Einführung in Visual Computing

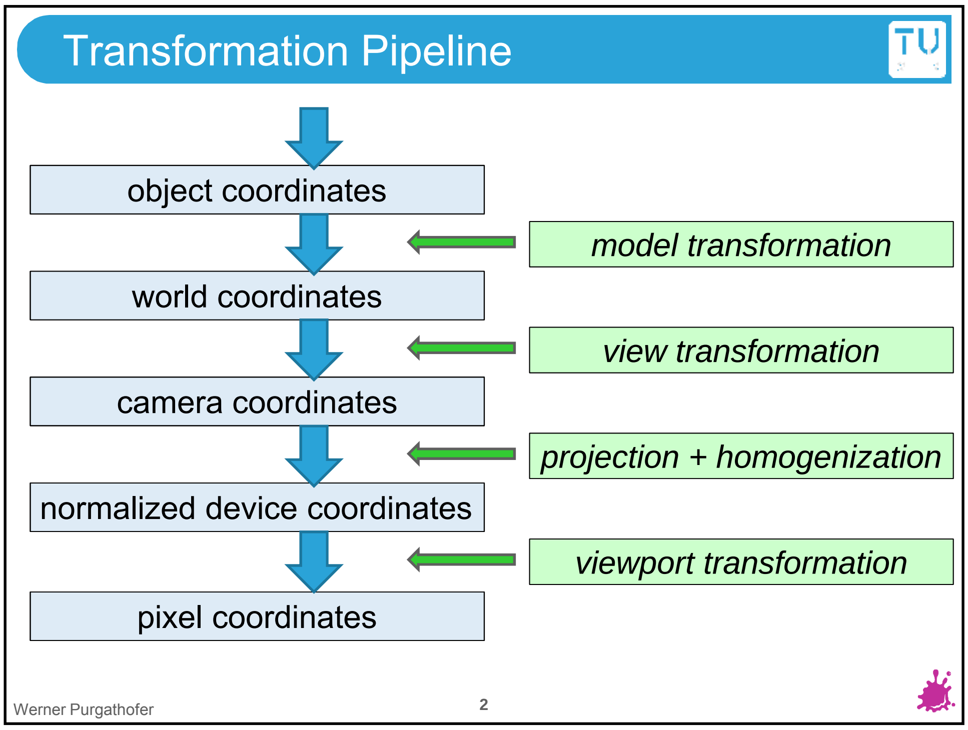
186.822

Graphics Pipeline

Werner Purgathofer

Graphics Pipeline

- information is transformed to an image in successive steps
  - object and scene creation
  - definition of view (camera)
  - projection
  - rasterization
- this is called the ***graphics*** ***pipeline***
  (also ***viewing*** *pipeline,*
  transformation *pipeline,*
  rendering *pipeline*, …)



Transformation Pipeline

object coordinates

model transformation

world coordinates

view transformation

camera coordinates

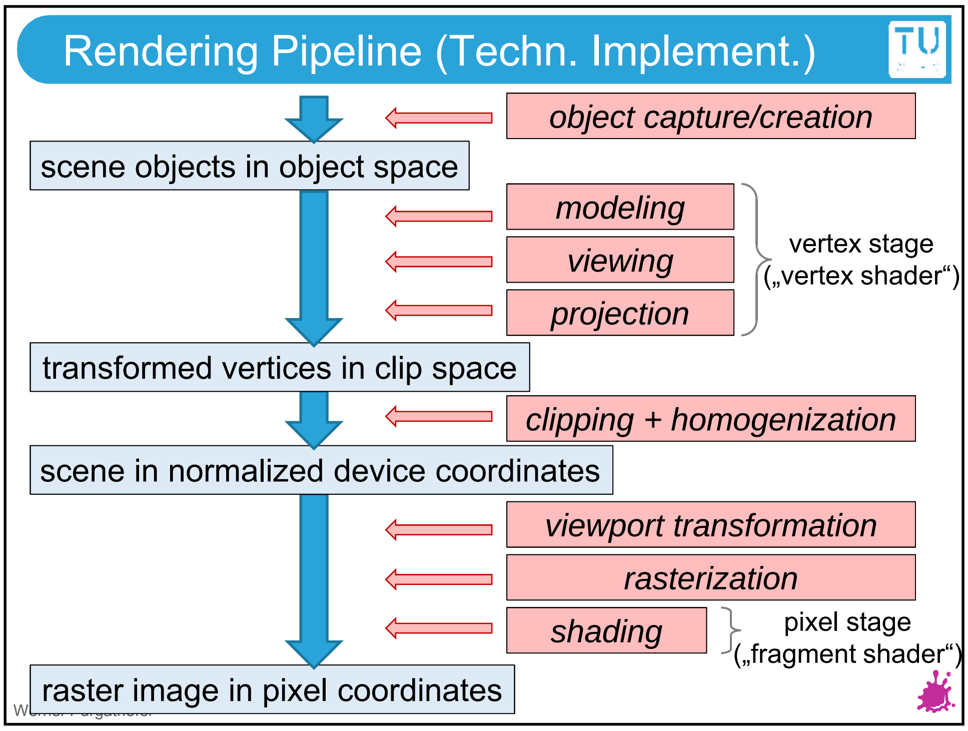
projection + homogenization

normalized device coordinates

viewport transformation

pixel coordinates

Werner Purgathofer 2



Rendering Pipeline (Techn. Implement.)

object capture/creation

scene objects in object space

modeling

viewing

projection

vertex stage („vertex shader")

transformed vertices in clip space

clipping + homogenization

scene in normalized device coordinates

viewport transformation

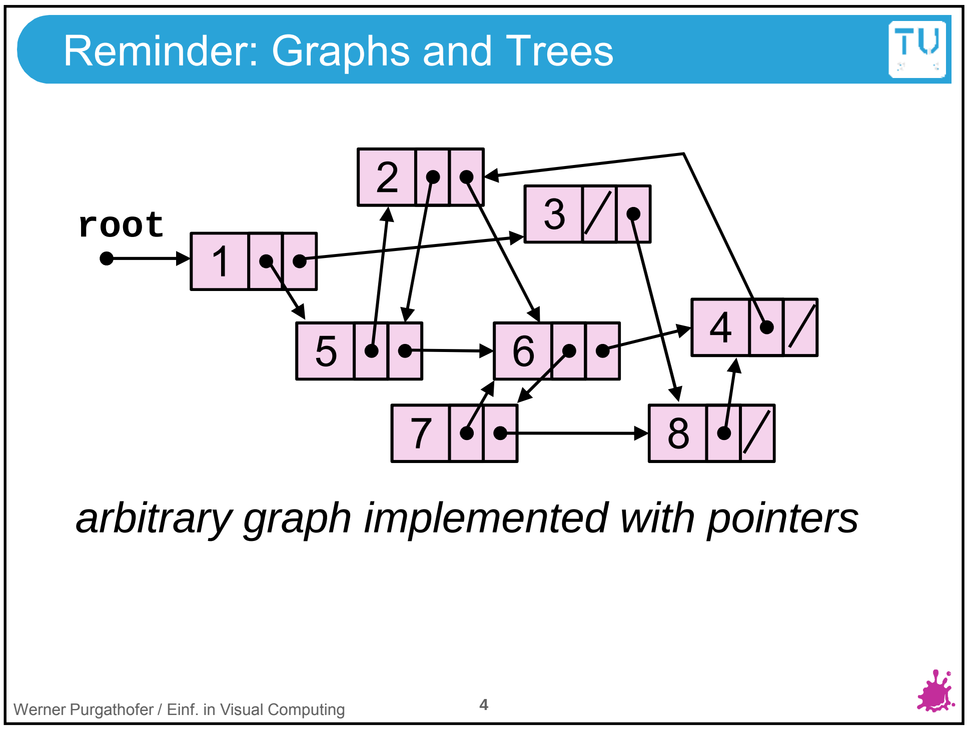
rasterization

shading

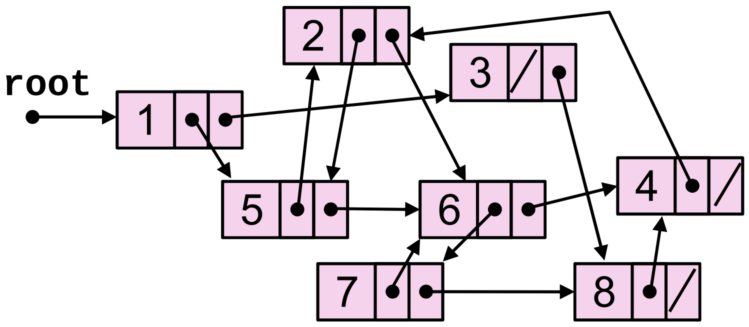
pixel stage („fragment shader")

raster image in pixel coordinates

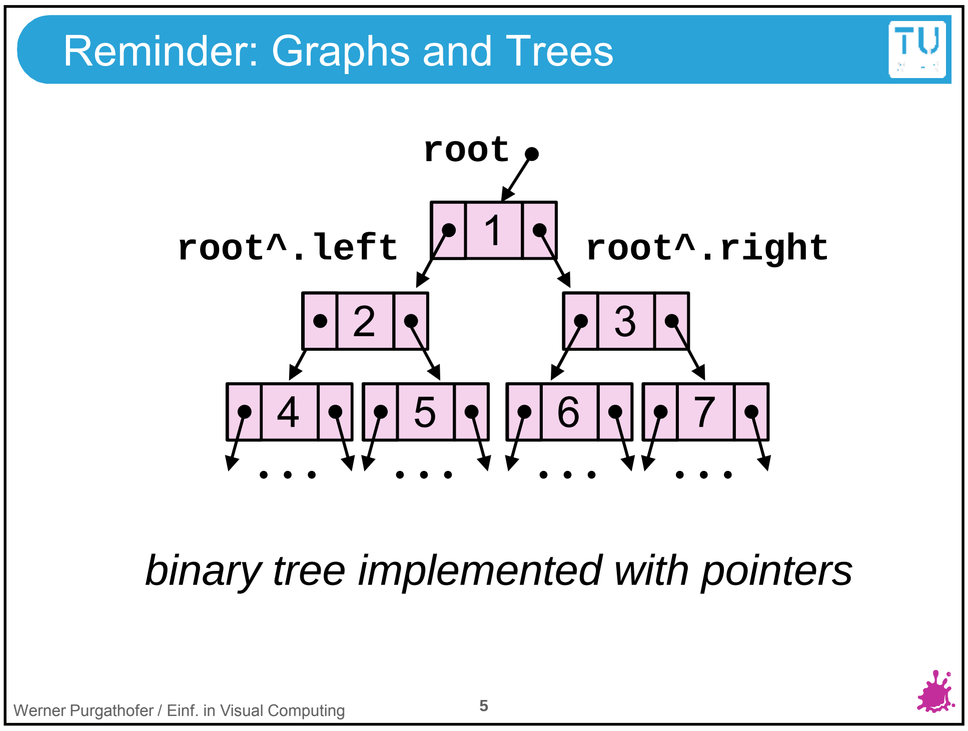
Werner Purgathofer



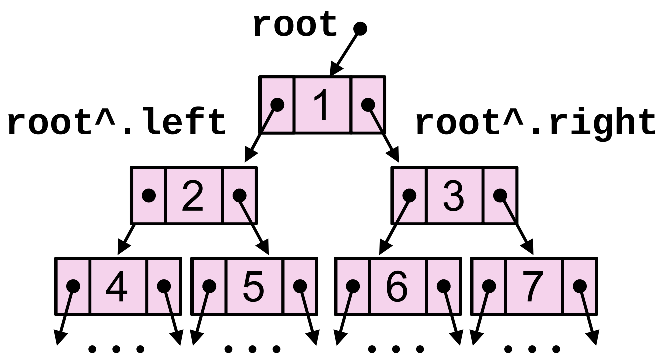
Reminder: Graphs and Trees



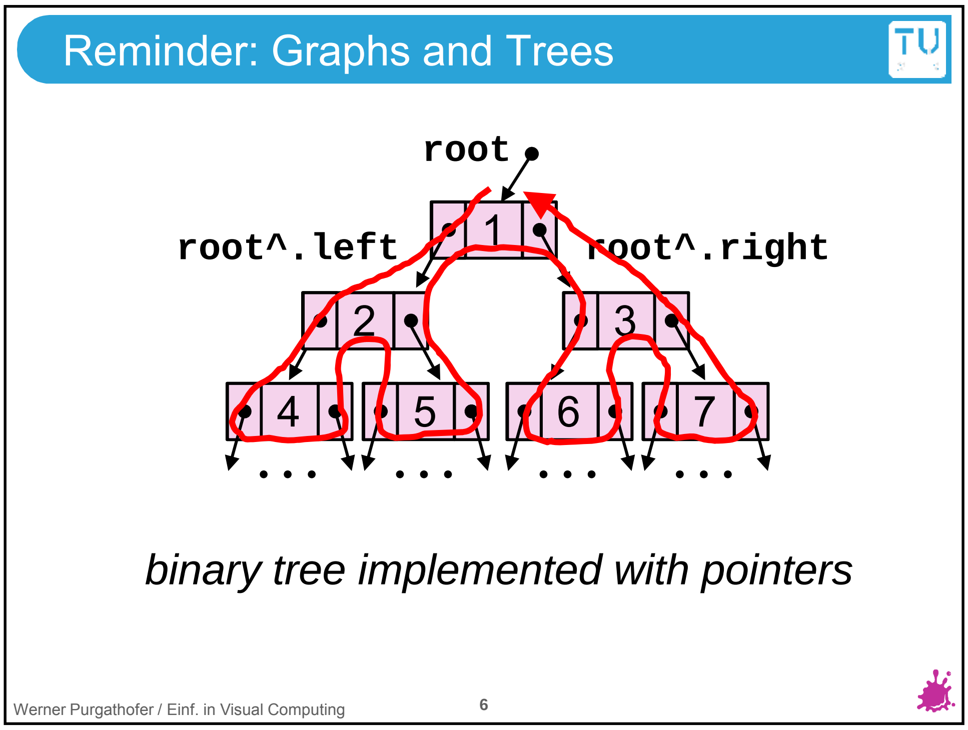
arbitrary graph implemented with pointers



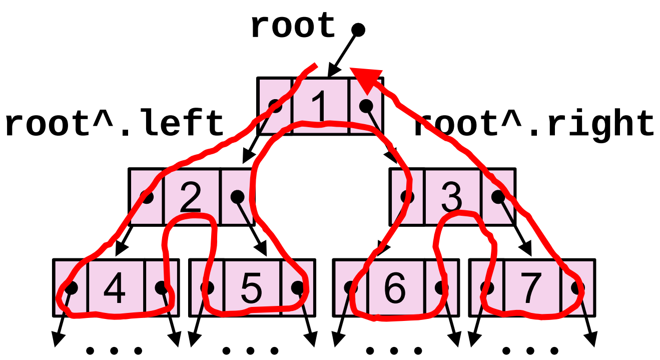
Reminder: Graphs and Trees



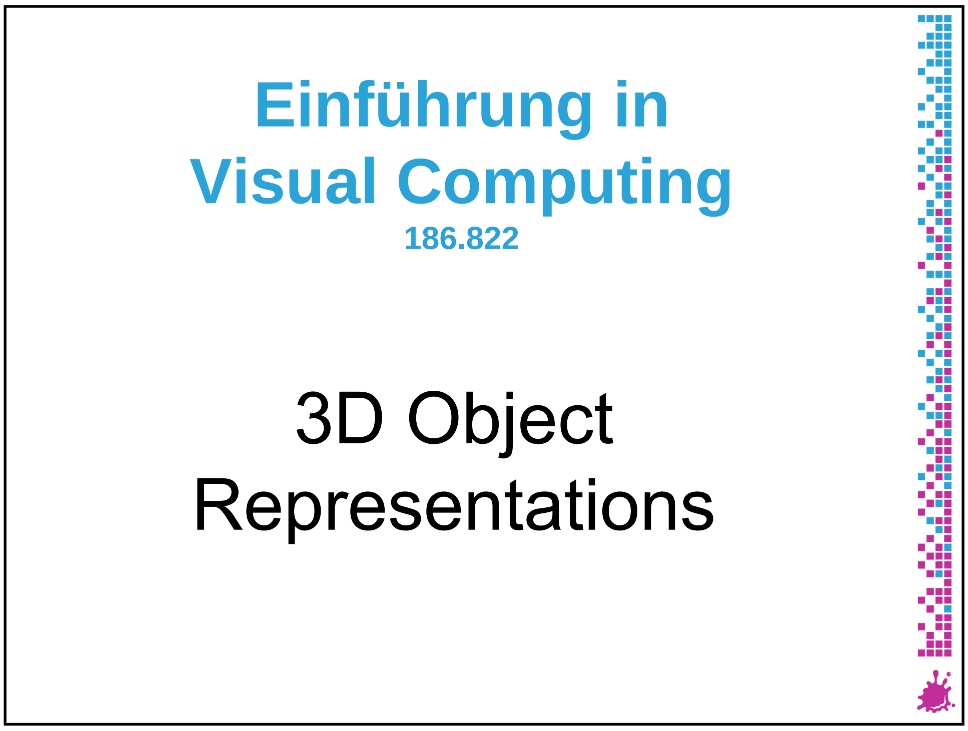
binary tree implemented with pointers



Reminder: Graphs and Trees



binary tree implemented with pointers

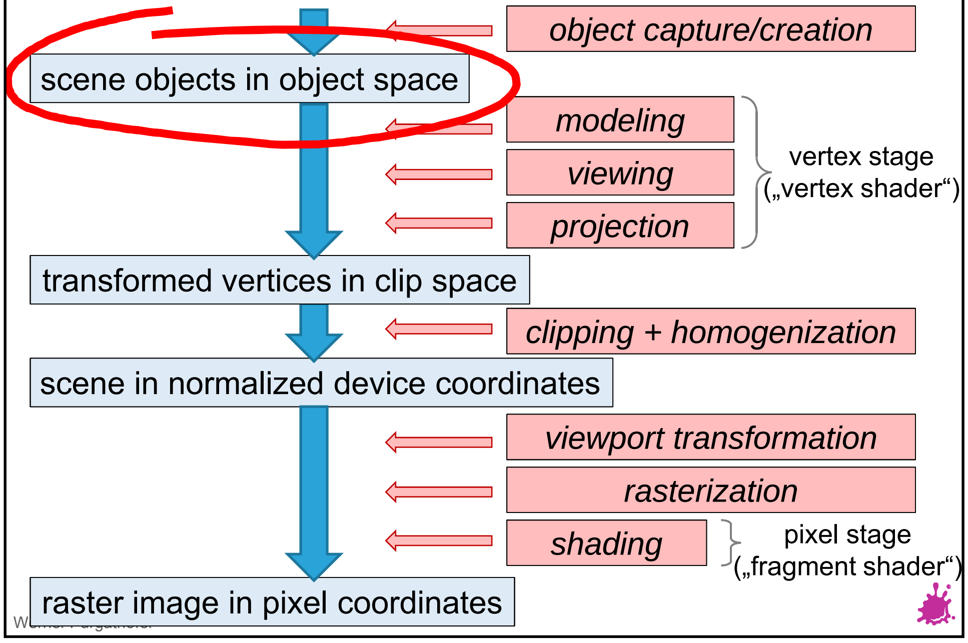


Einführung in Visual Computing

186.822

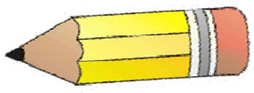
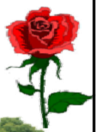
3D Object Representations

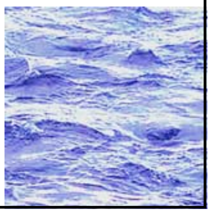
Object Repres. in the Rendering Pipeline


object capture/creation
scene objects in object space
modeling
viewing
projection
vertex stage („vertex shader“)
transformed vertices in clip space
clipping + homogenization
scene in normalized device coordinates
viewport transformation
rasterization
shading
pixel stage („fragment shader“)
raster image in pixel coordinates

3D Object Representations

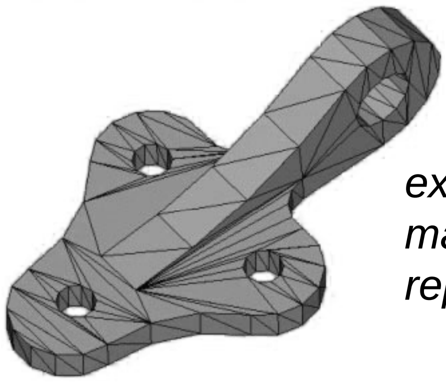
- graphics scenes contain
  - solid geometric objects
  - trees, flowers, clouds, rocks, water,…
- creation of models
  - surface ↔ interior models
  - explicit ↔ procedural models
  - heuristically ↔ physically based models
- representations
  - geometrical data structures
  - data structure organization



Polygon Surfaces (1)

- set of surface polygons enclose object interior
  = ***Boundary Representation***
  ("B-Rep")



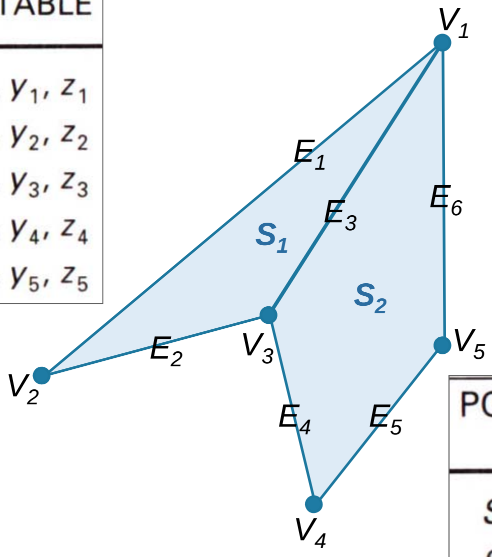
example:
machine part surface
represented by triangles

Polygon Surfaces (2)

- polygon tables (B-Rep lists)
  - geometric and attribute tables
  - vertex, edge, polygon tables
  - consistency, completeness checks

Polygon Surfaces: Data Structure

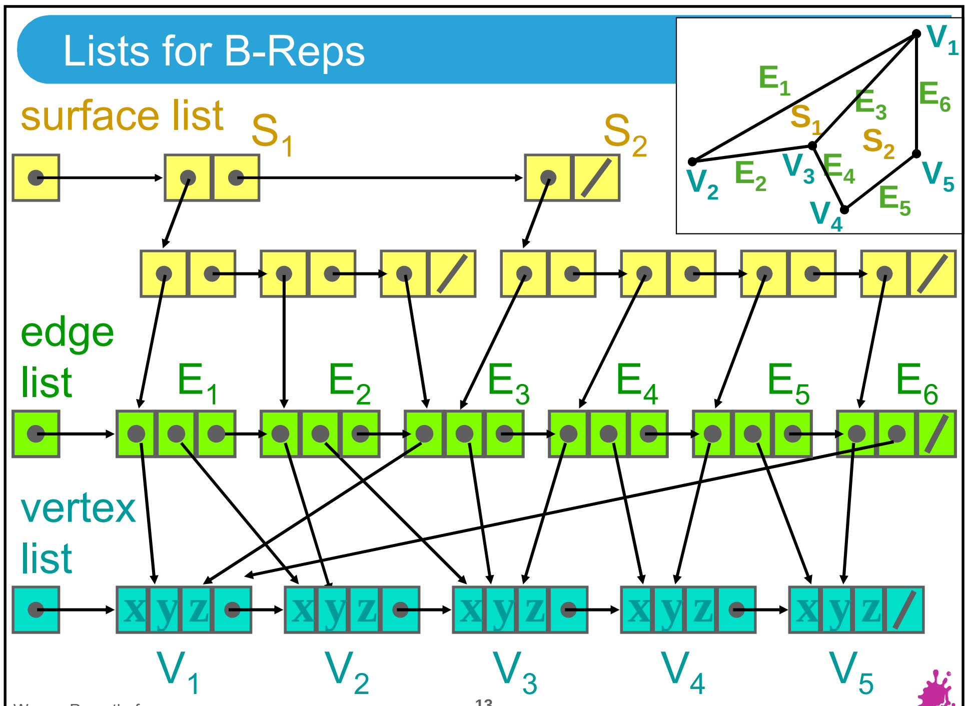
VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

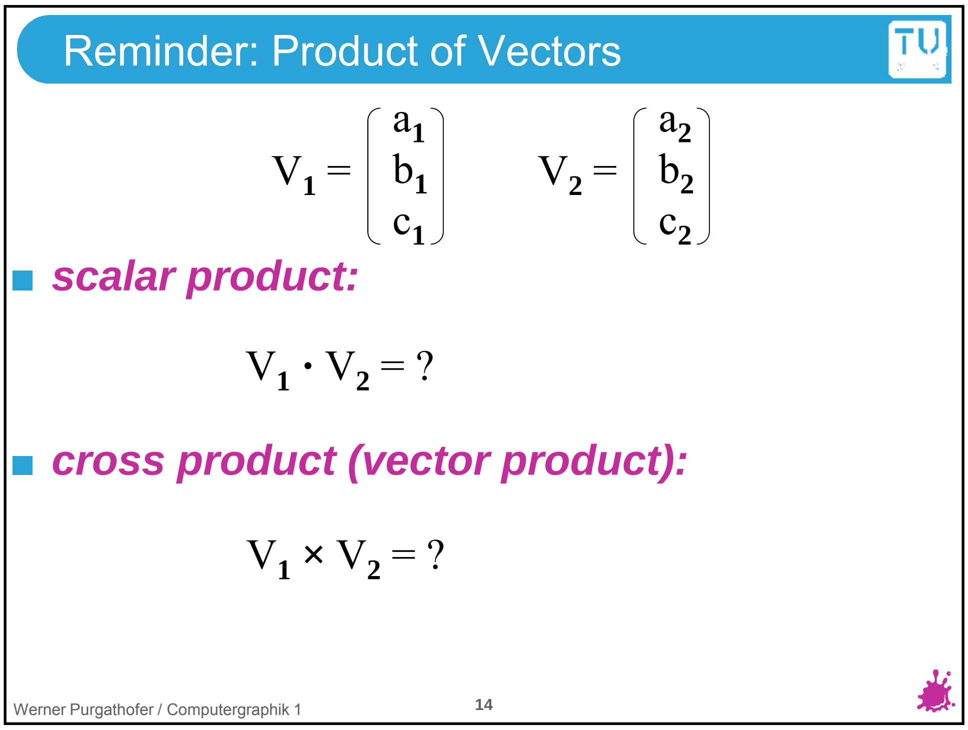


EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Lists for B-Reps





Reminder: Product of Vectors

$$V_1 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \qquad V_2 = \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix}$$

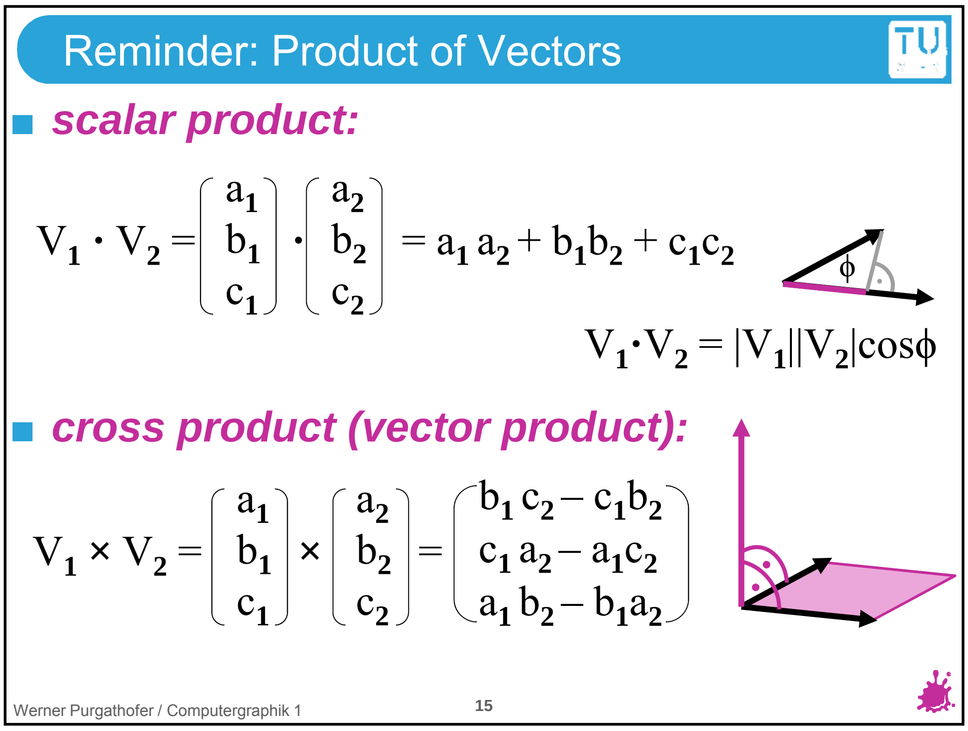
- ***scalar product:***

$$V_1 \cdot V_2 = ?$$

- ***cross product (vector product):***

$$V_1 \times V_2 = ?$$

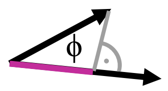
Werner Purgathofer / Computergraphik 1 14



Reminder: Product of Vectors

- ***scalar product:***

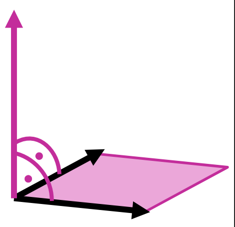
$$V_1 \cdot V_2 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \cdot \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = a_1 a_2 + b_1 b_2 + c_1 c_2$$



$$V_1 \cdot V_2 = |V_1||V_2|\cos\phi$$

- ***cross product (vector product):***

$$V_1 \times V_2 = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} \times \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} = \begin{pmatrix} b_1 c_2 - c_1 b_2 \\ c_1 a_2 - a_1 c_2 \\ a_1 b_2 - b_1 a_2 \end{pmatrix}$$



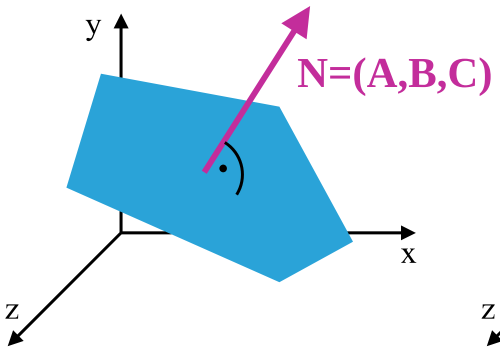
Werner Purgathofer / Computergraphik 1 15

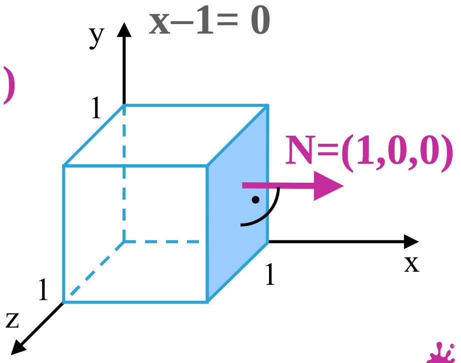
Polygon Surfaces: Plane Equation

$$\boxed{Ax + By + Cz + D = 0}$$

- plane parameters **A,B,C,D**
- normal (**A,B,C**)

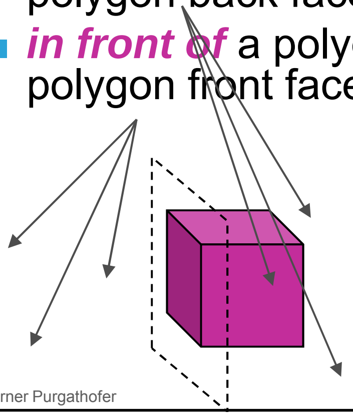
example:
x–1= 0





Front and Back Polygon Faces

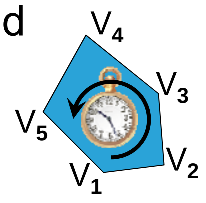
- ***back face*** = polygon side that faces into the object interior
- ***front face*** = polygon side that faces outward
- ***behind*** a polygon plane = visible to the polygon back face
- ***in front of*** a polygon plane = visible to the polygon front face

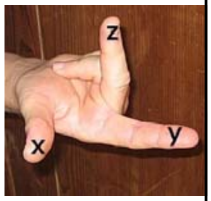


Front and Back Polygon Faces

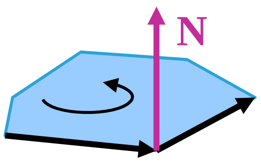
$Ax + By + Cz + D = 0$ for points on the surface
< 0 for points behind
> 0 for points in front

if (1) right-handed coordinate system
(2) polygon points ordered counterclockwise



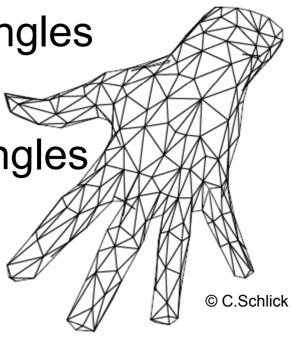


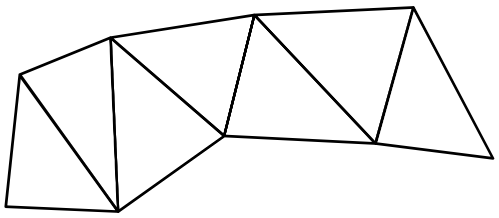
V_1, V_2, V_3 counterclockwise $\Rightarrow$
normal vector
$\mathbf{N} = (\mathbf{V}_2 - \mathbf{V}_1) \times (\mathbf{V}_3 - \mathbf{V}_1)$



Triangle Meshes

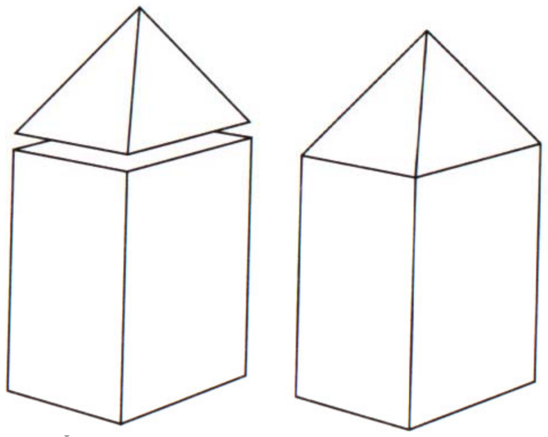
- most polygons are triangles
- ***triangle mesh*** = connected triangles
- ***triangle-strip*** = successive triangles (1 additional point per triangle)





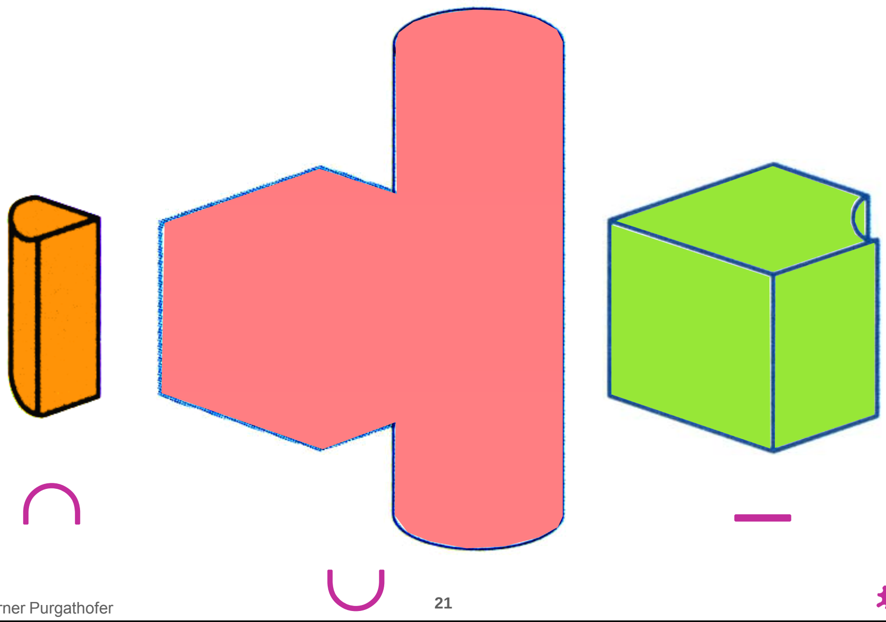
Constructive Solid Geometry

- **C**onstructive **S**olid **G**eometry (CSG)
  - boolean set operations on 3D objects
  - union, intersection, difference operation



combining 2 objects with a union operation, producing a single composite object

CSG: Different Set Operations

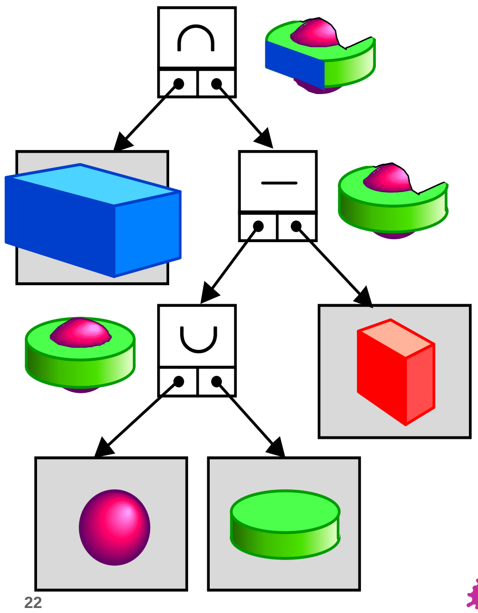


CSG Data Structure

Every object is assembled from simple solids with **set operations**

data structure: **binary tree**

recursive evaluation



Operations with CSG Trees

- **transformations**
  - multiplication of all transformation matrices with the matrix of this transformation
- **combinations**
  - generate a new node with the desired operator and link the operands as subtrees to it

A op B:

$$\begin{tikzcd} & \boxed{\text{op}} \arrow[dl] \arrow[dr] & \\ A & & B \end{tikzcd}$$

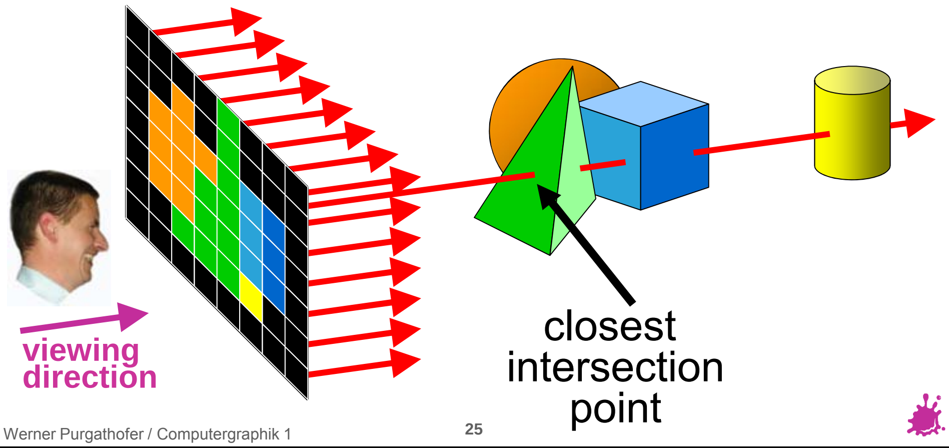
Rendering of CSG Trees

- transform into B-Rep and use normal hidden surface algorithm

  or
- render directly with ray casting (or with ray tracing)

Ray-Casting Method (1)

- line-of-sight of each pixel is intersected with all surfaces
- take closest intersected surface

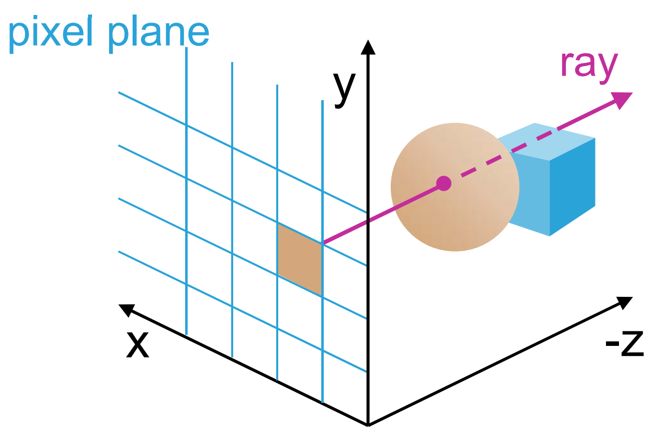


Ray-Casting Method (2)

- based on geometric optics, tracing paths of light rays
- backward tracing of light rays
- suitable for complex, curved surfaces
- special case of ray-tracing algorithms
- efficient ray-surface intersection techniques necessary
  - intersection point
  - normal vector

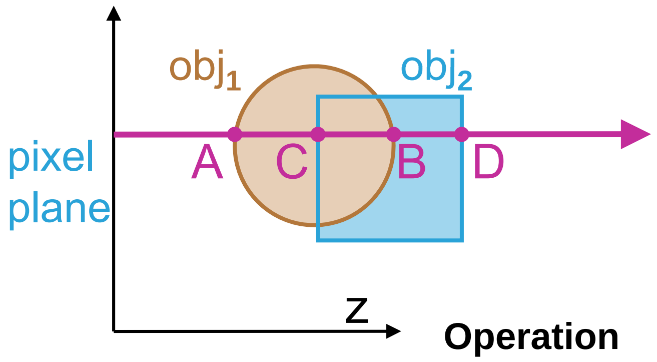
Ray-Casting Methods for CSG (1)

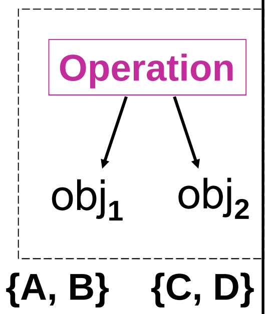
- visibility processing



Ray-Casting Methods for CSG (2)

- determining surface limits





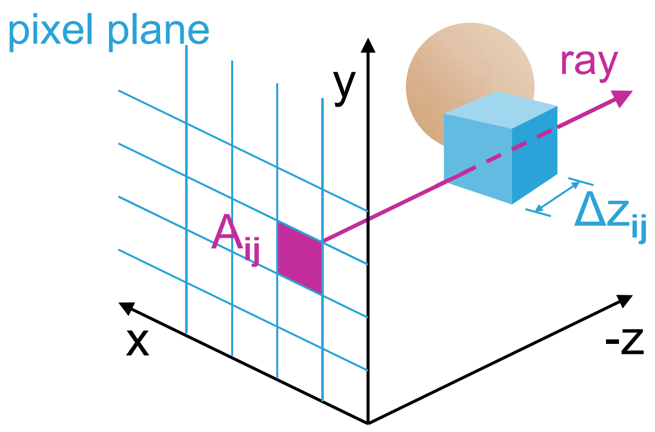
Operation	Result
Union	{A, D}
Intersection	{C, B}
Difference	{A, C}

Werner Purgathofer 28

Ray-Casting Methods for CSG (3)

- volume determination

$V_{ij} \approx A_{ij} \cdot \Delta z_{ij}$ $\qquad$ $V \approx \Sigma V_{ij}$



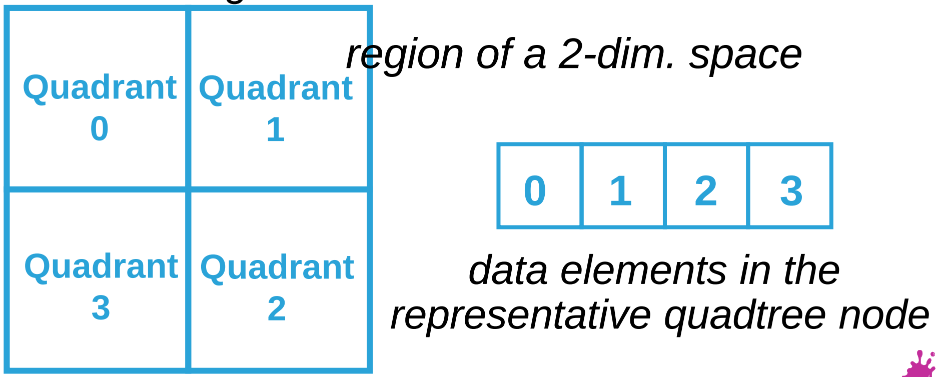
Werner Purgathofer 29

Properties of CSG

- advantages
  - exact representation
  - low memory cost
  - combinations and transformations trivial
- disadvantages
  - rendering effort is high

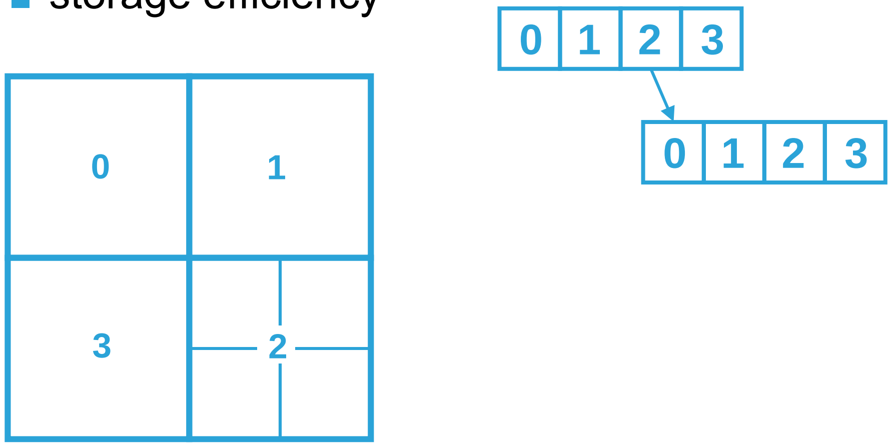
Quadtrees

- hierarchical enumeration of objects
- in 2D: quadtree
  - hierarchical subdivision until a region is homogeneous



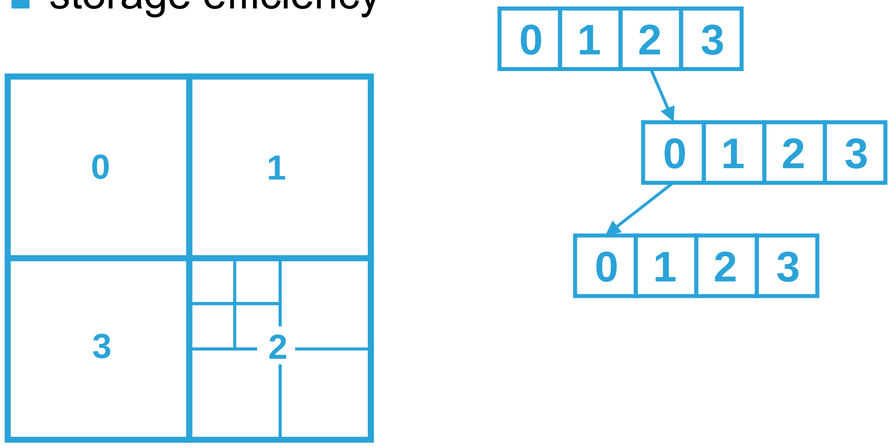
Quadtrees

- area with 2^n by 2^n pixels $\Rightarrow$ quadtree with n levels
- storage efficiency



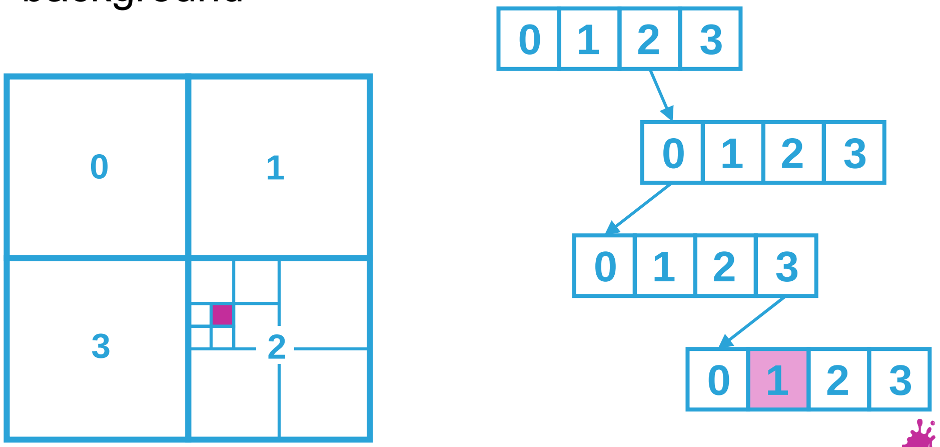
Quadtrees

- area with 2^n by 2^n pixels $\Rightarrow$ quadtree with n levels
- storage efficiency

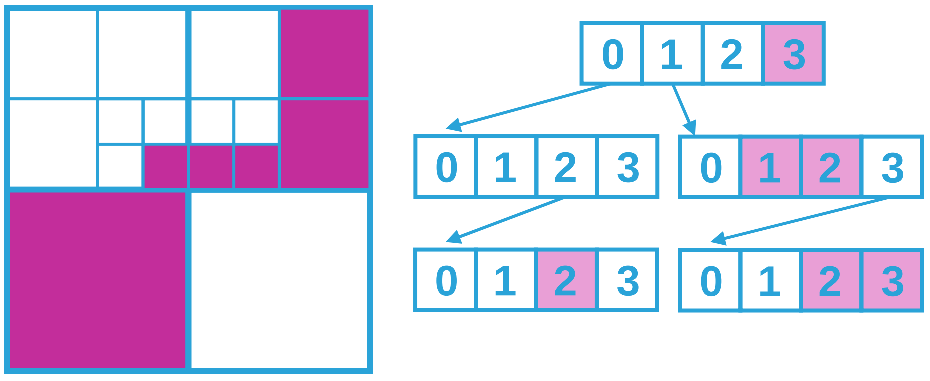


Quadtrees

quadtree representation for a region containing one foreground-color pixel on a solid background



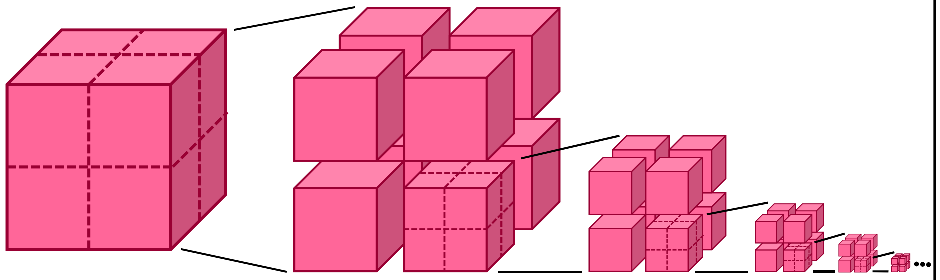
Quadtree Example



suitable for representing (2D) images

Octree

= extension to 3D

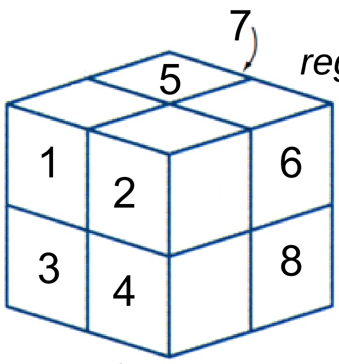


regular space subdivision:
- simple (empty or uniform) ⇒ leaf node
- complex (other cases) ⇒ divide further

Werner Purgathofer 36

Octrees

- octree divides 3D cube into octants
- volume elements (voxels)
- set operations easy on octrees
- geometric transformations difficult on octrees



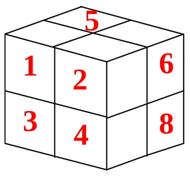
region of a 3-dim. space

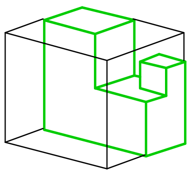
1	2	3	4	5	6	7	8

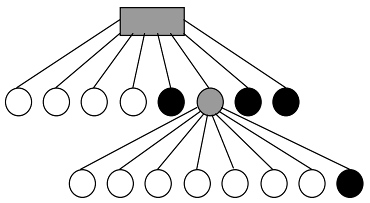
data elements in the representative octree node

Werner Purgathofer 37

Octree Simple Example







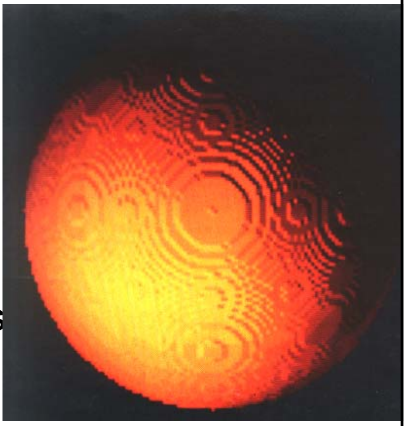
G (W W W W S G (W W W W W W W S) S S)

Operations with Octrees

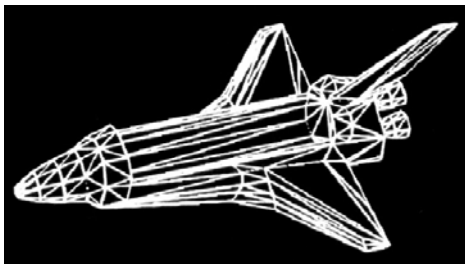
- **transformations**
  - very complicated except for a few special cases,
    e.g. rotation by 90°, mirroring at a subdivision plane, scalation by 2^n
- **combinations**
  - very simple:
    if A or B homogeneous ⇒ simple rules
    else combine recursively all 8 octants of A and B

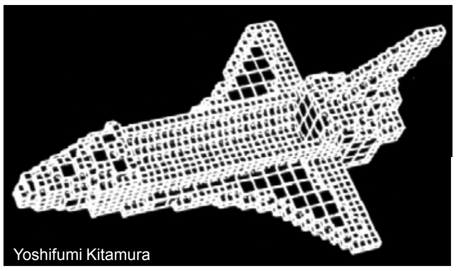
Properties of Octrees

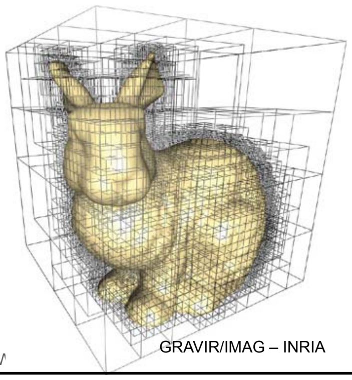
- **advantages**
  - combinations very simple
  - fast rendering
  - spatial search possible
- **disadvantages**
  - inexact representation
  - low image quality →
  - restricted transformations
  - high memory cost

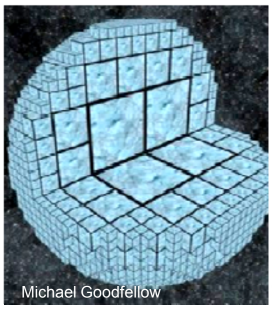


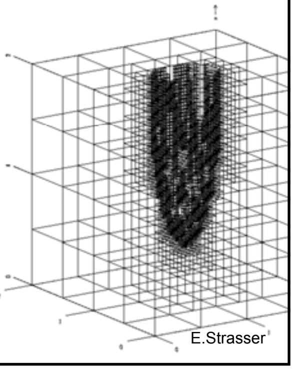
Octree Examples




Yoshifumi Kitamura


GRAVIR/IMAG – INRIA


Michael Goodfellow


E.Strasser

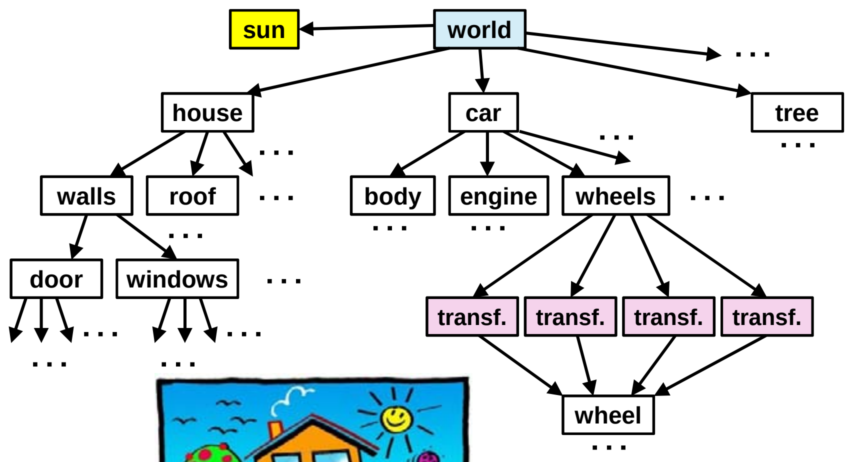
Other 3D Object Representations

- BSP trees
- fractal geometry methods
- shape grammars, procedural models
- particle systems
- physically based modeling
- visualization of data sets
- ...

Scene Graphs

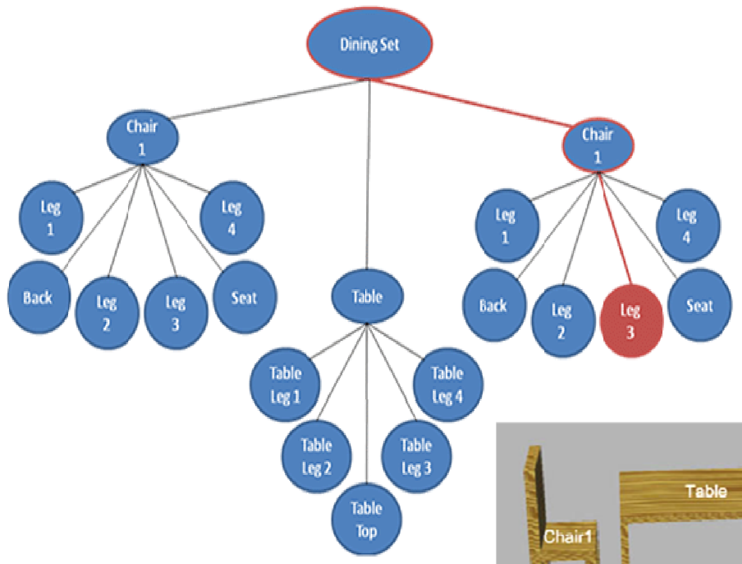
- object-oriented data structure
  - directed acyclic graph
- describes logical and/or spatial relationship of scene objects
- describes groups of (groups of …) objects
- no exact definition
- used in most graphics systems, e.g.
  - OpenSceneGraph
  - VRML
  - X3D …

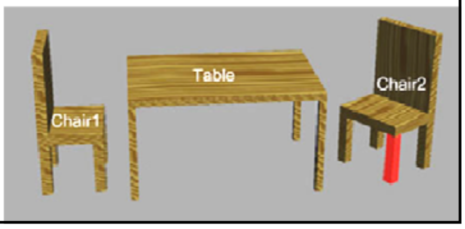
Scene Graph Example



Werner Purgathofer

Scene Graph Example

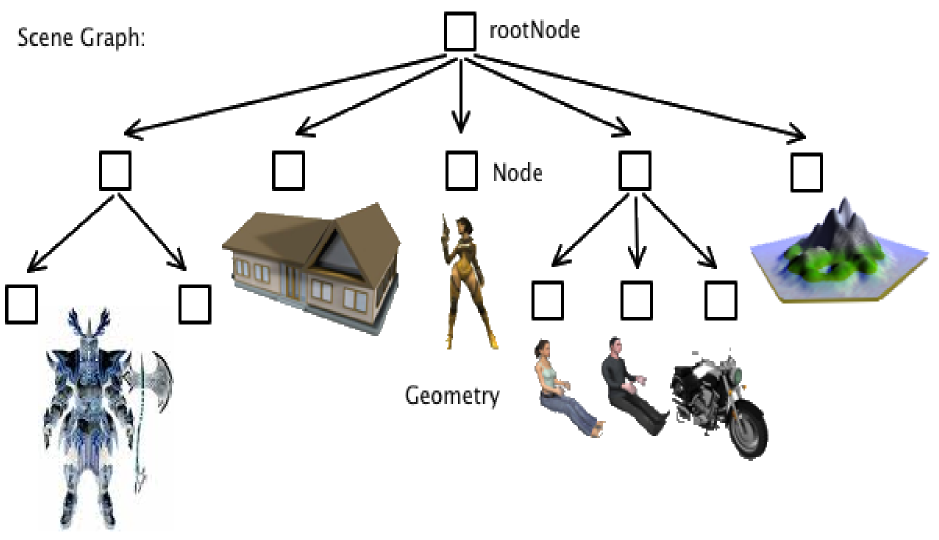




© hypercosm

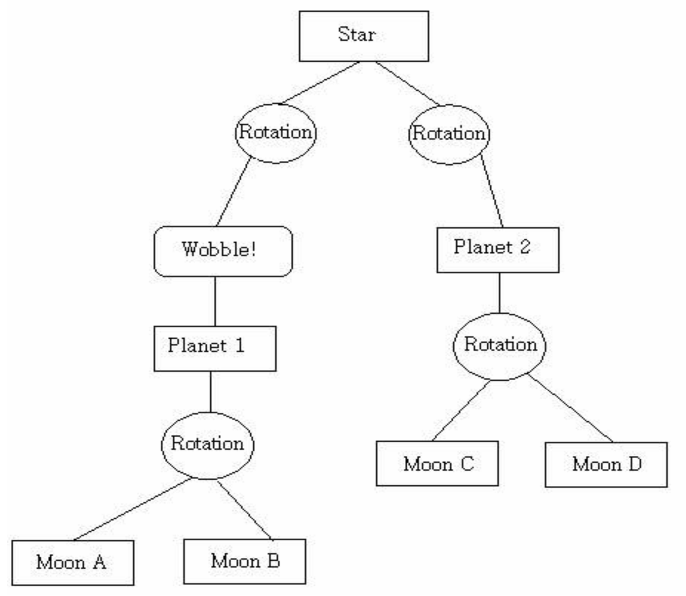
Werner Purgathofer

Scene Graph Example



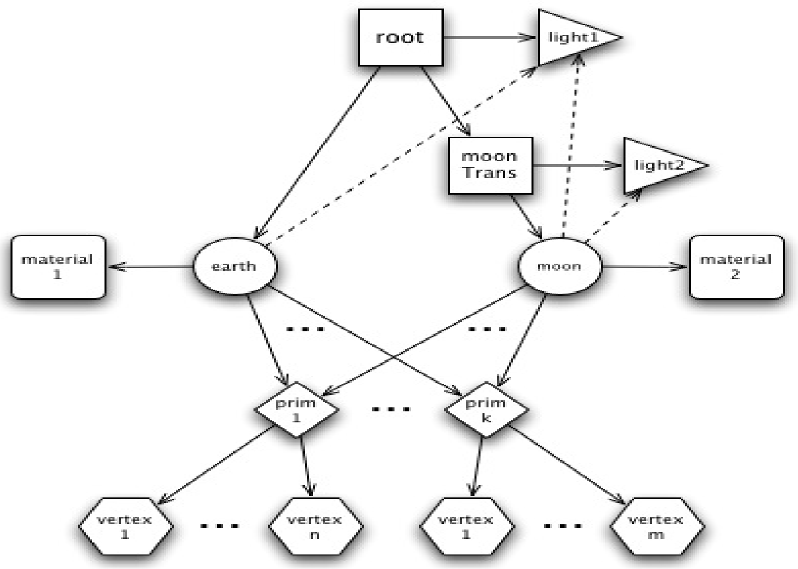
© jmonkeyengine.org

Scene Graph Example



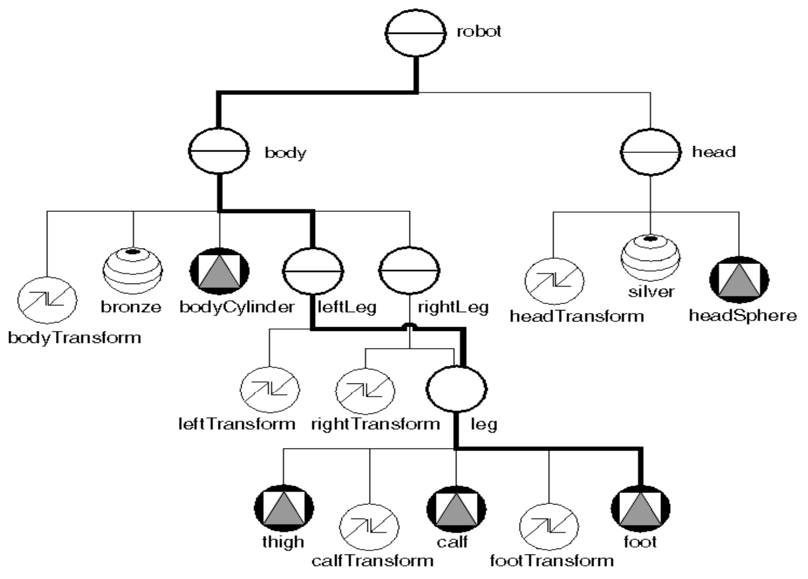
© Garret Foster

Scene Graph Example



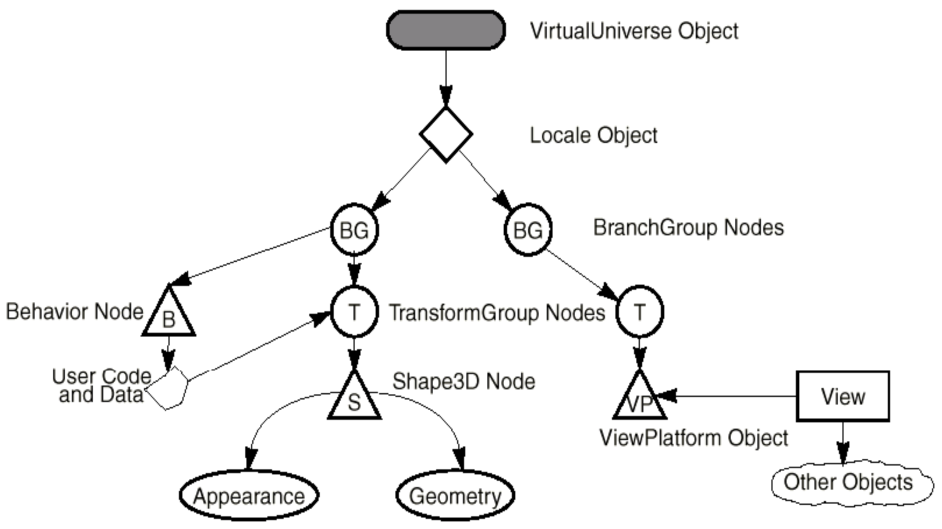
© B.B.Christensen

Scene Graph Example



© Francois Faure

Scene Graph Example



© S.Teichmann

End of 3D Object Representations