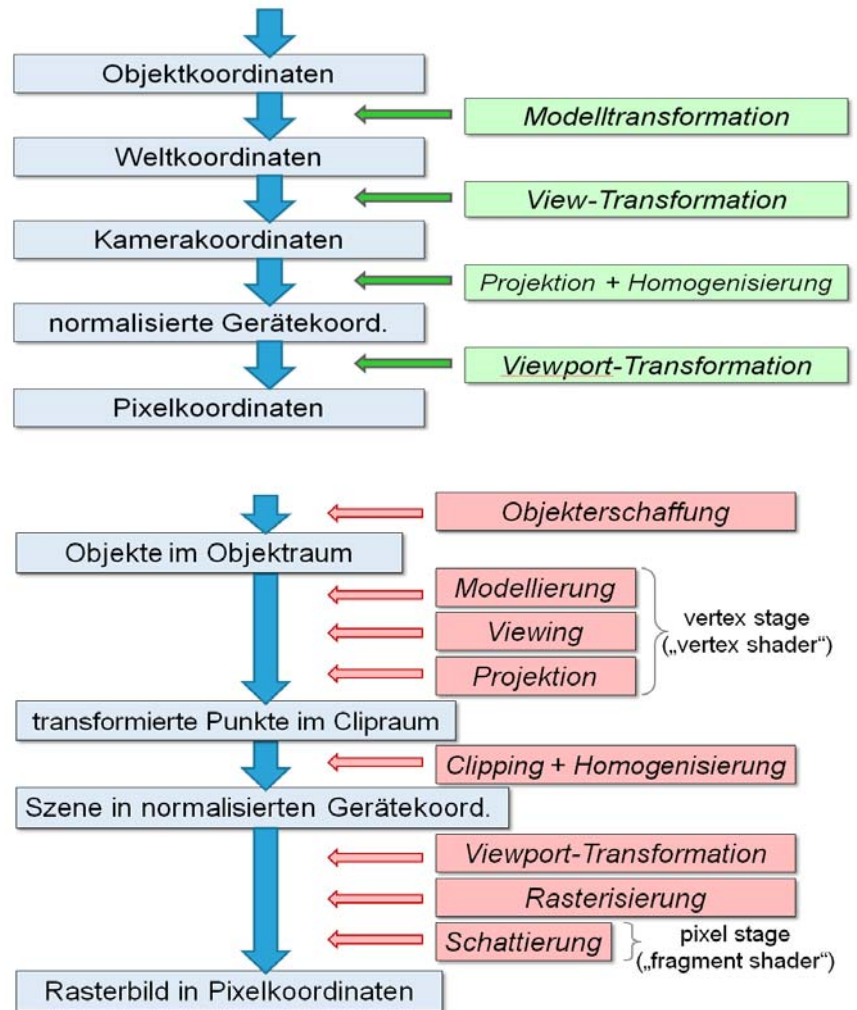
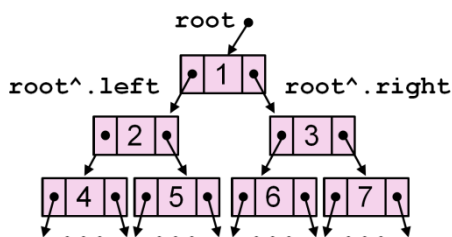


# Graphikpipeline

Objekterschaffung kann prinzipiell durch Modellierung oder durch Scanning erfolgen. Die Begriffe „vertex shader“ und „fragment shader“ werden für programmierbare Teile von Graphikkarten verwendet. Man beachte, dass es viele weitere ähnliche Abbildungen gibt, die alle das gleiche Grundprinzip der Graphikpipeline beschreiben. Die meisten weiteren Graphik-Kapitel dieses Skriptums lassen sich direkt in dieses Schema einordnen.



Ähnlich wie verkettete Listen lassen sich allgemeine Graphen, und im Speziellen auch Bäume, mit zeigerverketteten Strukturen repräsentieren. Die einzelnen Knoten müssen ausreichend viele Zeigerkomponenten enthalten, so dass die gewünschte Struktur erzielt werden kann, z.B. zwei Zeiger pro Knoten für einen binären Baum.

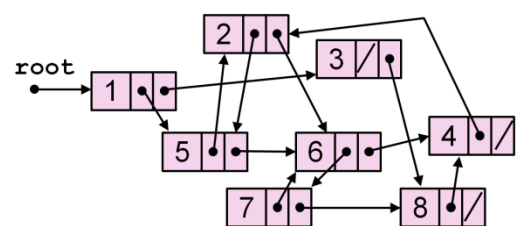


mit Zeigern implementierter binärer Baum

Operationen zum Einfügen  
und Entfernen von Knoten  
kann man sich analog zu  
den verketteten Listen

leicht überlegen. Weiters sind  
solchen Datenstruktur durch  
rekursive Abarbeitung eines

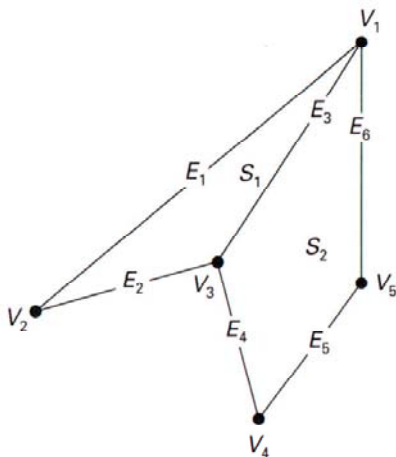
- „inorder“: 1. linker Nachfolger, 2. Wurzel, 3. rechter Nf. (→ 4251637)
- „postorder“: 1. linker Nachfolger, 2. rechter Nf., 3. Wurzel (→ 4526731)
- „preorder“: 1. Wurzel, 2. linker Nachfolger, 3. rechter Nf. (→ 1245367)



allgemeiner, mit Zeigern implementierter Graph

Dreidimensionale Objekte werden meist durch Polygonlisten repräsentiert (oft Dreiecke). Eine Menge von Polygonen, die die Oberfläche eines Objektes beschreibt, nennt man *Boundary-Representation* („B-Rep“).

Datenstrukturen für B-Reps enthalten neben geometrischer Information auch Attribute (Eigenschaften). Die Geometrie besteht aus Punktlisten, Kantenlisten, Flächenlisten und muss auf Konsistenz und Vollständigkeit überprüft werden.



Das Beispiel zeigt für eine ganz einfache Situation mit 2 Polygonen, wie die Punktliste (Vertex Table), Kantenliste (Edge Table) und die Flächenliste (Surface Table) sich gegenseitig referenzieren.

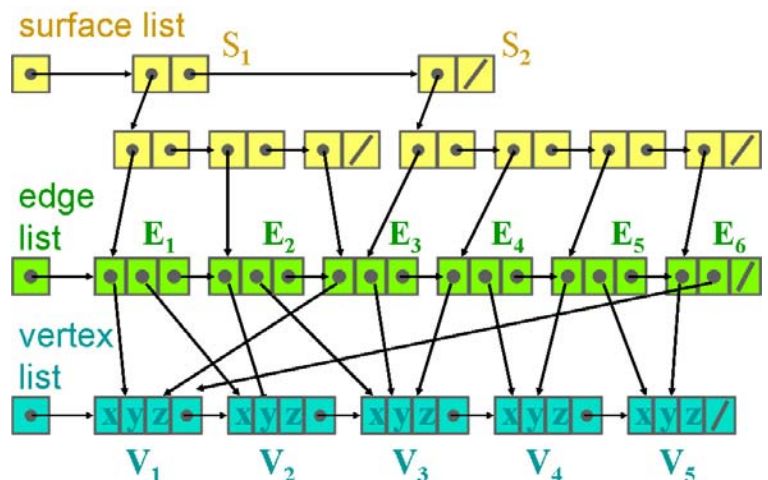
| VERTEX TABLE |                 |
|--------------|-----------------|
| $V_1$ :      | $x_1, y_1, z_1$ |
| $V_2$ :      | $x_2, y_2, z_2$ |
| $V_3$ :      | $x_3, y_3, z_3$ |
| $V_4$ :      | $x_4, y_4, z_4$ |
| $V_5$ :      | $x_5, y_5, z_5$ |

| 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 |                      |
|-----------------------|----------------------|
| $S_1$ :               | $E_1, E_2, E_3$      |
| $S_2$ :               | $E_3, E_4, E_5, E_6$ |

Dieselbe Struktur lässt sich auch mit Zeigerlisten darstellen (siehe Abbildung)

Die gesamte Koordinateninformation befindet sich in den Punktknoten (V steht für Vertex). Wenn ein Punkt an einen neuen Ort transformiert wird, so reicht es, die Koordinaten dieses Punktes zu verändern. Die Topologie bleibt dabei erhalten. Die lineare Verkettung der Kanten und Punkte erleichtert die Bearbeitung (z.B. alle Kanten einmal zeichnen, oder alle Punkte verschieben).



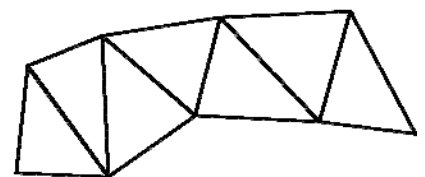
**Ein paar Begriffe:** Die Repräsentation jeder einzelnen Polygonfläche inkludiert die Trägerebene  $Ax + By + Cz + D = 0$  und die Eckpunkte  $V_1$  bis  $V_n$ . Aus den Ebenenparametern  $A, B, C, D$  erhält man sofort den Normalvektor auf die Ebene ( $A, B, C$ ). Als *Backface* bezeichnet man die Rückseite des Polygons, die also in das Objekt hinein schaut, als *Frontface* die Vorderseite, die also die Außenseite des Objektes mitformt. Mit „hinter dem Polygon“ meint man dann alle Punkte, die vom Backface aus sichtbar sind, „vor dem Polygon“ heißt, dass man von dort auf das Frontface sieht.

Wenn man ein rechtshändiges Koordinatensystem vorausschickt und die Eckpunkte jedes Polygons (von vorne betrachtet) im mathematisch positiven Sinn (also gegen den Uhrzeigersinn) anordnet, dann gilt für einen Punkt  $(x, y, z)$

- wenn  $Ax + By + Cz + D = 0$  dann liegt der Punkt **auf** der Ebene
- wenn  $Ax + By + Cz + D < 0$  dann liegt der Punkt **hinter** der Ebene
- wenn  $Ax + By + Cz + D > 0$  dann liegt der Punkt **vor** der Ebene

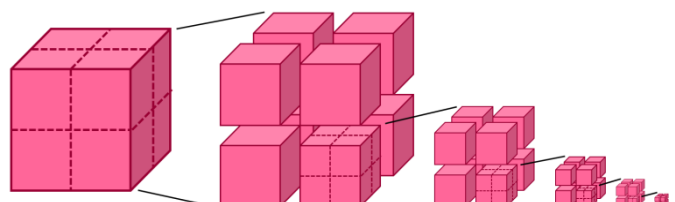
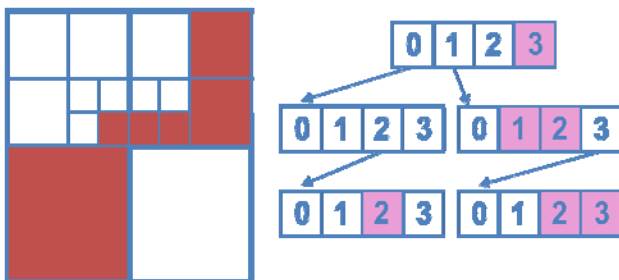
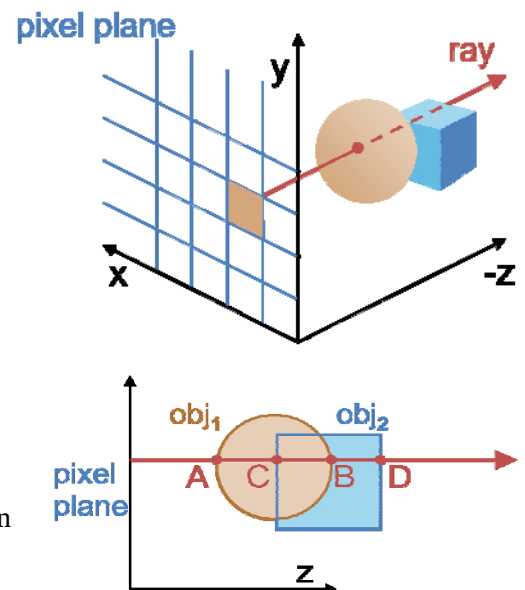
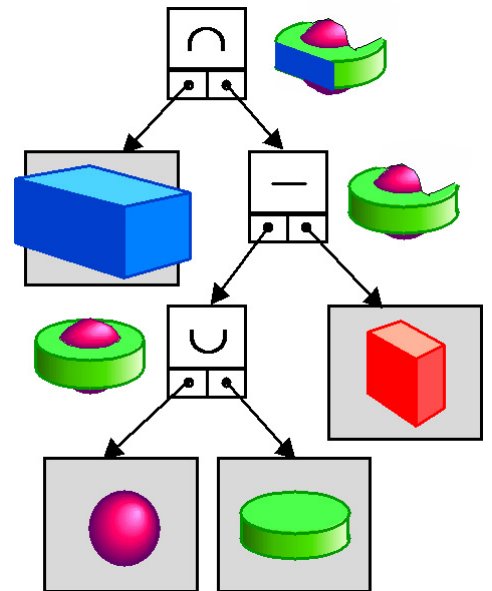
Ebenso lässt sich aus drei aufeinanderfolgenden Eckpunkten  $V_1, V_2, V_3$  ein nach außen gerichteter Normalvektor  $N$  durch  $N = (V_2 - V_1) \times (V_3 - V_1)$  errechnen. Dieses Wissen werden wir später noch brauchen.

Sehr häufig werden ausschließlich Dreiecke als Polygone verwendet, weil sie viele Eigenschaften haben, die die Algorithmen vereinfachen (z.B. sind Dreiecke immer eben). Eine Datenstruktur, die ausschließlich aus Dreiecken besteht, nennt man auch *Dreiecks-Mesh*; eine lineare Abfolge von Dreiecken (jedes weitere Dreieck wird durch nur einen Punkt angegeben!) nennt man *Dreiecks-Strip* (siehe Abb.).



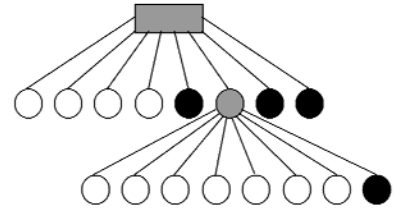
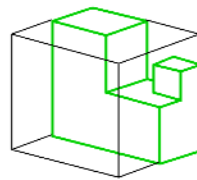
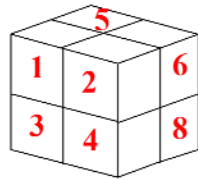
## Constructive Solid Geometry (CSG)

Bei CSG werden Objekte aus dreidimensionalen Primitiven mit Hilfe von Mengenoperationen konstruiert. Sie werden so angeordnet, dass sie in einer hierarchischen Datenstruktur angeordnet werden können, die – obwohl es sich eigentlich nur um einen kreisfreien Graphen handelt – normalerweise CSG-Baum genannt wird. Als Primitive dienen einfach geometrische Formen wie Kugel, Tetraeder, Würfel, Zylinder, die mit den Operatoren Vereinigung,



wird dadurch repräsentiert, dass „einfache“ Teilwürfel (leer oder ganz innerhalb eines Objektes) durch Endknoten beschrieben werden, und kompliziertere Teilwürfel (alle anderen!) in acht kleinere Teilwürfel (Oktanten) unterteilt werden, auf die wieder dieselben Regeln angewendet werden (rekursiv). Dadurch entsteht ein Baum, in dem jeder Knoten 8 Nachfolger hat („Octree“). Man hört mit der Unterteilung auch dann auf, wenn die Teilwürfel eine bestimmte Mindestgröße unterschreiten (z.B. ein Tausendstel der Gesamtgröße); in diesem Fall erhält der Knoten des Baumes die bestmögliche einfache Information. Das passiert zumindest bei allen schrägen Oberflächen irgendwann, und dann müssen diese Randwürfel entweder als innerhalb oder als außerhalb deklariert werden. Das nachstehende Beispiel mit nur zwei hierarchischen Ebenen zeigt auch das Problem der beschränkten räumlichen Auflösung. Die Zeichenkette ist ein Beispiel für eine Linearisierung der Octree-Information, z.B. zum Abspeichern in einer Datei.

Octrees haben den Vorteil, dass man beliebige Formen repräsentieren kann und dass man schnell untersuchen kann, was sich an einer bestimmten räumlichen Position befindet. Dem stehen als Nachteile gegenüber: ungenaue Repräsentation, hoher Speicherbedarf, komplizierte Transformationen. Octrees werden genauso wie Quadrees rekursiv bearbeitet. Mengenoperationen sind so ganz einfach, dafür sind geometrische Transformationen (von Ausnahmen abgesehen) sehr aufwändig, weil der Octree komplett neu generiert werden muss. Das Rendering von Octrees ist dagegen bei Verwendung eines überschreibbaren Speichers einfach:



G(WWWWSG(WWWWWWSS))

```
if Knoten ist einfach
then zeichne Knoten (d.h. tue nichts wenn Knoten leer ist)
else rekursiver Aufruf der 8 Oktanten von hinten nach vorne
```

## Andere Objektrepräsentationen

Es gibt noch eine Fülle von anderen Objektrepräsentationen und zugehörigen Datenstrukturen, die zum Teil für sehr spezielle Objekte und Anwendungen entwickelt wurden. Dazu zählen BSP-Bäume, Fraktale, Graphische Grammatiken und prozedurale Modelle, Partikelsysteme, physikalisch basierte Modelle, dreidimensionale Volumendaten, und einige weitere. Wir werden später noch gekrümmte Oberflächen und Freiformflächen näher betrachten. Die anderen Datenstrukturen werden in weiterführenden Lehrveranstaltungen behandelt.

## Szenengraphen

Als Szenengraph bezeichnet man eine objektorientierte Datenstruktur, mit der die logische und/oder räumliche Anordnung einzelner Elemente einer (zwei- oder) dreidimensionalen Szene hierarchisch beschrieben wird. Dieser Begriff ist nicht genau definiert, es ist eher ein Oberbegriff für alle möglichen hierarchischen Beschreibungsformen für Objekte, die man graphisch darstellen will. Graphentheoretisch handelt es sich um einen baumähnlichen gerichteten kreisfreien Graphen, dessen Wurzelknoten die gesamte Szene repräsentiert, jeder Zwischenknoten beschreibt eine Teilszene (ist ja die Wurzel eines Teilbaumes), und die Endknoten stehen für die einfachsten Objekte der Szene (das können auch sehr unterschiedliche Repräsentationen sein). In OpenSceneGraph, VRML und X3D sind Szenengraphen zentrale Konzepte.

Wir wollen uns als Beispiel eine Stadtszene denken, bei der der gesamte Szenengraph die Stadt repräsentiert. Jedes Haus entspricht einem Zwischenknoten, jedes Fenster entspricht ebenfalls einem Zwischenknoten, allerdings weiter unten im „Baum“, und einfache Objekte wie Fensterscheiben oder Schrauben könnten die einfachsten Objekte sein. Wenn ein Teil öfter vorkommt, dann kann man im Szenengraph die schon vorhandene Information mehrfach nutzen, allerdings braucht man dann die Information, wo und in welcher Lage und Größe diese Teile dann auftreten.

Zwischenknoten enthalten solche Informationen, die den gesamten Teilgraphen betreffen, wie das Material oder die Farbe, die Position und Lage im Raum, die Größe, eine eventuelle Verzerrung. Alle geometrischen Transformationen kann man sehr elegant in Matrizen ablegen, wie das geht werden wir im nächsten Kapitel sehen.

