§6 Representation and Modeling of Objects

6.1 Overview

- **Goal of computer graphics:** Generation of two-/three-dimensional images/animations of a scene or an object.
- **Question:** How are scenes or objects generated, represented, or modelled in the computer?
- Factors for the choice of a representation
  1. Object exists in reality or only virtually in a computer representation.
  2. Generation (**modeling**) of the objects is closely related to their visualization:
     - ➡ interactive CAD-systems,
     - ➡ modeling and visualization as tools in the production process,
     - ➡ more than just 2d-output required.

6.1 Overview

3. **Accuracy** of the internal computer representation depends on the application:
   - An exact description of geometry and shape for CAD-applications is mandatory,
   - An approximating description of geometry and shape for a renderer is sufficient.
4. For interactive applications the objects exist in multiple internal representations at the same time or are generated dynamically as and when required:
   - dynamic triangulation of objects,
   - Level-Of-Detail-methods,
   - hybrid Models.

6.1 Overview

Important aspects for the modeling and representation of objects:

- **Generation** of 3d geometry data
  - CAD-interface,
  - digitizer, laser-scanner (reverse engineering),
  - image(2d)- and video(3d)-analysis.
- **Representation**, efficient access and conversion.
  - Polygonal meshes (e.g. triangulation) as representation for rendering.
  - Other representations:
    - Finite EleMents,
    - implicit (iso-surfaces),
    - Constructive Solid Geometry,
    - Boundary-Representation in CAD,
    - Surface-Elements = Point + Normal (splats)
    - **Parametric representation.**

6.1 Overview

- **Manipulation**, i.e. change of the shape of objects (editing), e.g.
  - Boolean operations („set operations“),
  - local smoothing,
  - interpolation of certain features (boundary curves),
  - „engraving“ of geometric details,
  - simulation of mechanical deformations, etc.

6.2 Polygonal representation

6.2.1 Boundary Representation

- General: Representation of a 3d object by its bounding surfaces.
- Here: Only the most simple form of a BRep

Polygonal representation

  - Description of an object by a set or mesh of planar polygons/facets (usually triangles).

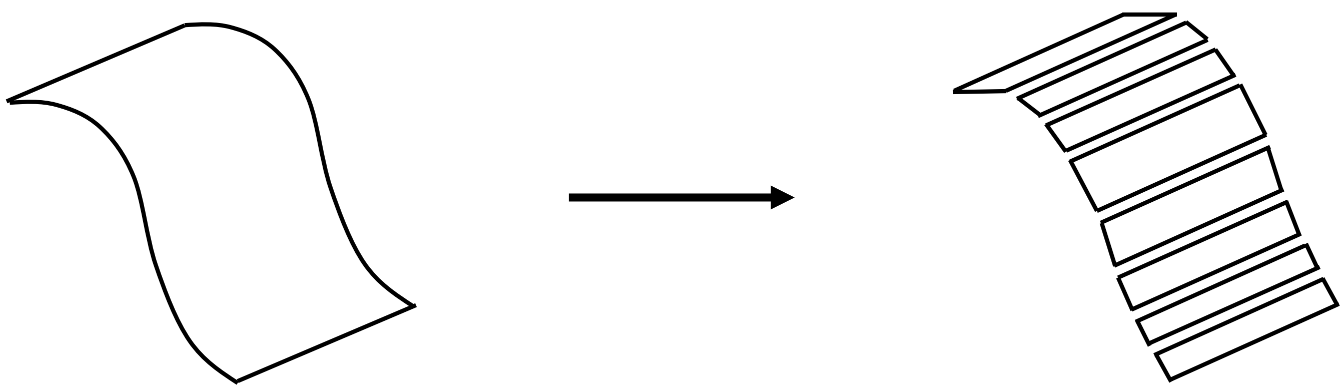
- **Polynomial representations** (Non-Uniform Rational B-Splines) are the main topic of the lecture „Geometric Modeling (MSI)“.

6.2 Polygonal representation

6.2.2 Polygonal representation

- Classical representation of three-dimensional objects in computer graphics.
- object is represented by mesh of polygonal facets (often triangles).

  ➡ piecewise linear approximation.
- The polygonal facets are only an approximation of the curved surfaces, that bound the respective object.



6.2 Polygonal representation

- Accuracy of the approximation (number and size of the polygons) has to be pre-determined, but causes often massive problems e.g.:
  - Which polygon resolution is required for a sufficiently exact representation?
  - Which polygon resolution is required for the renderer, to yield a smooth visual impression from a piecewise linear approximation?
  - What is the relation between number of polygons of an object and its size in the final representation?
  - ➡ Couple polygon resolution to the local curvature of the surface
    - ➡ Lecture „Geometric Modeling (MSI)“.

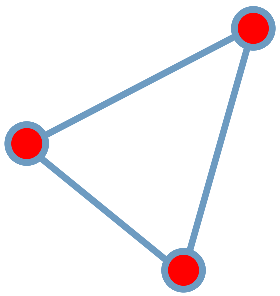
6.2 Polygonal representation

Topology:

Set of properties of an object, that do **not** change under rigid body transformations.

➡ **The structure of the model.**

In the example: The polygon has three vertices, which are adjacent via edges.



Geometry:

The „instantiation“ of the topology by specification of its spatial position.

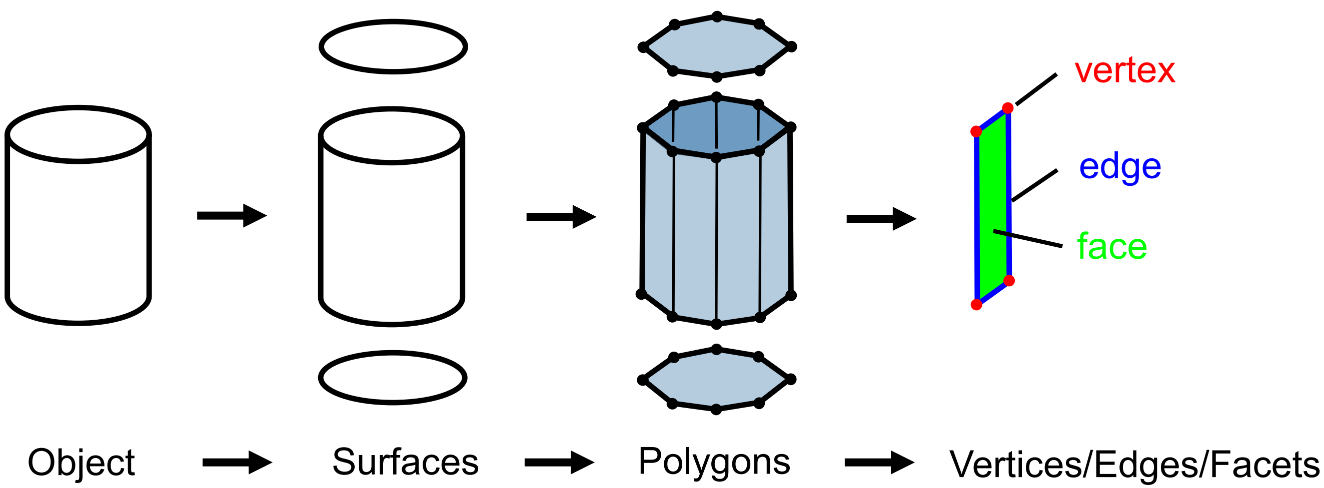
➡ **The shape of the model.**

In the example: The coordinates of the corners.

6.2 Polygonal representation

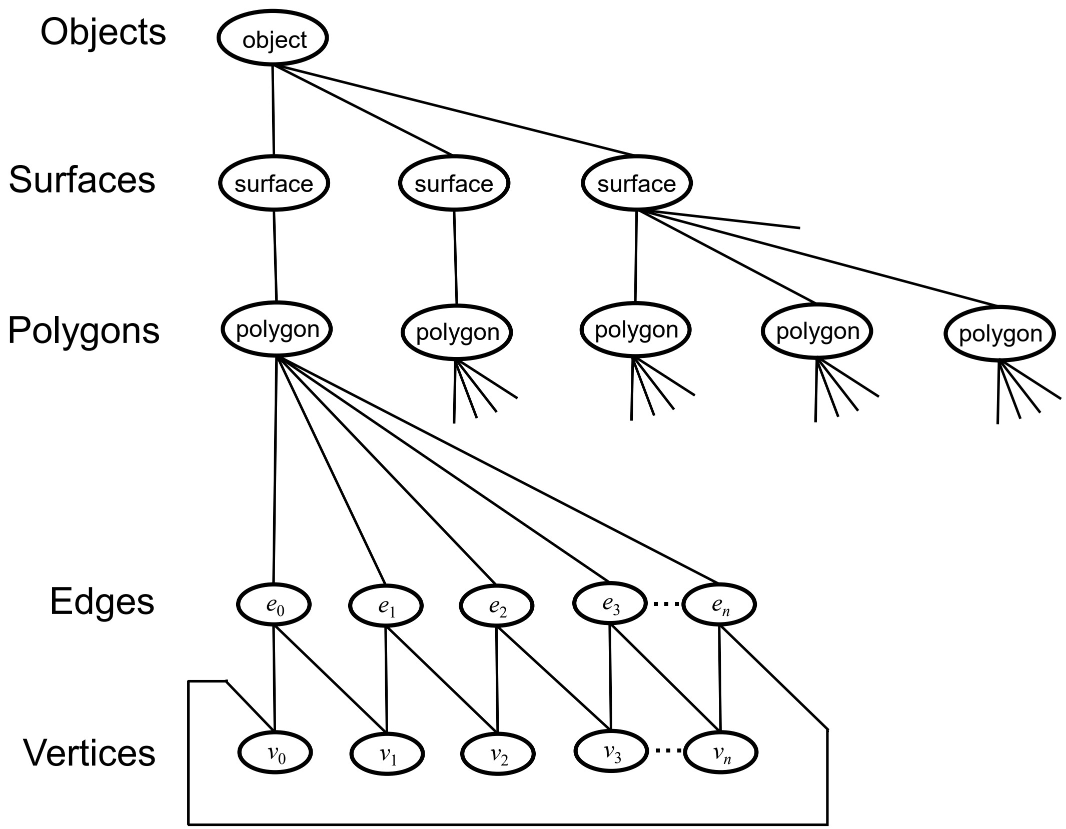
6.2.3 **Representation hierarchy** (conceptually)

- Object consists of surface.
- Surfaces consist of polygons (faces, facets).
- Polygon consists of corners (vertices) and edges.

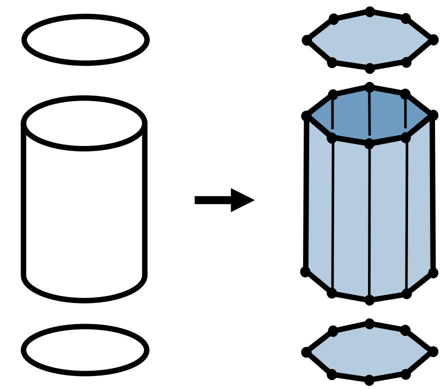


6.2 Polygonal representation

Representation hierarchy (topologically)



6.2 Polygonal representation



Edges

There are two different kinds of edges for an approximating polygonal representation:

- Sharp feature edges (feature lines)
  - ➡ Should be visible as edges in the rendered image.
- Virtual edges (in the interior of surfaces)
  - ➡ Should vanish in the rendering process.
  - ➡ 70s: interpolative shading algorithms)
  - ➡ Flat, Gouraud, Phong shading (see § 5)
- The edge type has to be defined in the data structure, e.g. multiple storage (per facet) of feature vertices and edges.

6.2 Polygonal representation

6.2.4 Representation hierarchy (data structure)

What type of manipulations of the polygonal mesh are allowed?

- Removal of measurement or sampling errors using smoothing filters?
- Refinement of meshes for better accuracy and more details in the object.
- Coarsening of meshes for data compression of level-of-detail-representation.

6.2 Polygonal representation

Questions for the choice of the data structure?

- How is the topological information separated from the geometric information?
- How is the information about vertices, edges, and facts stored?
- Which operations are required?

General operations:

- Vertices, edges, facets: selection, insertion, removal, merging, rotation, scaling, ...
- Merging of meshes, differences of meshes, trimming (cutting of at) of meshes, ...

6.2 Polygonal representation

Topological operations:

- Determine all edges emanating from a given vertex.
- Determine all faces, that share a certain edge or vertex!
- Determine the vertices connected by an edge!
- Determine all edges of a given facet!
- Consistency checks
  - Is something missing? Are there holes?
  - Is there redundant information?

6.2 Polygonal representation

In practice, data structures do not only hold topological and geometrical information of the polygonal representation, but also attributes necessary for the renderer or application:

- Face-attributes:
  triangle?, surface, surface normal?, coefficients of the surface?, convex?, holes?
- Edge-attributes:
  length?, adjacent polygons or surfaces?, boundary edge?
- Vertex-attributes:
  adjacent polygons, vertex normal, texture coordinates

Average of face normals
of adjacent faces

6.2 Polygonal representation

6.2.5 Approaches

There are various options, depending on the application:

- Regular structures with implicit topology and geometry,
- Mixed structures with implicit topology and explicit geometry,
- Unstructured data with explicit topology and geometry
  - General polygonal meshes.

6.2 Polygonal representation

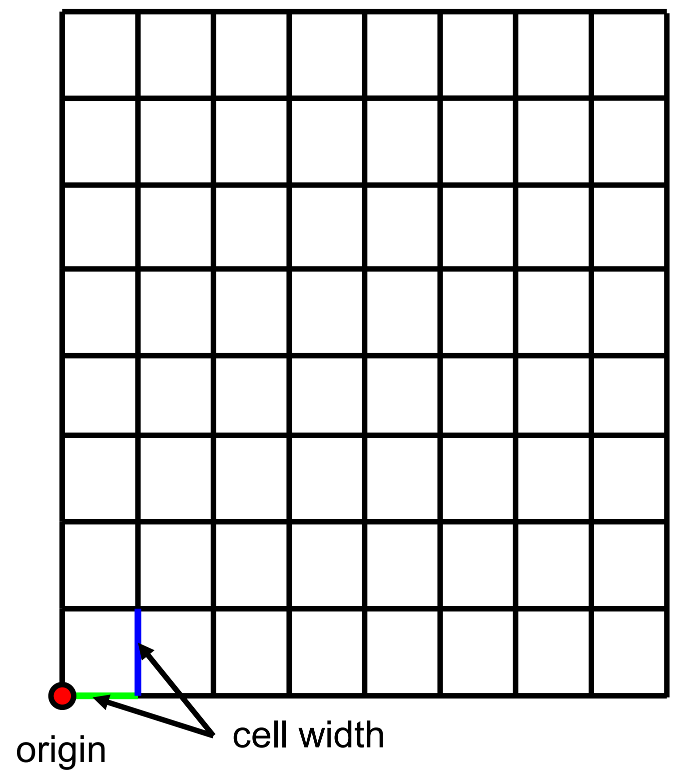
6.2.6 Regular structures

- Orthogonal, uniform grid.
- Topology is determined by number of points in x- and y-direction (and z-direction).
- Geometry is determined by origin and cell width.
- **Example:** Origin = $(-1,-1)$,

  $x_{\text{dim}} = 0.2, \; y_{\text{dim}} = 0.1,$

  $n_x = 10, \; n_y = 20$

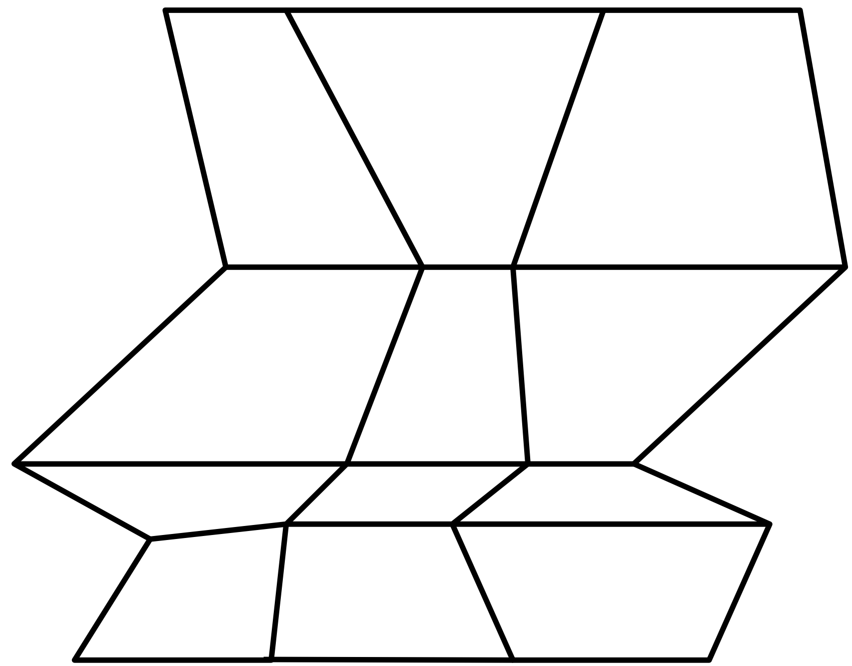
  ➡ What are the coordinates of the top right point?



6.2 Polygonal representation

6.2.7 Mixed structures

- Topology is determined by number of points in x- and y-direction (and z-direction).
- Geometry is stored explicitly in an array by each point's coordinates.



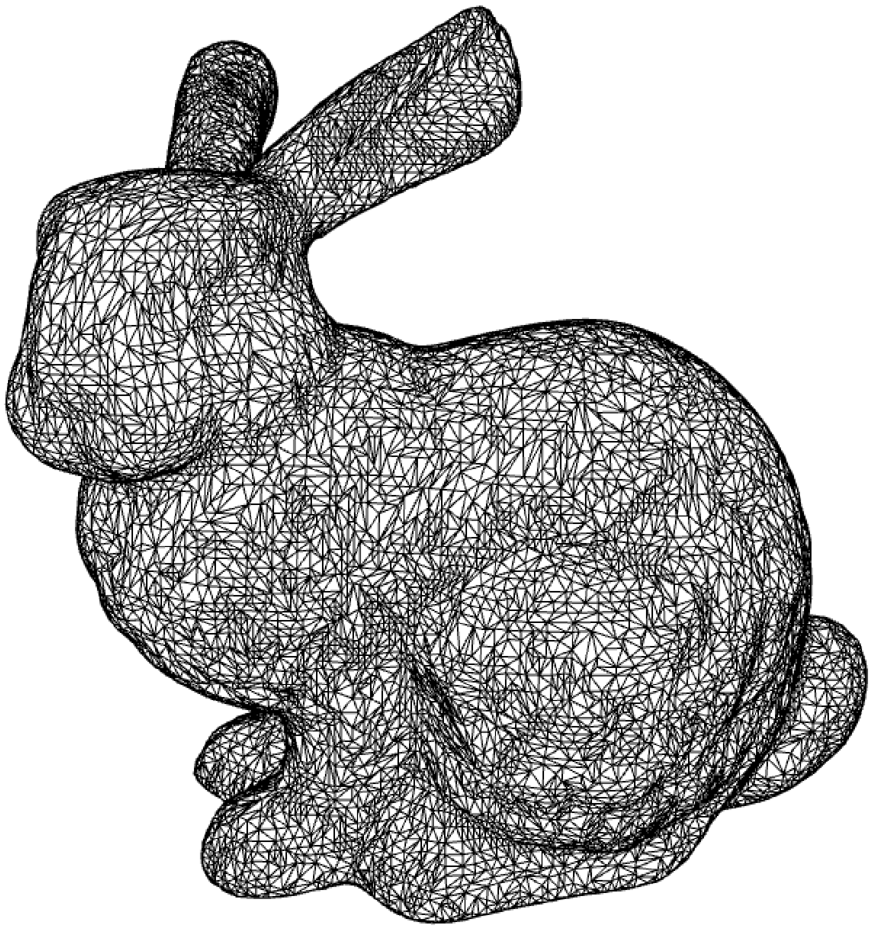
6.2 Polygonal representation

6.2.8 Polygonal meshes

- Topology and geometry have to be stored explicitly!
- The problem is the topology; the geometry is just a list of point coordinates!

Different options:

- *explicit storage,*
- *vertex list,*
- *edge list,*
- *winged-edge,*
- *half-edge, etc.*

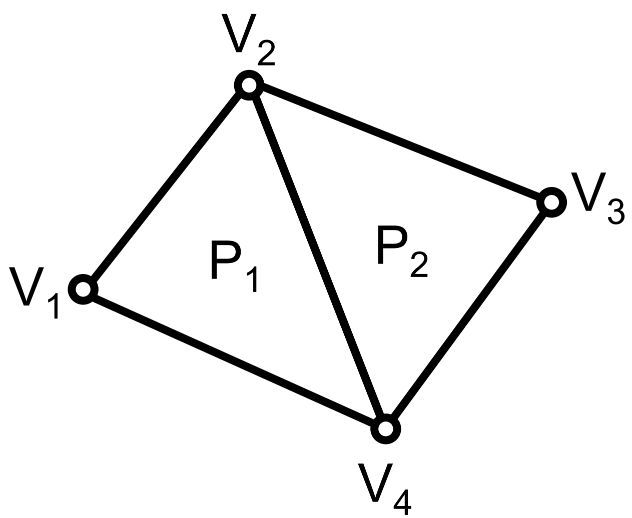


6.2 Polygonal representation

6.2.8.1 Polygonal mesh, explicit storage, "triangle soup"

- Each polygon is determined by a list of its vertex' coordinates.
- Between every pair of vertices is an edge, also between the last and the first vertex.

Example:



$$P_1 = ((V_{2x}, V_{2y}\ V_{2z}),\ (V_{1x}, V_{1y}\ V_{1z}),\ (V_{4x}, V_{4y}\ V_{4z}))$$

$$P_2 = ((V_{4x}, V_{4y}\ V_{4z}),\ (V_{3x}, V_{3y}\ V_{3z}),\ (V_{2x}, V_{2y}\ V_{2z}))$$

6.2 Polygonal representation

Remarks

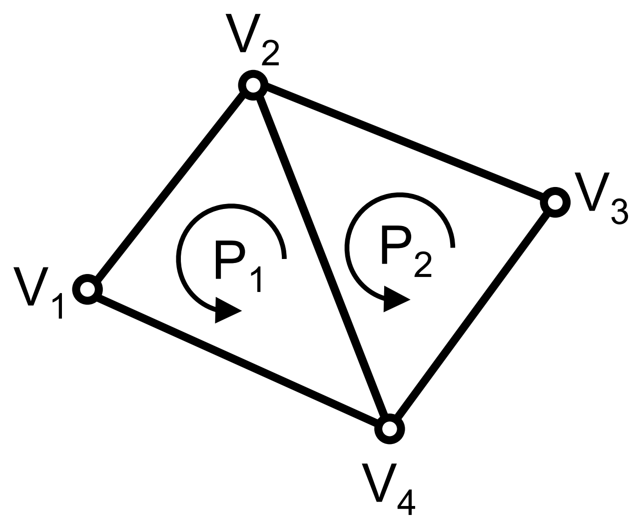
- Memory expensive representation:
  - Coordinates of vertices are stored multiple times.
- The is no explicit information about common vertices or edges:
  - How do you determine the common edge of two triangles?
  - How do you determine all edges emanating from a vertex?
- Geometry cannot be changed independent from the topology, because common vertices have to be determined.
- Is used in the stl-format (SurfaceTesselationLanguage, StandardTriangulationLanguage, StandardTesselationLanguage)
  - Stores also face-normals (redundant).

6.2 Polygonal representation

6.2.8.2 Polygonal mesh, vertex list

- All vertices are stored in a point list (vertex list).
- A polygon is determined by a list of indices (links) into the point list.

Example:



$V = (V_4, V_2, V_1, V_3)$

$P_1 = (3, 1, 2)$

$P_2 = (1, 4, 2)$

Care for consistent orientation!

6.2 Polygonal representation

Remarks

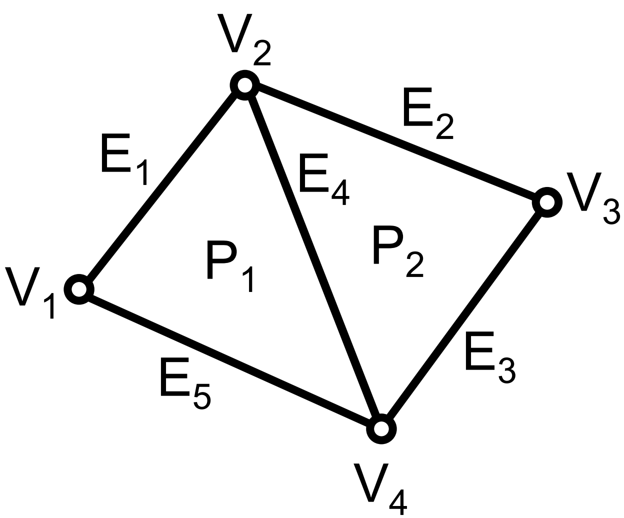
- Every vertex is stored exactly once.
- Geometry can be changed independent from the topology.
- For the graphical representation edges are drawn multiple times!
- To determine common edges and vertices of polygons is still difficult!
- Is used in the VRML- (Virtual Reality Modeling Language) and off-formats (object file format).
  - Both offer much more features, e.g. polynomial representation!

6.2 Polygonal representation

6.2.8.3 Polygonal mesh, edge list

- All vertices are stored in a point list (vertex list).
- All edges are stored in an edge list.
- A polygon is determined by a list of indices into the edge list.

Example:



$V = (V_1, V_2, V_3, V_4)$

Index of start vertex
Index of end vertex
Left polygon
Right polygon

$E_1 = (2, 1, P_1, N)$,
$E_2 = (3, 2, P_2, N)$,
$E_3 = (4, 3, P_2, N)$,
$E_4 = (4, 2, P_1, P_2)$,
$E_5 = (1, 4, P_1, N)$,
$E \;= (E_1, E_2, E_3, E_5, E_4)$

$P_1 = (1, 4, 5), P_2 = (2, 5, 3)$

6.2 Polygonal representation

Remarks

- Geometry can be changed independent from the topology.
- For the graphical representation edges are drawn exactly once!
- To determine common edges of polygons has become simpler:
  - Simply test the second polygon index in the data structure for an edge.
- To determine common vertices of polygons is still difficult!

6.2 Polygonal representation

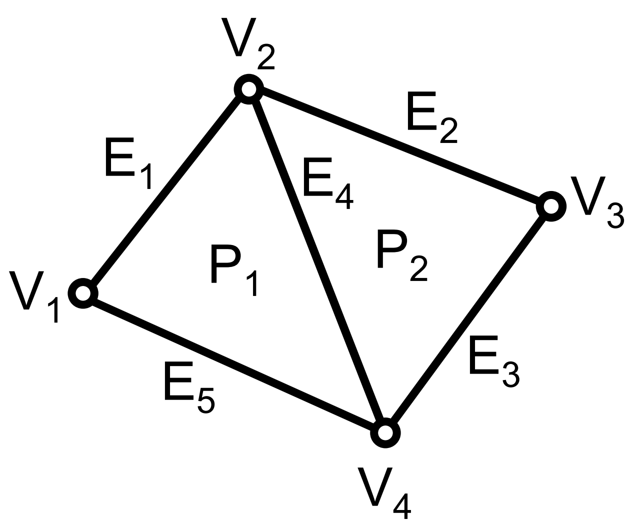
Input	Output	Procedure	$O(?)$
Vertex (Valence k)	Emanating edges	Check all n edges for vertex and traverse edges in corresponding facets.	$O(n)$
	Neighbor vertices	Vertices of emanating edges.	$O(n)$
	Adjacent facets	Polygons of emanating edges.	$O(n)$
Edge	End points	Start-/end-vertex in edge.	$O(1)$
	Successor/predecessor	Traverse polygon (k-gon).	$O(k)$
	Adjacent facets	Polygons in edge data.	$O(1)$
Facet (k-gon)	Vertices (corners)	Vertices in edge data.	$O(k)$
	Edges	Edges in polygon data.	$O(k)$
	Adjacent facets	Polygons in edge data.	$O(k)$

6.2 Polygonal representation

6.2.8.4 Polygonal mesh, doubly linked edge list (winged-edge)

- Vertices, edges and polygons as in the edge list representation.
- An edge has pointers to the successor and predecessor edges in both adjacent polygons (**winged-edge**).

Example:



Succ. left
Pred. left
Succ. right
Pred. right

$V = (V_1, V_2, V_3, V_4)$

$E_1 = (2, 1, P_1, N, 4, 5, N, N)$,
$E_2 = (3, 2, P_2, N, 5, 3, N, N)$,
$E_3 = (4, 3, P_2, N, 2, 5, N, N)$,
$E_4 = (4, 2, P_1, P_2, 1, 4, 3, 2)$,
$E_5 = (1, 4, P_1, N, 5, 1, N, N)$,
$E = (E_1, E_2, E_3, E_5, E_4)$

$P_1 = (1, 4, 5), P_2 = (2, 5, 3)$

6.2 Polygonal representation

Remarks

- Geometry can be changed independent from the topology.
- There are nine adjacency relations:
  - Which facet, edge or vertex belongs to each facet, each edge or each vertex?
- Determining edges and facets of an edge is possible in constant run-time.
- Determining all edges or facets of a vertex is still difficult.

6.2 Polygonal representation

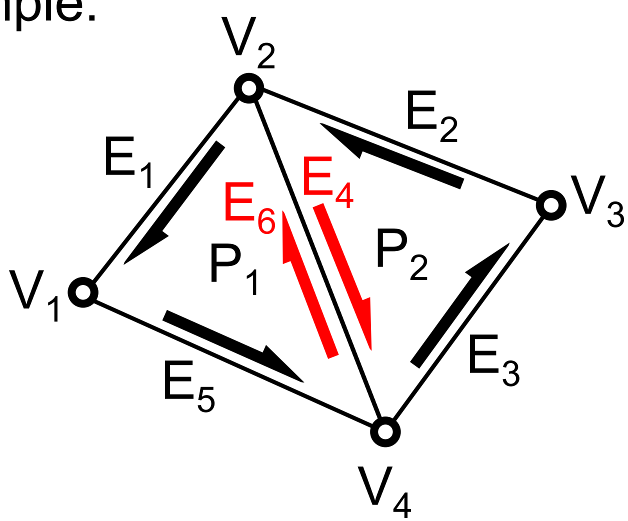
Input	Output	Procedure	$O(?)$
Vertex (Valence k)	Emanating edges	Check all n edges for vertex and use successor/predecessor pointers.	$O(n)$
	Neighbor vertices	Vertices of emanating edges.	$O(n)$
	Adjacent facets	Polygons of emanating edges.	$O(n)$
Edge	End points	Start-/end-vertex in edge.	$O(1)$
	Successor/predecessor	Successor/predecessor pointers.	$O(1)$
	Adjacent facets	Polygons in edge data.	$O(1)$
Facet (k-gon)	Vertices (corners)	Vertices in edge data.	$O(k)$
	Edges	Edges in polygon data.	$O(k)$
	Adjacent facets	Polygons in edge data.	$O(k)$

6.2 Polygonal representation

6.2.8.5 Polygonal mesh, half-edge

- Edges become half-edges containing pointers to
  - Start vertex, polygon, predecessor- and partner-half-edge.
- Vertex contains pointer to (almost) arbitrary initial start-half-edger.
- Polygon contains pointer to arbitrary start-half-edge.

Example:



$V = ((V_1,4), (V_2,1), (V_3,2), (V_4,3)),$

$E_1 = (2, P_1, 6, N), E_2 = (3, P_2, 3, N),$

$E_3 = (4, P_2, 5, N), E_4 = (2, P_2, 2, 6),$

$E_5 = (1, P_1, 1, N), E_6 = (4, P_1, 4, 5),$

$E = (E_1, E_2, E_3, E_5, E_4, E_6)$

$P_1 = (1), P_2 = (2)$

6.2 Polygonal representation

Remarks

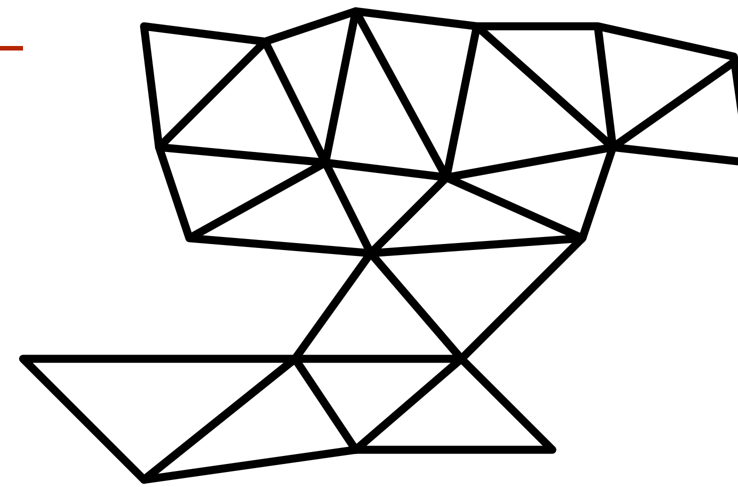
- Geometry can be changed independent from the topology.
- All adjacency relations can be determined in constant run-time.
- Special treatment of boundary edges:
  - Boundary edge: partner-half-edge is null-pointer.
  - Allow only two boundary edges per boundary vertex.
  - Start-half-edge is right most edge (as seen from the vertex looking towards the mesh).

6.2 Polygonal representation

Input	Output	Procedure	$O(?)$
Vertex (Valence k)	Emanating edges	Use start-half-edge and partner-half-edge of predecessors.	$O(k)$
	Neighbor vertices	Vertices in partner of emanating edges.	$O(k)$
	Adjacent facets	Polygons in emanating half-edges.	$O(k)$
Edge	End points	Start vertex in half-edge and its partner.	$O(1)$
	Successor/predecessor	Predecessor in half-edge and traverse predecessors in k-gon for the successor.	$O(k)$
	Adjacent facets	Polygons in half-edge and its partner.	$O(1)$
Facet (k-gon)	Vertices (corners)	Vertices in half-edge.	$O(k)$
	Edges	Traverse predecessors of start-half-edge.	$O(k)$
	Adjacent facets	Polygons of partners of facet's half-edges.	$O(k)$

6.2 Polygonal representation

6.2.9 Triangle meshes



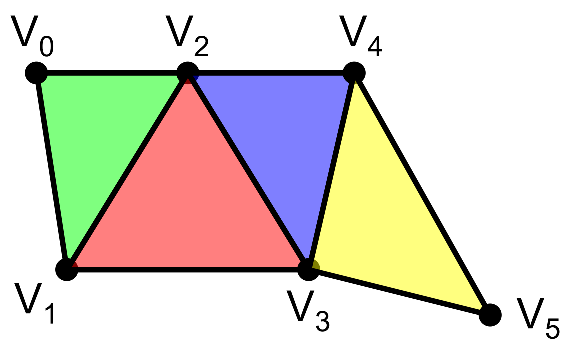
- A special form of polygonal meshes!
- The geometric primitive at the end of the graphics pipeline is a triangle or a structured set of triangles.
- For triangle meshes exist **special data structures** (e.g. **triangle strip**, **triangle fan**), which
  - code/store topology implicitly and thus
  - generate less memory costs and
  - yield a better performance on the graphics hardware (e.g. supported by OpenGL, Direct3D and Java3D).

6.2 Polygonal representation

6.2.9.1 Triangle Strip

- List of at least three vertices.
- Each triple of consecutive vertices determines one triangle.
- $n + 2$ vertices determine n triangles.

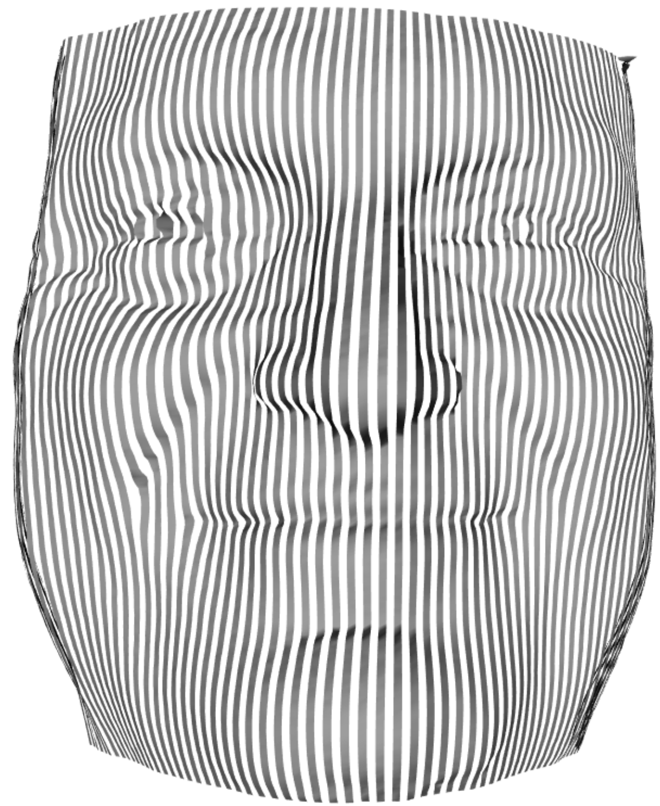
Example: $(V_0, V_1, V_2, V_3, V_4, V_5)$



6.2 Polygonal representation

Example: Triangle Strips

2297 strips with an average length of 3.94 triangles, the longest strip with 101.



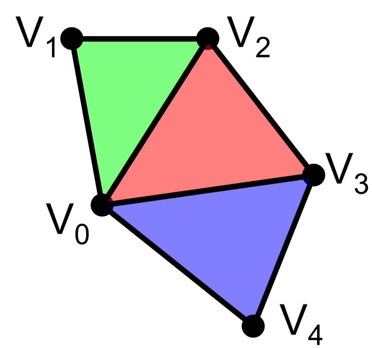
134 strips with a length of 390 triangles.

6.2 Polygonal representation

6.2.9.2 Triangle Fan

- List of at least three vertices.
- Each pair of consecutive vertices together with the first vertex in the list determine one triangle.
- $n+2$ vertices determine n triangles.

Example: $(V_0, V_1, V_2, V_3, V_4)$



6.2 Polygonal representation

6.2.10 Generation of polygonal objects

- **Manual methods:** Manual manipulation of (groups of) vertices using 3d input devices or 3d input interfaces:
  - complex, hard to handle,
  - only applicable for simple objects or simple manipulations.

FARO

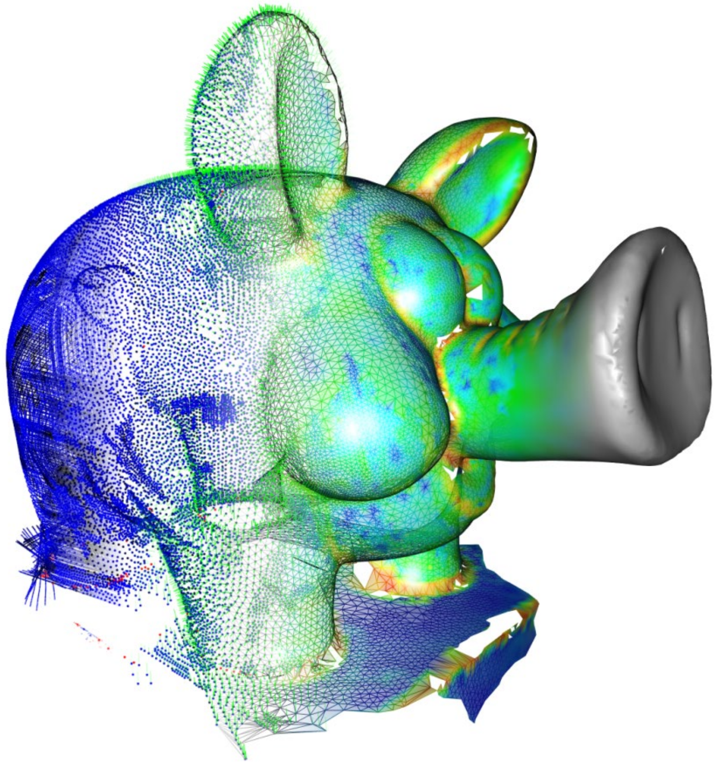
6.2 Polygonal representation

- **Half-automatic methods** (3d-digitizer)**:** Manual or automatic mounting of patterns on the object, which are used to digitize points on the surface.
  - Example: “pull” mesh over the surface.
  - First 3d-representations of car bodies (1974).

David-Scanner

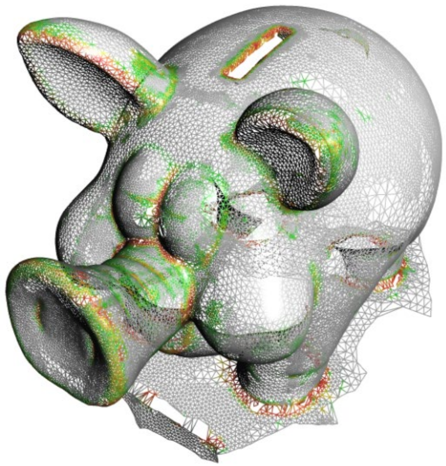
6.2 Polygonal representation

- **Automatic methods** (e.g. laser scanner):
  - Object is sampled in slices/scan-lines with a laser, which measures the distance to the object surface.
  - From these measured points suitable neighboring points are triangulated using skinning-algorithms.
  - Application: Reverse engineering, virtual garments, etc.

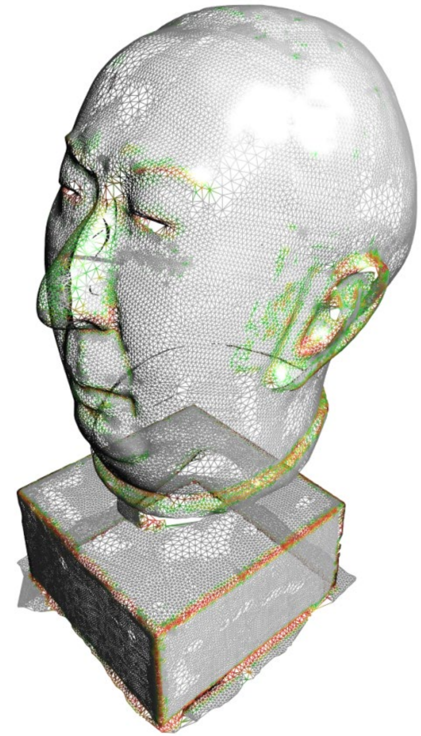


6.2 Polygonal representation

- **Problem:** Tends to generate too many points and triangles.



ca. 1.3 Mio points,
ca. 10.000 triangles



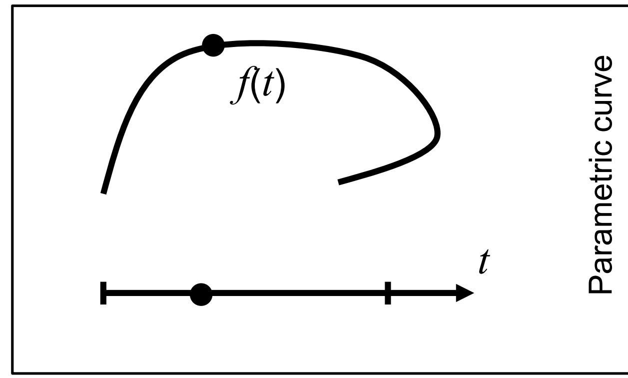
Gerhard Marcks:
Bildnis Theodor Heuss

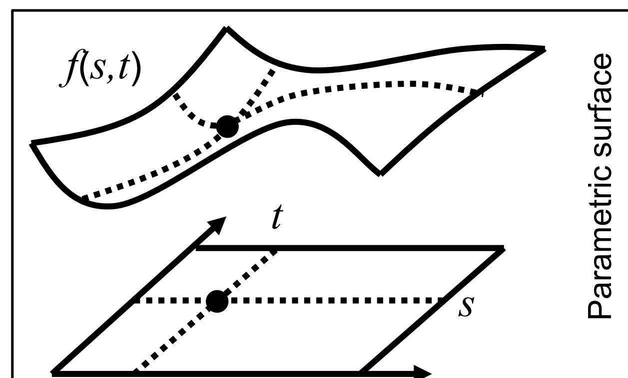
ca. 2.5 Mio points,
ca. 15.000 triangles

- **Problem:** For non-convex objects not every part of the object surfaces can be seen by the laser, i.e. it cannot be scanned.

6.2 Polygonal representation

- **Mathematical methods:**
  - Generation of polygonal representations from parametric curves and surfaces.
  - **Application:** CAD
  - **Advantage:**
    - User works with the high-level object representation.
    - Shape of the object is directly coupled with its mathematically exact representation.
  - **Example:** parametric surfaces (piecewise polynomial), rotational surfaces, sweep-surfaces, etc.

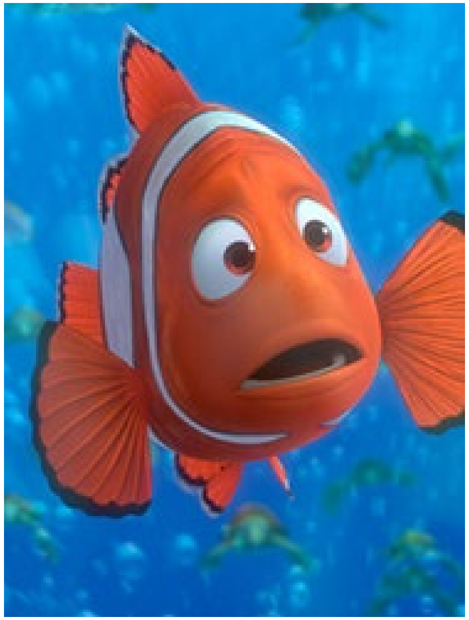




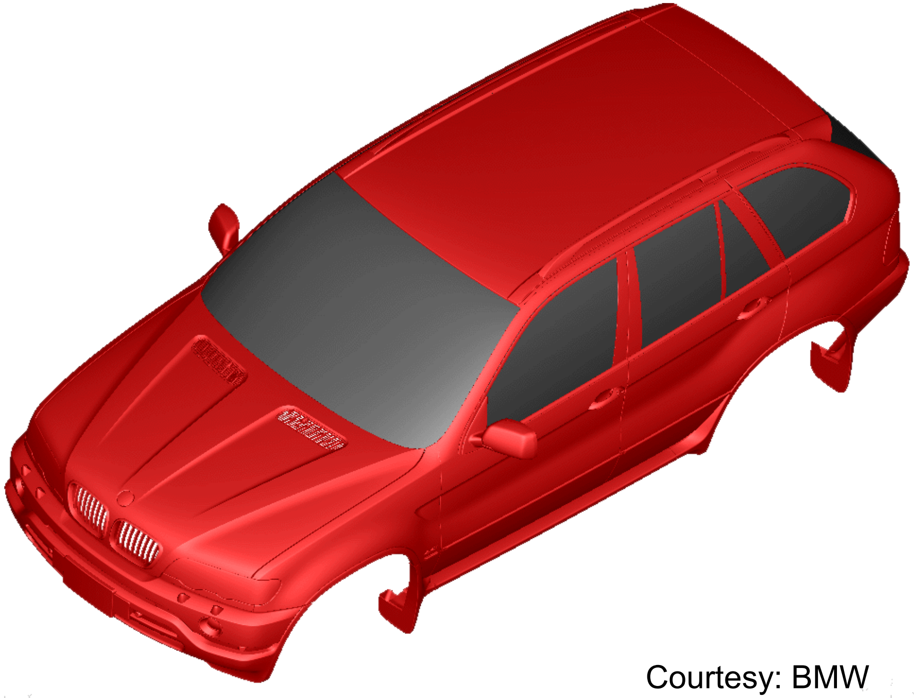
6.2 Polygonal representation

Parametric curves and surfaces are (besides polygonal representations) the most used representations.

➡ **Lecture „Geometric modeling (MSI)“**



Courtesy: Pixar



Courtesy: BMW

6.2 Polygonal representation

- **Procedural methods**
  - A popular methods to generate polygonal objects on the fly (procedurally) are so-called fractals.
  - Fractals are theoretically founded in the Mandelbrot-geometry and are used in computer graphics for the modelling of geographical height fields (terrain models).
  - Fractals are very efficient and are used in applications such as professional flight simulators for pilot training.

6.2 Polygonal representation

6.2.11 Level-Of-Detail methods

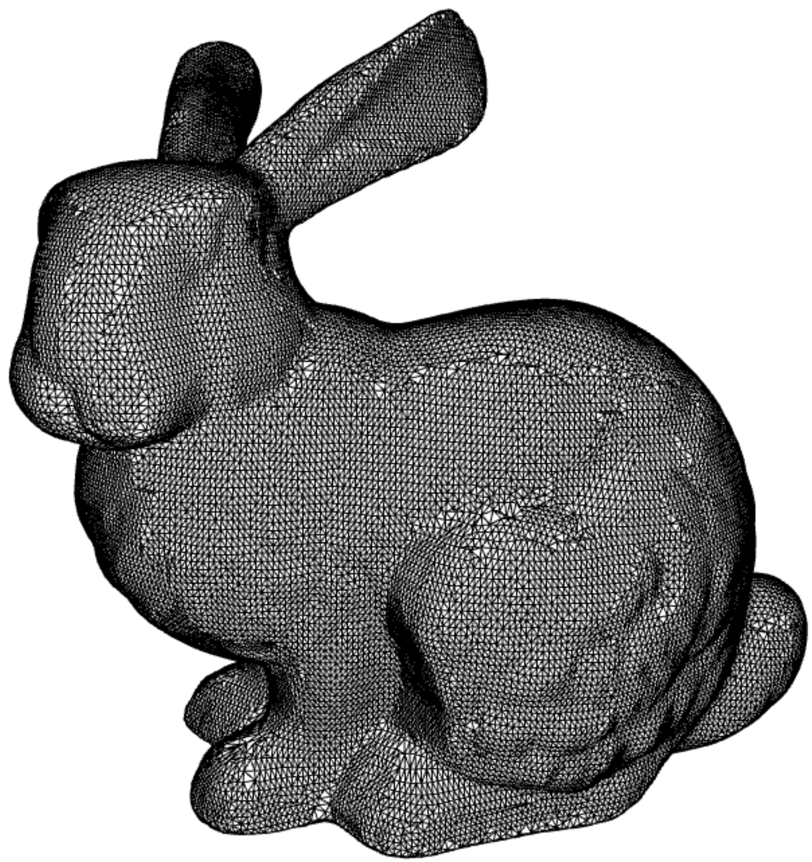
- The root represents the initial cube (containing the scene).
- Automatic methods tend to generate (too) many polygons
  - Problem: The ratio

    (number of polygons in object) : (size of projected area on object)

    is too large.
- Overhead for the storage, transmission, manipulation, and visualization of „unnecessary“ polygons:
  - Use different polygonal resolution in the object representation: level of detail.
  - Maintained in a **detail pyramid**.

6.2 Polygonal representation

- **mesh simplification**
  Reduce number of polygons, such that mesh quality become sufficient for actual task.
- **level of detail approximation**
  Avoid „popping“, i.e. visual jumps at the transitions between different resolutions
  - geomorphs (smooth visual transitions)



6.2 Polygonal representation

- **progressive transmission**
  3d-equivalent to progressive transmission of different resolutions for 2d-bitmap-images.
- **mesh compression**
  Minimization of memory for the point coordinates using compression methods (transformation, quantization, and coding of the coefficients)
  - Wavelet-approach: Base-mesh and meshes of local differences.
  - ➡ Lossy compression: small coefficients are not stored.
  - ➡ Problems:
    - How do you store a 3d-difference/-change?
    - What is a good predictor?

6.2 Polygonal representation

- **selective/adaptive refinement**
  Dynamic context dependent LOD-technique.
  - **Example:** Plane flies over landscape, which is represented in full detail only at the actual location of the plane or what is in the pilot's field of view.

6.3 CSG-Representation

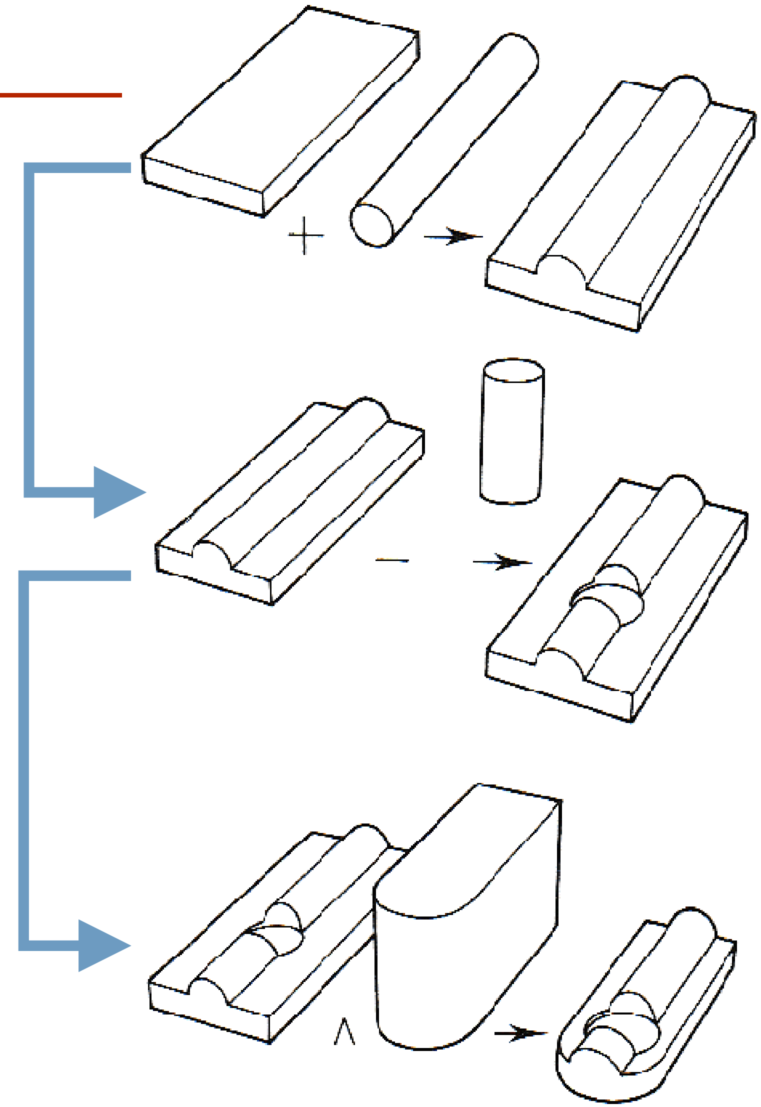
Constructive Solid Geometry

- Volume representation of 3d objects.
- **Advantage:** Representation of complex objects via combination of simple primitive objects using Boolean operations and affine transformations.
- Geometric primitives: Sphere, cone, cylinder, cuboid, ...
- **Advantage :** Allows for an interactive construction.
- **Disadvantage:** Representation of CSG-objects requires special rendering methods (e.g. ray-tracing) or conversion to a polygonal representation (boundary evaluation).

6.3 CSG-Representation

Boolean operations

1. Union of a cuboid and a cylinder, ...
2. ... subsequent different with a second cylinder, and ...
3. ... subsequent intersection with an object, that is defined as union of a cuboid and a cylinder.

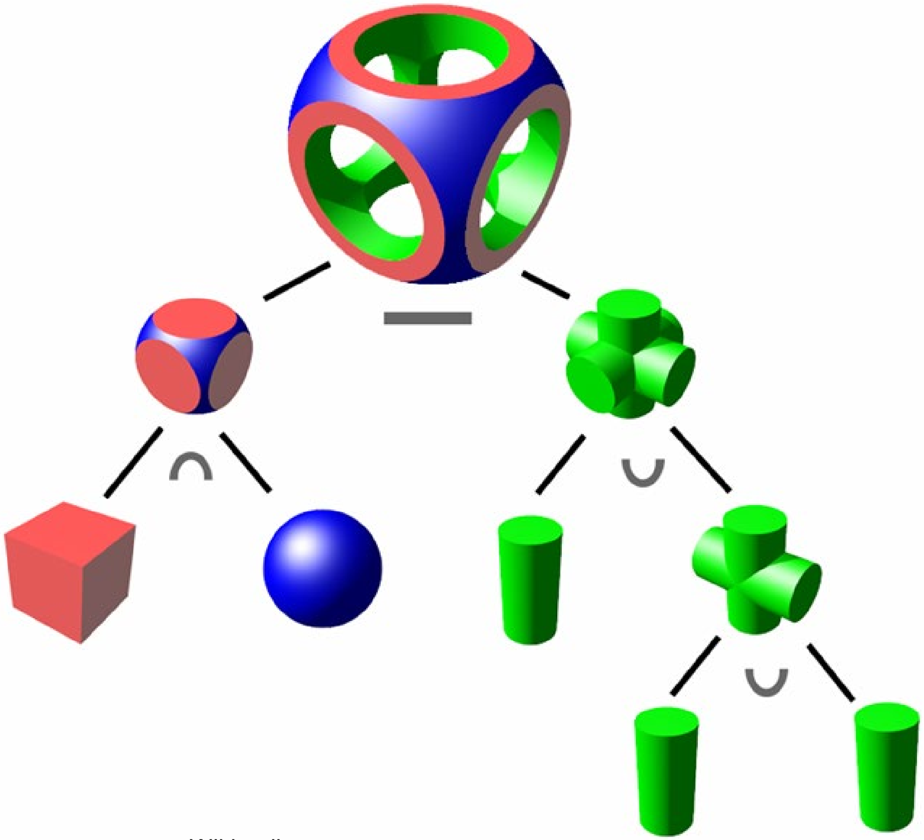


6.3 CSG-Representation

- One particular object can be represented with different CSG-operations.
  - Here: Steps 2 and 3 can be exchanged!
  - But: CSG-operations are in general not commutative!
- A CSG-representation is described by a tree:
  - Inner nodes:
    - Information about the Boolean operations and
    - information about the spatial relation between its siblings (given as an affine transformation).
  - Leaves:
    - Name of the primitive and
    - its dimensions and attributes.

6.3 CSG-Representation

Example: Construction of a complex objects from primitives.



source: Wikipedia

6.4 Space partitioning

Representation of an object using space partitioning:

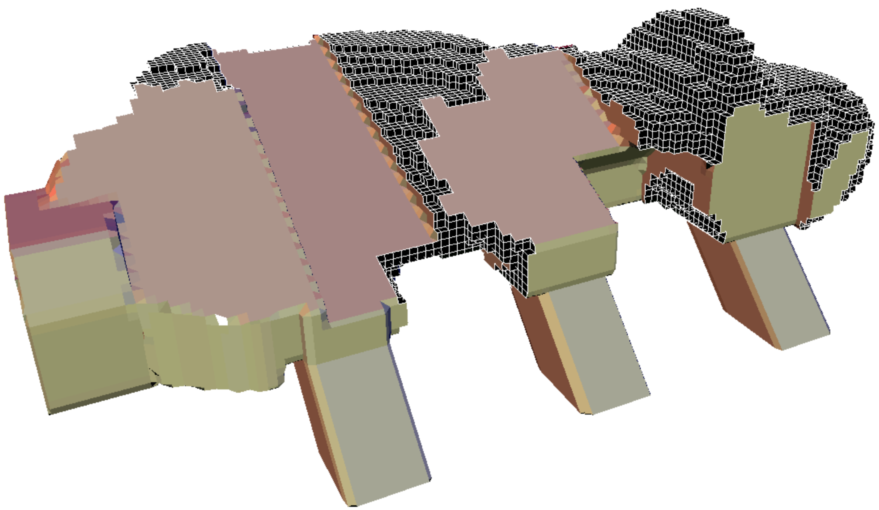
- Object space is subdivided into small elements.
- For each element store a state, if it overlaps with the object.

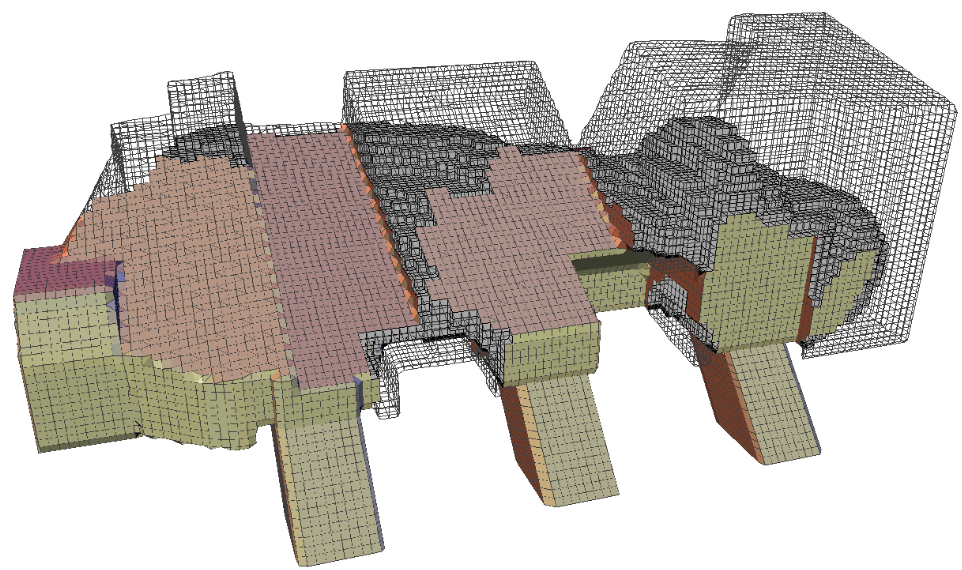
Standard approach:

- Subdivide object space into a fixed, regular grid using cells with identical geometry.
- In 3d this yields cube-shaped cells, so-called **voxels** (volume elements).

➡ The analog in 2d are pixels.

6.4 Space partitioning





Source: Daimler

6.4 Space partitioning

- **Advantages:**
  - It is very simple to test if a point is inside or outside of an object.
  - It is very simple to test if two objects touch/intersect.
  - The representation of a particular object is unique.

- **Disadvantages:**
  - There are also only partially overlapped cells.
  - Objects are (in general) only approximated.
  - At a resolution of n voxels, n^3 voxels are required:
    - **extremely memory insensitive,**
    - cheaper representation using octrees.

6.4 Space partitioning

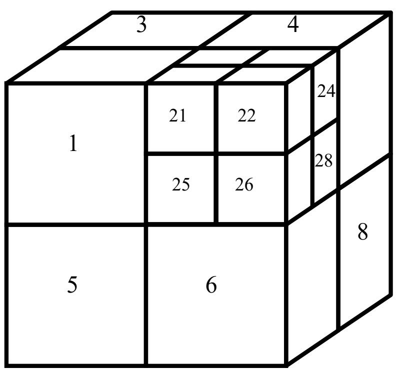
6.4.1 Octrees

An octree is a hierarchical data structure for the efficient storage of an irregular subdivision of 3-space.

- It is a tree with eight siblings per inner node

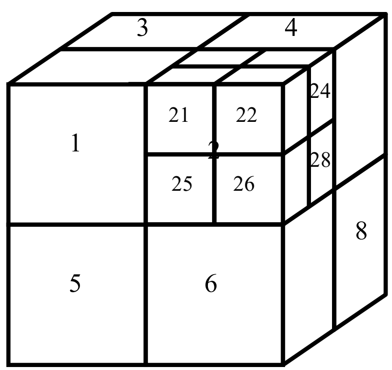
Principle:

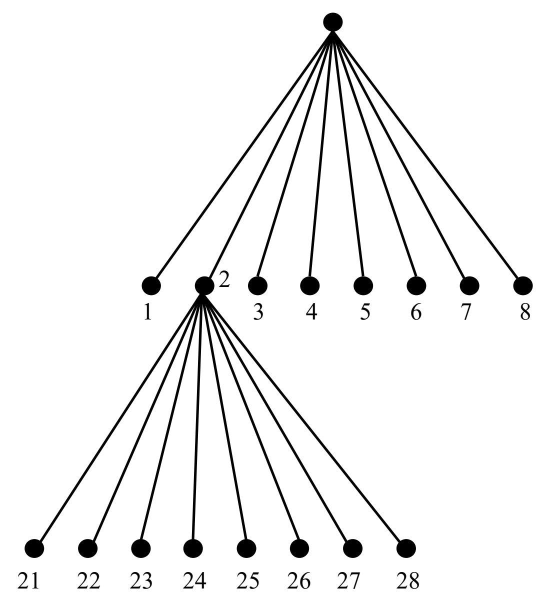
- The initial element is a cube, that contains the object space and accepts the occupancy-states ***occupied / not-occupied***.
- First the occupancy-state of the cube is determined.
- If it is only **partially occupied**, the cube is halved along every edge.
- Apply this strategy recursively to every sub-cube until the target resolution is reached.



6.4 Space partitioning

- The root represents the initial cube.
- All inner node have exactly eight siblings.
- At each subdivision, for each subdivided cube the generated siblings are inserted using a fixed ordering of sub-cubes.
- Each leave stores the occupancy-state of its respective sub-cube.
- Each inner node represents a partially-occupied sub-cube.





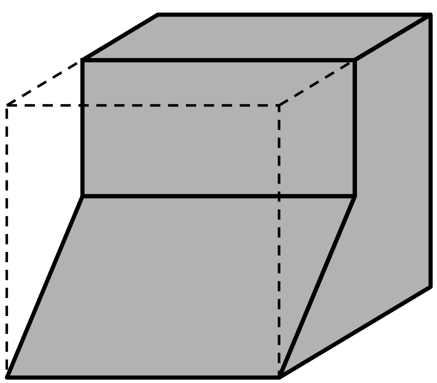
6.4 Space partitioning

Example: Representation of a 3d-object using an octree

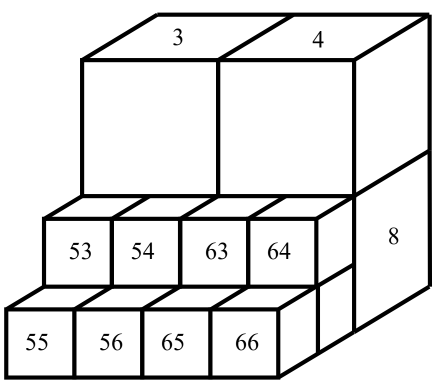
a) Object, embedded into the initial cube.

b) Representation of the object using only two subdivisions of the space.

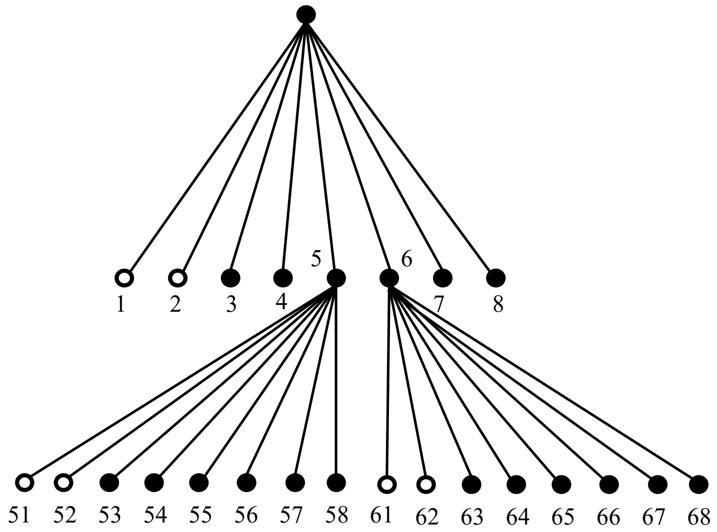
c) Resulting octree-data-structure.



a)



b)



c)

6.4 Space partitioning

Application: Octrees are used for the spatial subdivision of objects within a 3d-scene.

- The individual objects are represented using their standard data-structures (e.g. polygonal mesh).
- The nodes of the octree store a list containing all overlapping objects or polygons, respectively.

➡ Significant acceleration of algorithms, that process the geometric data within local regions of the space, e.g. ray tracing, collision detection, etc.

➡ Fast navigation in octrees:

Lecture „Computational Geometry (MSI)".

6.4 Space partitioning

6.4.2 Quadtrees

- Generalization of octrees to n dimensions.
- $n = 2$: Subdivision of the plane, i.e. so-called **quadtrees**:
  - Every inner node of the tree has exactly four siblings.
- Historically older than octrees.
- **Applications**
  - Image representation,
  - Face indexing (e.g. in GIS-programs),
  - Efficient collision detection in 2d,
  - Hidden surface removal for terrain data.
  - Grid generation for the generation of 2d-triangulations.

6.4 Space partitioning

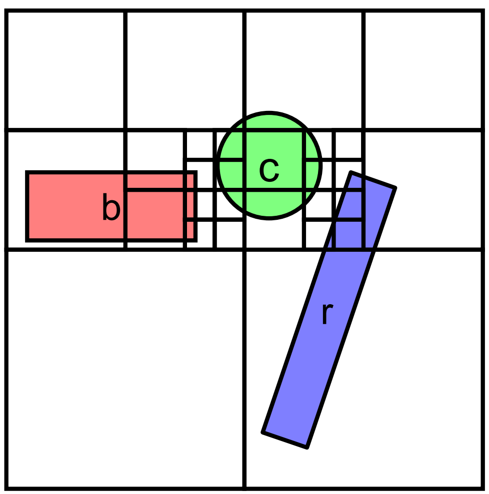
Example: Subdivision of a 2d-scene by a quadtree.

a) Subdivision of 2d-space, until every cell contains only one object.

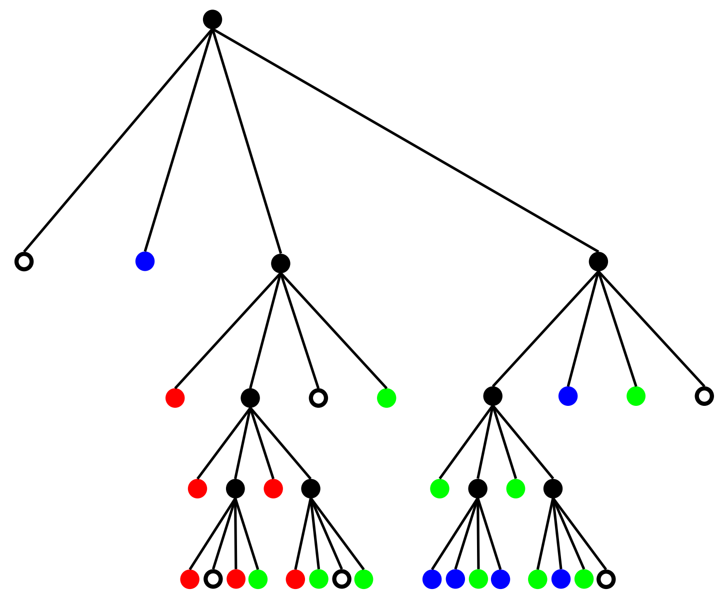
➡ Problem: Fragmentations!

b) Resulting quadtree-data-structure.

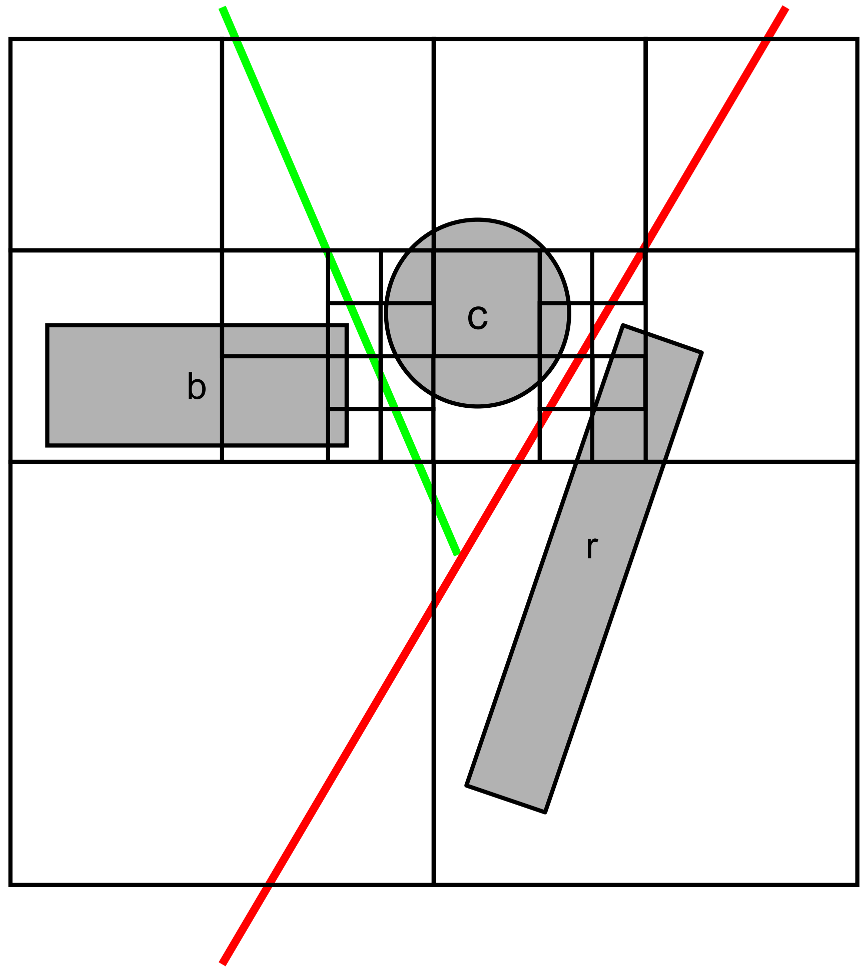
a)



b)



6.4 Space partitioning



6.4 Space partitioning

6.4.3 Binary Space-Partitioning Trees

- Octrees/Quadtrees subdivide the space along pairwise perpendicular hyper-planes in all dimensions at the same time.
  - **Disadvantage:** Sometimes badly balanced!!
- Alternative: The space is recursively subdivided into **two** sub-spaces along arbitrary (hyper-) planes

  BSP-trees.
- If one sub-spaces if defined as “inside” an the other as “outside”, arbitrary convex polyhedrons can be modelled using oriented bounding hyper-planes.
- Via unions of convex “inside”-regions, arbitrary non-convex polyhedrons with holes can be defined.

6.4 Space partitioning

- Application in computer graphics:
  Determine visibility of objects
  - BSP-trees can be used (as octrees) for the subdivision of a scene.
    - However, they are not bound to a certain grid!
  - Space is recursively subdivided along planes, such that each region contains at most one object.
  - The relative position of these regions to an observer/camera can be used for a efficient depth sorting of the objects.
    - View direction v, m objects in the tree, runtime for the sorting of the objects in direction v: $O(m)$.

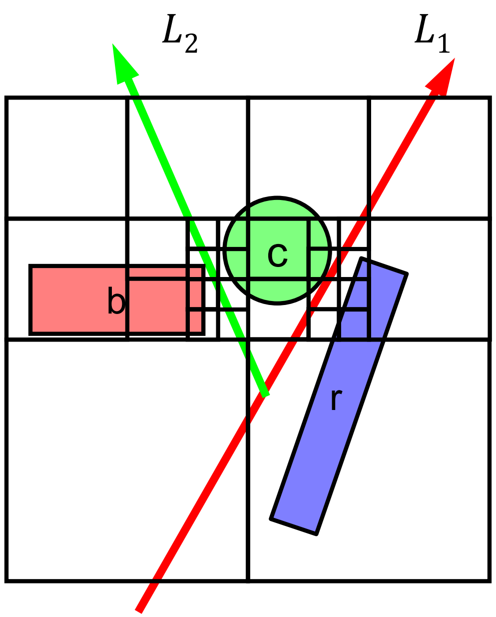
➡ Which objects are visible / occluded at all?

6.4 Space partitioning

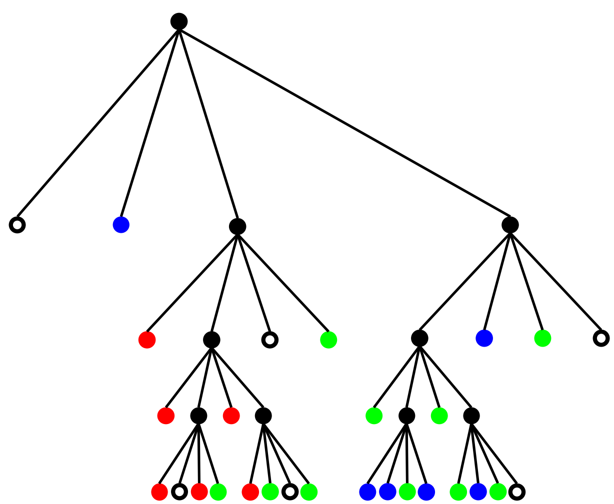
Example: Subdivision of a 2d-scene

a) by a quadtree and

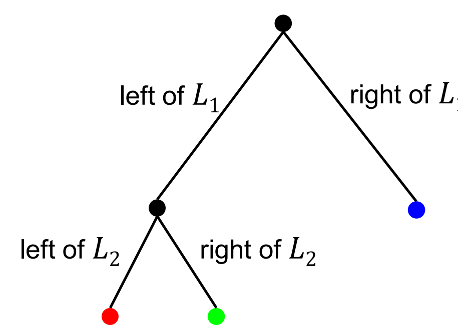
b) by a corresponding BSP-tree.



a) Quadtree



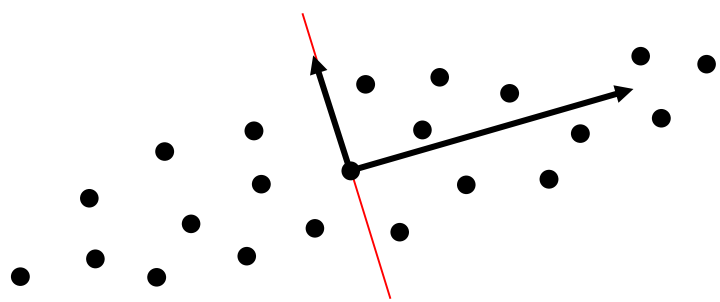
b) BSP-tree



6.4 Space partitioning

6.4.4 Principal Component Analysis (PCA)

- How to choose the hyper-planes for the partitions in a BSP-tree?
- Complex scene is given by point cloud P_i in $\mathbb{R}^3$, $i = 1, \dots, n$,
  - e.g. Object centers or vertices of polygons.
- PCA yields orthogonal coordinate system e_1, e_2, e_3, which is aligned with the point cloud.



6.4 Space partitioning

- The point average becomes the origin: $c = \frac{1}{n}\sum_{i=1}^{n} P_i$.
- Matrix $B = \frac{1}{n-1} A^t A \in \mathbb{R}^{3\times 3}$ with

$$A = \begin{pmatrix} P_{1x} - c_x & P_{1y} - c_y & P_{1z} - c_z \\ P_{2x} - c_x & P_{2y} - c_y & P_{2z} - c_z \\ \vdots & \vdots & \vdots \\ P_{nx} - c_x & P_{ny} - c_y & P_{nz} - c_z \end{pmatrix}$$ has

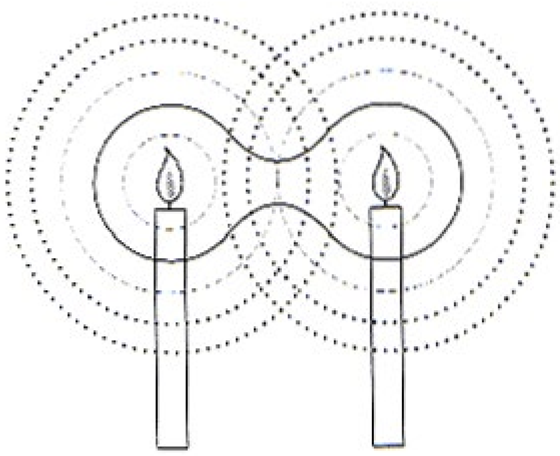
  - real eigenvalues $\lambda_1, \lambda_2, \lambda_3$ with $\lambda_i e_i = B e_i$ and
  - pairwise orthogonal eigenvectors e_1, e_2, e_3, i.e. $e_i \perp e_j, i \neq j$.
- Eigenvectors and origin c define a suitable system.
- Elongation of point cloud in direction e_i is proportional to $\sqrt{\lambda_i}$.
  - Analog for arbitrary dimensions.

➡ Chose plane normal to eigenvector of maximal eigenvalue.

6.5 Implicit representation

- **Idea:** Description of object surfaces of volumes using scalar fields (i.e. one scalar value for every space point) as iso-surfaces (e.g. surfaces of equal values).
  - Scalar fields can be generated in a controlled way using generating primitives (functions).
- Very well suited for physical phenomena.

- **Example:**
  - Point heat sources generate spherical field function.
  - Addition of both fields generates global scalar field.
  - *Other examples:* CT, MRT, ultrasound, isobars, isotherms, contour lines, etc.



6.5 Implicit representation

6.5.1 „Blobs“, Meta-balls

- Field function $F_d\colon \mathbb{R}^3 \to \mathbb{R}$ at center point $P \in \mathbb{R}^3$.
  - F_d is basically composition of a
    - monotonically decreasing **influence function** and a
    - **distance measure** (hear Euclidean distance) between free parameter $x \in \mathbb{R}^3$ and center point P.
  - The index d indicates that the field function is defined at a discrete center point.

- Blob, Meta-ball: The surface that is defined implicitly at a given **iso-value** of the scalar field F_d.

6.5 Implicit representation

- Multiple blobs with field functions $F_d(i, \cdot): \mathbb{R}^3 \to \mathbb{R}$ at the centers $P_i \in \mathbb{R}^3, i = 1, \ldots, n$, interfere with each other following the **superposition principle**:
  - Resulting field function F of the resulting scalar field is given by the sum of the individual field functions:

$$F: \mathbb{R}^3 \to \mathbb{R}, \quad F(x) = \sum_i^n F_d(i, x).$$

- Pre-defined constant c smaller than the maximum in the scalar flied:
  - Set of all $x \in \mathbb{R}^3$ such that

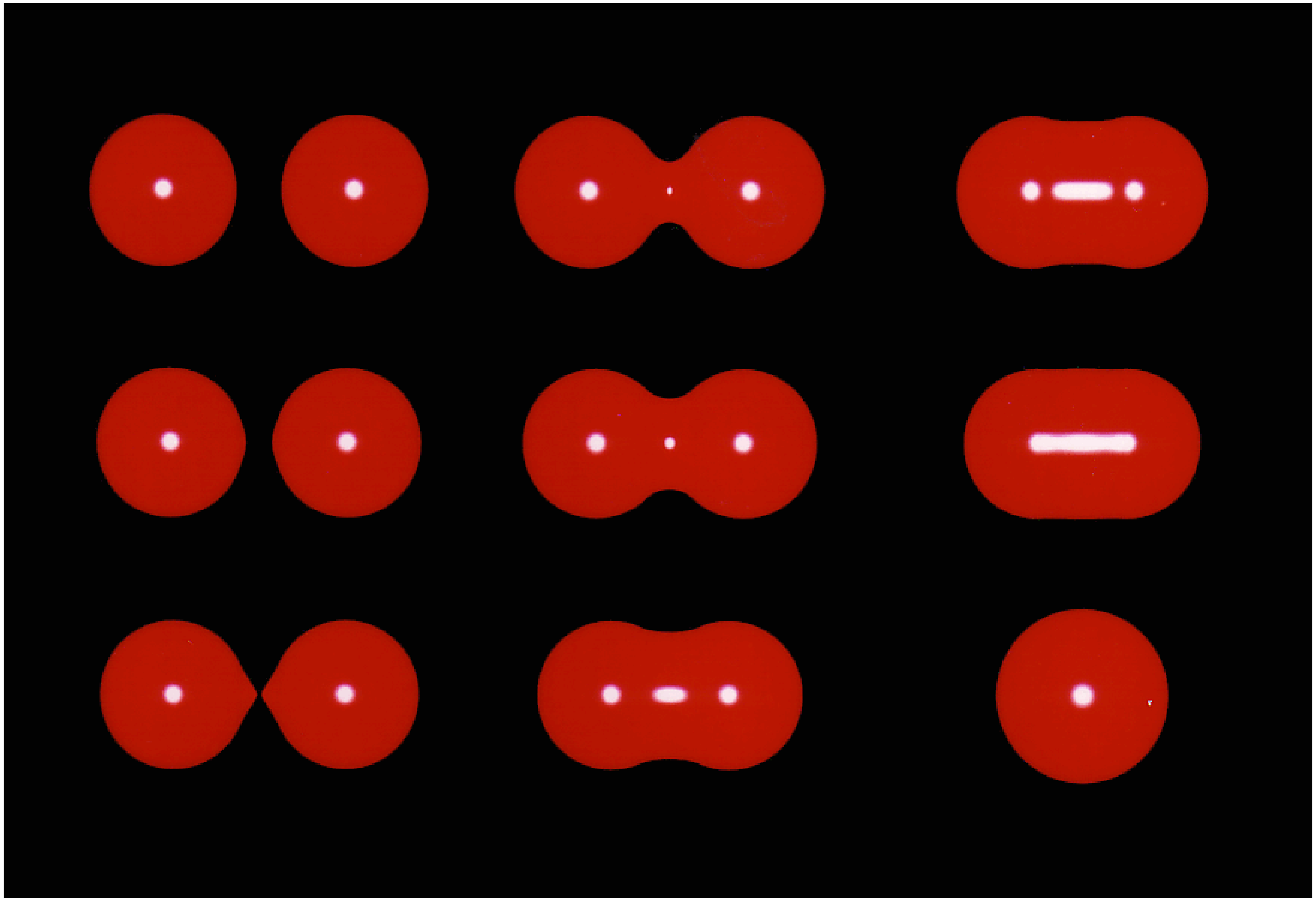
$$F(x) = c$$

    is called **iso-surfaces** (contour surface, level set) S of level (iso value) c.
  - S is determined by this equation implicitly.

6.5 Implicit representation

- The shape of the iso-surface can be controlled easily using reasonable center points and simple field functions.
- Next slide:
  - Iso-surface generated by two radial-symmetric field functions.
  - The respective center points are moved towards each other onto the same final position.
  - Fusion-effect, where the two separated iso-surfaces are merged smoothly (here C^1-continuously) into each other, when the distance of the center points becomes small enough.
  - The inverse motion will tear the iso-surface into two separated objects.
- **Advantage:** Topology and topology changes are defined implicitly!

6.5 Implicit representation



6.5 Implicit representation

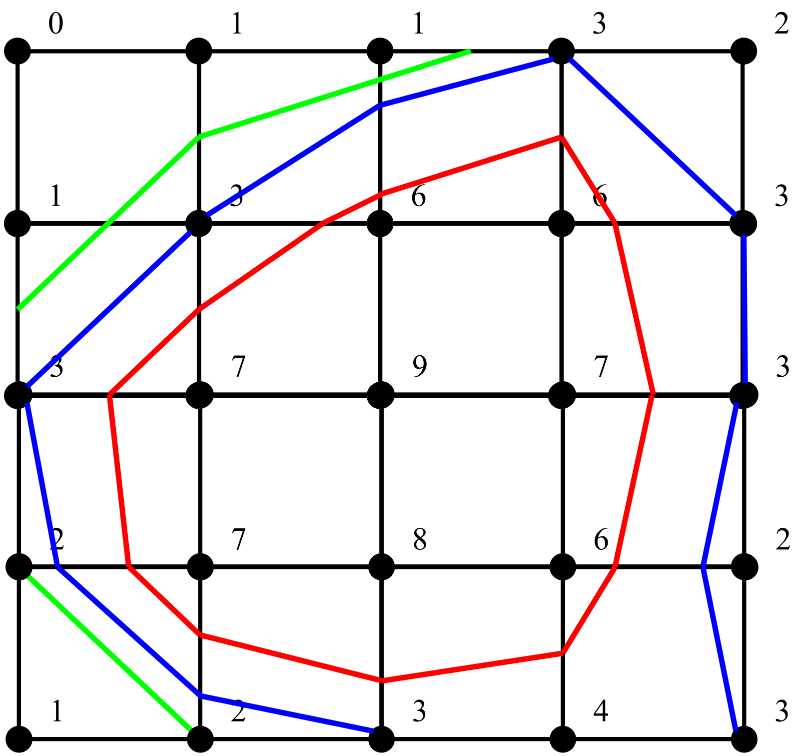
6.5.3 Marching Cubes Algorithm

How to determine $F(x) = c$?

Principle:

- Computation of contour surfaces.
- Extension of marching squares from 2d to 3d.
- Marching Squares: Computation of contour- rsp. iso-lines.
- Input: Discrete „measurements“ on a grid.

$\boldsymbol{F(x) = 2}$
$\boldsymbol{F(x) = 3}$
$\boldsymbol{F(x) = 5}$



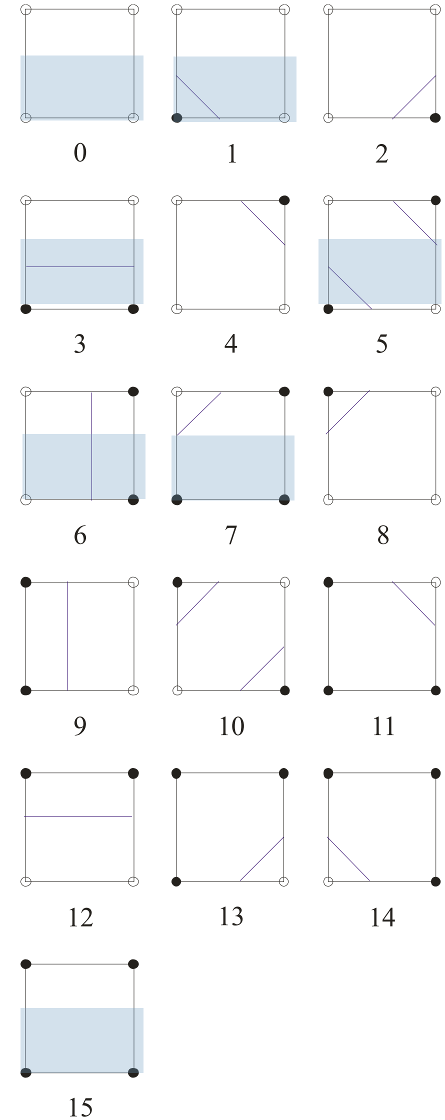
6.5 Implicit representation

Assumption: Contour line in linear inside a cell (Marching-Squares).

1. Choose cell.
2. Determine the state of the cell
   ➡ 4-bit-vector.
3. Lookup-Table
   ➡ Principle course of contour line.
4. Compute intersections with cell edges

$$S = (1-\lambda)V_i + \lambda V_j \text{ with } \lambda = \frac{c - f(V_i)}{f(V_j) - f(V_i)}.$$

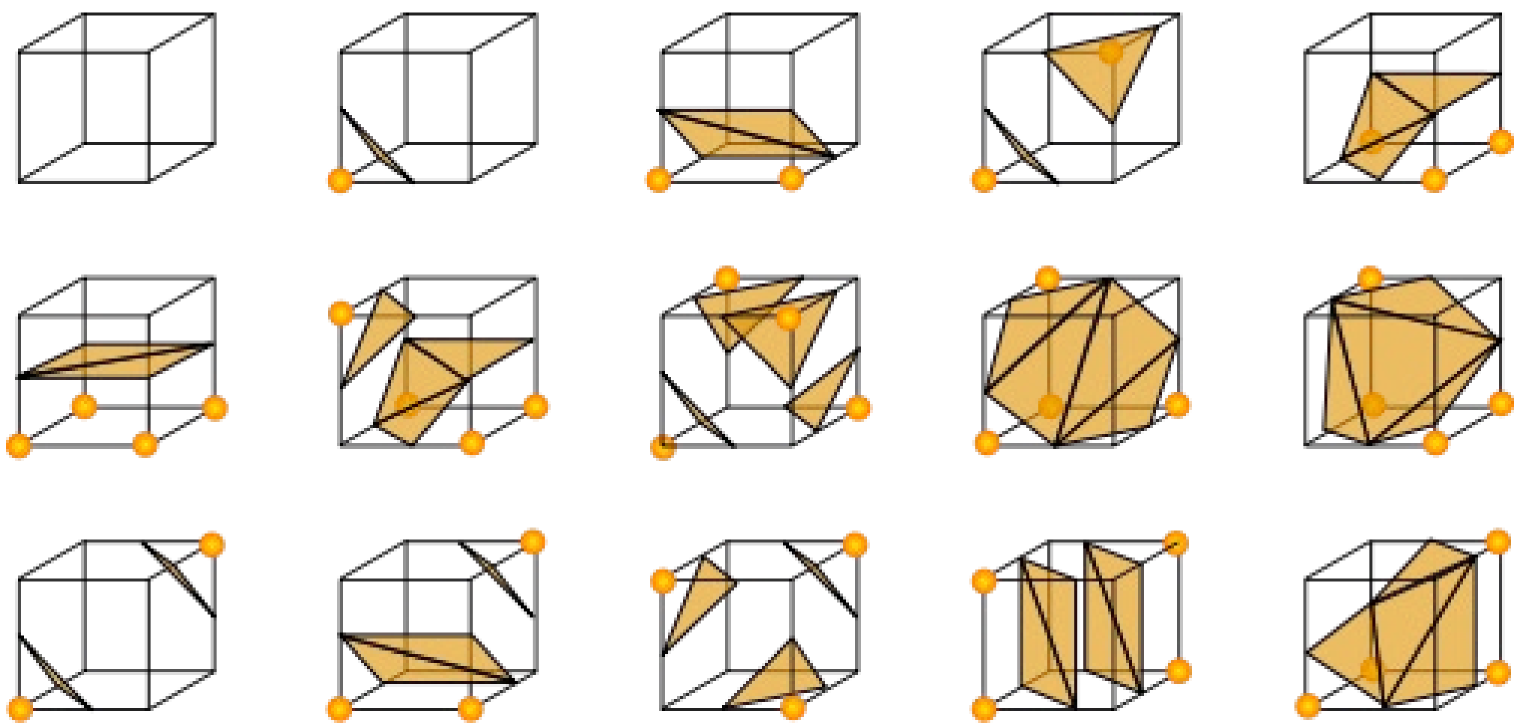
5. Continue wit adjacent cell.



6.5 Implicit representation

Marching Cubes in 3d:

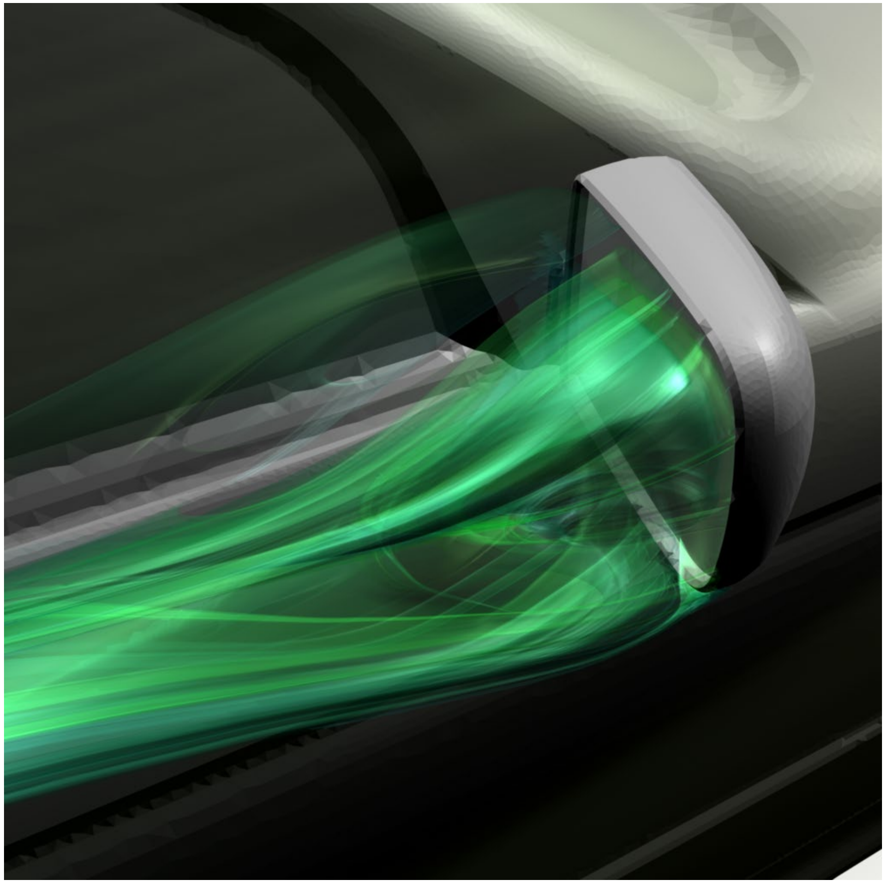
- 15 different unique configurations (up to rotations and reflections)



Source: Wikipedia

6.5 Implicit representation

Application: Flow simulations



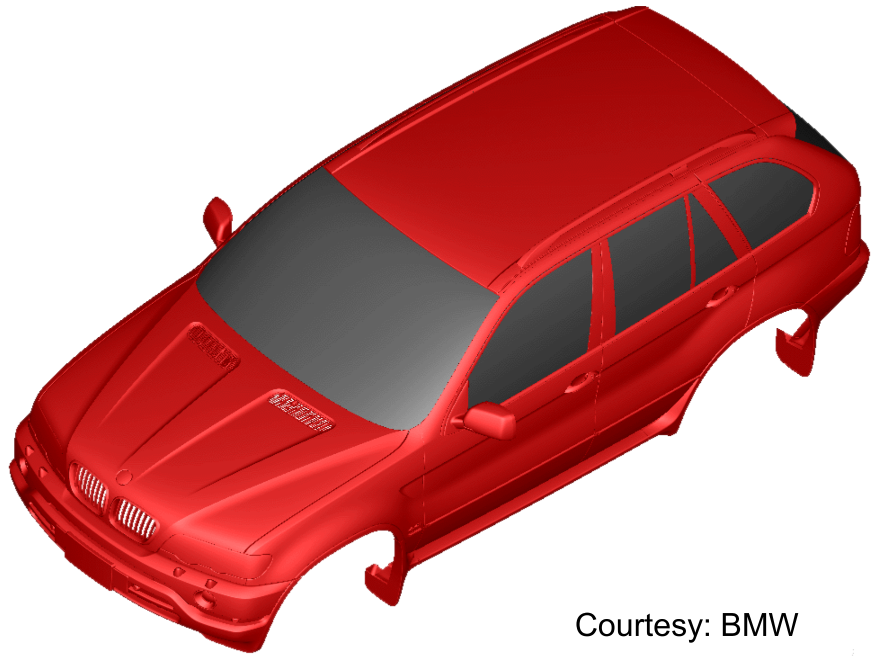
Courtesy: BMW/Garth

6.6 Parametric representation

Parametric curves and surfaces are the most popular representation in CAD besides polygonal representations.

➡ **Lecture „Geometric Modeling (MSI)“.**

Courtesy: Jaguar



Courtesy: BMW

6.7 Scene management

- For high-quality real-time computer graphics, especially for computer games or virtual reality applications, a

  efficient representation, organization and management of the virtual scene

  is necessary.
- At the same time the basic rendering of individual objects is outsourced to the graphics hardware (GPU).
- The representation and organization of the individual objects of a scene is managed using a hierarchical tree structure, the so-called **scene graph**.
  - **LOD**-methods support the rendering of complex scenes and highly detailed objects.

6.7 Scene management

- Efficient organization and maintenance of a scene is called

  scene management.

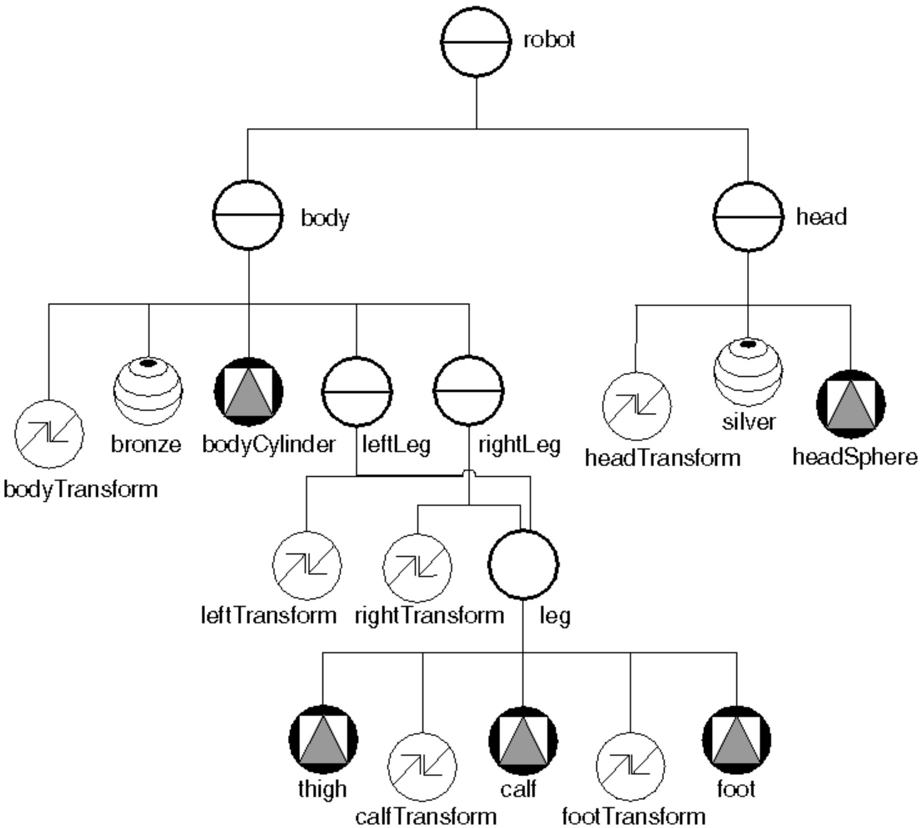
- Combination and organization of individual objects of a scene is stored in a hierarchical tree structure, the so-called

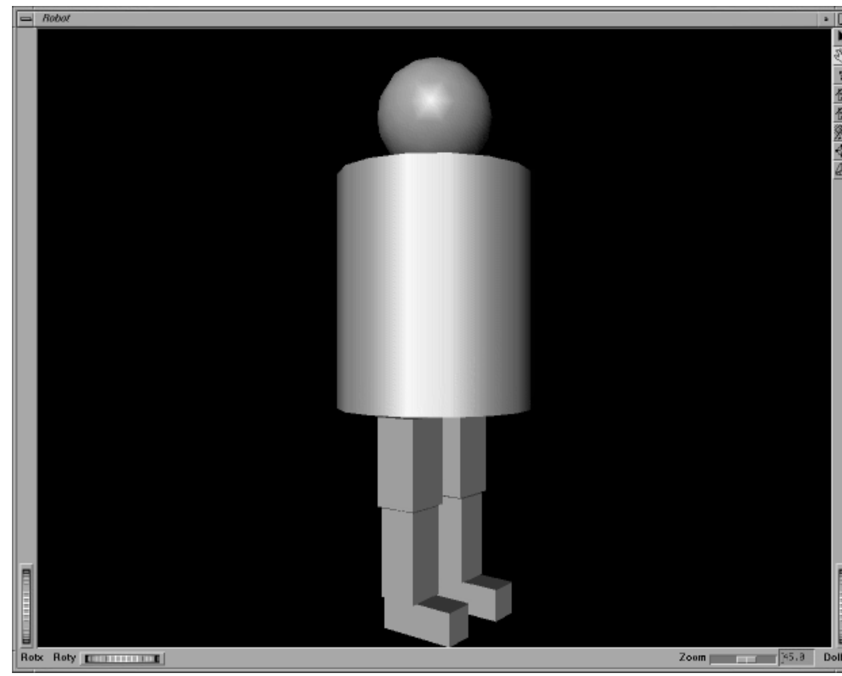
  scene graph.

- In particular the scene graph stores information about:
  - shapes (form, geometry, appearance (material, texture, etc.)),
  - groups of objects,
  - transformations, positions, orientations, etc.,
  - light sources, background, fog, atmospheric scattering, etc.,
  - view definitions, cameras, etc.,
  - behavior (e.g. in animations), etc.,
  - application specific attributes, sound, etc.

6.7 Scene management

Example: Open inventor-scene-graph





Goals

- What is a BRep?
- What is the representation hierarchy for polygonal models?
- Why do edges in polygonal meshes need special attention/treatment?
- What approaches do you know to generate polygonal objects?
- What is a CSG-representation and what is stored in the data structure?
- What are examples for space partition data structures?
- How can the (hyper-)planes for BSP-trees determined?
- What is the principle of implicit representation?
- What is the principle of the marching-cubes-algorithm?
- What is a scene graph?