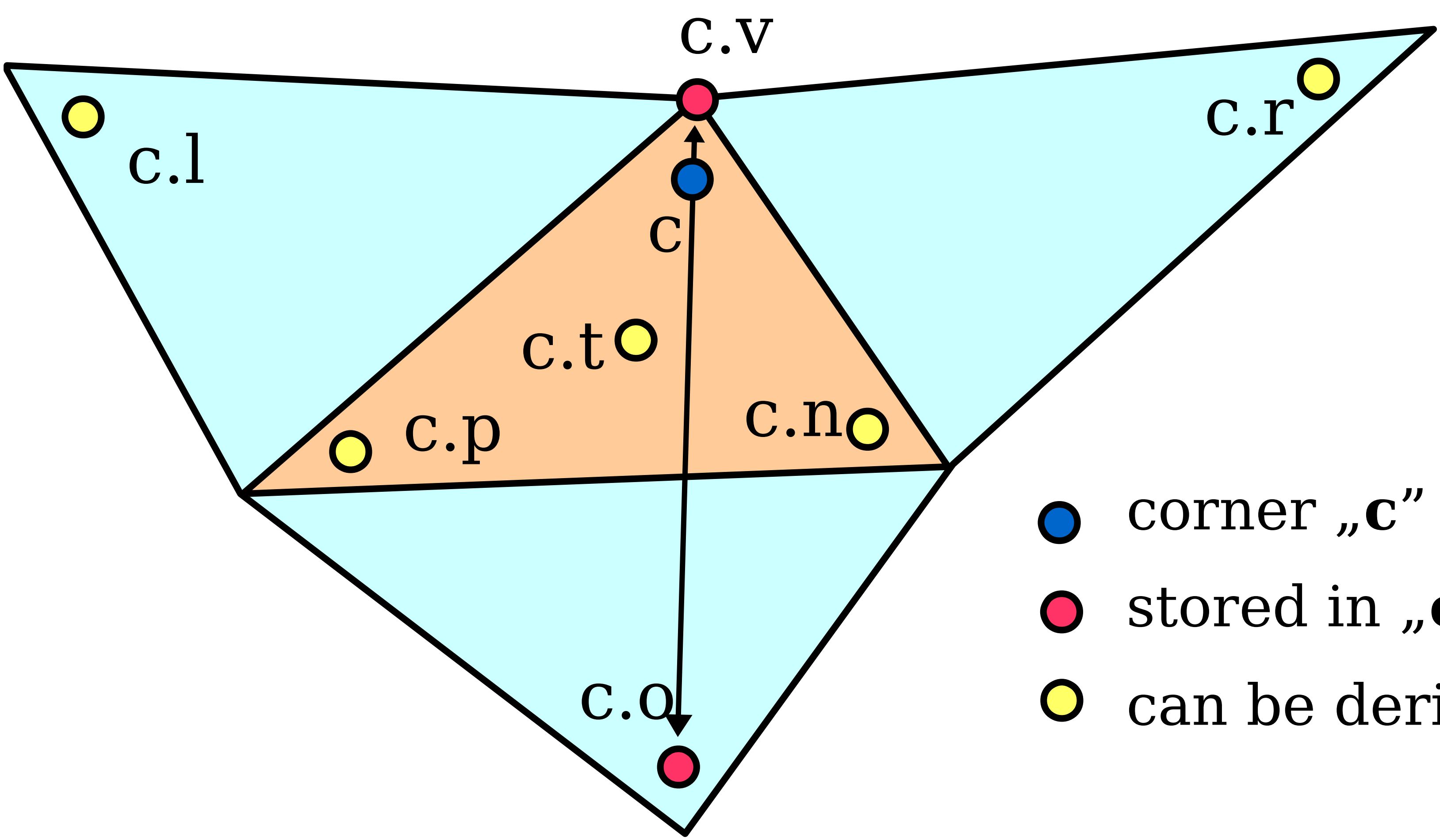
Additional Information

- **Vertices**
  - Colour, texture coordinates
  - Normal vector
- **Edges**
  - Flags for artificial borders
- **Polygons**
  - Colour, material, texture
  - Normal vector
- **Objects**
  - Colour, material, texture

„Corner Table“ (triangles)

- **Vertex table G[v]**
  - Coordinates, normals, texture coordinates, …
- **Corner table V[c]**
  - One inner corner of a triangle
  - **Vertex index** („**c.v**“)
  - Stored in 1D array, CW orientation of polygon
  - **Opposite corner of** („**c.o**“)
  - Implicit data:
    - Number of triangley **t** = c div 3
    - Other corners **c.n** = (c mod 3 == 2) ? c-2 : c+1, **c.p** = c.n.n
    - Other neighbouring triangles **c.l** = c.n.o, **c.r** = c.p.o

„Corner Table“



- corner „**c**”
- stored in „**c**”
- can be derived

Euler's Laws

- The following holds for a **simple polyhedron** (without holes):

$$\mathbf{V - H + S = 2}$$

V - # of vertices, **H** - # of edges, **S** - # of faces

- **Generalised formula** (that allows holes)

$$\mathbf{V - H + S - D = 2 \cdot (T - G)}$$

D – # of holes in faces, **T** - # of solids, **G** - # of holes that traverse the entire object (Genus)

Euler Operators

- **Step by step construction of 2-manifolds**
  - At every step the validity of the Euler formulas is guaranteed (the object is topologically correct)
  - There is an inverse for each operator (easy implementation of „Undo")

- Examples of **Euler operators**:

  Msfevv(P_1,P_2): „make solid, face, edge, vertex, vertex"

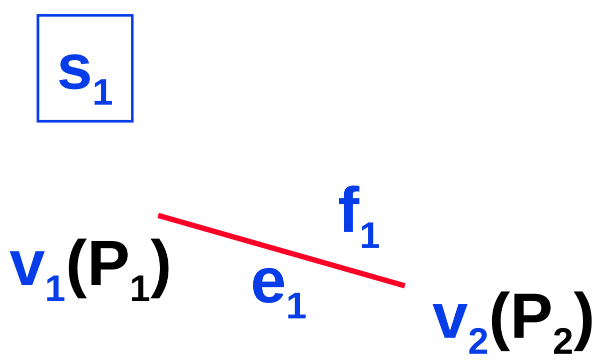
  Mev(f_1,v_1,P_2): „make edge, vertex"

  Mef(f_1,v_1,v_2): „make edge, face"

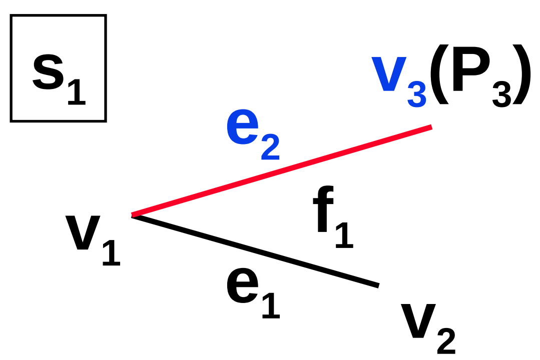
  Kef(e): „kill edge, face"

Tetrahedron Construction

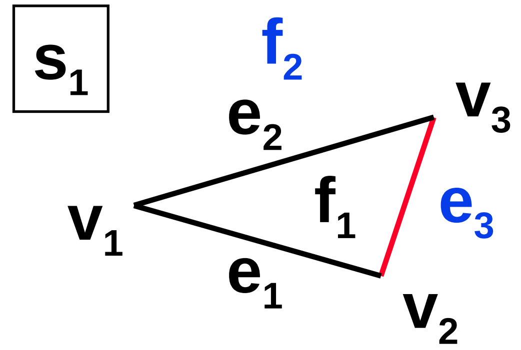
1. Msfevv(P_1,P_2)



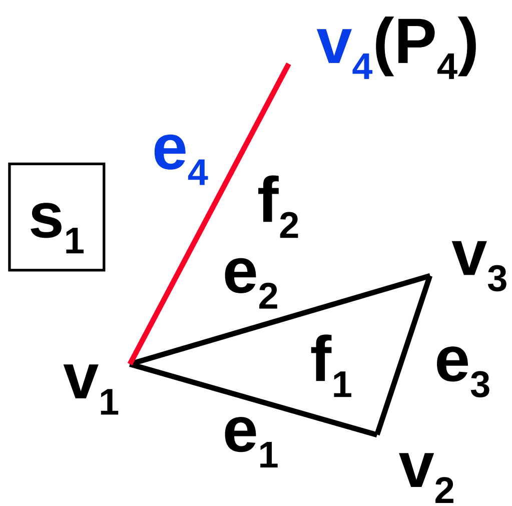
2. Mev(f_1,v_1,P_3)



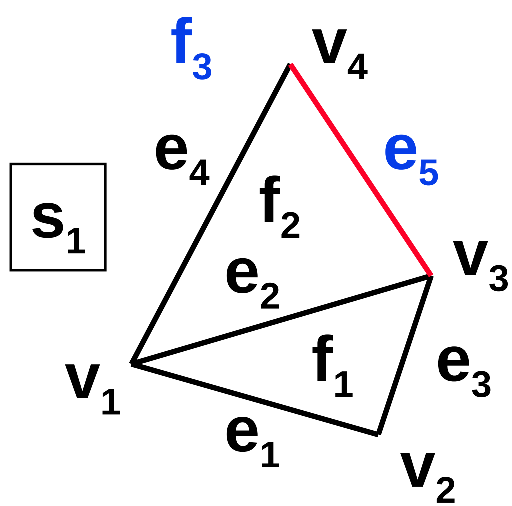
3. Mef(f_1,v_2,v_3)



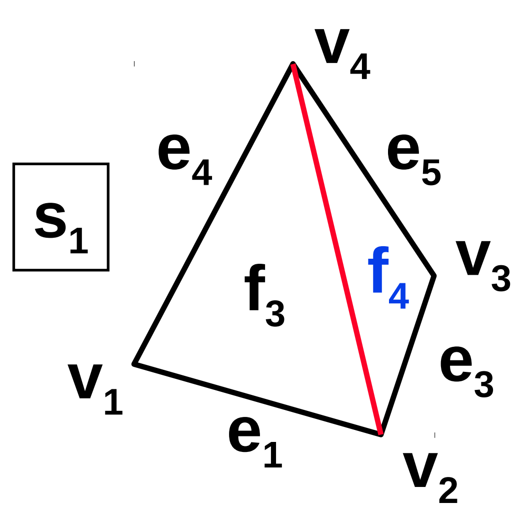
4. Mev(f_2,v_1,P_4)



5. Mef(f_2,v_3,v_4)



6. Mef(f_3,v_2,v_4)



End

Further information:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 534-562, 712-714
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 234-238