Computer Graphics: 1-Modeling

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

Geometric Primitives

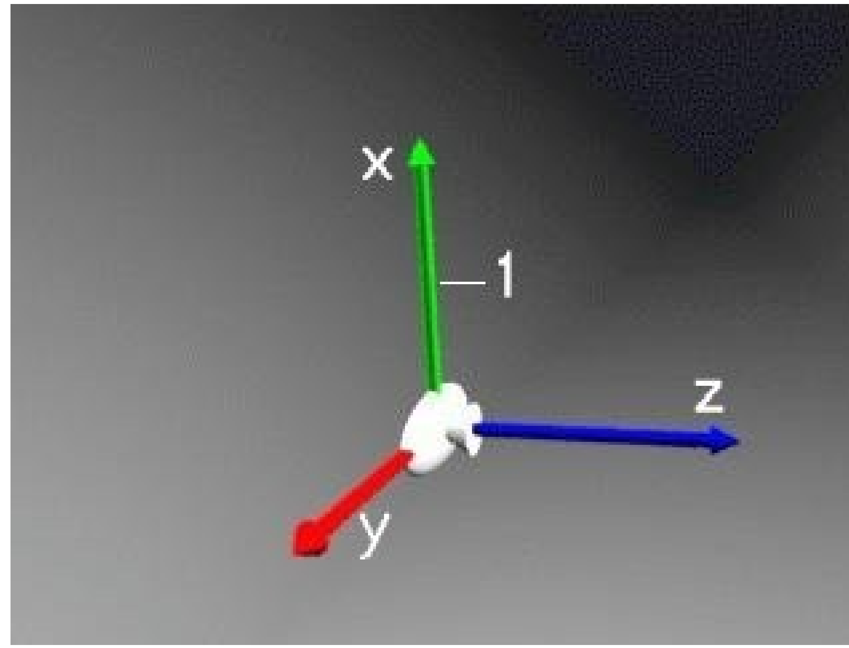
Lesson 1

Models and Coordinate spaces

- In the beginning....  .... there was an idea...
- Modeling an idea means making it understandable for a computer
- In Computer Graphics, models are generally
  - 3-dimensional AND
  - Include Color Modeling
  - For animation they also include the modeling of movement
- In this course, we shall limit ourselves to 3D models

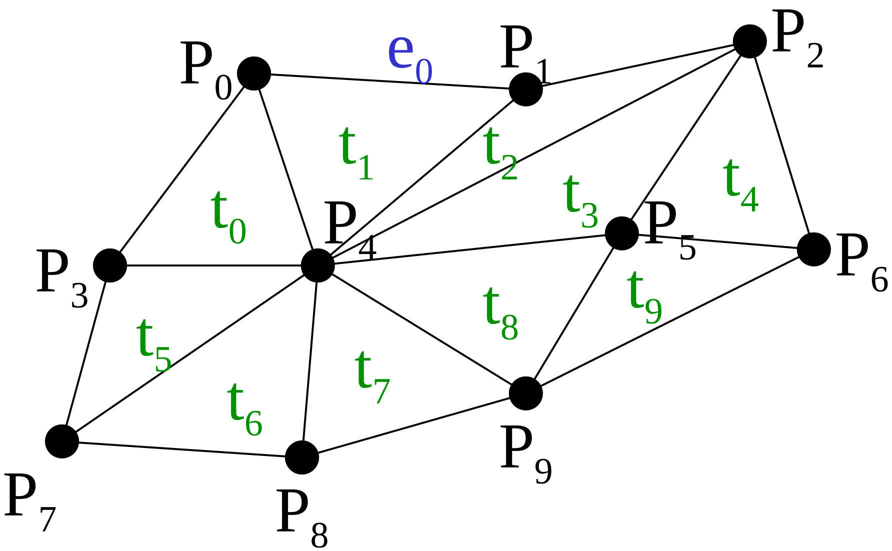
Creating a 3D space to work with

- The idea here is to be able to represent three-dimensional objects in a computer
- The first thing necessary, of course, is to define a proper 3D space for it:
  axes and the units
- Right handed axes
- Units same on all axes

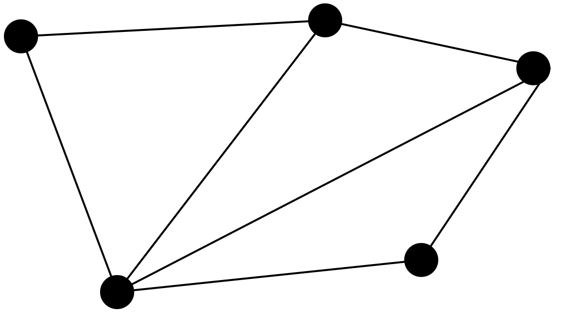


Adding elements to the space

- Points in space have three coordinates P(x,y.z)
- Two points P_1P_2 build a segment, which form a triangle *edge* e
- In Computer Graphics, objects are generally represented as triangle *meshes*
- A mesh is a set of contiguous triangles t_i
- If the triangles of the mesh have one vertex in common the set is called a triangle *fan*

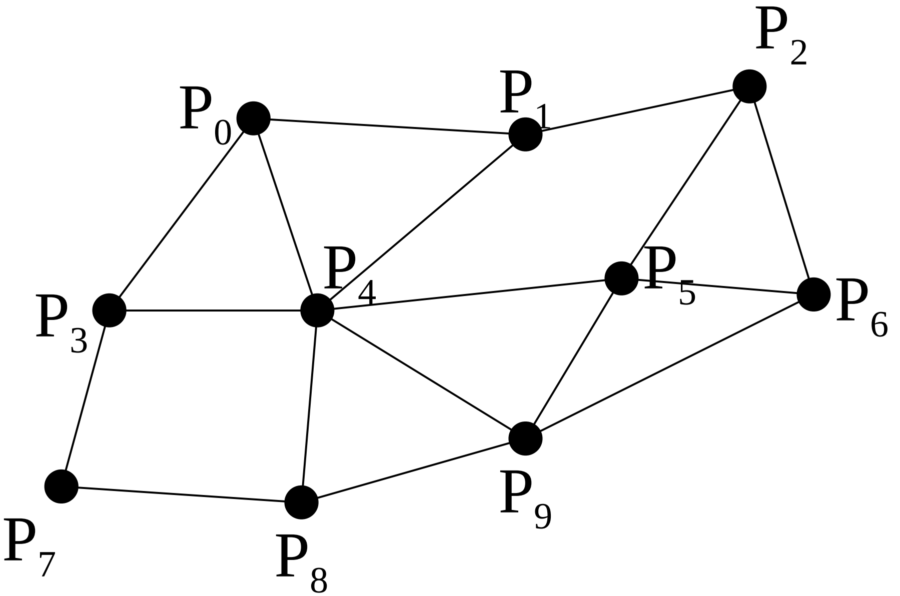


A triangle mesh


A triangle fan

Adding elements to the space

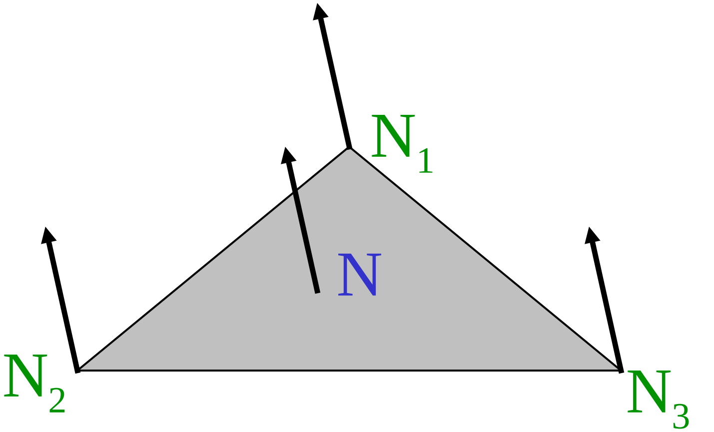
- Of course, triangles are not the only possible basic element of a 3D geometry
- One can have more complex polygons, like quadrangles of polygons with a higher number of edges
- Whereby, one must recall that polygons are FLAT
- Hardware reduces everything to triangles anyhow



A generic mesh

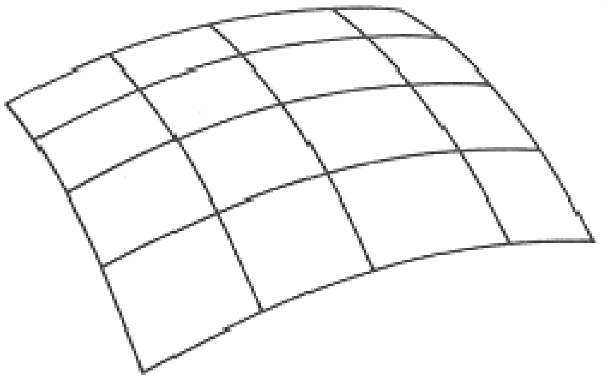
Normals

- For each polygonal element of the 3D model, attributes are added
  - Normal to the surface containing the polygon
  - Colour of the element
- Sometimes, instead of having ONE normal N for a polygon, a normal N_i is assigned to each of its vertices
- This is necessary for illumination computations

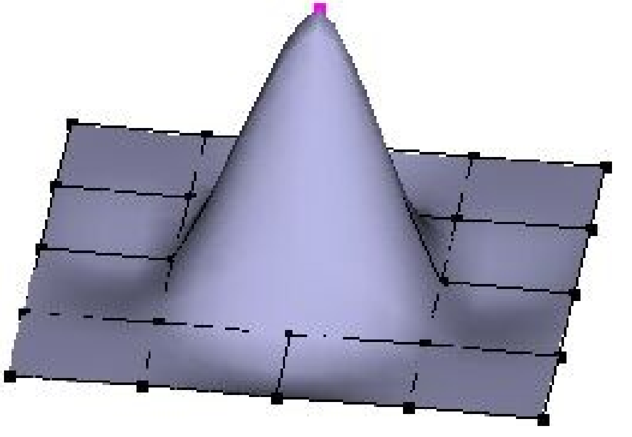


Higher order representation

- Another way to representing surfaces is to use instead of linear functions (=polygons) higher order functions joined suitably at the edges
- Spline patches do exactly this: the object is represented by piecewise defined „patches“ joined at their definition edges so that they are continuous at the joins, like a „patchwork“
- Splines are very flexible in shape modeling
- But what is behind spline patches?



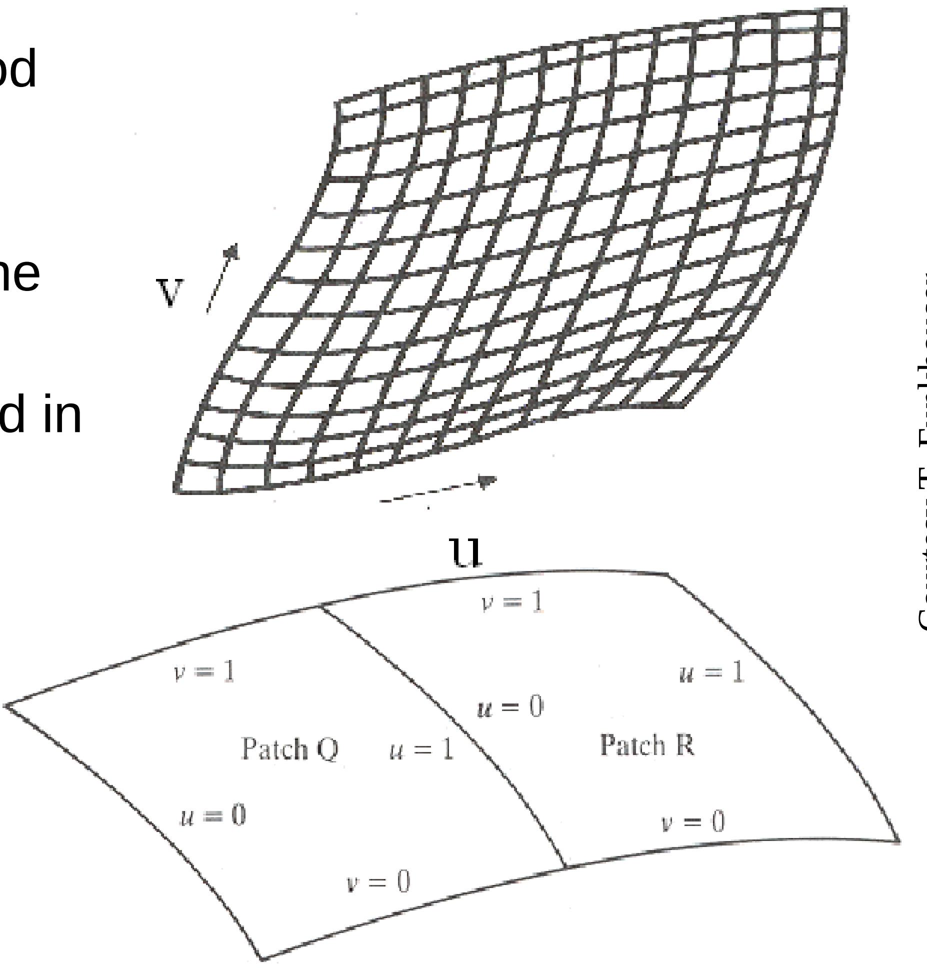
Courtesy T. Funkhouser, Princeton University



Courtesy Russian Academy of Sciences

BRep representation: patches

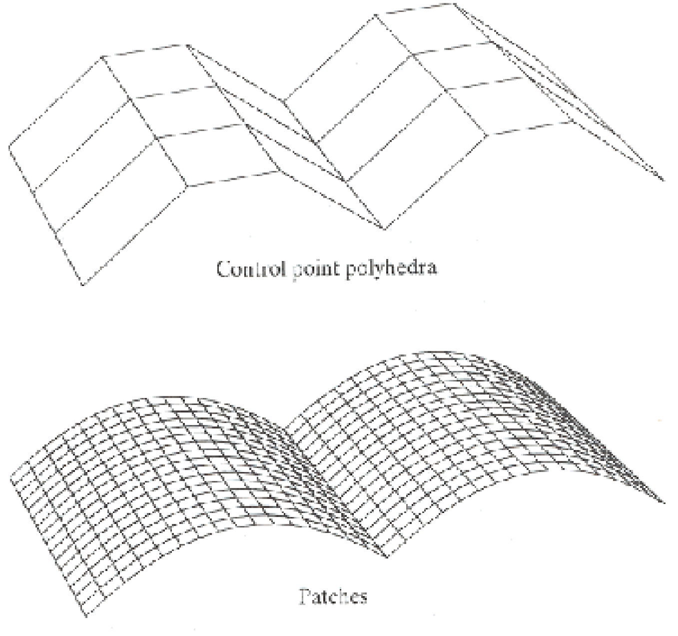
- The idea is to find families of piecewise parametric functions that allow a good control on shape
- Patches are joined at the edges so as to achieve the desired continuity
- Each patch is represented in parametric space



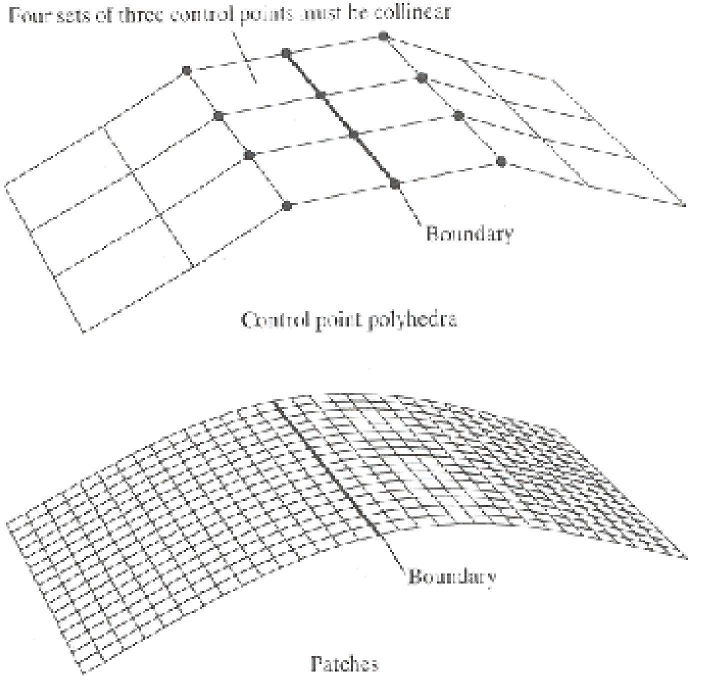
Courtesy T. Funkhouser, Princeton University

BRep representation: patches

- C^0 continuity



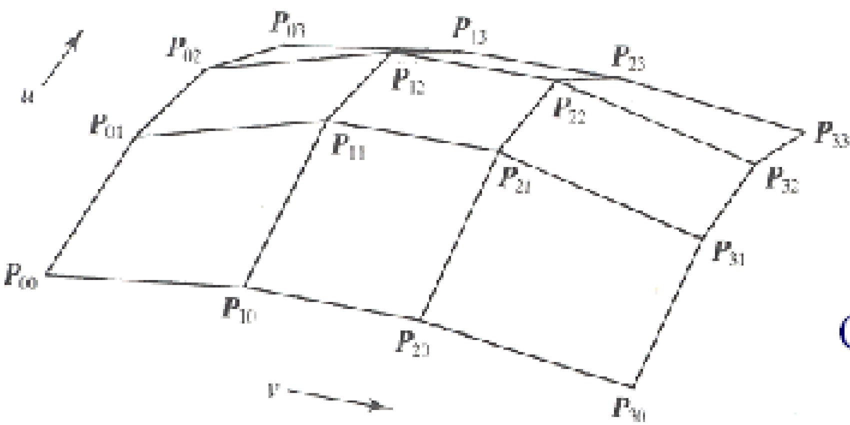
- C^1 continuity

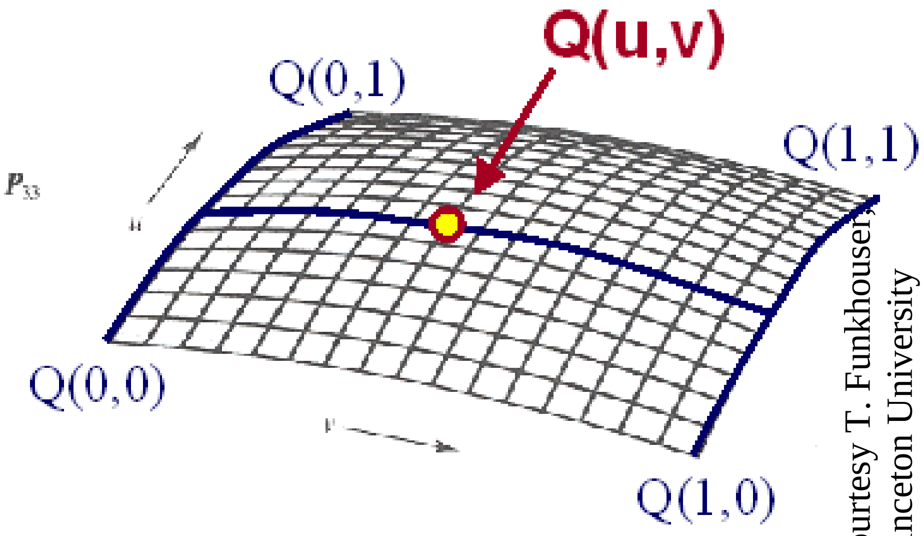


Courtesy T. Funkhouser,
Princeton University

Spline patches

- A point Q on a patch is the tensor product of parametric functions defined by control points





Courtesy T. Funkhouser,
Princeton University

Spline patches

- A point Q on any patch is defined by multiplying control points by polynomial blending functions

$$Q(u,v)=UM\begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix} M^T V^T$$

$$U=[u^3\ u^2\ u\ 1]$$

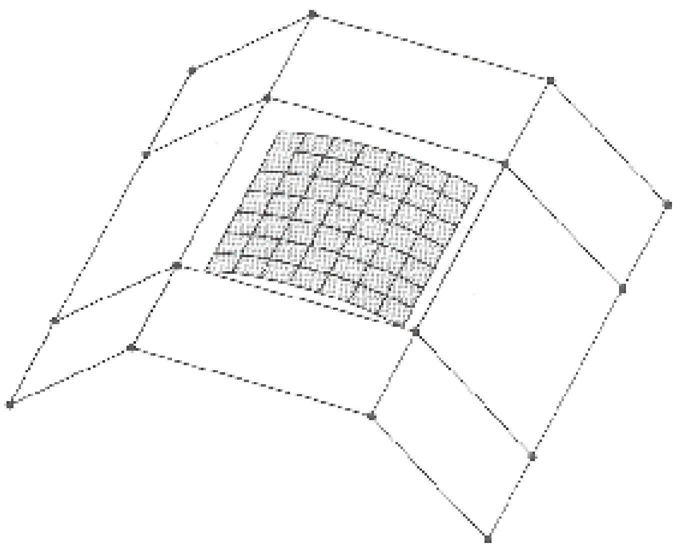
$$V=[v^3\ v^2\ v\ 1]$$

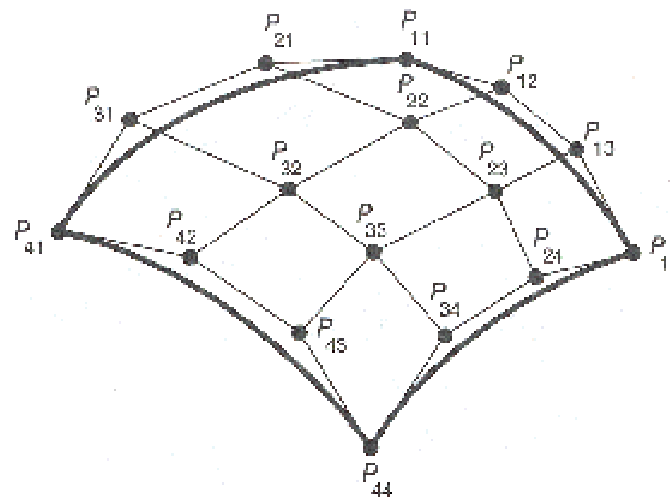
- What about M then? M describes the blending functions for a parametric curve of third degree

Spline patches

$$M_{B-spline} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 1/2 & -1 & 1/2 & 0 \\ -1/2 & 0 & 1/2 & 0 \\ 1/6 & 2/3 & 1/6 & 0 \end{bmatrix}$$

$$M_{Bezier} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

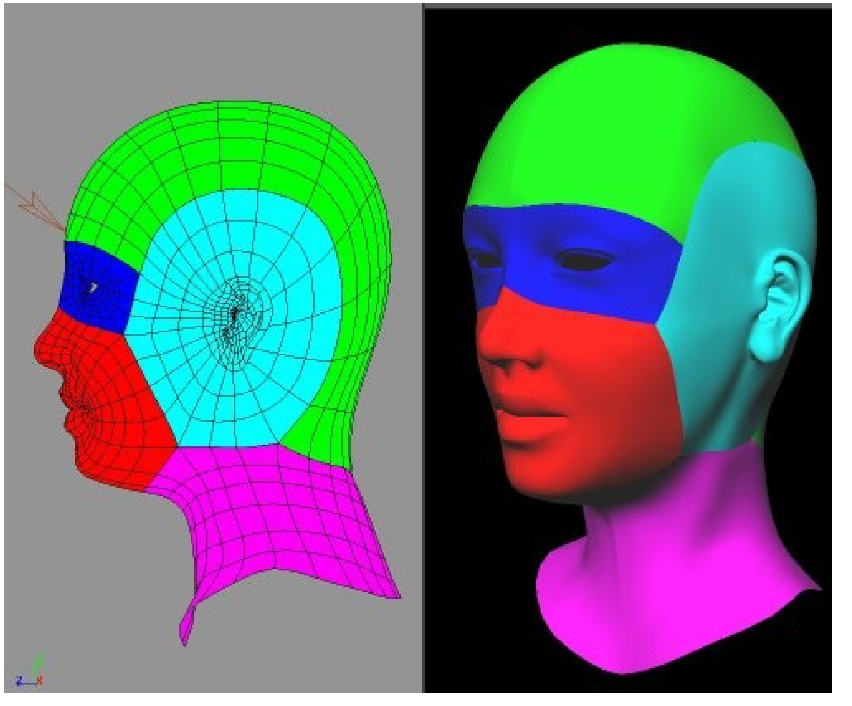




Courtesy T. Funkhouser, Princeton University

Spline patches

- Third order patches allow the generation of free form surfaces, and easy controllability of the shape
- Why third order functions?
  - Because they are the minimal order curves allowing inflection points
  - Because they are the minimal order curves allowing to control the curvature (= second order derivative)



Courtesy Softimage Co.

Basic transformations (2D)

- In the modeling process, it is important to be able to apply to objects in space transformations.
- Most important transformations:
  - Translation of a point P: P´=T+P
  - Rotation of a point P: P´=R · P
  - Scaling of a point P: P´=S · P
  - Where (in 2D):

$$R = \begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix} \qquad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Basic transformations (2D)

- Problem is that translation has to be treated differently
- The solution is to use homogeneous coordinates:

  [x y] →[x y 1]

  [a b c] →[a/c b/c]
- What we have done, is basically adding a third coordinate representing infinity
  - (when c → 0, the other two coordinates become big)

- This is called *projective geometry space*, and the new coordinates are called *homogeeous coordinates*

- Translations can be seen as rotations around the infinity, because a the circumference of a circle of infinite radius is a straight line

Basic transformations (2D)

- With homogeneous coordinates, the transformations become 3x3 matrices applied to the single point coordinates

  $$P' = M \cdot P$$

  where M is one of the following matrices

$$T = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 \\ \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Such transformations can be concatenated to obtain complex transformations.
- Concatenate means apply one after the other one, which is done by multiplying the correspondent matrices

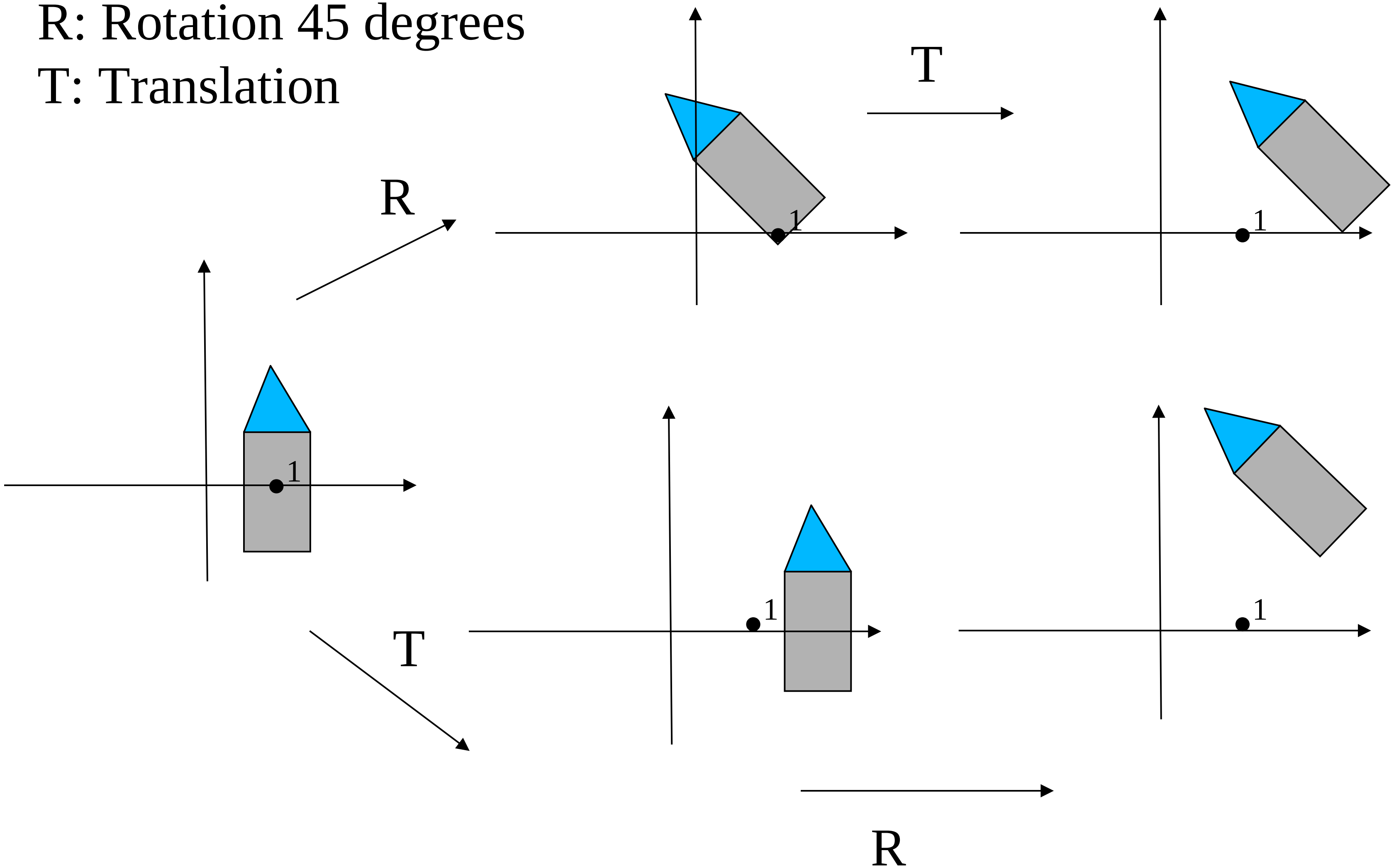
  $$P' = M_1 M_2 \ldots M_n \cdot P$$

- CAUTION! Matrix multiplication is NOT commutative!

Example

R: Rotation 45 degrees

T: Translation



Basic transformations (3D)

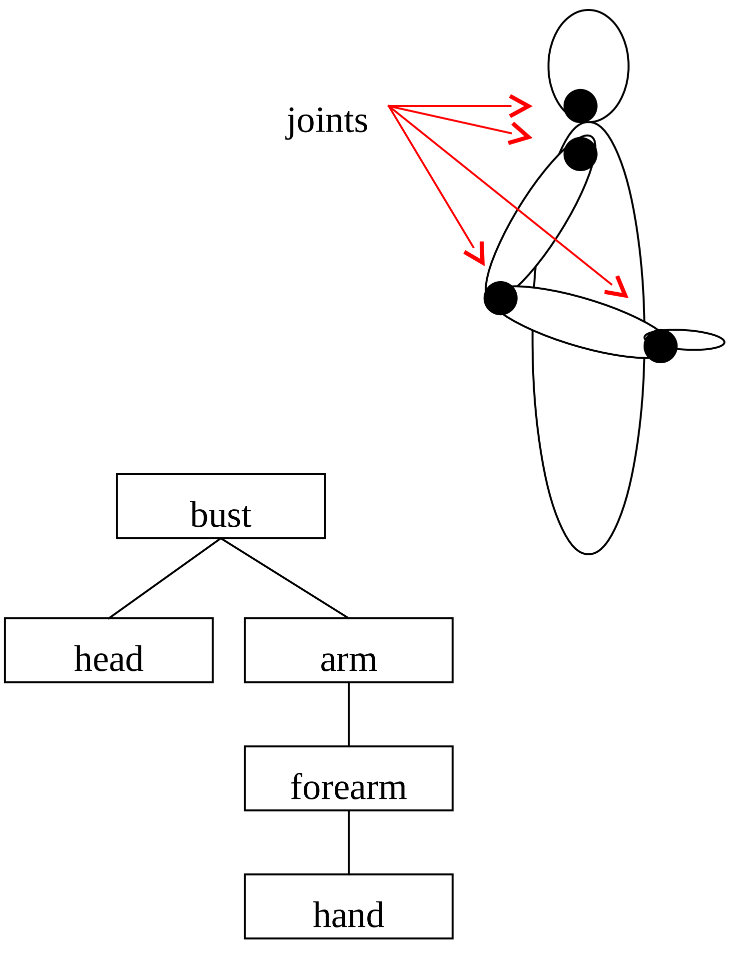
- In 3D, the math is similar:
  [x y z] →[x y z 1]
  [a b c d] →[a/d b/d c/d]

$$T = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\vartheta) = \begin{bmatrix} \cos\vartheta & -\sin\vartheta & 0 & 0 \\ \sin\vartheta & \cos\vartheta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\vartheta) = \begin{bmatrix} \cos\vartheta & 0 & \sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_x(\vartheta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta & 0 \\ 0 & \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

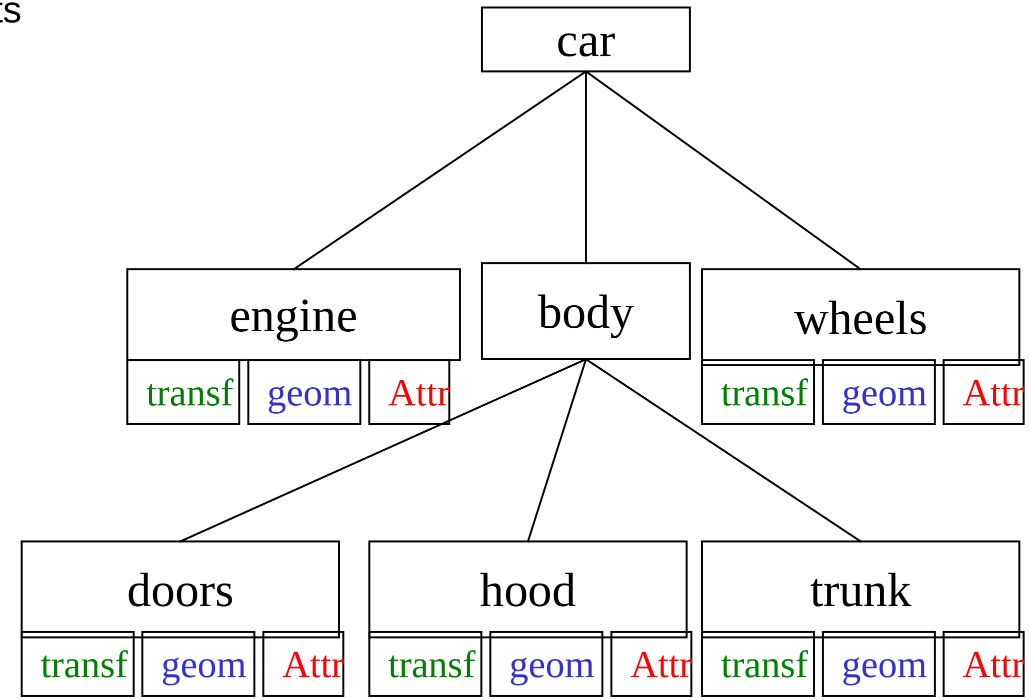
Hierarchical objects

- Of course it is not always practical to have a flat polygonal structure for your 3D world
- Scenes are usually structured in an object oriented hierarchical way
- The object is represented like a tree.
  - One of its parts is chosen as root, and is represented in global coordinates
  - The other elements are represented as children moving in the local in the local coordinate system of the parent
- This is done by matrix multiplication



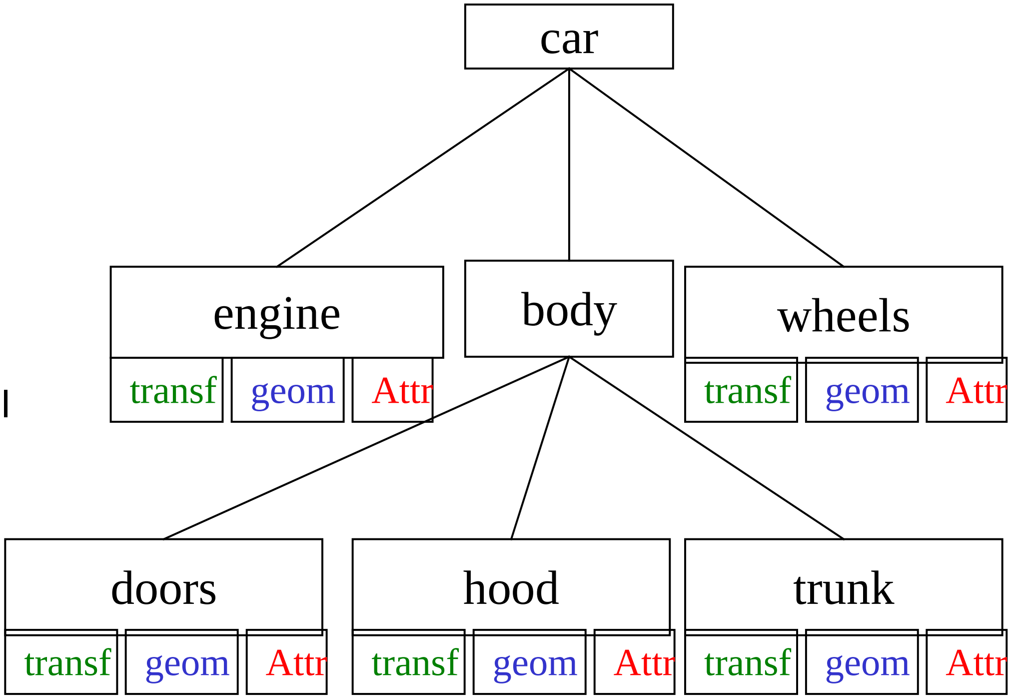
Scene Graphs

- Similarly, in a scene, storing is made hierarchically in a tree
  - Polygons will be grouped into parts of objects
  - Parts of objects into objects
  - Objects into group of objects
  - Group of objects into a scene
- Each node of the scene graph will have
  - its transformation matrix WRT parent
  - geometry (point coordinates)
  - attributes (colour, transparency, texture, …)
- Attributes can be inherited from the father node



Traversing Scene Graphs

- Drawing is done by traversing the tree
- For traversing, different techniques can be used
  - Start from one node (usually root)
  - Move downwards left, multiplying transformations (and inheriting attributes), and apply rendering
  - Until leaf is reached
  - Retrace back, undoing transformations and attributes, until first unprocessed child
  - Move down and leftmost….
  - Until whole tree is processed



End

+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++