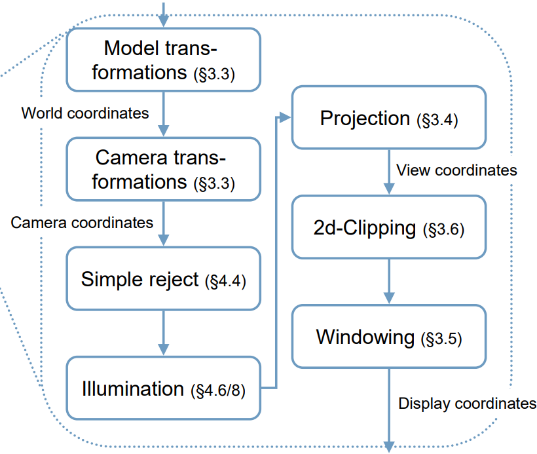
Rendering: Mapping den Geometrische Modellen, Objekten oder der Szene auf ein Bild auf dem Display.

Rendering Pipeline: Konkrete Implementation des Rendering in Soft oder Hardware.

De-Facto Standard Pipeline: Application – Geometry – Rasterization – Fragments – Display

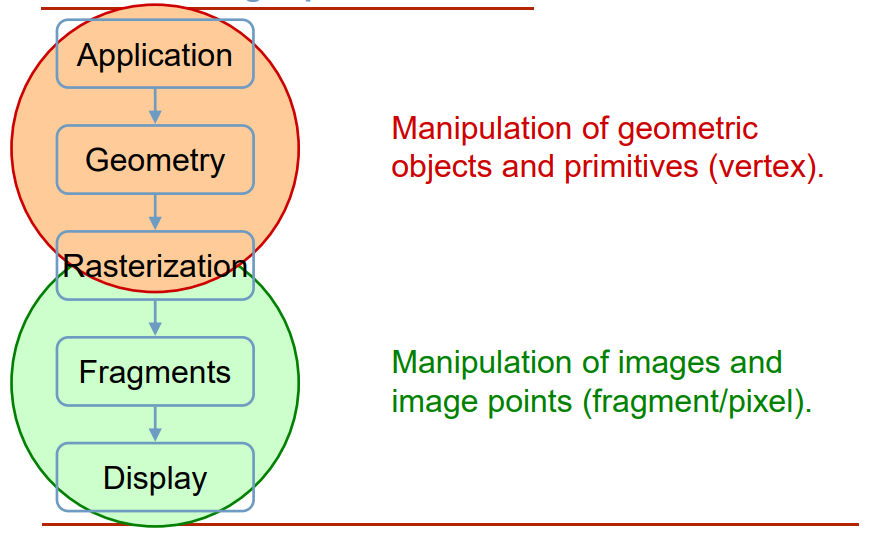
Application

* Receiving and sending of commands and data
* Command buffering/interpretation

Geometry

* Model transformations
* Camera transformations
* Simple reject (Back-Clipping)
* Illumination
* Projection
* 2d-Clipping
* Windowing

Rasterization

* Primitive assembly (Triangles, lines etc)
* Triangle setup (Tiefen/-Farbwerte, Textur Koordinaten)
* Rasterization
* Anti-Aliasing
* Z-Buffer

Fragments

* Transparency
* Texturing

Display

* Frame buffer
* Gamma-correction
* Analog/digital-converter

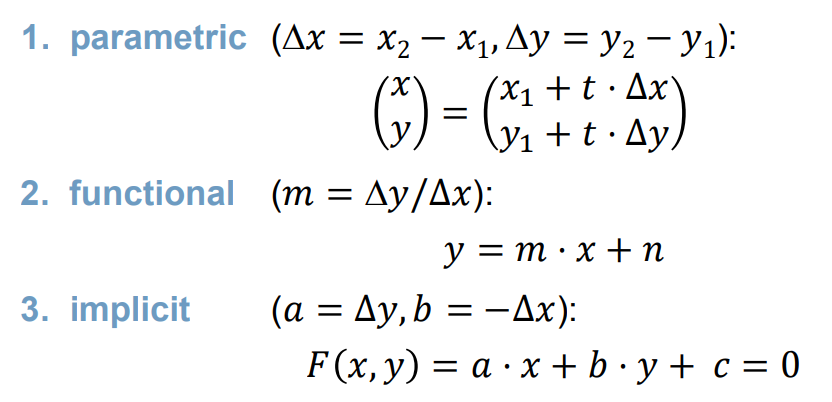
**Kapitel 2 – Rasterization**

Dekomposition von geometrischen 3D Objekten in ein 2D Image mit im Raster angeordneten Pixeln.

Problem: Linie die abgebildet werden soll überlappt die Pixel mal mehr, mal weniger.

Anforderungen:

* Linie muss konstante Dicke, Helligkeit und keine Lücken haben
* Endpunkte muss exakt sein
* Berechnung muss schnell sein, Algorithmus muss sich in Hardware implementieren lassen.

1. Stütz + Richtungsvektor + Parameter t
2. Steigung m \* X-Wert + Y-Offset n
3. Nach Y umstellen, oder Punkt einsetzen

zum Prüfen

**Naive algorithm**:

Probleme:

Floating point values 𝑦 and 𝑚,

divisions and multiplications,

rounding,

vertical lines,

appearance of lines depends on the slope 𝑚.

Use functional representation: 𝑦 = 𝑚 \* 𝑥 + 𝑛

1) Iterate from 𝑥1 to 𝑥2 in pixel-steps

2) Compute 𝑦-values

3) Round

4) Draw

**Bresenham Algorithmus:**

Aufteilung in Oktanten, nur im 1. Zeichnen. Bei Linie in anderen Oktanten die Gerade durch Transformation in den 1. Verlegen.

Entscheidungsvariable d gibt an, ob Mittelpunkt zwischen nächsten Pixelkandidaten (NE/E) über oder unter der Linie liegt. ( d >= 0 => Mittelpunkt unter Linie, d < 0 => Mittelpunkt über Linie)

Initialisierung: Umformulierte Gleichung, die prüft, ob Steigung > oder < 0.5 ist. \* 2 um Division zu vermeiden.

**Bresenham Algorithmus für Kreise:**

Diesmal im 2. Oktanten, andere Punkte extrapolieren.

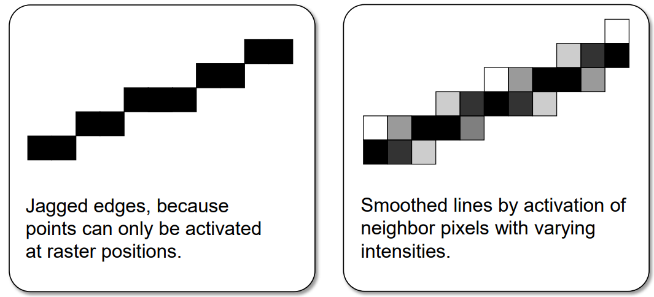
Ähnliches Vorgehen, mit Kreisgleichung prüfen, ob Punkt innerhalb oder Außerhalb vom Kreis liegt.

Implizite Gleichung: 𝐹(𝑥, 𝑦) = (𝑥 – 𝑥𝑀)2 + (𝑦 – 𝑦𝑀)2 − 𝑟2 = 0

Initialisierung von d: d = 5 - 4 \* r;

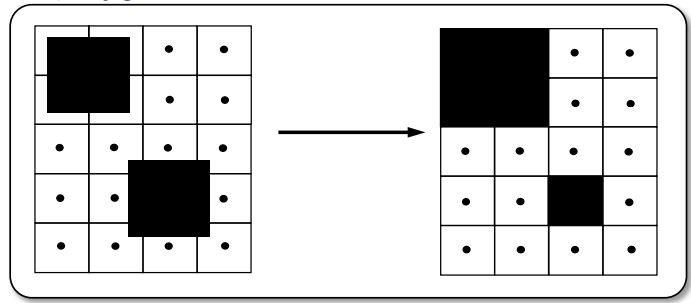
**Aliasing**

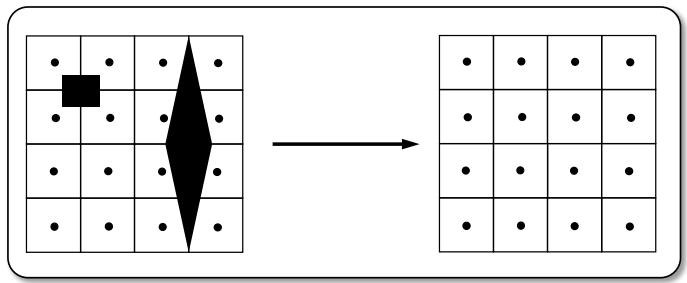
Tritt auf wenn die Abtastfrequenz im Vergleich zum abgetasteten Signal zu klein ist. (Abtastfrequenz = Pixel)

Visual Effects: Aliasing Effekte, ausgelöst durch undersampling. Objekte werden mit niedrigerer, falscher Frequenz dargestellt.

Visual Artifacts (Spatial Aliasing):

* Jagged edges/jaggies

Falsche Größe, Verschwindende Objekte:



Temporal Aliasing: Rad eines Autos scheint während der Fahrt stillzustehen

Anti-Aliasing Techniken:

Edge Smothing: Anti-Aliasing Methoden, die durch Rasterisierung hervorgerufene Artefakte bereinigen

Super-Sampling: Pixel wird in Sub-Pixel unterteilt. Die tatsächliche Pixelfarbe richtet sich nach dem Durchschnitt der Sub-Pixel.

Gewichtete/Ungewichtete overlap area: Overlap von Line mit Pixeln berechnen => Grauwert stat schwarz/weiß

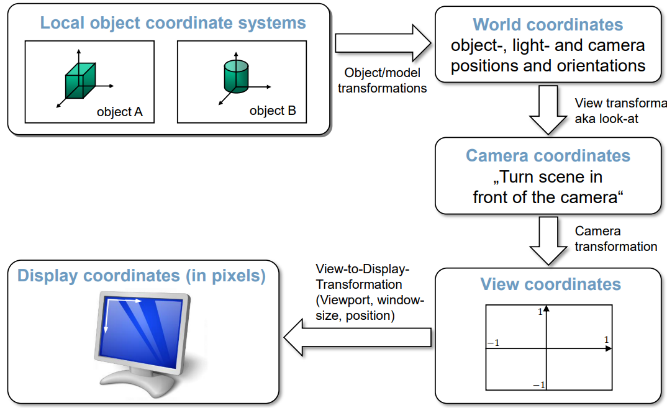
Stochastische Methode: Anti Aliasing nur an Zufälligen Punkten. Effizienter, aber kann bei Animationen flackern

FSAA (Full scene anti-aliasing): Super-sampling for every pixels of the scene, very expensive.

TXAA (Temporal anti-aliasing): Filtering of multiple samples inside and outside the pixel with pixels of previous frames.

DLSS (Deep learning super sampling): A NN is trained (an stored on the driver) to render the anti-aliased high resolution image.

**Kapitel 3 - Transformations and Projections**



A window defines the visible sub-area of the image plane

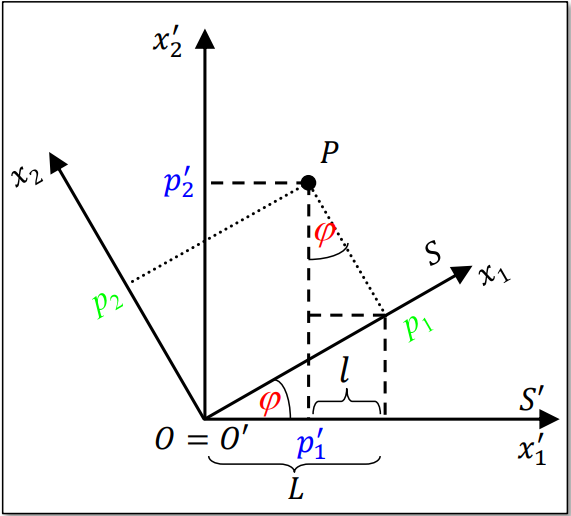
A viewport defines a sub-area of the display, where the content

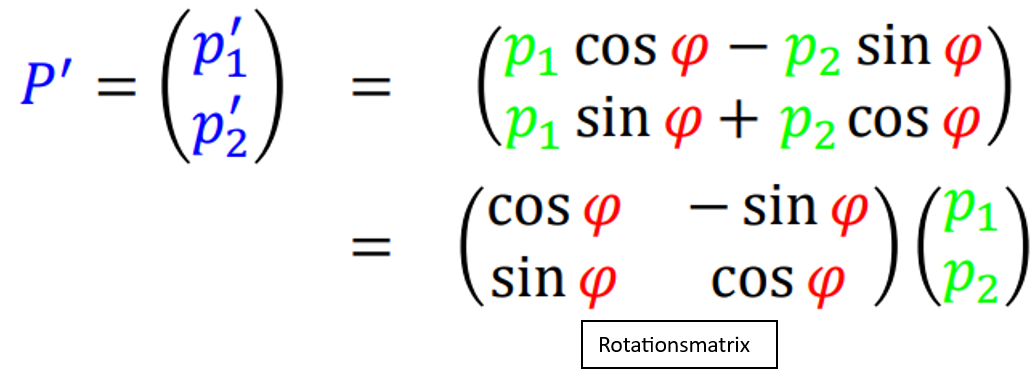
of the window is displayed.

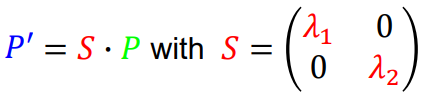
Window-viewport-transformations (windowing) are

2d-2dtransformations.

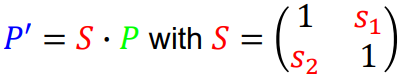
Wir nehmen an, dass ein Koordinatensystem Kartesisch (orthonormal + right handed) ist.

Für Rotation: Translation (Local System to (0,0) of Global System), Rotation (around origin), Back Translation (of Local System) Matrixmultiplikation: Zeile \* Spalte



**Matrix multiplication is not commutative.**

Skalierung eines Koordinatensystems:



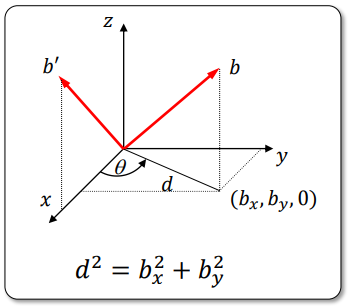
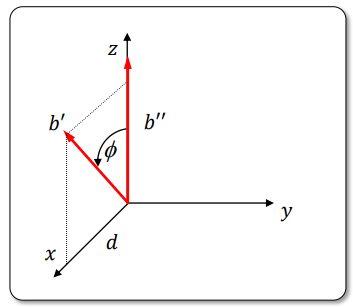
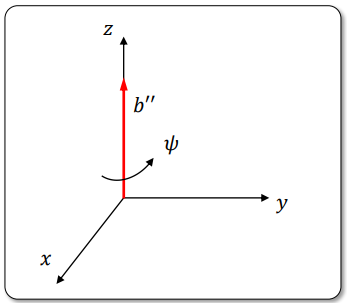
Scherung eines koordinatensystems:

Under an affine transformation parallel lines remain parallel.

Windowing: Translation of image into the (World)-origin, Scaling to size of viewport, Translation of image to viewport position

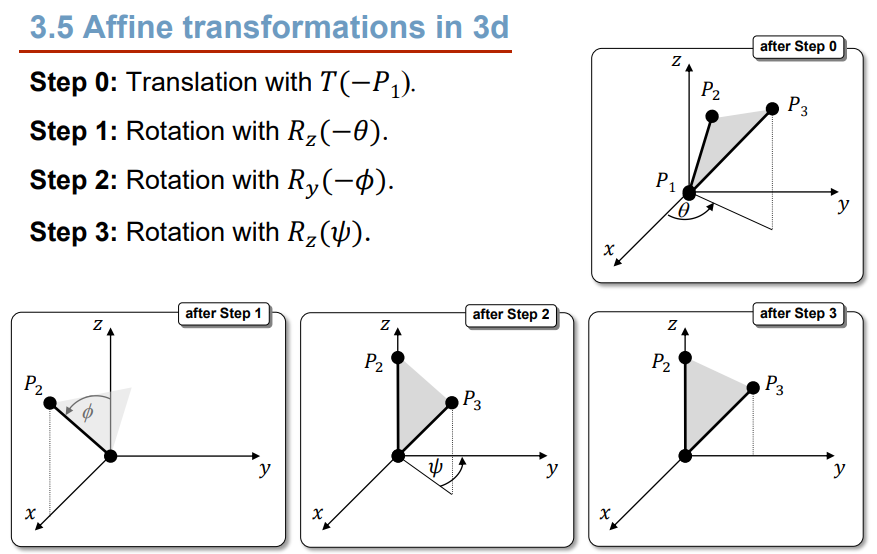
Transformation in die nächst höhere Dimension heben um auch für Translation Matrixmultiplikation nutzen zu können. (Homogen)

***Rotation GEGEN Uhrzeigersinn***

Rotation um beliebige Achse G: G um Z und Y rotieren, damit sie auf Z liegt. Eigentliche Rotation um Z ausführen. Punkt wieder um Y und Z zurück rotieren.

Falls G nicht durch Ursprung geht

Translation auf Ursprung, zum Schluss

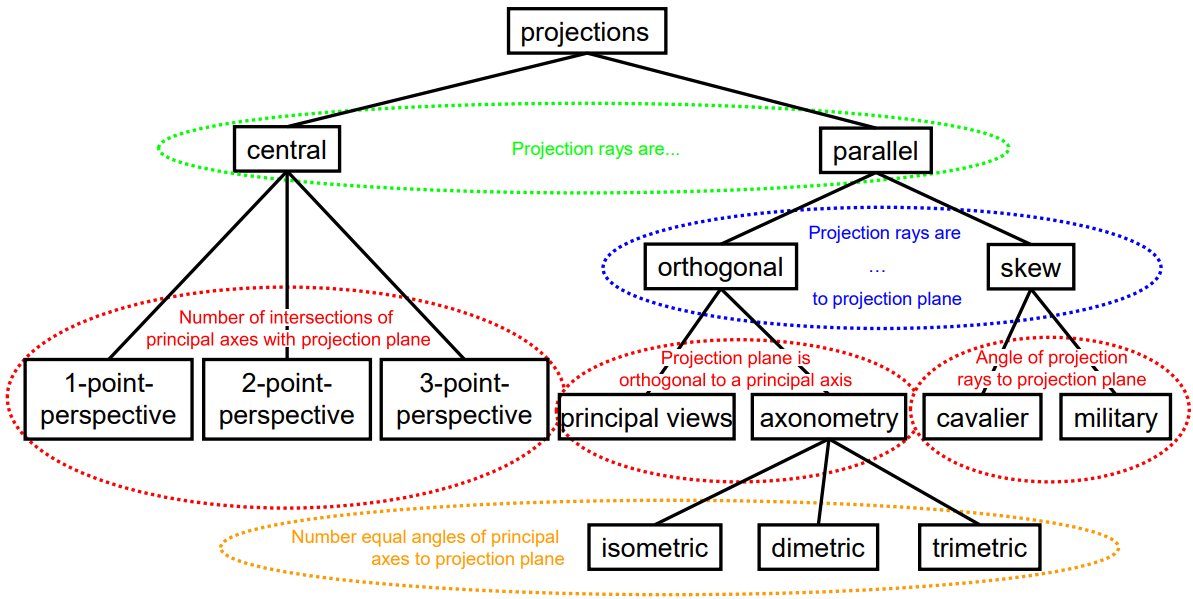
Backtranslation. G muss normalisiert sein.  
Wenn ein Objekt in m Globalen System um sich selbst  
gedreht werden soll (Orientation ändern), dann benötigt   
man dafür das Lokale System. Durch 3 (nicht kolineare)  
Punkte kann dieses Rekonstruiert werden.

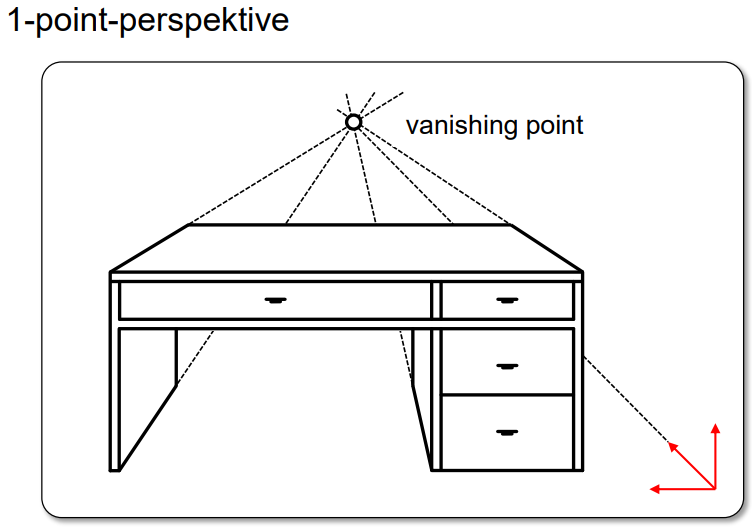
**Euler Winkel**: Stellen die Orientierung eines Objektes  
Im 3D Raum dar. Am intuitivsten.

**Gimbal Lock**: Tritt auf, wenn sich bei Rotation um Euler  
Winkel 2 Achsen Alignen, so dass man einen Freiheitsgrad verliert.

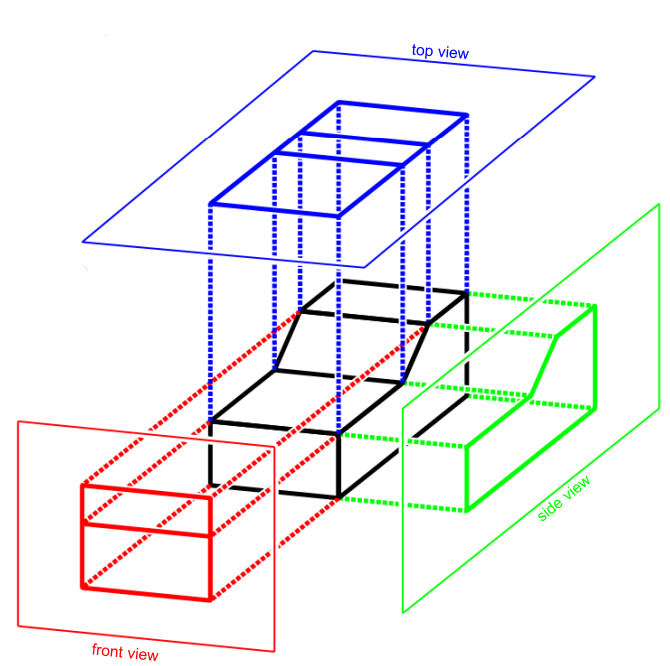
**Quaternions**: Alternative Darstellung einer Rotation.  
Theorie: Einheitsvektor stellt Rotationsachse G dar, Orthogonal zur Achse geht eine Ebene durch P. P wird (in der Ebene) um Winkel θ um die G rotiert.  
Normalisiertes Quaternion kann dargestellt werden als: 𝑞 = (cos(𝜃/2), sin(𝜃/2) 𝒏), wobei 𝒏 = G.  
Für Rotation Quaternion Produkt: 𝑞 ∘ (0, 𝒑) ∘ 𝑞′ 𝑞′ = q konjugiert.  
Attention: The quaternion multiplication is not commutative!

**Projection:** A projection is a mapping, that maps a space of dimension 𝑛 to a space of dimension < 𝑛.



****

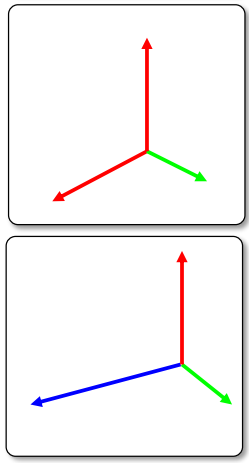
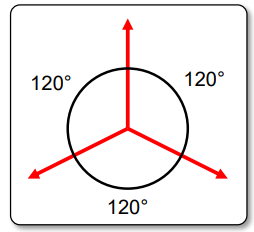
**Perspective projection / central projection**: „Normal“,   
wie in der Übungsaufgabe. Parallele Linien die nicht Parallel  
 zur Projektionsebene sind laufen in einem Fluchtpunkt   
zusammen. Klassifiziert nach Anzahl der Hauptachsen (x,y,z)   
die durch die Projektionsebene gehen   
(also nicht parallel dazu sind). Schreibtisch von   
Vorne = 1-Punkt-Perspektive, da zwei Hauptachsen  
 (Hoch/Runter + Links/Rechts).

**Parallel projections:** Spezialfall der Central Projection  
(Projektionspunkt = ∞). Weniger realistisch, dafür sind Messungen  
möglich.

**Orthographic projections:**

**Principal views:** Technische Zeichnung, wie 1-Punkt-Perspektive,  
aber ohne die nicht parallele Dimension. Winkel und Distanz  
bleiben erhalten.

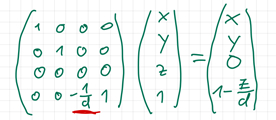
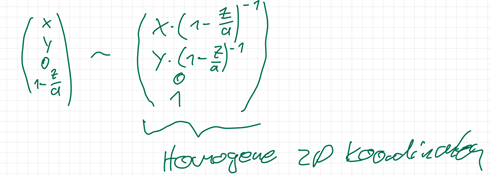
For an **axonometry** the projection plane is not orthogonal  
to any of the coordinate axes (of the world system).

* Parallel lines are mapped to parallel lines.
* Angels are not preserved.
* Distances can be measured along principal axes,   
  although using different scales for each axis.

Isometric, dimetric , trimetricaxonometry:

Winkel zwischen Allen (3), 2 bzw. keiner Achse und der   
Projektionsebene sind gleich.

Projektionsmatrix wird auf Punkt losgelassen



The projectors through the corners of the view window define the socalled viewing frustum.

Front and back clipping plane

Clipping:

Beide Endpunkte im Fenster: Plotten

Beide Endpunkte Oberhalb/Unterhalb/Links/Rechts vom fenster: nicht Plotten

Ansonnsten: Compute the intersections of the line with the window boundaries and determine the visible part of the line.

**Kapitel 4 – Rendering and Visibility**

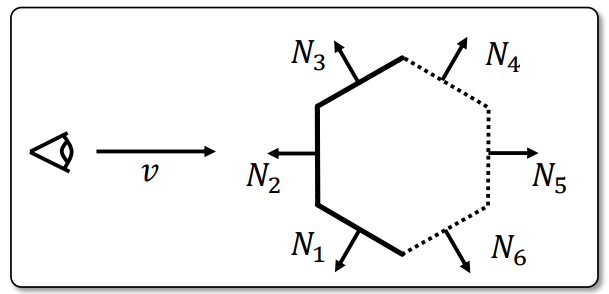
**Visibility test**: Determine all visible and invisible (occluded) parts of a scene

Objekte werden verdeckt, wenn verschieden Punkte aus der 3-D Szene auf denselben Pixel der 2D Szene gemapped werden.

Coherence 4-S36

Object space method: In der 2D Szene wird berechnet welche Objekt(Teile) verdeckt werden

Image space method: Bei der Projektion auf den Image Space wird geschaut, welches Objekt als erstes vom projection-ray getroffen wird.

**Backface-Culling:** Rückseite von 3D-Objekten „entfernen“, das diese nicht gerendert werden muss.  
 Skalarprodukt zwischen Viewdirection und face normal berechnen:

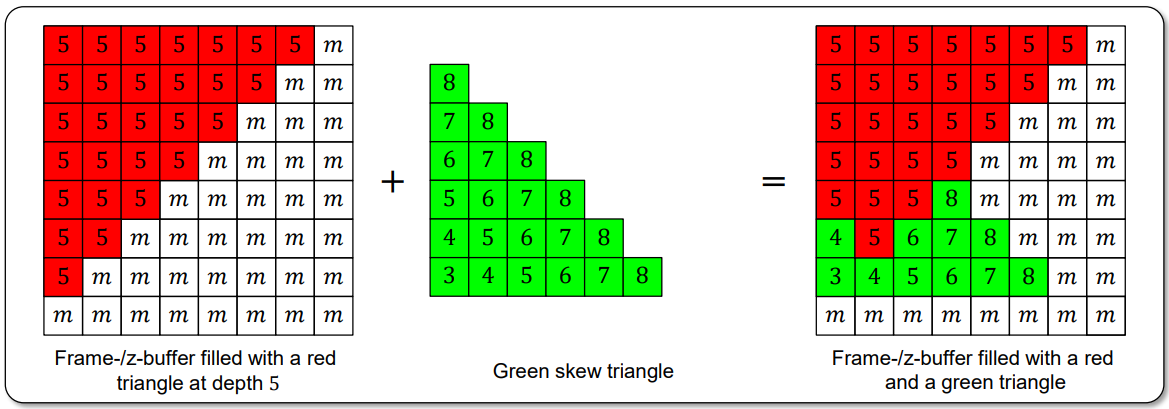
* + v \* N < 0 = Visible
  + v \* N > 0 = Invisible

Skalarprodukt billig, Complexity reduced by ~ Factor 2  
Bei Komplexeren Objekten kann BC nicht alle nicht-sichtbaren Oberflächen entfernen.

**Bounding Box:** Box um komplexeres Objekt, um komplexe Berechnung zu vereinfachen. Je nach bounding Box ungenau.

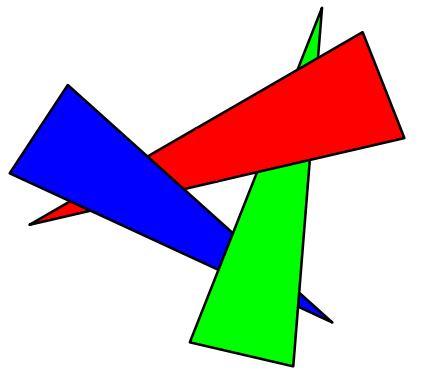
**Frustum Culling:** Test Intersection of Bounding box with View-Frustum.

**Occulsion Culling:** Pretend to render Objects bounding Box, count visible Pixels. If the number of potentially visible pixels is > 𝟎 (or some threshold), render the object. Otherwise, discard the object from the rendering pipeline. Pop-in/pop-out effects in animations possible.

**Z-Buffer:** Additional memory, that stores for   
each pixel the smallest 𝑧-value.

Computation of 𝑧(𝑥, 𝑦) for planar polygons:  
Plane equation: 𝑎 ⋅ 𝑥 + 𝑏 ⋅ 𝑦 + 𝑐 ⋅ 𝑧 + 𝑑 = 0  
Nach Z umstellen: 𝑧(𝑥, 𝑦) = (−𝑑 − 𝑎𝑥 − 𝑏𝑦) / 𝑐  
Mithilfe von Depth Coherence kann der Z-Wert   
inkrementel berechnet werden:  
𝑧 𝑥 + 𝑑𝑥, 𝑦 = 𝑧 𝑥, 𝑦 − 𝑑𝑥 ⋅ 𝑎/𝑐.

Nachteile: Begrenzte Auflösung des Z-Buffers(Anzahl Z-Werte) kann zum Problem werden, bei Szenen mit sehr weit entfernten Objekten.  
Braucht extra Speicher. Komplizierter bei transparenten Objekten.

Alpha-Buffer: Wie Z-Buffer, aber für opacity (durchsichtigkeit) Werte.   
Von hinten nach vorne Aufbauen, schwierig bei:

**Ray Casting:** Solves the visibility problem, ends at first Object hit.

**Ray tracing:** Ray Casting + tracing of reflected and refracted rays.

Example Sphere: In die Kreisgleichung wird Geradengleichung des View-Rays  
eingesetzt. Nach T auflösen ergibt 1-2 Schnittpunkte, kleinster ist nächster.

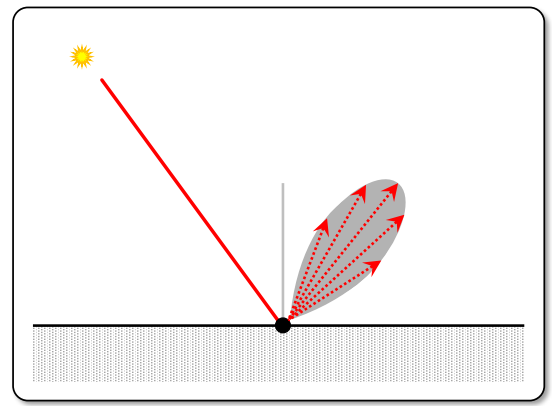
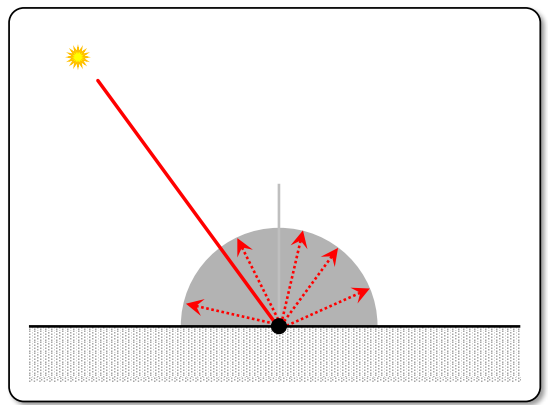
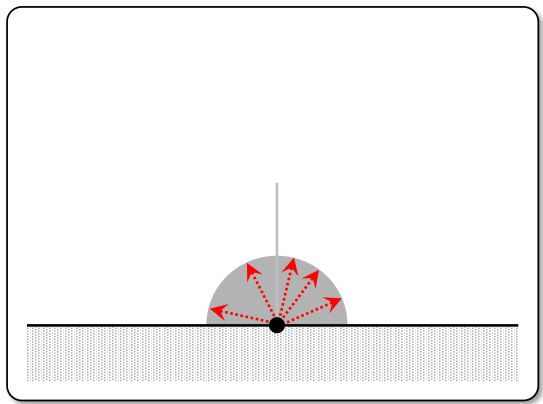
Ray Casting ist sehr aufwendig. Back-Face Culling und Boundign Boxes reduzieren Komplexität.

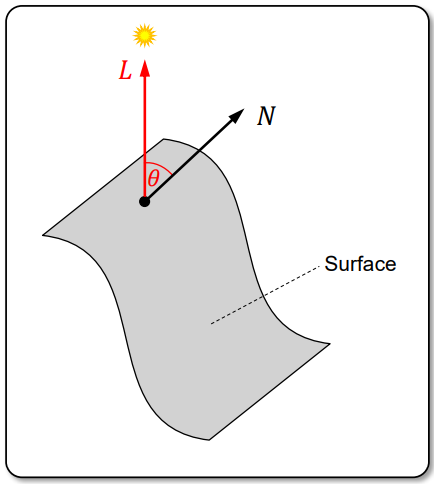
**Illumination**  
Typen: Point light, Direction light, Spot light, Surface light source

Local illumination model: Nur für direct light (direkt von Lichtquelle)  
Global illumination model: Direct and indirect light.

Shading Model Optionen: Itensity für jeden Punkt vs Itensity für manche Punkte, interpolation dazwischen.  
Bei Interpolation: Illumination wird in in Object-Space berechnet, Mittelwert wird aber in Display-Space berechnet.

=> Da Central projections nicht affine, sind die Ergebnisse nicht 100% korrekt, aber close enough.

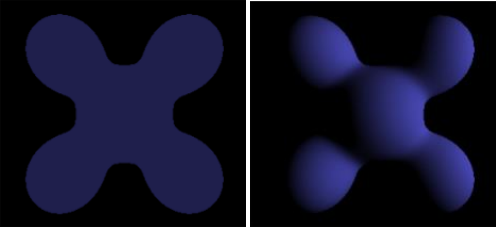
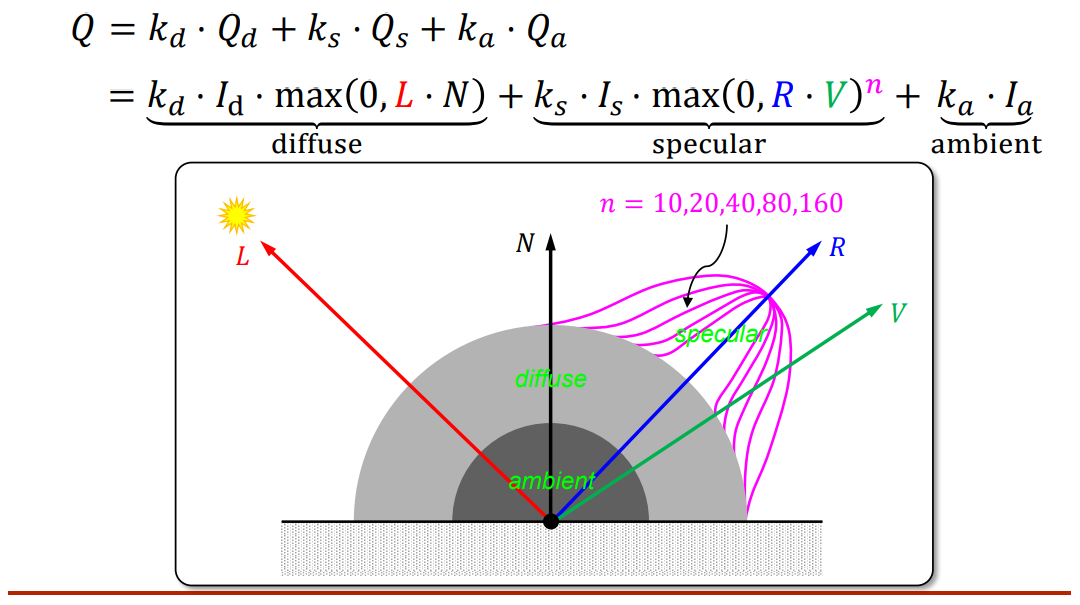
Phong Illumination: = imperfect specular component, perfect diffuse component, ambient light

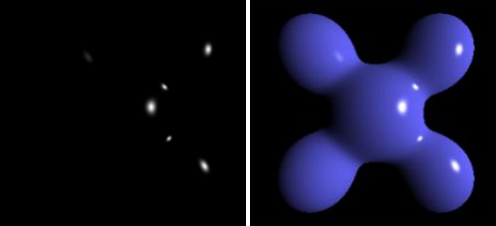
**Diffuse Reflection** (the term 𝑘𝑑 · 𝑄𝑑)

𝑄𝑑 = 𝐼𝑑 ⋅ cos 𝜃 = 𝐼𝑑 ⋅ (𝐿 · 𝑁)  
𝐼𝑑 Intensity of incident, diffuse light.   
𝜃 Angle between point normal 𝑁 and light vector 𝐿.  
Wenn L = N, dann ist cos 𝜃 = 1 = Maximale Diffusion

**Specular reflection** (the term 𝑘𝑠 · 𝑄𝑠 )  
Physically a specular reflection is the mirror image of the light source, smeared over an area of the surface

𝑄𝑠 = 𝐼𝑠 ⋅ cosn (ω) = 𝐼𝑠 ⋅ (𝑅 · 𝑉)n𝜔 Angle between 𝑉 and 𝑅.  
 𝑛 Simulates degree of perfection of the surface (shininess)  
𝑛 → ∞ is a perfect mirror

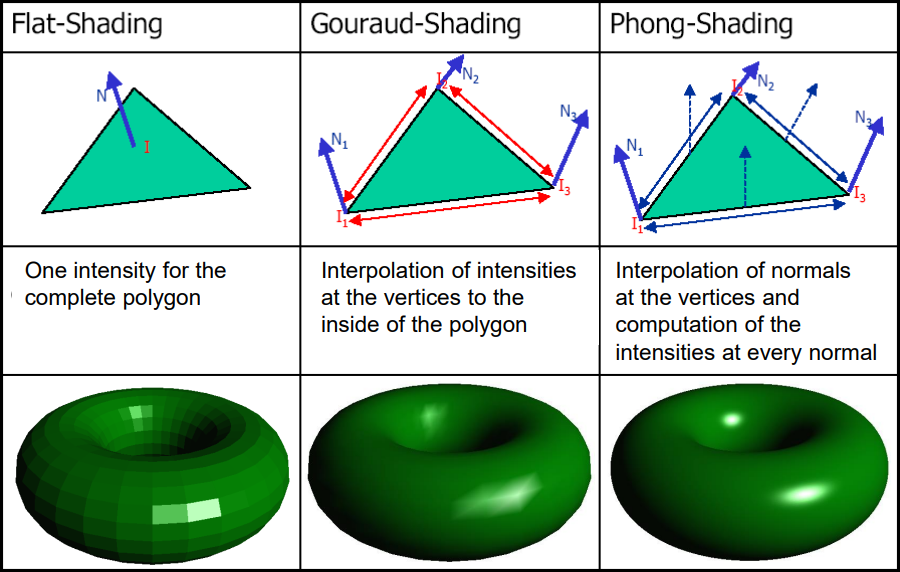
**** **Combined model (without color)**

****

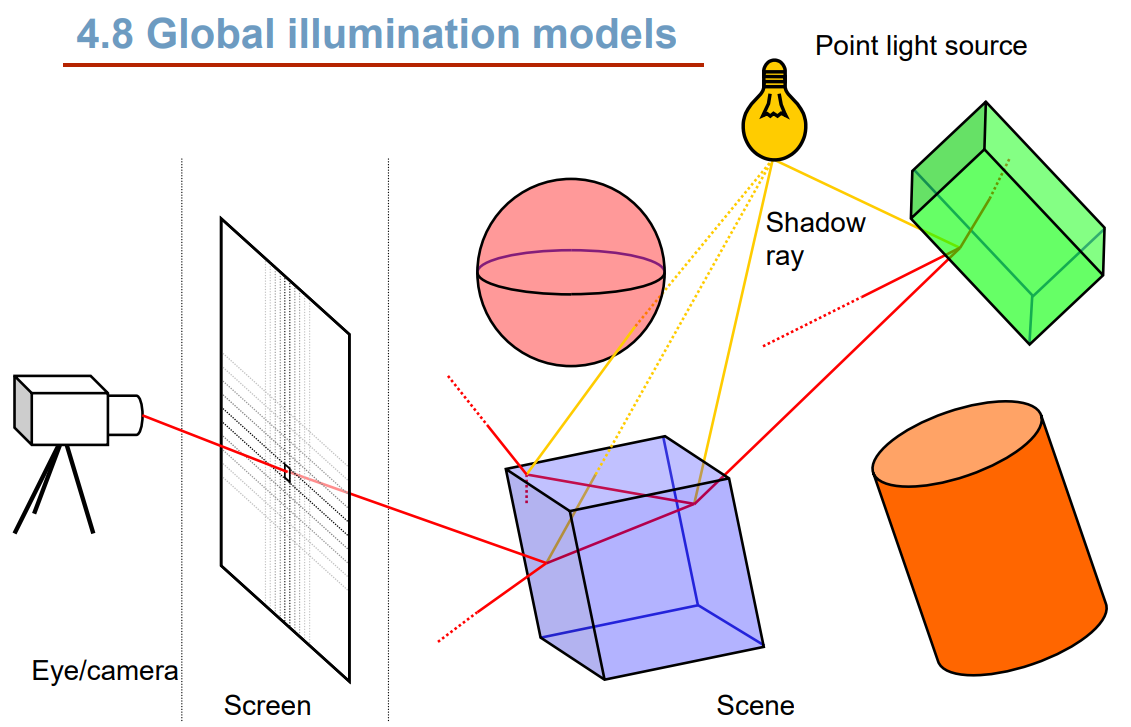
Phong gut für Plastik, ok für  
Metal, schlecht für Organisches

Blinn-illumination model: Bei Reflection statt R den Vektor H verwenden, welcher in der mitte zwischen V(iew) und L(ight) liegt.

BRDF(bi-directional reflection distribution function)  
This function describes the relation of the light that is reflected in an arbitrary direction 𝑉 in dependence to the direction of the incoming light 𝐿.

Interpolatory shading models:

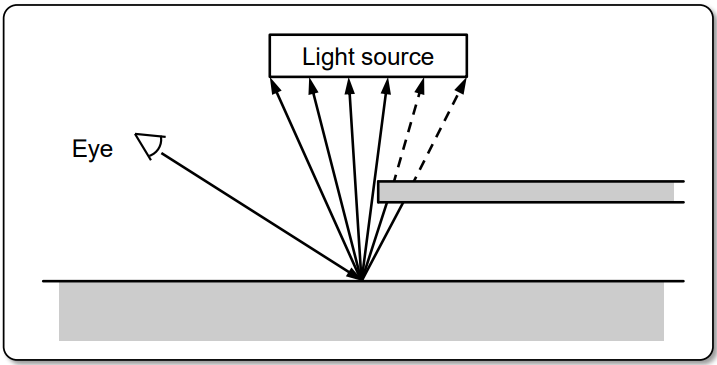
Objekt in Flächen aufteilen,   
Intensity wird einmal pro Fläche berechnet.

(Backwards) ray Tracing: Ray Casting + „verfolgung“ der   
Relfected und reflected rays. An der ersten Intersection vom  
gecasteten ray wird ein lokales Model z.B. Phong evaluiert  
(mit besserer light source intersection, wegen schatten).  
Reflected und refracted komponente kommen drauf.

Pro gecastetem ray der ein Objekt trifft wird ein Baum  
aufgespannt. Dieser enthält: Light, Reflected und Refracted  
Component, die wiederum selbst einen Baum aufspannen.

Refraction entsteht am Treffpunkt zweier benachbarten  
Materialien mit unterschiedlicher Dichte.

Es gibt nicht nur einen Reflected/Refracted ray, sondern mehrere, die um den „Perfekten„ (Einfallswinkel=Ausfallswinkel) Reflection Ray herum gestreut werden. Diese werden dann gemittelt.

2-D Lightsource: Lichtquelle nicht nur als einen Punkt ansehen,  
so kann man simulieren, dass nur ein Teil des Lichts ankommt.

Für mehr Realismus kann die Brennweite der Linse simuliert  
werden um verschwommene Bereiche zu erzeugen.

Adaptive supersampling: Statt Ein ray Cast pro Pixel, ein ray pro  
sub-Pixel, um Aliasing zu verhindern

Stochastic ray-tracing: Supersampling, aber statt feste Sub-Pixel zufällig verteilte Sub-Pixel

Ray-tracing – Properties   
Advantages:

* (1) The physical phenomena of illumination using geometric optics are very effectively simulated.
* (2) Very good for specular reflections.
* (3) Visibility is solved automatically.
* (4) High realism for geometric optics.

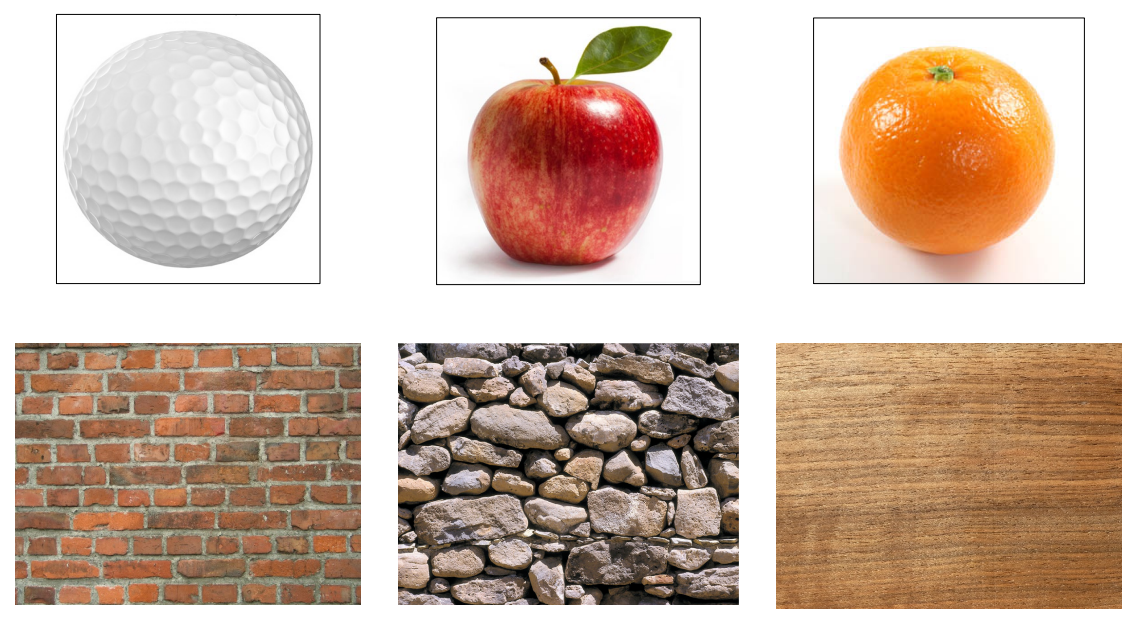
Disadvantages:

* (5) Not suitable for diffuse reflections.
* (6) Large computation costs.
* (7) Intersection computations are very inefficient.
* (8) Prone to numerical problems.

**Radiosity**The radiosity method computes the energy transfer caused by diffuse radiation between individual surface components in a scene.

The radiosity function wird für jeden Punkt gelöst, zur initialisierung bekommt jeder Punkt einen Startwert.  
Dann wird die Radiosity iterativ berechnet, bis sich die Werte stabilisieren.

**Rendering:** Image Synthese, d.h. mapping des geometrischen Models auf ein Display.  
Rendering-Pipeline: Siehe S1

**Kapitel 5 – Mapping**

Alle diese Objekte müsste naus Millionen Dreiecken   
Bestehen um die Farben/Oberfläche richtig abzubilden.  
Lösung: Texturen.

Vereinfacht das Rendering von Oberflächendetailles, ohne  
diese in derGeometrie abbilden zu müssen.

**Texel:** Pixel einer Texture Map

Interpretationen:

**Forward mapping:** The mapping from texture to screen space (via object space).

**Inverse Mapping:** The mapping from screen space to texture space.

Mapping wird in 2 Schritte geteilt:

**s-mapping:** First use a suitable (simple) mapping of the texture onto a simple intermediate, auxiliary surface, e.g.  
 rectangle, cube, cylinder, sphere, etc.

**o-mapping:** From the intermediate surface the texture is mapped onto the final surface of the object.

Texture Mappign ist Sampling -> Aliasing möglich (Minification/Magnification)

Gegenmaßnahmen:   
 Integral/ Sampling + Filter wenn 1 Pixel auf mehrere Texel fällt.  
 Texture map um größeres Bild erweitern falls 1 Texel auf mehrere Pixel fällt.

**Bumb-Mapping:**Zusätzlich Surface Normal hinterlegen. Strukturen können simuliert werden, sind aber immer noch flach.

**Displacement Mapping:** Overly the surface with a height field, such that surface points are translated along the surface normal.  
 Changes Silhouette.

**Opacity Mapping / Transparency Mapping:** Using opacity mapping the 𝛼𝛼-value of transparent surfaces is manipulated locally. Object ist an  
 entsprechenden stellen teilweise Transparent.

**Procedural Mapping:** Proceduale generierung von irregularities.

**Environment Mapping:** Statt Raytracing wird die gespiegelte Szene als TextureMap hinterlegt. Im Gegensatz zu anderen  
 Mappings ist dieses von der Kameraposition abhängig.