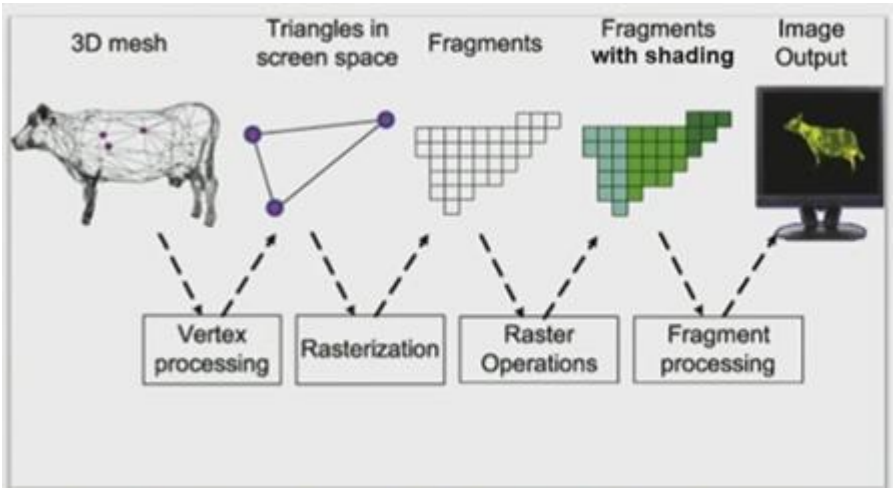
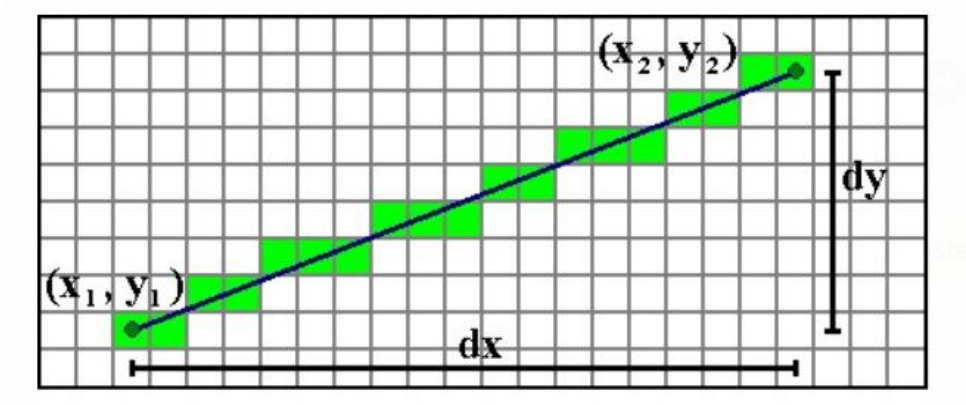
Konstantin Lackner 03.07.2020

Grafikpipeline



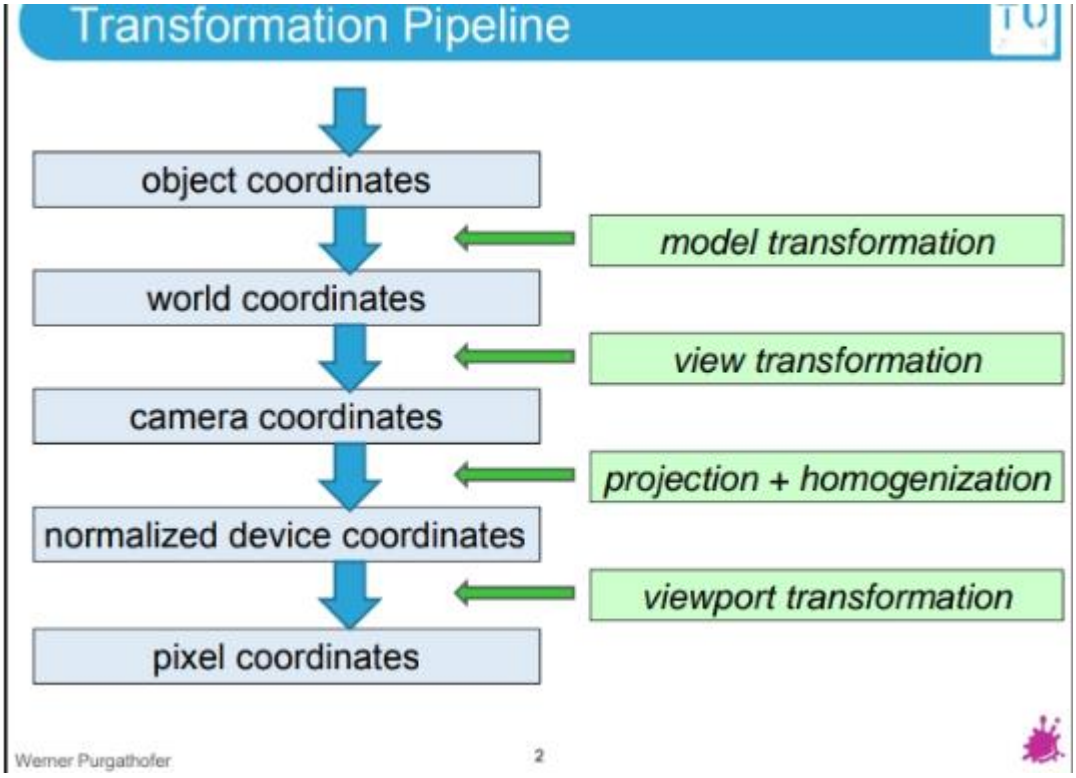
Zuerst hat man ein Mesh, dann muss man das verschieben, drehen, usw.. also Vertex processing im Vertex Shader.

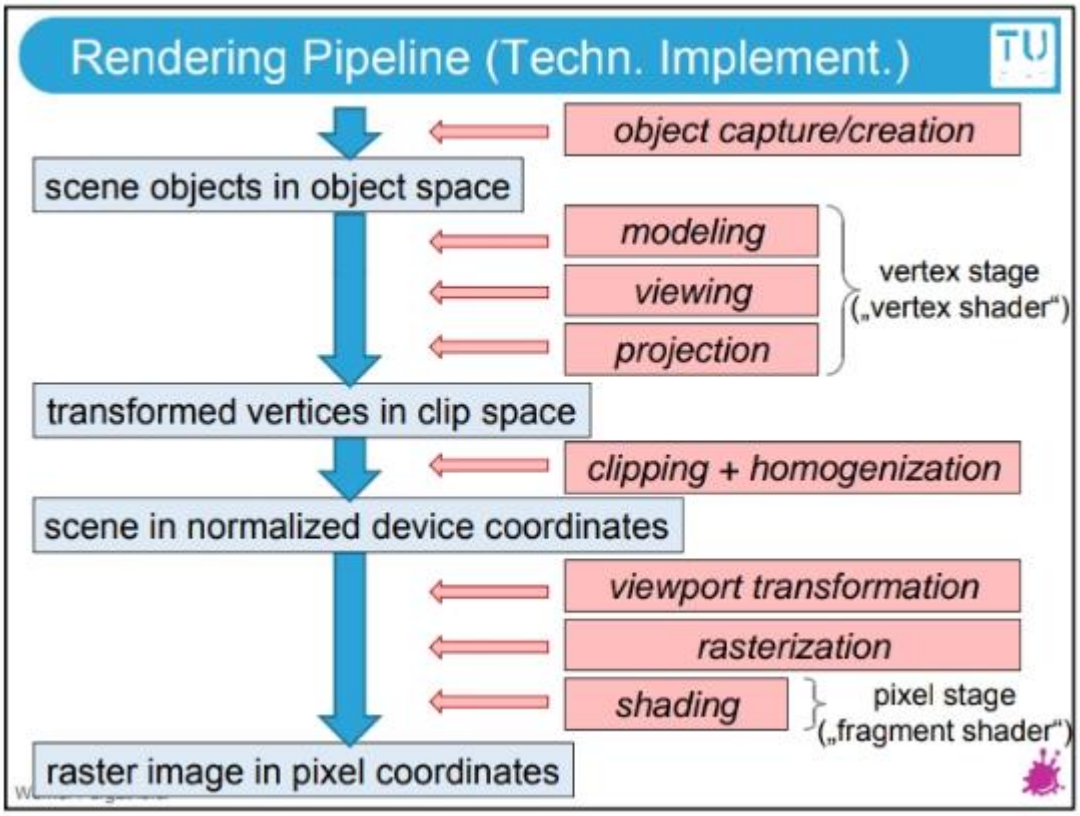
Rasterization = Sampling bei dem man aus den Dreiecken ein Pixelmuster bekommt. Da gibt es den Bresenheim Algorithmus der aus Linien Pixel macht.

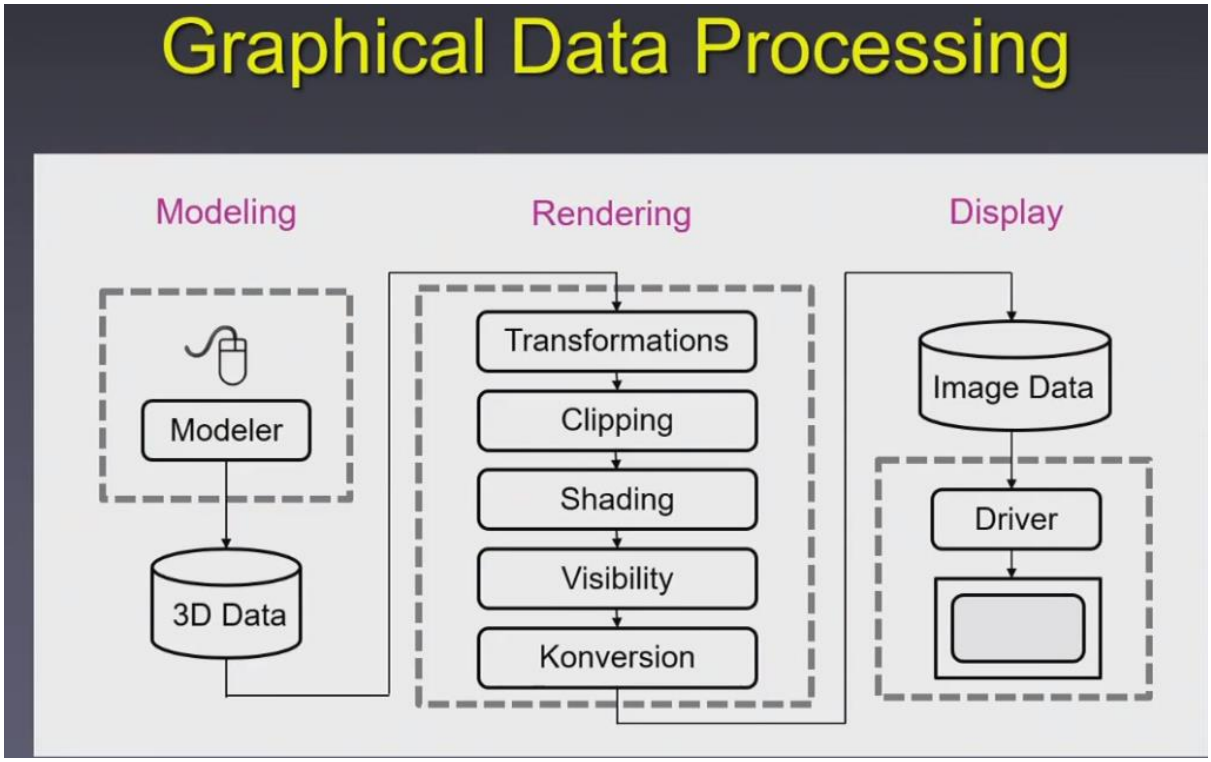


Rasteroperationen sind solche, wo Sichtbarkeit, Farbe usw. festgelegt werden, das sind Pixelshader oder auch Fragment Shader.

Das Ablegen im Speicherbereich der dann am Bildschirm abgelegt wird, ist der letzte Schritt.


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

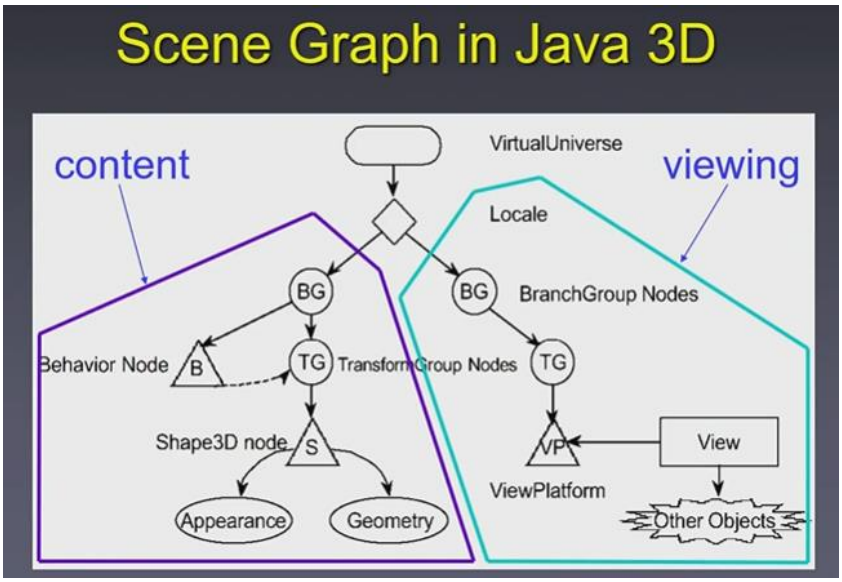


Graphics Programming

- Low-level (OpenGL, DirectX, Vulkan, Metal)
  - Direct call to API functions
  - Developed with C++, C#, Java, Python…
- Mid-level
  - Abstraction layer on top of graphics APIs
  - Java3D, Qt3D, bgfx
- High-level
  - Render engines
  - Visualization toolkits
  - Game engines

Konstantin Lackner 03.07.2020

Scene Graph



Strukturierung aller Objekte in der Scene. Dadurch kann man dann Transformationen gemeinsam anwenden. Zwei Zweige, einerseits der Contentzweig wo alle Objekte drinnen sind, andererseits der Viewing Zweig wo all die view Sachen drinnen sind.

Monte Carlo Path tracing

Bei Raytracing schaut man nur Richtung Lichtquelle oder Spiegelungsrichtung, also Richtungen die vielversprechend für Reflektionen sind usw.. Beim Monte Carlo Pathtracing schaut man in eine Vielzahl zufällig normalverteilter Richtungen.

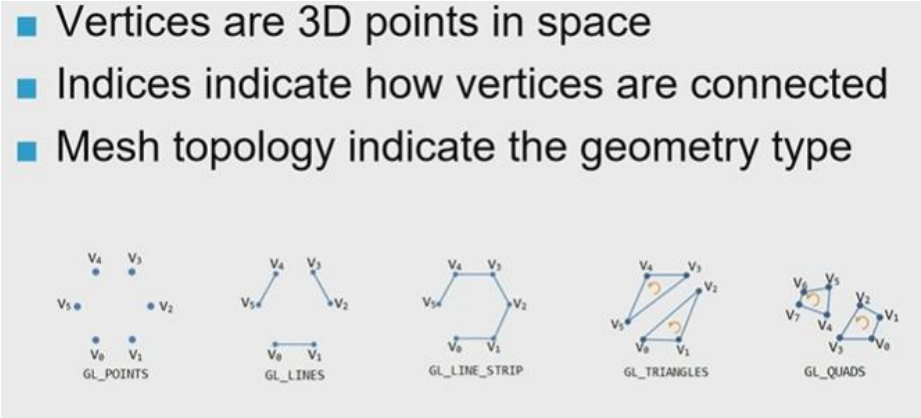
Unity

High-level System, das aber auch mid und low Sachen zulässt. Ist mit C# scripting aufgebaut. Cross-platform Unterstützung. Scene editor. WYSIWYG principle - Man hat eine Scene view und eine Game view und sieht all die assets usw. die man macht.

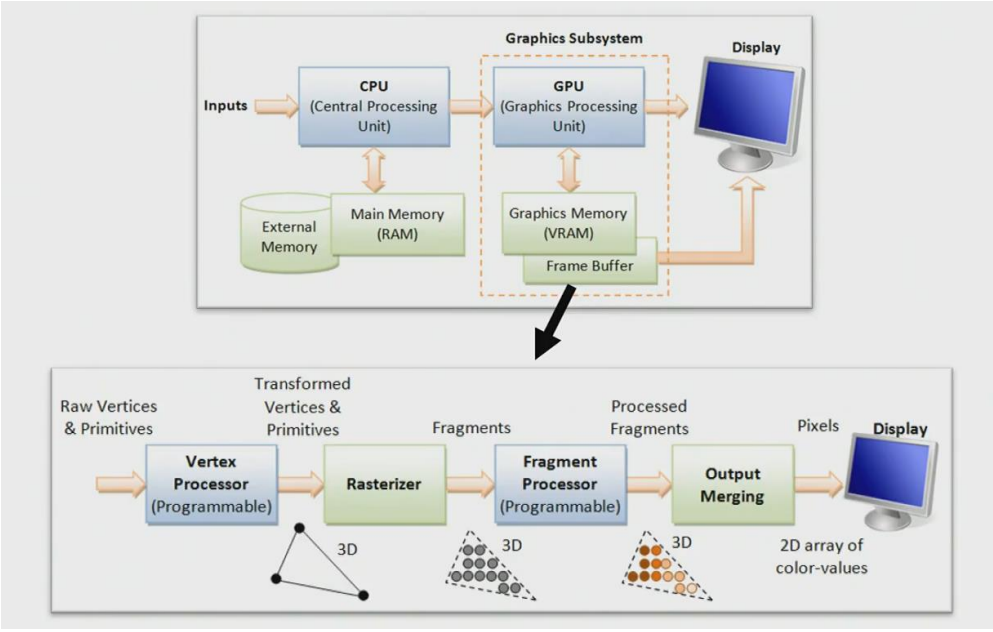
Konstantin Lackner 03.07.2020

Mesh

ist eine Menge von Dreiecken mit Nachbarschaftsinformation. Mit Indices werden die Eckpunkte indiziert. Man merkt sich also für jedes Dreieck nur einen Index dieses Punktes. (damit muss man ned mehrmals speichern)
Es gibt auch noch Normalvektoren bspw. an Eckpunkten oder im Dreieck. Meist wird bei Normalen im Eckpunkt einfach eine gewichtete Berechnung aller anstoßenden Dreiecke genommen.
UV Koordinaten sind die Koordinaten in 2D auf einem Rasterbild für die Texturen.



Shader



Vertex shader dann fragment shader

Quarternion

Vektor aus drei Rotationsachsen und einem Rotationswinkel.

Konstantin Lackner 03.07.2020

Macro, Micro und Meso scale

Macro sehr große Sachen, also bspw. der Ast eines Baumes
Micro sehr kleine, bspw. die Fasern in Blättern – können mit Texturen gemacht werden
Meso das dazwischen… also die sachen, die man nicht nur mit Texturen machen kann, sondern mit 3D machen muss, aber als mesh wären sie auch wieder zu komplex.

Particle Systems

Eh klar – Sorry, Renaj, musst selbst lernen

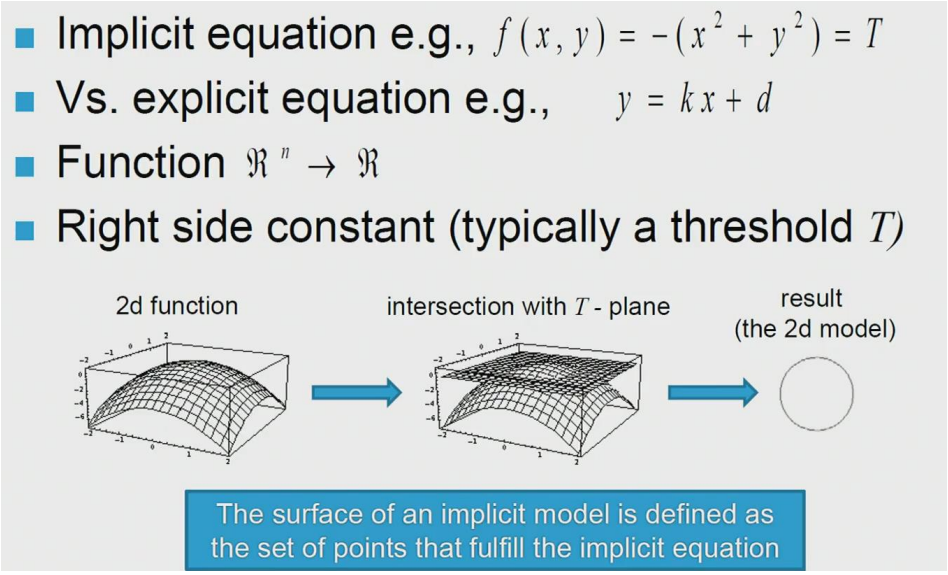
Lagranger Ansatz, Eulerscher Ansatz

Lagrange – Partikel bewegen sich im übergeordneten System weiter

Euler – Man platziert eine Messstelle und diese wird nicht verändert. Man misst dann zu gewissen Zeiten an diesen genauen Punkten.

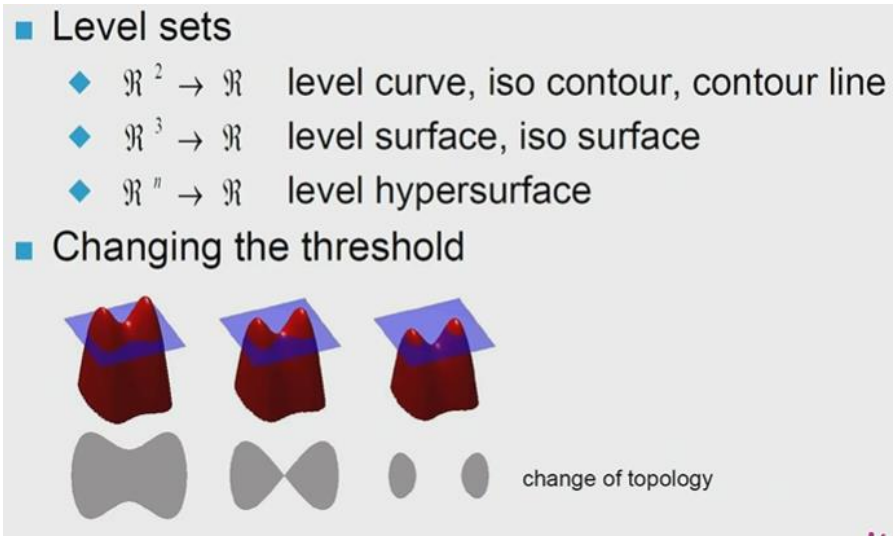
Implizite modellierung

Oberfläche ist nicht explizit gegeben sondern wird nach einer physikalischen Simulation berechnet und erst später bekannt.



Bei der physikalischen Funktion wird die Funktion verändert und dann mit der Fläche die stets bei T bleibt geschnitten um eben auch den Kreis damit zu verändern mit der Zeit. Der Vorteil ist, dass man damit nicht nur einen Kreis bekommen kann (man könnte das ja auch mit änderndem Radius), sondern, dass man auch andere Formen bekommen kann, bspw. einen Donut, theoretisch auch ein Dreieck, usw..

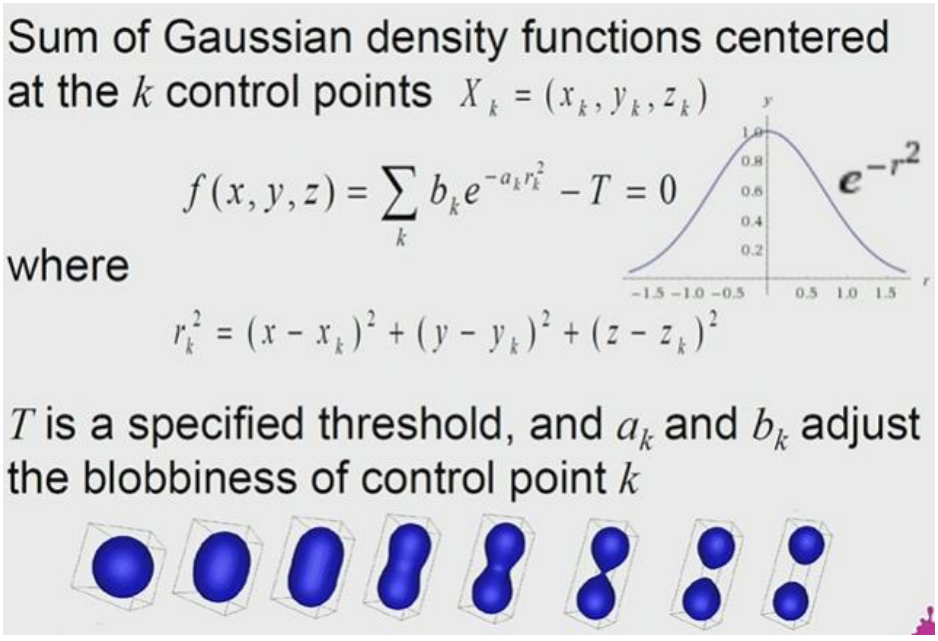
Konstantin Lackner 03.07.2020



Änderung der Topologie indem man nur das T ändert. Hätte man eine Linie von Punkten, wäre das viel aufwändiger.

Blobby objects

Particles werden definiert durch ein Zentrum zu dem sie nahe liegen. Der Einfluss des Partikel auf andere ist abhängig davon wo die anderen stehenm je näher man zu einem Partikel ist, desto größer ist dessen Einfluss. Daher ist der Einfluss im Zentrum abhängig von der Entfernung der Partikel zum Zentrum.



T ist ein Schwellwert, man verbindet alle Punkte wo dieses T ein gewisser Wert ist.

Konstantin Lackner 03.07.2020

Superquadrics

Quadrics sind Formeln zweiten Grades. Von einer Analytischen Definition geht man aus und verändert gewisse Parameter. Man kann mit diesen Kegelschnitte ausdrücken.

Beispiel Superellipse:

- Exponent of x and y terms of a standard ellipse are allowed to be variable:

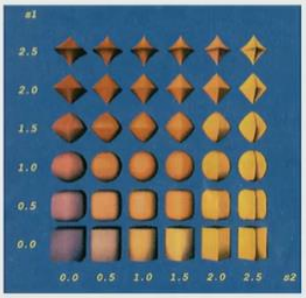
$$\left(\frac{x}{r_x}\right)^{2/s} + \left(\frac{y}{r_y}\right)^{2/s} = 1$$

- Influence of s:

- Exponent of x, y and z terms of a standard ellipsoid are allowed to be variable:

$$\left[\left(\frac{x}{r_x}\right)^{2/s_2} + \left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1} + \left(\frac{z}{r_z}\right)^{2/s_1} = 1$$

- Influence of s_1 *and* s_2:



Procedual modeling

Einfache Regeln für die Erzeugung von Objekten. Endergebnis wird nicht gespeichert, sondern on the fly generiert. Durch das Anwenden der Regeln entstehen hoch komplexe Strukturen. Die Regeln sind aber meist eher einfach.

Sweeps

Objekte die bspw. wie beim Töpfern rotationssymmetrisch gebaut wird. Man kann bspw. den Querschnitt eines Objektes entlang einer Kurve bewegen.

Translational sweep = Verschieben des Querschnitts.

Rotational sweep = Töpfern

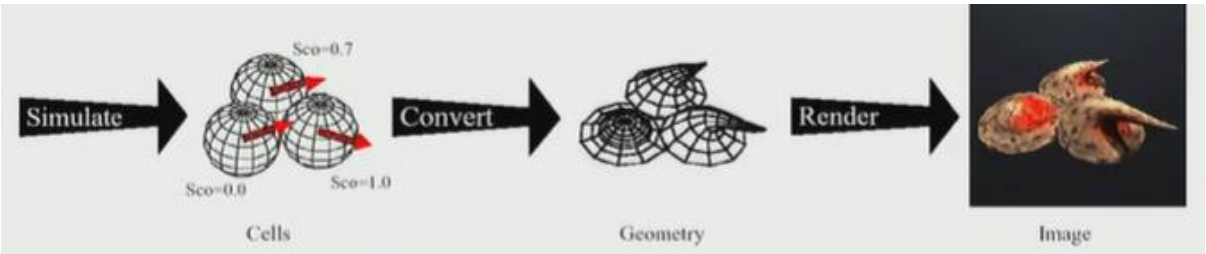
Der Querschnitt wird bspw. durch Kontrollpunkte definiert.

So hab ich die Map in Blender gebaut.

Sind schwer zu modellieren und schwer zu rendern, können dafür aber sehr komplexe Formen leicht darstellen.

Cellular Texture Generation

Man hat Elemente wie bei einem Partikelsystem, aber Zellen mit fixer Position, diese werden durch die Position beeinflusst und auch durch die, die die Zelle umgeben. Man nimmt eine Oberfläche mit einer Menge von Zellen, diese Zellen haben einen Zustand und gewisse Parameter. Das Zellprogramm verändert Parameter, dann wenn das fertig ist, wird die Geometrie erzeugt.



- Cell state: position, orientation, shape, chemical concentrations (reaction-diffusion)
- Cell programs:
  - Go to surface, die if too far from surface, align, adhere to other cells, divide until surface is covered, ...
  - Differential equations
- Extra cellular environment: neighbor orientation, concentration, ...

Zellparameter und Programme

Terrain Simulation

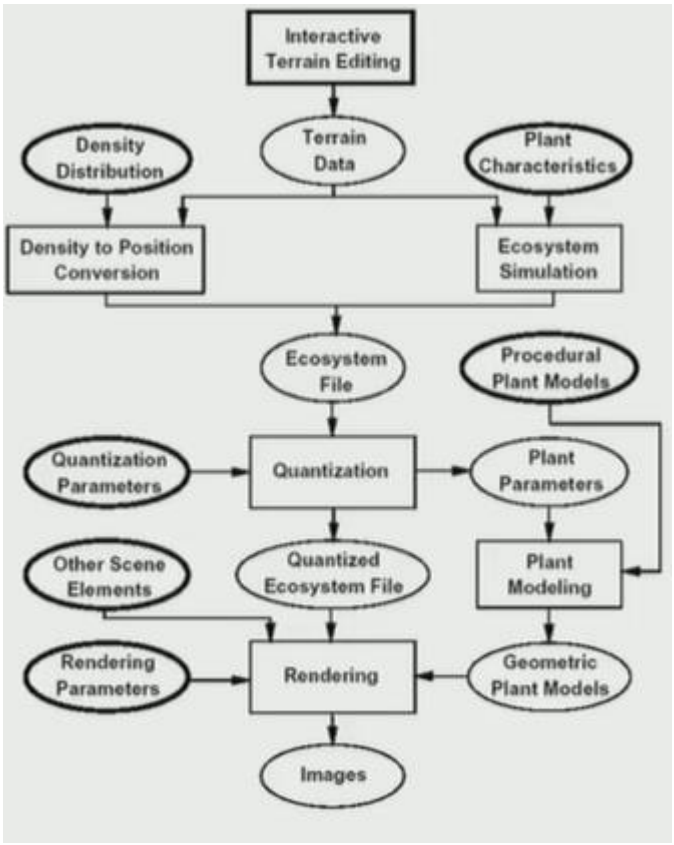
Hohe geometrische Komplexität aber einfaches Bildungsgesetz (entspricht natürlich nicht wahren biologischen Prozessen)

Man nimmt ein Dreieck und dann ersetzt man prozedural. Man nimmt zu jder der Kanten den Mittelpunkt und verschiebt dann jeden der Mittelpunkte ein bisschen nach oben oder unten. Damit bekommt man dann vier Dreiecke, das macht man so lange, bis man ein Gebirge hat.

Man muss sich natürlich mit den Zufallswerten spielen, damit man realistischere Gebirge bekommt. Bspw. von der Höhe abhängig machen.

Die Fraktale Dimension gibt dabei die Rauigkeit an. Je höher desto rauer. = Wie sich die Fläche ändert in Relation zu den Unterteilungsfaktoren.

Scene Synthesis System



Manche Dinge, bspw. Grashalme, werden nur als billboards gezeichnet, manche auch anders als bspw. heightmap beim terrain.

Pflanzenpopulationen kann man generieren, indem man:

Individual based – Bspw. Heightmaps so simulieren, dass man schaut, wo bei Regen in der heightmap Wasser gesammelt werden würde und dort in der Nähe dann mehr Pflanzen erzeugt. Ebenso kann man das von den bereits anwesenden Pflanzen abhängig machen. Bspw. schließen sich manche Pflanzen aus, manche begünstigen sich, usw..

Space occupancy – da wird definiert an welcher Stelle welche Pflanzen stehen

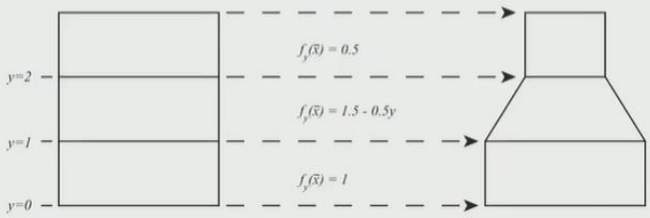
Struktur-deformierende Transformation

Um mehr Formen zu erhalten kann man auch einfach ein Objekt hernehmen und dieses verformen.

Tapering:

■ Scale factor is a function:

$$\vec{x}' = \begin{pmatrix} f_x(\vec{x}) & 0 & 0 \\ 0 & f_y(\vec{x}) & 0 \\ 0 & 0 & f_z(\vec{x}) \end{pmatrix} \cdot \vec{x}$$

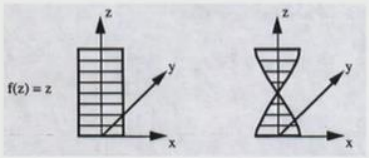


Hier hängt der Skalierungswert von der Position ab.

Twist:

■ Angle of rotation is a function
e.g., for rotation about z-axis

$$\vec{x}' = \begin{pmatrix} \cos f(\vec{x}) & -\sin f(\vec{x}) & 0 \\ \sin f(\vec{x}) & \cos f(\vec{x}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \vec{x}$$



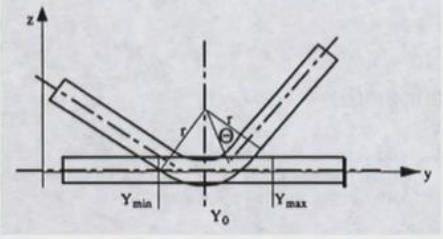
Auch hier hängt der Winkel vom X ab.

Bend:

■ Also non-linear rotation

$$Y = \begin{cases} -\sin(\Theta)(z-r)+y_0 & ,y_{min}<y<y_{max} \\ -\sin(\Theta)(z-r)+y_0+\cos(\Theta)(y-y_{min}), & y<y_{min} \\ -\sin(\Theta)(z-r)+y_0+\cos(\Theta)(y-y_{max}), & y>y_{max} \end{cases}$$

$$Z = \begin{cases} \cos(\Theta)(z-r)+y_0+r & ,y_{min}<y<y_{max} \\ \cos(\Theta)(z-r)+y_0+r+\sin(\Theta)(y-y_{min}), & y<y_{min} \\ \cos(\Theta)(z-r)+y_0+r+\sin(\Theta)(y-y_{max}), & y>y_{max} \end{cases}$$



Konstantin Lackner 03.07.2020

Kurvenarten und Surfaces

Parametrische Kurven

Solche wo ich einen Parameter angebe und dann einen Punkt auf der Kurve bekomme. (wie im Spiel zwischen 0 und 1)

Polynomiale Kurven

Kurven mit gewichteten Kontrollpunkten.

Rationale Kurven

Gewichtsfunktionen hängen von Parametern in Bruchform ab, können genauer Kegelschnitte darstellen als Polynomiale Kurven.

Tensor Produkt

Wie kommt man von Kurve zu Fläche? Ein Sweep. Man bewegt die Kurve durch den Raum. Diese Fläche die entsteht nennt man Tensor Produkt Fläche.

Triangular Surfaces

Triangulierte Freiformflächen. Und Subdivision surfaces und so.

Parametrische Kurven

Parametrische Kurve im 3D ist eine Kurve in deren Definition ein Parameter vorkommt (u)

$$c: \quad c(u) = \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}; \quad u \in [a,b] =: I \subset IR$$

x, y und z sind alle Funktionen. Gut daran ist, dass man einfach das z weglassen kann und eine 2D Kurve bekommt.

Mit diesem Parameter kann man schauen, wo man auf der Kurve ist. Man ist aber bei 0.5 nicht zwangsweise in der Mitte.

Mit dem Tangentenvektor kann man die Kurve an der Stelle approximieren.

$$- \quad \text{Tangent vector}: \quad t(u) = \frac{d}{du} c(u)$$

Konstantin Lackner 03.07.2020

Parametrische Flächen

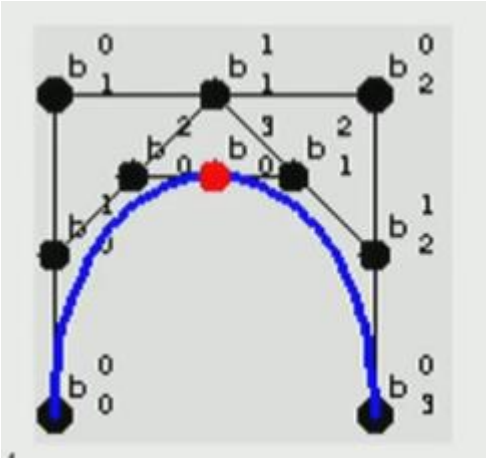
Dasselbe mit zwei Unbekannten.

$$s: \quad \mathbf{s}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}; \quad (u,v) \in [a,b] \times [c,d] =: I \times J \subset \mathrm{IR}^2$$

- $x(u,v)$, $y(u,v)$, $z(u,v)$ are differentiable functions in u and v
- Tangent plane: $\mathbf{t}_0(l,m) = \mathbf{s}(u_0,v_0) + l\,\mathbf{s}_u(u_0,v_0) + m\,\mathbf{s}_v(u_0,v_0)$
- Normal vector: $\mathbf{n}(u_0,v_0) = \mathbf{s}_u(u_0,v_0) \times \mathbf{s}_v(u_0,v_0)$
- s regular in $\mathbf{s}(u_0,v_0) \Leftrightarrow \mathbf{n}(u_0,v_0) \neq \mathbf{o}$
- s regular $\Leftrightarrow$ s can be parametrized in a way that all surface points are regular.

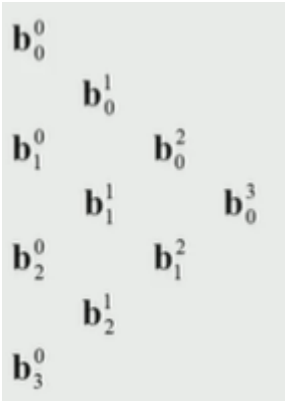
Bezier Kurven

Man hat Kontrollpunkte. Hier vier (die Randpunkte)



Man hat einen Parameter t, bei t=0 is man am Anfangspunkt, bei t=1 am Endpunkt. 0.5 ist nicht in der Mitte!

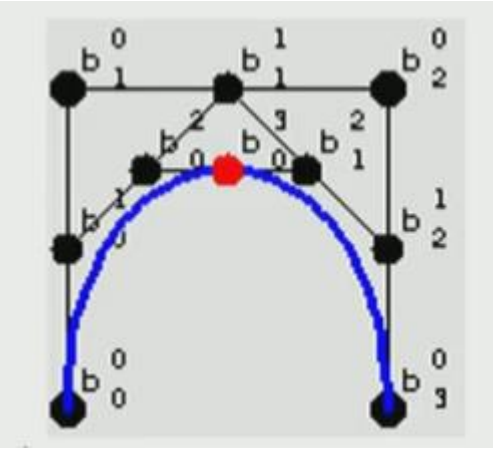
Der de Casteljau Algorithmus:



Konstantin Lackner 03.07.2020

Man schreibt amal alle Eckpunkte am linken Rand hin

Dann interpoliert man jeweils zwei Punkte mit einem gewählten t und schreibt rechts dazwischen den interpolierten. Man teilt also die Strecke in t und 1-t, hier ungünstig 0.5 gewählt…



Hier wurde für t 0.5 genommen. Der letzte Punkte bei diesem Interpolieren ist dann der Bezierpunkt, also der Punkt auf der Kurve. (hier rot)

$$\mathbf{b}_i^r := (1\text{-}t)\,\mathbf{b}_i^{r-1} + t\,\mathbf{b}_{i+1}^{r-1} \quad \text{with } \mathbf{b}_i^0 := \mathbf{b}_i.$$

Das r zeigt die Ebene (Vergleich Bild mit Dreieck) der untere Index i zeigt einfach der wievielte Punkt es ist.

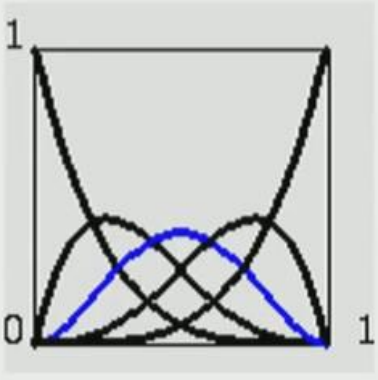
Die ganze Kurve lässt sich dann schreiben als:

Bézier curve with respect to Bézier points b_i, $i = 0,\ldots,n$:

$$b(t) = \sum_{i=0}^{n} b_i B_i^n(t)$$

Bernstein polynomials:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$



Die b sind nun nur die äußeren Kurvenpunkte, also die Kontrollpunkte.

B sind Bernsteinpolynome, das sind diese Funktionen hier. Die sind die Gewichtsfunktionen der Kurvenpunkte. Der erste Kurvenpunkt hat also, entsprechend dem einen Strich da, am Anfang 100% Einfluss, später dann am Ende 0. Man merkt aber, dass die Funktionen nie 0 sind außer bei Anfang und Ende, darum hat man den globalen Einfluss.

Beim Transformieren einer „normalen“ Kurve, müsste man tausende Punkte transformieren, bei Bezierkurven muss man lediglich die Kontrollpunkte transformieren.

Die Beierkurve liegt immer in der konvexen Hülle ihrer Kontrollpunkte

Man kann auch einfach die äußeren Kontrollpunkte löschen und nur die inneren nehmen, damit kann man die Kurve dann genauer steuern. Die Grade der Kontrollfunktion sind damit aber erhöht.

Ändert man einen Kontrollpunkt, ändert sich die gesamte Kurve.

B-Splines

Ändert man einen Kontrollpunkt, so ändert sich nur noch ein Teil der Kurve. Und mehr Kontrollpunkte erhöhen nicht auch den Grad.

Der Parameter k ist einfach wie viele Kontrollpunkte an einer Stelle Einfluss haben. Also die Differenzierbarkeit.

n ist die Anzahl der Kontrollpunkte plus 1 (weil es bei 0 beginnt) und k ist wie oft die Kurve stetig differenzierbar ist (siehe oben).

Damit kann man Kontrollpunkte ändern ohne die gesamte Kurve zu ändern. Auch kann man sehr viele Kontrollpunkte haben, aber k ist nur 4, also an einer Stelle haben nur 4 Kontrollpunkte Einfluss.

Kontrollpunkte / Knotenvektor

Kontrollpunkte sind Stützpunkte die man verändert.

Der Knotenvektor

Given :

- $n+1$ control points $\mathbf{d}_i \in I\!E^3,\ i = 0,\ldots,n$
- knot vector $U = (u_0,\ldots,u_{n+k})$

B-spline curve :

$$\mathbf{s}(u) = \sum_{i=0}^{n} \mathbf{d}_i N_i^k(u), \quad u \in [u_0, u_{n+k})$$

with the B-spline basis functions $N_i^k(u)$ of order k.

Der Parameterbereich geht nicht mehr von 0 bis 1 sondern von u_0 bis u_{n+k}.

Man hat keine Bernsteinpolynome, sondern diese Grundfunktion N.

Die Kontrollpunkte sind beim Bild oben d_i.

Wenn man in der B-Spline k auf 0 setzt, bekommt man die Kontrollpunkte heraus als Werte, keine Kurve.

Konstantin Lackner 03.07.2020

$$k=0: \quad N_i^0(u)=\begin{cases}1 & u\in[u_i,u_{i+1})\\ 0 & sonst\end{cases}$$

$$k>0: \quad N_i^k(u)=\frac{u-u_i}{u_{i+k-1}-u_i}N_i^{k-1}(u)+\frac{u_{i+k}-u}{u_{i+k}-u_{i+1}}N_{i+1}^{k-1}(u)$$

Das N ist die basis function B-Spline = basis spline

Das u-u/ui+k-1 usw. ist ein Hinnormieren damit man damit insgesamt, also mit beiden wieder auf ein Gewicht von 1 kommt.

An jedem Punkt Ui fängt eine Gewichtsfunktion an btw. Hört eine auf.

De Boor Algorithmus

Man muss wie beim De Casteljau der Bezier Kurve(ka wie wirklich geschrieben) schauen, aber nur k Kontrollpunkte kontrollieren, weil ja nur K Einfluss haben. Die Interpolation macht man hier indem man den Parameter so wählt, dass er auf die Abstände der Knoten normiert ist. Also muss man:

$$\alpha_i^r := \frac{t-u_i}{u_{i+k-r}-u_i}$$

das so berechnen. Das ist im Endeffekt wie auch beim De Casteljau, nur ist der Bereich hier zwischen u_0 und u_n statt 0 und 1. Dann gewichtet man den einen Punktk mit a^r_i und den anderen mit 1 - a^r_i .

Knot insertion

Man kann das K erhöhen indem man einen Knotenpunkt einfügt. Damit kann man Ableitungen berechnen. So wie das Zerlegen der Bezierkurven.
Man kann das auch zur Approximation benutzen, je mehr Punkte man einfügt, desto näher an die Kurve kommen die Punkte.

Rationale Kurven

Bei normalen B-Spline und normalen Bezier Kurven kann man Kontrollpunkte nur nehmen oder nicht (mit dem Einfluss). Wenn man aber 2.3 mal nehmen will bspw., muss man was neues machen. Ebenso kann man perfekte Kegelschnitte von Kreisen und Ellipsen nicht exakt machen so. All das geht mit rationalen Kurven.

Rational curve:

$$\mathbf{c}(u) = \frac{1}{w(u)} \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}, \quad u \in I \subset IR$$

Homogeneous representation:

$$\mathbf{c}_H(u) = \begin{pmatrix} w(u) \\ x(u) \\ y(u) \\ z(u) \end{pmatrix}, \quad u \in I \subset IR$$

Homogene Koordinaten macht man um die perspektivische Geometrie reinzubringen. Einfach eine extra Koordinate.

Rationale Kurven kann man wenn man sie homogenisiert einfach eine Dimensionen höher dann als ganz normale ordinäre polynomiale Kurve anschauen! WOW – Wer ist begeistert??? Wen freut das?
Man hat dann im 4D Raum einfach eine normale Bezier oder B-Spline Kurve. Wunderbar!

Wie kommt man zu der rationalen Kurve?

Rationale Bezierkurve

A rational Bézier curve is defined as

$$\mathbf{b}(u) = \frac{\sum_{i=0}^{n} w_i \mathbf{b}_i B_i^n(u)}{\sum_{i=0}^{n} w_i B_i^n(u)}, \quad u \in I \subset IR$$

The $w_i > 0,\ i = 0,\ldots,n$ are called weights.

Homogeneous representation:

$$\mathbf{b}_H(u) = \sum_{i=0}^{n} \mathbf{b}_{H\,i}\ B_i^n(u), \quad u \in I \subset IR$$

with the homogeneous Bézier points

$$\mathbf{b}_{H\,i} = \begin{pmatrix} w_i \\ w_i \mathbf{b}_i \end{pmatrix}$$

18

Zusätzliches Gewicht w nun dabei. Das ist einfach ein Gewicht für die Bernsteinpolynome. Der Nenner muss nur sein, weil die Summe der Gewichte wieder 1 ergeben würde und das tut sie nur, wenn man durch die Summe der Gewichte dividiert.

Es ist also eigl. wieder gleich wie bei den normalen Bezierkurven. Nur hat man dazu noch den Nenner.

Projektive Invarianz

Um eine rationale Bezierkurve in eine Perspektive zu transformieren, muss man nur die Kontrollpunkte mal der Projektionsmatrix rechnen. Das is noch besser als nur das Transformieren bei Bezierkurven und B-Splines.

Non Uniform Rational B-Splines = NURBS

Gleiches Prinzip.

A NURBS curve with respect to the control points $d_i,\ i = 0,\ldots,n$ and the knot vector $U = (u_0,\ldots,u_{n+k})$ is defined as

$$\mathbf{n}(u) = \frac{\sum_{i=0}^{n} w_i \mathbf{d}_i N_i^k(u)}{\sum_{i=0}^{n} w_i N_i^k(u)}, \quad u \in [u_0, u_{n+k}) \subset IR$$

The $w_i > 0,\ i = 0,\ldots,n$ are called weights.

Homogeneous representation :

$$\mathbf{n}_H(u) = \sum_{i=0}^{n} \mathbf{n}_{H\,i}\ N_i^k(u), \quad u \in [u_0, u_{n+k}) \subset IR$$

with the homogeneous B - Spline points

$$\mathbf{n}_{H\,i} = \begin{pmatrix} w_i \\ w_i \mathbf{n}_i \end{pmatrix}$$

20

Man hat wieder diese Gewichte dabei.

Non Uniform bezieht sich auf die u_i die müssen nämlich nicht gleichen Abstand haben.

Tensorproduktflächen

Ist ein Sweep. Man nimmt eine Kurve, bewegt die entlang einer zweiten Kurve und die Fläche ist dann die Tensorproduktfläche. Man nimmt die Kurve approximierend. Heißt man kann auch einfach die Kontrollpunkte bewegen.

- "A surface is the locus of a curve that is moving through space and thereby changing the shape"

  Given a curve

  $$\mathbf{f}(u) = \sum_{i=0}^{n} \mathbf{c}_i F_i(u), \quad u \in I \subset IR$$

  moving the control points yields

  $$\mathbf{c}_i(v) = \sum_{j=0}^{m} \mathbf{a}_{ij} G_j(v), \quad v \in J \subset IR$$

  Combining both yields a tensor - product surface

  $$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{a}_{ij} F_i(u) G_j(v), \quad (u,v) \in I \times J \subset IR^2$$

Man definiert sich also für jeden Kontrollpunkt eine Kurve und bewegt ihn anhand dieser.

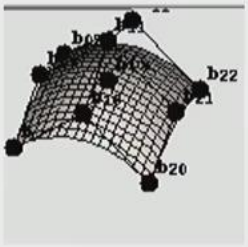
Konstantin Lackner 03.07.2020

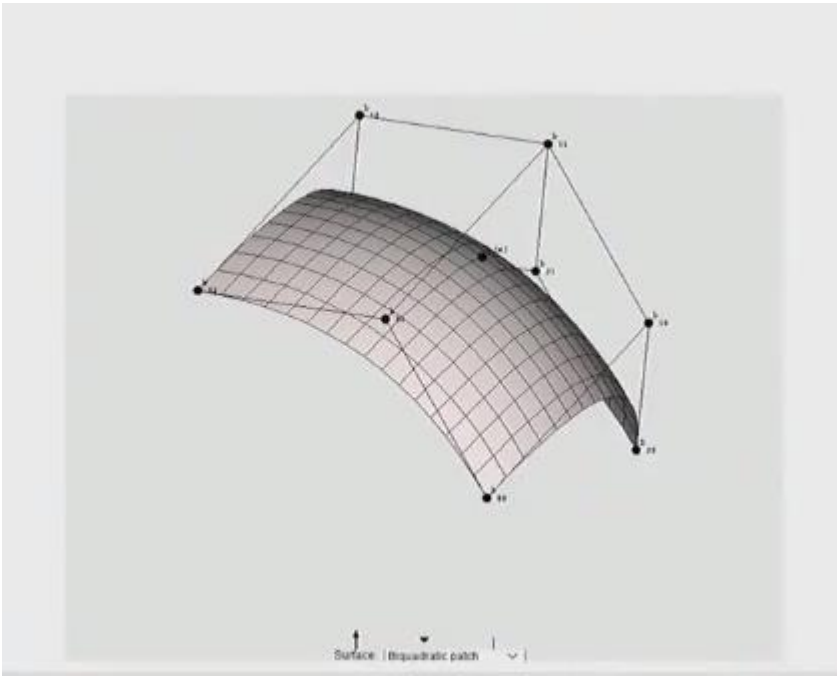
Tensorproduktbezierflächen

A tensor - product Bézier surface is given by

$$\mathbf{b}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{b}_{ij} B_i^n(u) B_j^m(v), \quad (u,v) \in I \times J \subset IR^2$$

The Bézier points $\mathbf{b}_{ij}$ form the control net of the surface.





Im Endeffekt sind das einfach in beide Richtungen Kurven…

TensorproduktBezierflächen

Man braucht für U und V Richtung Knotenvektoren.

Konstantin Lackner 03.07.2020

A tensorproduct B - spline surface with respect to the knot vectors

$U = (u_0,\ldots,u_{n+k}), V = (v_0,\ldots,v_{m+l})$

is given by

$$\mathbf{d}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{d}_{ij} N_i^k(u) N_j^l(v), \quad (u,v) \in [u_0, u_{n+k}) \times [v_0, v_{m+l}) \subset IR^2.$$

The control points $\mathbf{d}_{ij}$ form the control net of the surface.

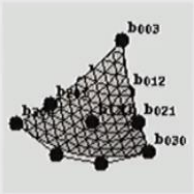
Properties and Algorithms are analogue to the description for Bézier tensor product surfaces.

Man kann auch den k Faktor da variieren. (hier hat man zusätzlich für die andere Richtung l)

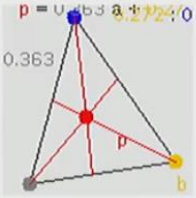
Bezier Triangles

A triangular Bézier patch is defined by

$$b(u,v,w) = \sum_{\substack{i+j+k=n \\ i,j,k>0}} \mathbf{b}_{ijk} B_{ijk}^n(u,v,w)$$

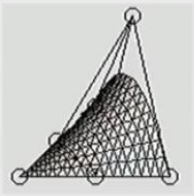


The u,v,w are barycentric coordinates of the triangular parameter domain.



Generalized Bernstein polynomials:

$$B_{ijk}^n(u,v,w) = \frac{n!}{i!\,j!\,k!} u^i v^j w^k$$



Hier wird mit Baryzentrischen Koordinaten gearbeitet. Man hat drei Parameter, u, v und w.

De Casteljau:

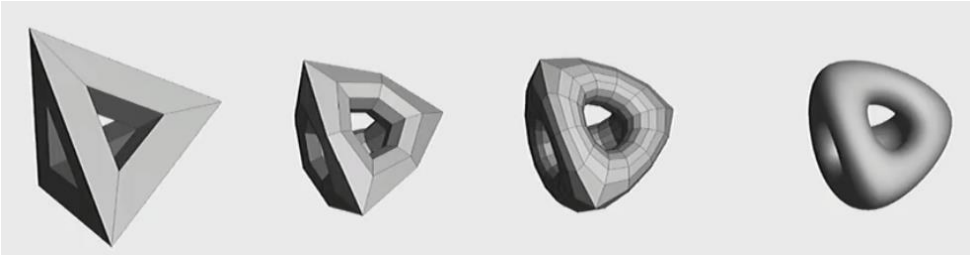
$$b_{ijk}^l = u b_{i+1\,jk}^{l-1} + v b_{ij+1k}^{l-1} + w b_{ijk+1}^{l-1}$$

Den De Casteljau macht man indem man einfach alle Dreieckspunkte nimmt und mit allen zusammen ein kleineres Dreieck errechnet.

Subdivision Surfaces

Konstantin Lackner 03.07.2020

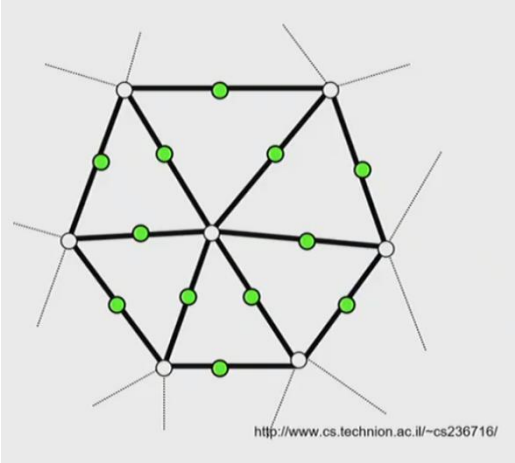
Man hat ein polygonales Netz und einen Iterativen Prozess bei dem man Ecken wegschneiden wie beim Schnitzen.



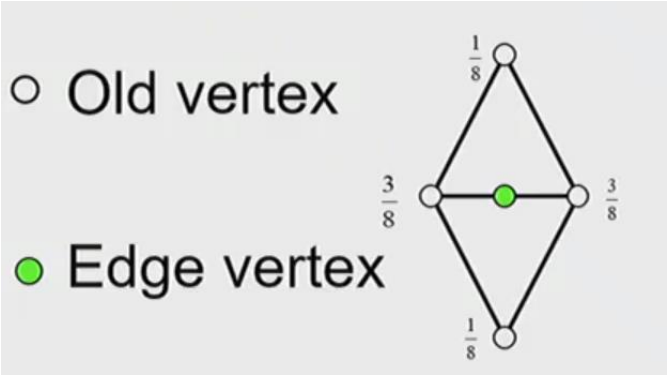
Konstantin Lackner 03.07.2020

Loop Schema

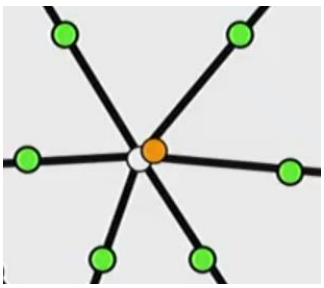
Man hat ein Dreiecksnetz, in einem Unterteilungsschritt macht man mehr Dreiecke. Man fügt an jeder Kante einen Eckpunkt hinzu

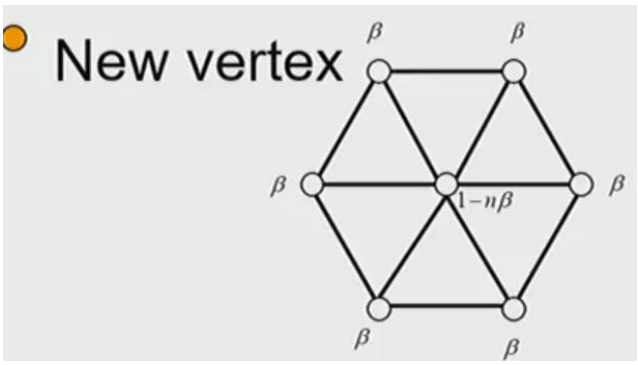


An einer normalen Kante hängen zwei Dreiecke an. Den Punkt sollte man stetes so setzen, dass er das gewichtete Mittel der umgebenden Punkte. Da die auf der Kante dem Punkt näher liegen, werden die größer gewichtet:



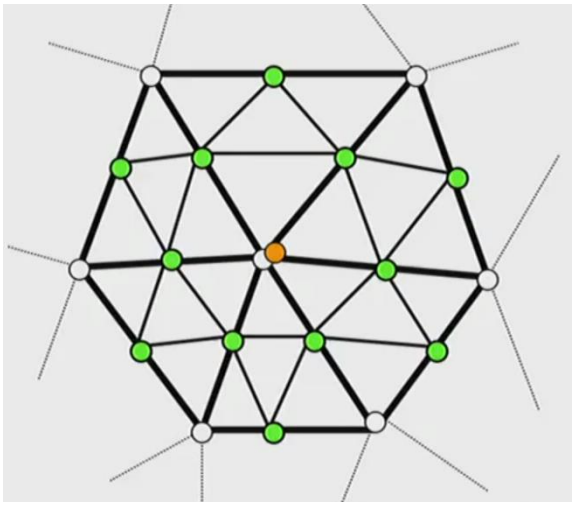
Jetzt hat man die grünen Punkte dabei, jetzt verschiebt man jeden zuvor existierenden Punkt ein bisschen indem man wieder die Umgebungspunkte nimmt und diese gewichtet.



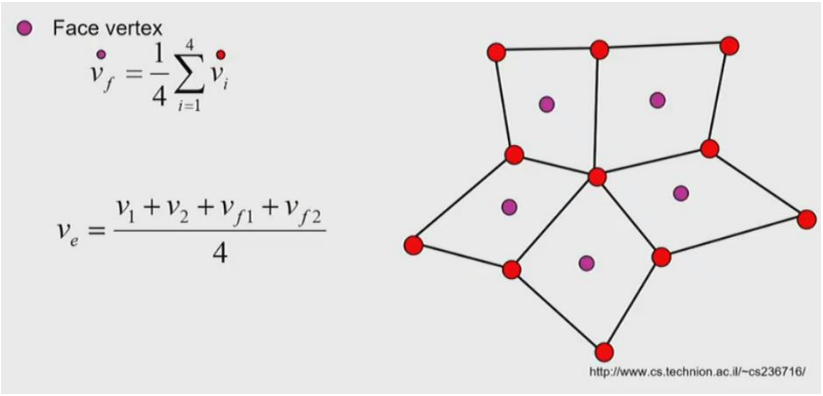


Dann fügt man neue Kanten zwischen den neuen Punkten ein:

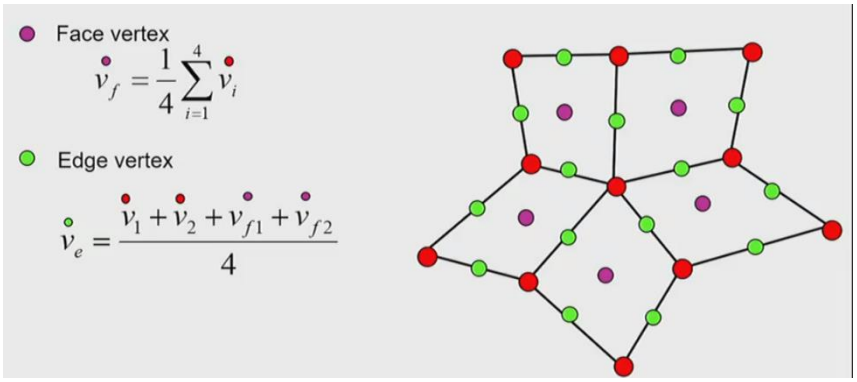
Konstantin Lackner 03.07.2020



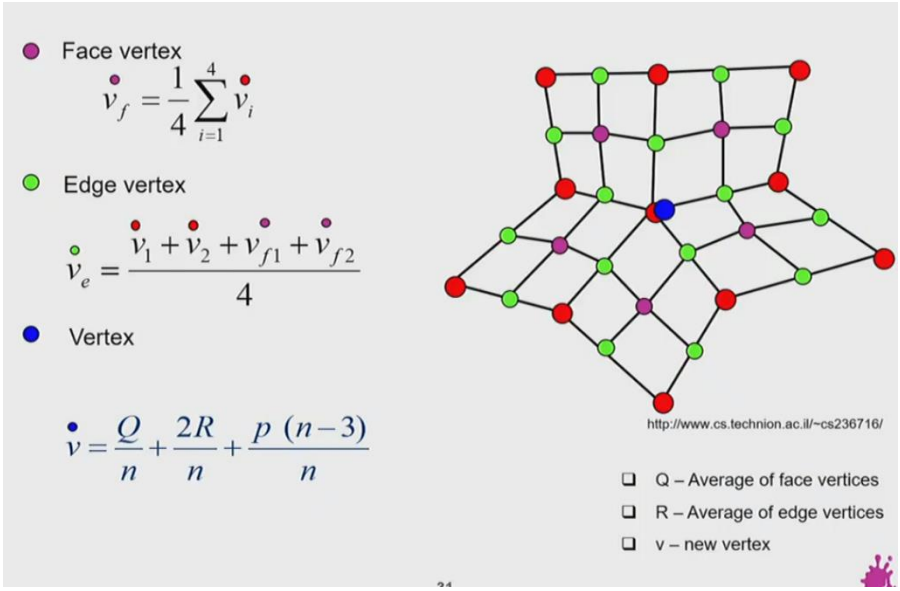
Catmull-Clark Scheme



Zu jedem Viereck ein gewichtetes Mittel als Punkt auf der Fläche berechnen. Dann zu jeder Kante einen Mittelpunkt. Der wird beeinflusst von den ursprünglichen roten Punkten und den zwei benachbarten neuen violetten.

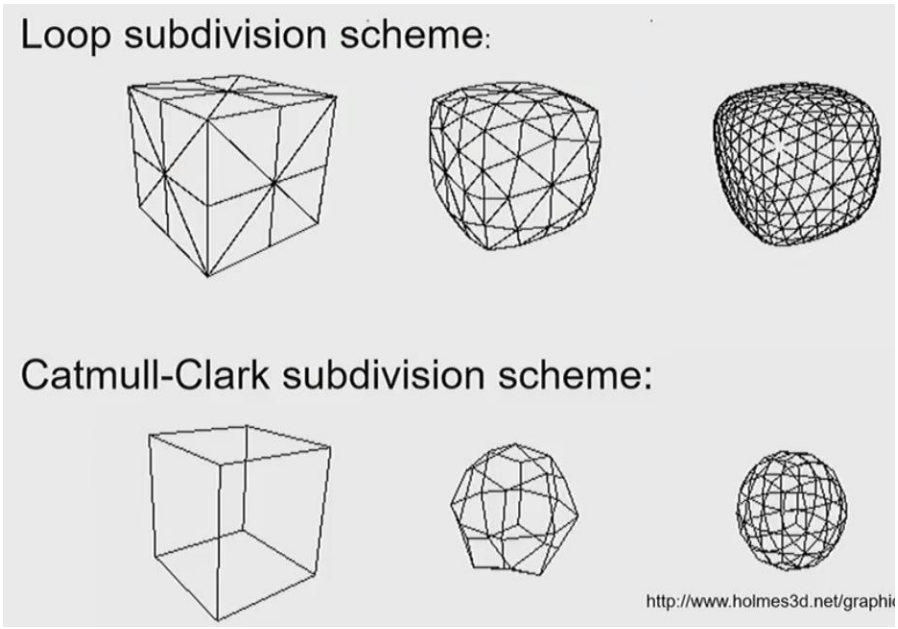


Dann müssen die ursprünglichen weggeschnitten werden, also verschoben hier. Man nimmt also den ursprünglichen Punkt mit großem Gewicht plus die grünen Punkte mit „R/n und dann noch mit noch weniger Gewicht die Flächenpunkte mit p(n-3)/n.



Dann noch Kanten rein und man hat wieder Vierecke.

Unterschied:



Subdivision Surfaces Classification

Typ des refinements:

Vertex insertion (beide besprochenen Methoden)

Corner cutting (Ersetzen von vertices)

Typ de generierten meshs

Triangular

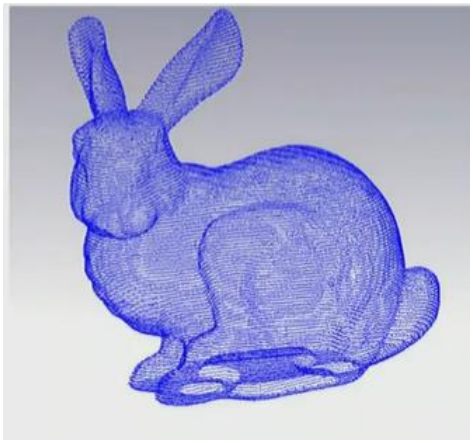
Quadrilateral

Konstantin Lackner 03.07.2020

Approximating vs. Interpolating

Point clouds

Ein Objekt wird von einer Menge an Punkten repräsentiert, die Punkte liegen auf der Oberfläche des Objektes.



Damit kann man auch Durchsichtigkeit simulieren. Vgl. Fliegengitter.

Transformationen gehen einfach, man kann einfach die Punkte linear transformieren. Kombinationen sind schwierig, man kann die Punktlisten natürlich vereinen, aber Durchschnitt und sowas is ned so easy.
Das Rendern geschieht einfach übers Zeichnen von Punkten am Screen.

Vorteile:

- Kein Rücksichtnehmen auf Nachbarschaft beim Rendern = sehr schnell. (billboarding)
- Realtiv genaues Abbild mit vielen Punkten.
- Leichtes Transformationen.

Nachteile:

- Man braucht viele Punkte für exakte Darstellungen.
- Hoher Speicherbedarf.
- Schlechte Kombinierbarkeit von Objekten.

Wireframemodell

Man hat ein Objekt, das besteht aus Ecken und Kanten. Man hat eine Kantenliste, jede Kante besteht aus zwei Einträgen für Ecken, diese Einträge sind aber nur Verweise auf eine Eckliste, wo alle Ecken einbaut sind.

Transformationen sind easy, weil man einfach die Punkte bewegt. Einfache Kombinationen sind wieder ok, weil man Ecken und Kantenlisten zusammenfügen kann, evtl. Kanten und Ecken zusammenfassen, wenn diese nahe genug beieinander sind. Beim Rendern werden einfach die Punkte auf ein Plane gezeichnet und dann die Kanten zwischen denen gezogen und gezeichnet.

Sichtbarkeitsberechnungen, Schattierungen und so sind nicht möglich.

Vorteile:

- Schnelle Transformationen.
- Schnelles Rendern.
- Man kann durch Digitalisierung ein Model erzeugen.

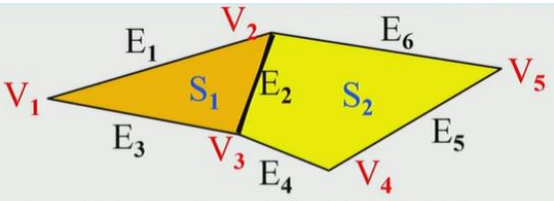
Nachteile:

- Ungenau (keine surfaces, keine occlusion)
- Nicht alle Kombinationen möglich.
- Kurven nicht genau darstellbar, braucht wieder viele Punkte.

Boundary Repräsentation (B-rep)

Ausweitung des Wireframemodells

Zusätzlich gibt es eine face list.



vertex list	edge list	face list
V_1: x_1 y_1 z_1	E_1: V_1 V_2	S_1: E_1 E_2 E_3
V_2: x_2 y_2 z_2	E_2: V_2 V_3	S_2: E_2 E_4 E_5 E_6
V_3: x_3 y_3 z_3	E_3: V_3 V_1	
V_4: x_4 y_4 z_4	E_4: V_3 V_4	
V_5: x_5 y_5 z_5	E_5: V_4 V_5	
	E_6: V_5 V_2	

Auf oberster Ebene eine Facelist aus Dreiecken. Darin wird auf die Kanten verwiesen, die wiederum verweisen jeweils auf zwei Ecken.

Muss man hier bsp. alle Kanten an einem Eckpunkt haben, muss man alle Kanten durchwandern.

Konstantin Lackner 03.07.2020

Winged Edge

Kante is das zentrale Element. Man speichert für jede Kante einen Pointer zu Start und Endpunkt, einen Pointer auf beide faces die anliegen und pointer zu den vorgehenden und nachfolgenden Kanten des faces.

Transformationen

Transformieren geht wie bei wire-frame.

Rendering

Hier kann man Verdeckung usw. anwenden.

Vorteile:

Gut zum Festlegen für Nachbarschaften.

Kann aus digitalisierten Dingen erstellt werden.

Easy Transformationen.

Nachteile:

Hoher Speicherverbrauch.

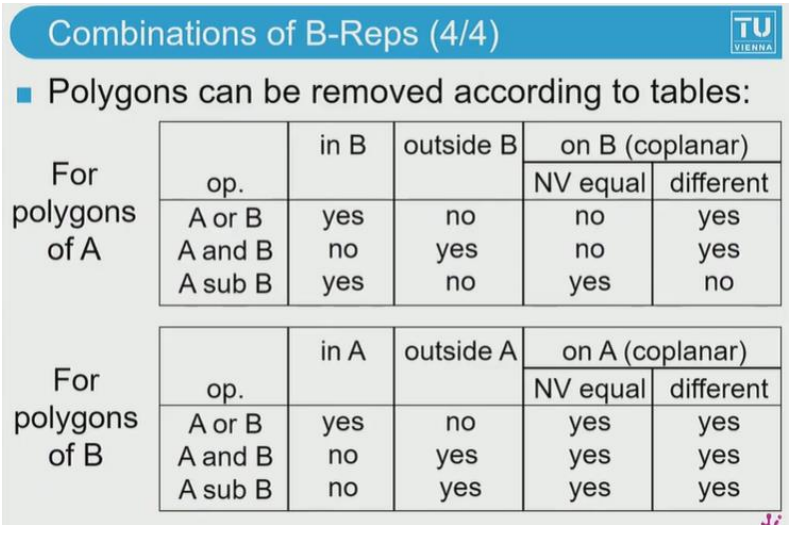
Kombination sehr teuer.

Kurven müssen approximiert werden.

Kombinationen

Man nimmt jedes Polygon und schaut, ob es in, außerhalb, bissl drinnen, oder genau auf der Oberfläche des anderen Objekts ist. Ragt etwas teilweise rein, unterteilt man es. Dann kann man, je nach Vereinigungsart, z.B. alle die drinnen sind wegwerfen. Die die drauf sind, sind wieder bissl schwierig.

Außerhalb und innerhalb wird festgestellt, indem man die Normalvektoren ansieht. Man schießt einen Strahl in Richtung Normalvektor und schneidet mit allen anderen Polygonen. Wenn man dann mit dem Strahl im zweiten Objekt ankommt und der Strahl in Gegenrichtung des Normalvektors schaut, ist man außen.



Voraussetzung für diese Operationen sind water tight Objekte, also solche, die keine Löcher haben. Konvexe Polygone sind auch eine Voraussetzung. Wenn man zwei konvexe Polygone schneidet, können nicht zwei Zipfel entstehen.

Partitioning von Object Surfaces

Hat man parametrisierte Flächen, kann man den Parameterbereich unterteilen und hat damit Unterteilungen. Wenn aber die Flächen nicht parametrisiert sind, braucht man Tesselation. Alle Polygonen werden einfach in kleinere Polygonen unterteilt. Alle bekommen einen neuen Normalvektor und wenn man dann zwei Normalvektoren nebeneinander vergleicht, und diese fast gleich sind, ist die Unterteilung schon fortgeschritten genug.

Binary Space Partitioning Tree

B-Reps sind ein aufzählender Ansatz, man beschreibt wo das Objekt ist, ein binary space partitioning tree beschreibt wie die Situation in einem gewissen Raumteil gegeben ist. Hier wird also ein Raum unterteilt.

Man nimmt ein beliebiges Polygon des Objekts, das liegt dann auf einer Ebene, diese Ebene teilt dann den Raum in dem das Objekt liegt. Das ist das Wurzelpolygon. Zu jedem Polygon kann man dann sagen, ob es vor oder hinter der Trägerebene liegt. Is es davor kommt es in den linken Teilbaum, dahinter in den rechten. Die die dazwischen sind, werden in zwei Teile geschnitten. Man hatte bspw. Glück und die Polygone wurden genau in die Hälfte geteilt. Aus jeder Hälfte nimmt man wieder ein Polygon und macht dasselbe. Dann hat man vier Gruppen jeweils mit 25% (reines Beispiel). Damit weiß man dann auch gleich für jeden Augpunkt, welche Polygonen welche verdecken.

Um am Anfang das ideale Polygon zu nehmen (Teilung in jeweils 50%) kann man ned exakt rechnen, das is zu aufwändig. Man nimmt einfach eine kleine Anzahl an Polygonen am Anfang, sag ma 10, und testet einfach amal wie gut geteilt wird.

Vorteile:

- Schnelles Rendern.
- Schnelle Transformation – wieder nur manche.
- Schnellere Kombinationen als bei B-Reps.

Nachteile:

- Kurven schwer darzustellen.
- Nur konvexe Polygonen, weil nicht konvexe würden evtl. mehr als zwei Geben beim Teilen.
- Hohe Speicherkosten.

Abarbeiten

Painter's Algorithm

Von hinten nach vorne AUSGEBEN. Man hat einen Augpunkt und schaut zuerst bei der Wurzel, ob der vor oder hinter der Trennebene liegt (A+ is davor, A- dahinter). Liegt er davor, zeichnet man zuerst alle Polygonen dahinter, liegt er dahinter, zeichnet man zuerst alle von vorne, dann jeweils die Wurzel und dann das, was noch nicht gezeichnet wurde.

```
IF eye is in front of a (in A+)
THEN BEGIN draw all polygons of A-;
     draw a;
     draw all polygons of A+ END
ELSE BEGIN draw all polygons of A+;
     (draw a);
     draw all polygons of A- END;
```

Dadurch stellt man sicher, dass man nie zwei Polygonen aus verschiedenen Teilbäumen vergleichen muss. Das muss natürlich alles dann für alle darunter liegenden Teilbäume auch gemacht werden.

Transformationen

Affine Transformationen auf Punkte sind natürlich easy. Da transformiert man einfach die Punkte.

Kombinationen

Möglichkeit 1

Aus beiden BSP Bäumen B-reps machen und dann kombinieren, danach wieder einen BSP Baum draus machen.

Möglichkeit 2 (besser)

Man nimmt einen BSP Baum her und sortiert den zweiten in den ersten ein. Dabei versucht man möglichst viel Info des ersten BSP wiederzuverwenden.

- A or B homogeneous (full or empty)
  ⇒ simple rules
- Else:
  - 1. Divide root polygon a of A at object B in a_{in}, a_{out}
  - 2. Root node c of C: if op="and" then c:=a_{in} else c:=a_{out} (with its plane)
  - 3. Divide B at plane of a in B_{in}, B_{out}
  - 4. Recursive evaluation of the subtrees:

    $C_{left}=A_{out}$ op B_{out} $C_{right}=A_{in}$ op B_{in}

Wenn bspw. A oder B leer, is es natürlich easy. Ansonsten…

C ist das Ergebnis aus beiden Bäumen. Für C braucht man auch eine root. Für diese Wurzel nimmt man als Trägerebene dieselbe wie die bei Wurzel von A. Bei A hat man bei der Trägerebene ein Polygon a. Man schaut sich jetzt a an und schaut, ob das innerhalb, außerhalb oder teilweise innerhalb von B ist. In Bezug auf B ist dann a zerlegt in zwei Teile a in und a out – also der Teil von a der in B liegt und der der außerhalb liegt. Dann schaut man, was für eine Operation man machen will.

Bei „and“ – kann man als Polygon c für die Trägerebene von C einfach ain nehmen.

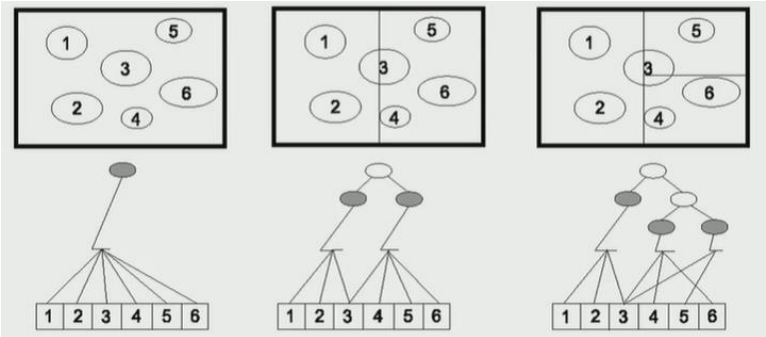
Bei „oder“ – c = a out

op	A	B	A op B
or	inhom.	full	full
	inhom.	empty	A
	full	inhom.	full
	empty	inhom.	B
and	inhom.	full	A
	inhom.	empty	empty
	full	inhom.	B
	empty	inhom.	empty
sub	inhom.	full	empty
	inhom.	empty	A
	full	inhom.	–B
	empty	inhom.	empty

kD Baum

= k-dimensionaler Baum

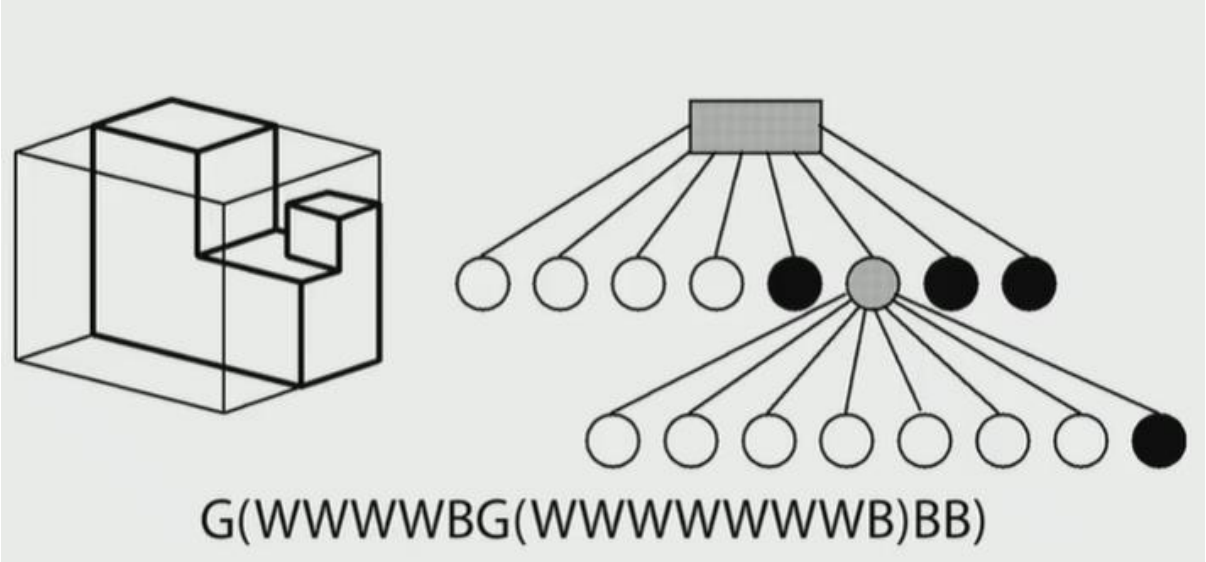
Ist ein Sonderfall des BSP Baums. Man kann mit der Trennebene beliebig verfahren. Wenn man keine Trägerebene des Polygons nimmt, sondern diese achsenparallel nimmt, also bspw. x=5 als Ebene, dann hat man einen einfachen Koordinatenvergleich um festzustellen, wo ein Punkt liegt. Die Unterteilungsebenen sind also bei einem kD Baum immer achsenparallel.



Konstantin Lackner 03.07.2020

Octree

Ist eine Spezialform des kD Baums. Man teilt einfach sofort 3 mal auf in X, Y und Z Richtung und immer in der Mitte. Das ergibt dann aus einem Würfel 8 Teilwürfel. (2^3)



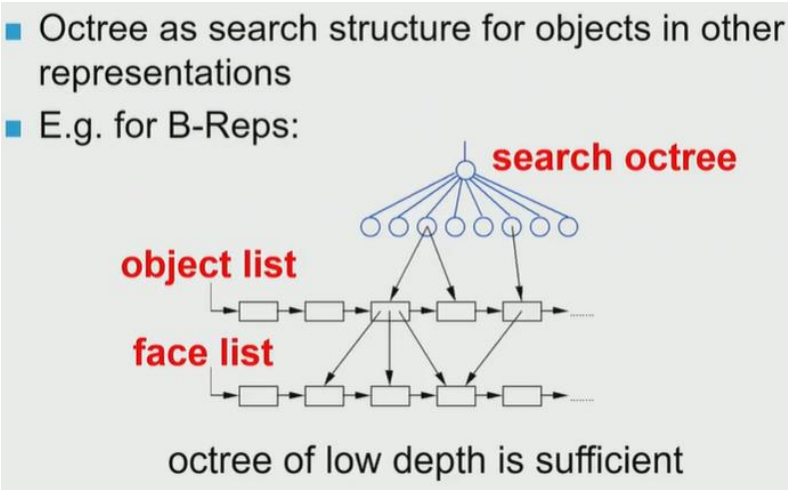
Operationen mit Octrees

Transformation ist schwer zu implementieren, bis auf bspw. Rotation.

Kombinationen sind leicht aber die Octrees müssen aneinander angepasst werden.

Beim Rendering wird zuerst der subspace gerendert, der am weitesten weg ist vom Viewer.

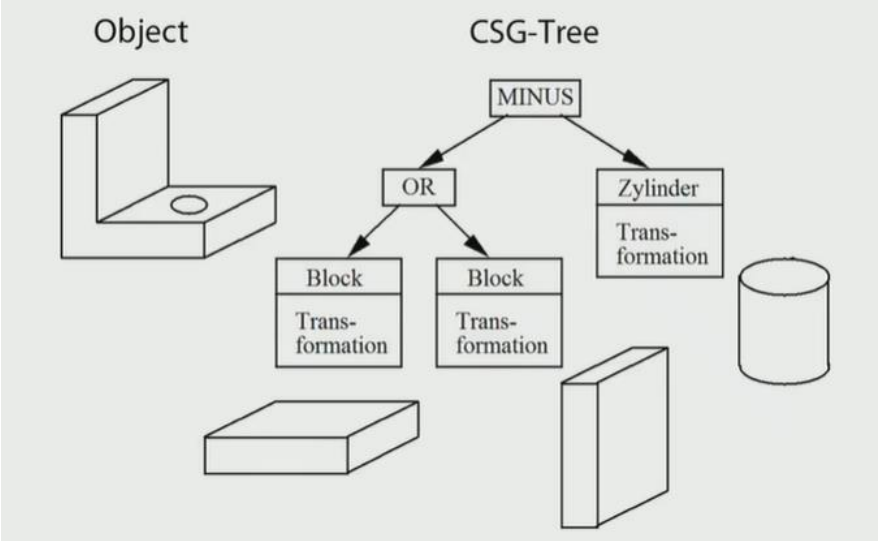
Octrees können auch als einfache Suchstruktur benutzt werden



Konstantin Lackner 03.07.2020

Constructive solid geometry

Man hat einen Baum mit Zwischen und Endknoten.



Transformationen

Die Transformation muss einfach auf alle Objekte drunter angewandt werden.

Combinations

Einfach die beiden Objekte als Children in einen neuen Baum.

Rendering

Muss zu einer B rep transformiert werden oder via raytracing gerendert werden

Vorteile:

Wenig Speicherplatzbedarf.

Kombinationen und Transformationen sind easy.

Objekte können exakt dargestellt werden.

Durch die Baumstruktur ist die Suche sehr einfach und schnell.

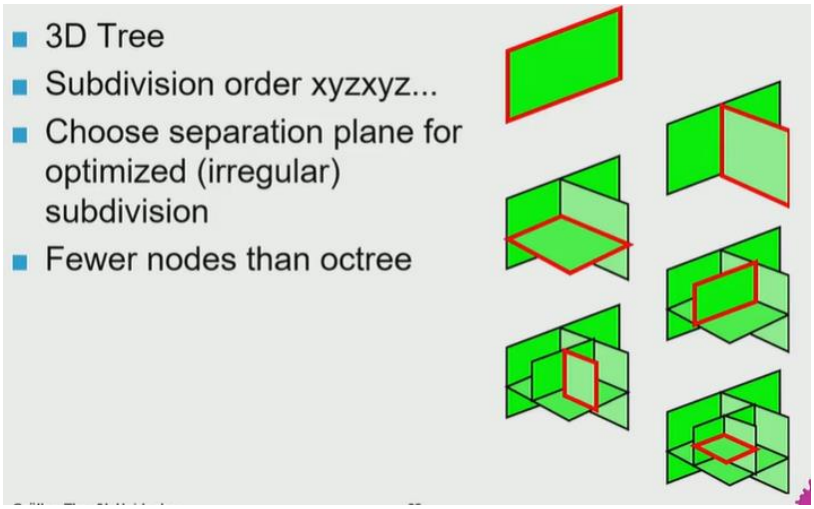
Nachteile:

Kann nicht direkt gerendert werden.

Kann nicht durch Digitalisierung erzeugt werden.

Bintree

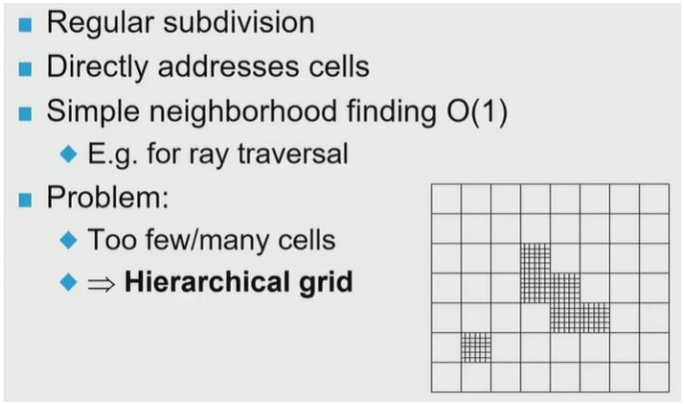
Ist ein kD Baum bei dem die Unterteilungsreihenfolge vorgegeben ist. Also immer zuerst x, dann y, dann z (bspw.)



Ist wie ein Octree, nur dass man nicht immer die 3 Unterteilungen machen muss.

Grid

Man unterteilt den Raum in ein regelmäßiges Gitter. Entweder als Suchstruktur, oder für Objekte. Das macht natürlich Kollisionsbehandlung einfach. Man kann durch richtige Einteilung auch mit konstanten Suchaufwand durchgehen, hat man bspw. 10 Elemente in jedem Element.

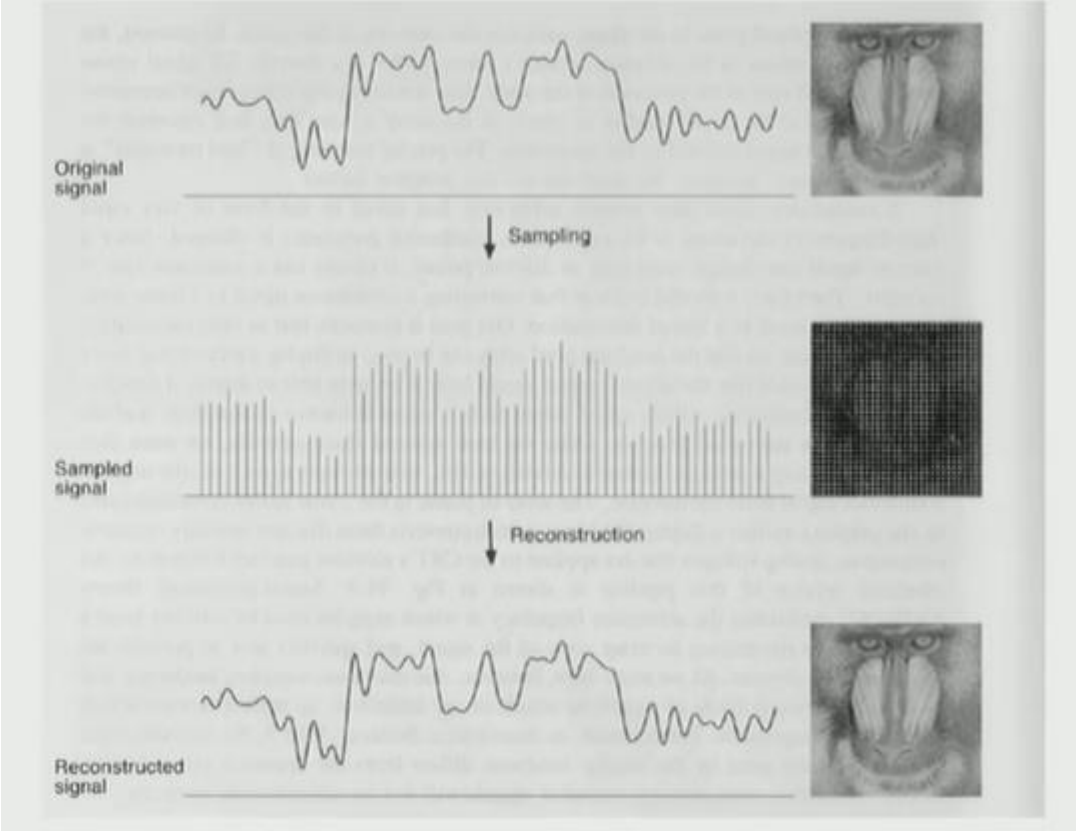


Sampling

Ich hab ein kontinuierliches Signal und stelle das durch ein paar diskrete Werte dar.

Rekonstruktion

Ich hab diskrete Werte und will wieder ein kontinuierliches Signal rausbekommen.



Hier wird in regelmäßigen Abständen gesamplet. In diesem Fall hat man ein Abtastraster, an diesen Stellen hat man dann den Helligkeitswert.

Man kann zum Rekonstruieren bspw. nearestNeighbour machen, also einfach den nächstgelegenen Wert nehmen.

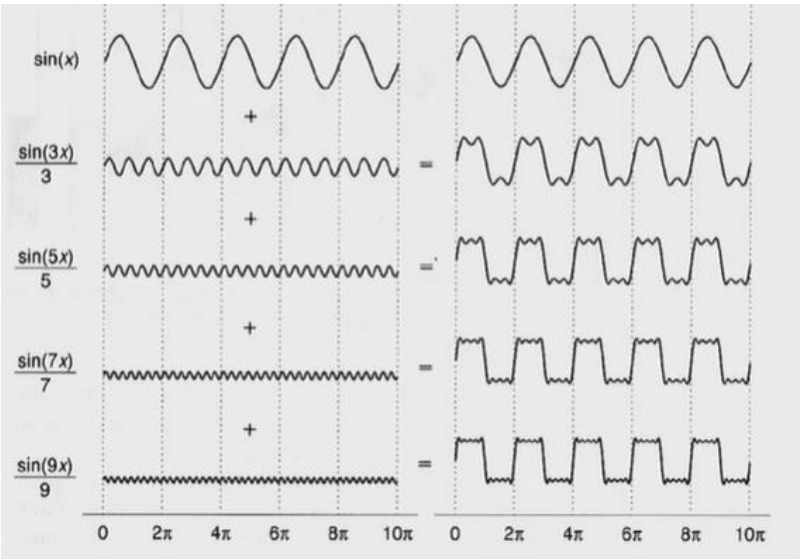
Zusammenhang zwischen Komplexität des Signals und Abtastrate (=Wie viele Punkte des Gesamten muss ich nehmen) das beschreibt das Samplingtheorem.

Wenn man beim Abtasten Fehler macht, nennt man das Aliasing – das Beheben demnach Antialiasing.

Sampling Theorie

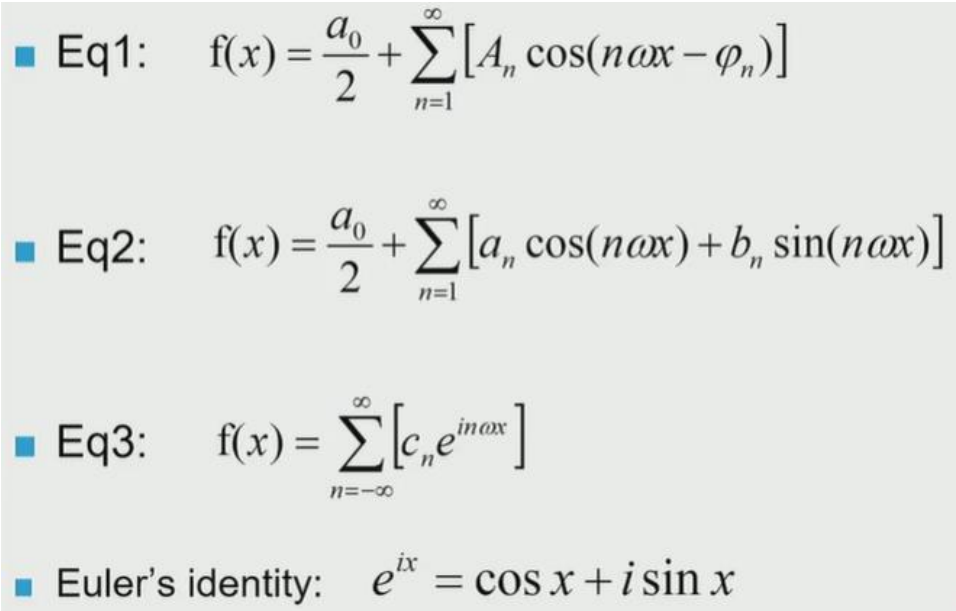
Man betrachtet das Signal entlang einer Scanline, hat also ein eindimensionales Signal. Das Signal ist auf einer Achse aufgelistet und horizontal hat man die Intensität. Um in den Frequenzraum zukommen, muss man das Signal als Kombination von Sinuswellen modellieren. Bei Sinuswellen hat man Frequenz und Amplitude (außerdem Phase = Verschiebung am Anfang) Die Fouriertransformation macht genau das. Sie wandelt ein Signal in eine Kombination von Sinusfunktionen um.

Konstantin Lackner 03.07.2020



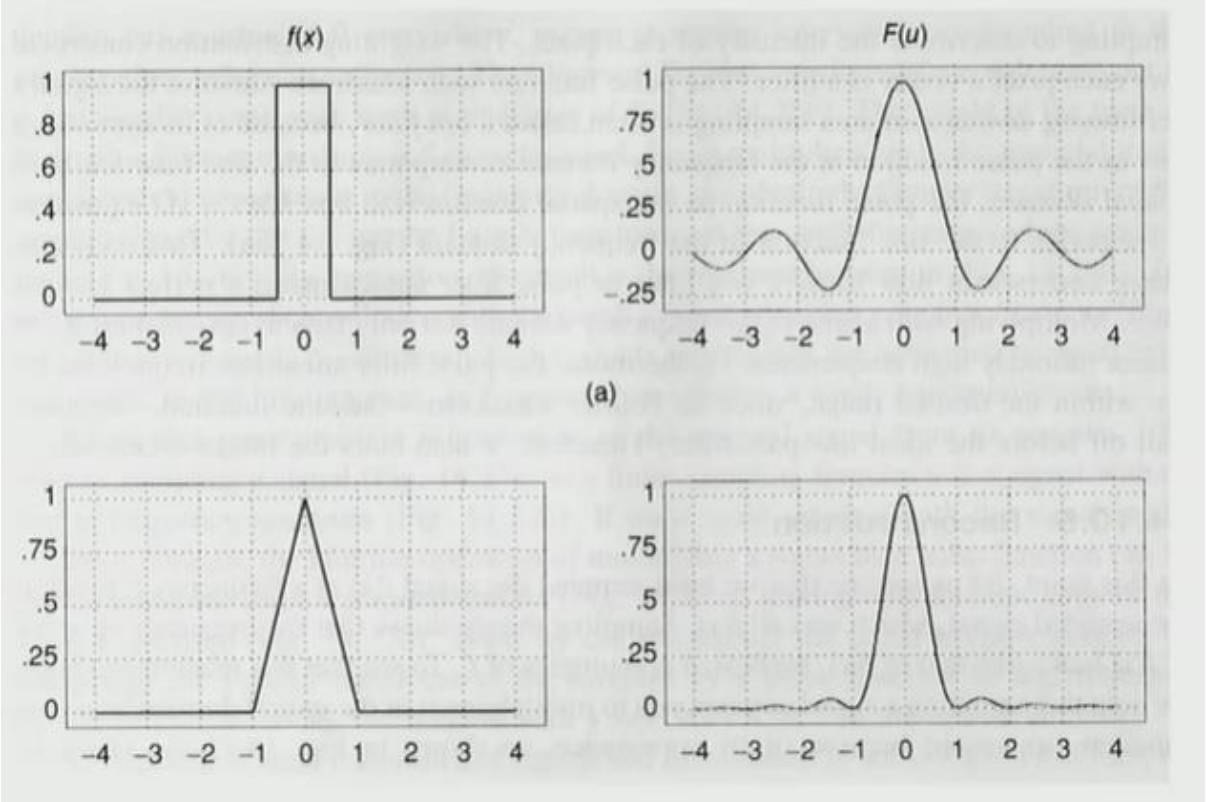
Man nimmt die Grenzfrequenz des höchstfrequenten Sinus als Zeichen für die Komplexität. Das Samplingtheorem fragt dann, wie viele Abtastpunkte man für einen Sinus braucht. Für eine Periode eines Sinus braucht man eine Spur mehr als zwei Abtastpunkte. Man macht also auf ein Signal eine Fouriertransformation, bekommt damit eine Kombination aus Sinüssen und dann nimmt man einfach eine Abtastrate die bisschen mehr als doppelt so groß ist wie die für den höchstfrequenten aus der Sinussumme.

Fouriertransformation

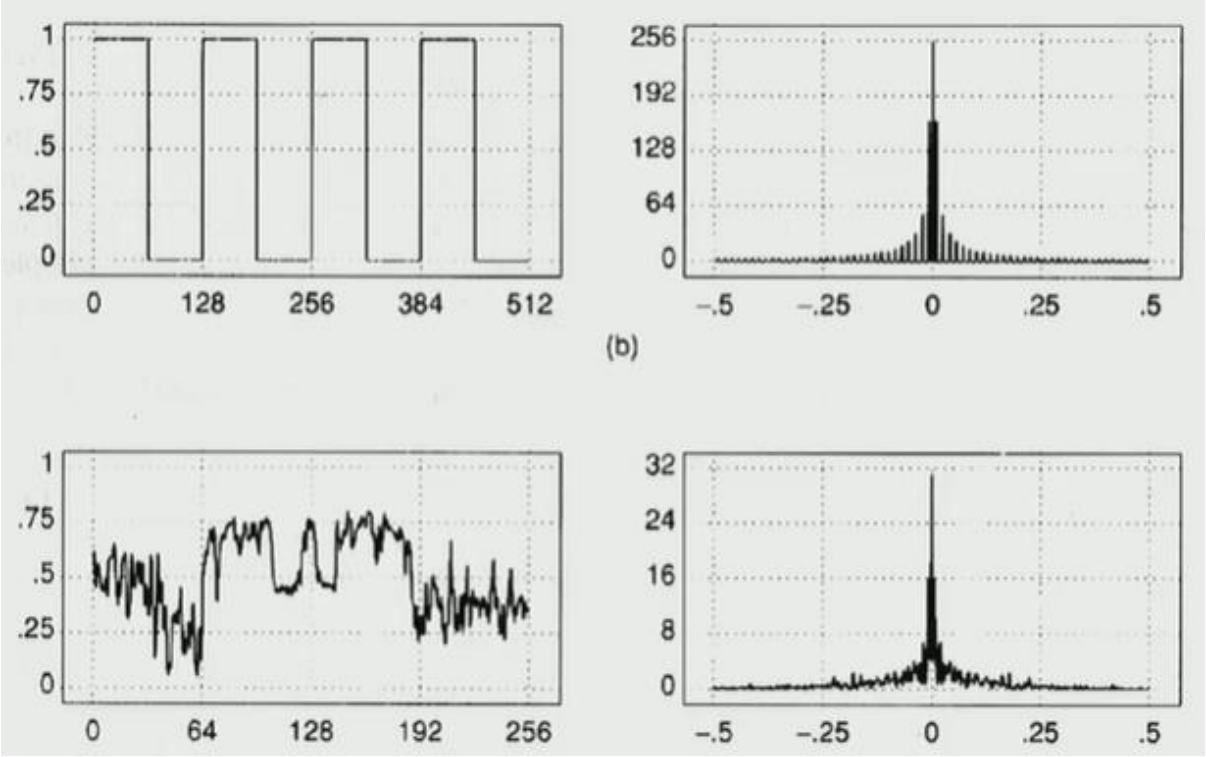


Verschiedene Schreibweisen. Die letzte ist die häufigste.

Man geht vom Ortsraum aus und kommt in den Frequenzraum. Man hat also im Ortsraum x = Scanline, f(x) = Helligkeit
Im Frequenzraum dann hat man auf der horizontalen Achse, die entspricht der Frequenz, die Frequenzachse, vertikal Amplituden und Phaseninformation.



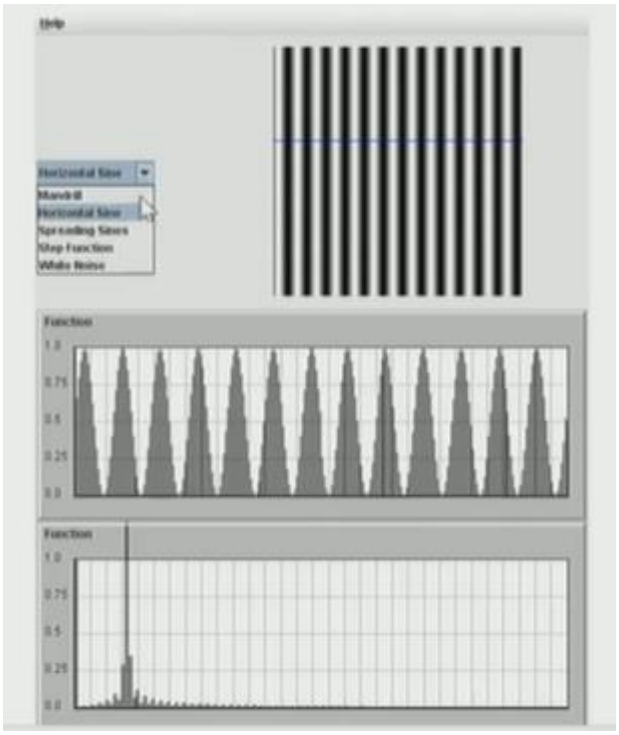
Man hat links den Ortsraum, rechts den Frequenzraum.

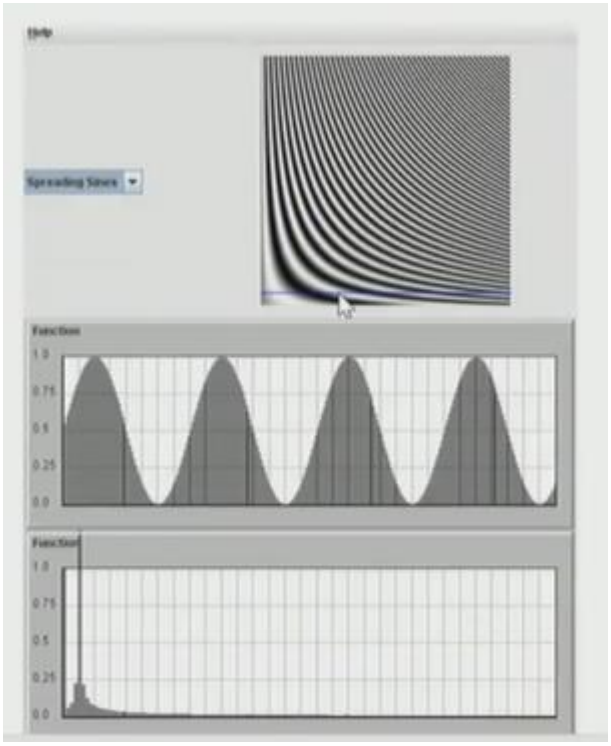


Links Ortsraum, rechts Frequenzraum.

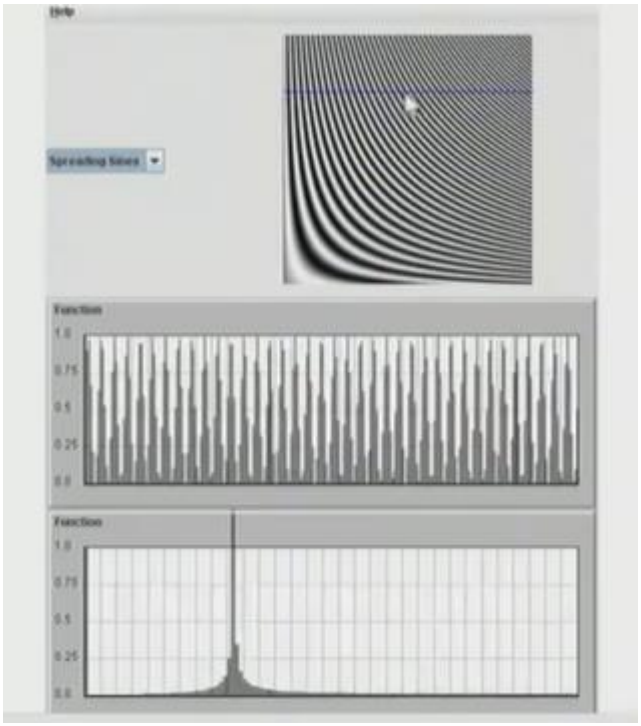
Hat man etwas periodisches im Ortsraum, bekommt man im Frequenzraum ein diskretes Spektrum = zwischen zwei Frequenzen immer ein Frequenzbereich für den man keine Sinüsse braucht.

Konstantin Lackner 03.07.2020



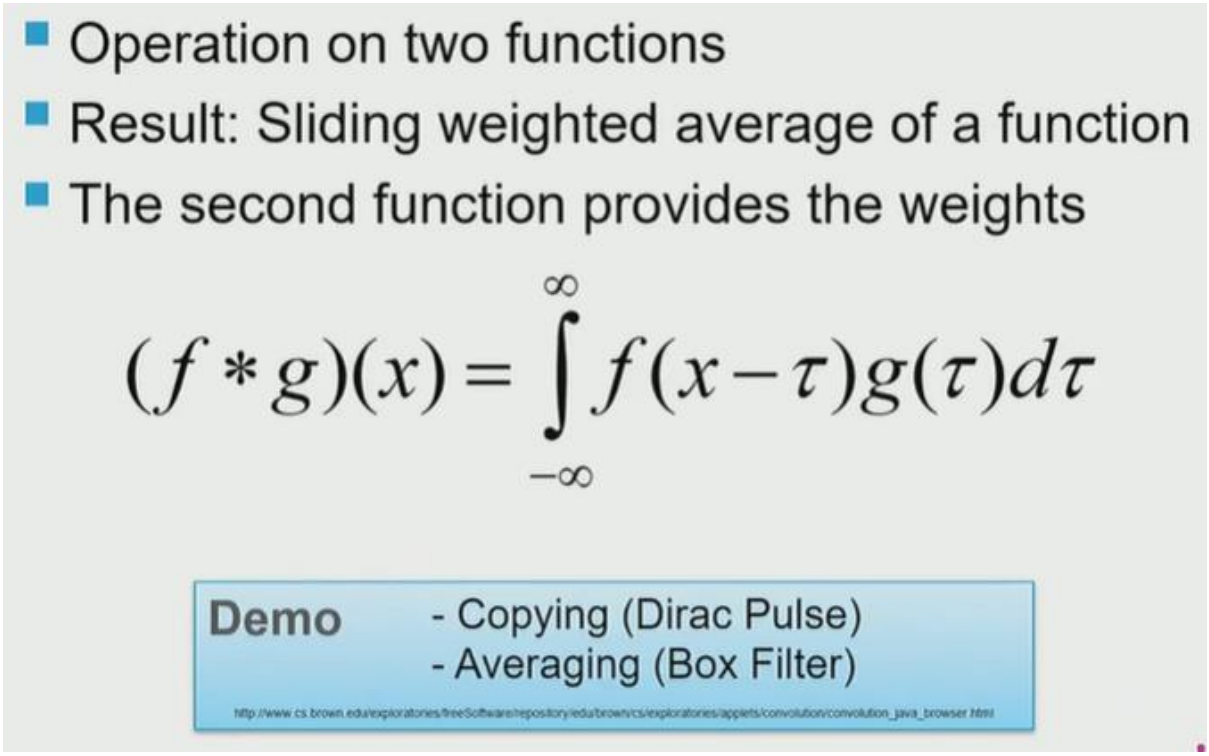


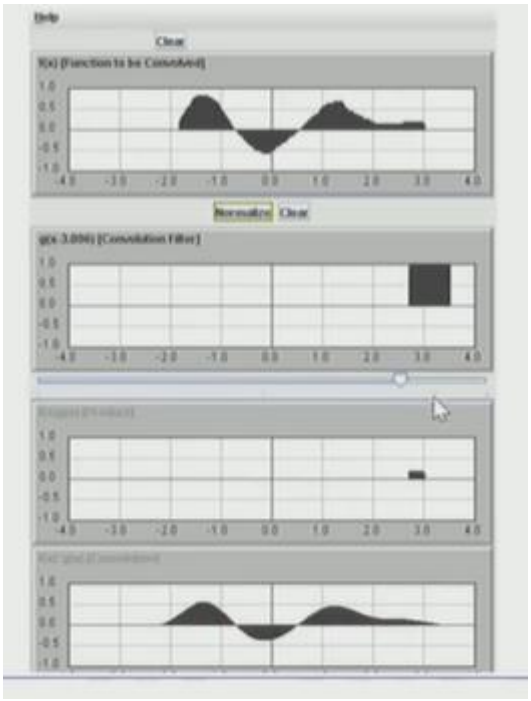
Konstantin Lackner 03.07.2020

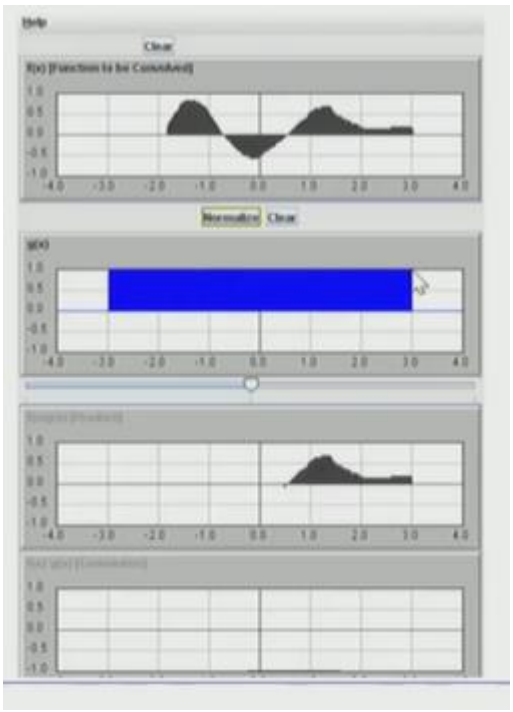


Faltung

Zwei Funktionen, zwei Signale, das erste f(x) das zweite spielt Rolle der Gewichtsfunktionen. Man nimmt also das Signal f her, nimmt einen gewissen Bereich, rechne ein neues Signal aus, das ist das gefaltete, im Endeffekt einfach ein Mittelwert.







Das glättet das Signal.

Man fährt mit der G Funktion, der Maske, über das F Signal drüber und macht eine Mittelwertsberechnung.

Faltungstheorem

Wenn man im Ortsraum zwei Signale hat und diese faltet, hat man im Frequenzraum eine Multiplikation der dazugehörigen Spektren.

Andersrum, wenn man im Frequenzraum eine Faltung macht, hat man im Ortsraum eine Multiplikation.

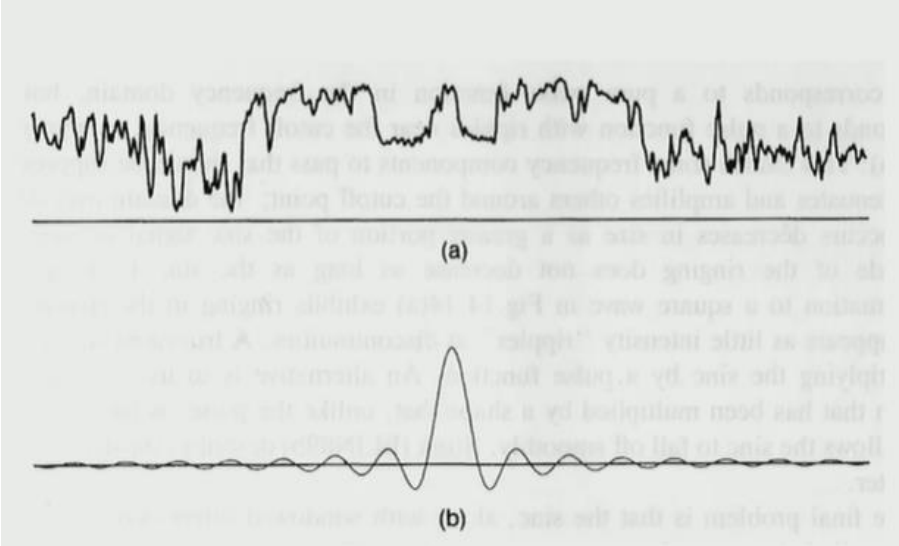
$$f_1 * f_2 \equiv F_1 F_2$$
$$F_1 * F_2 \equiv f_1 f_2$$

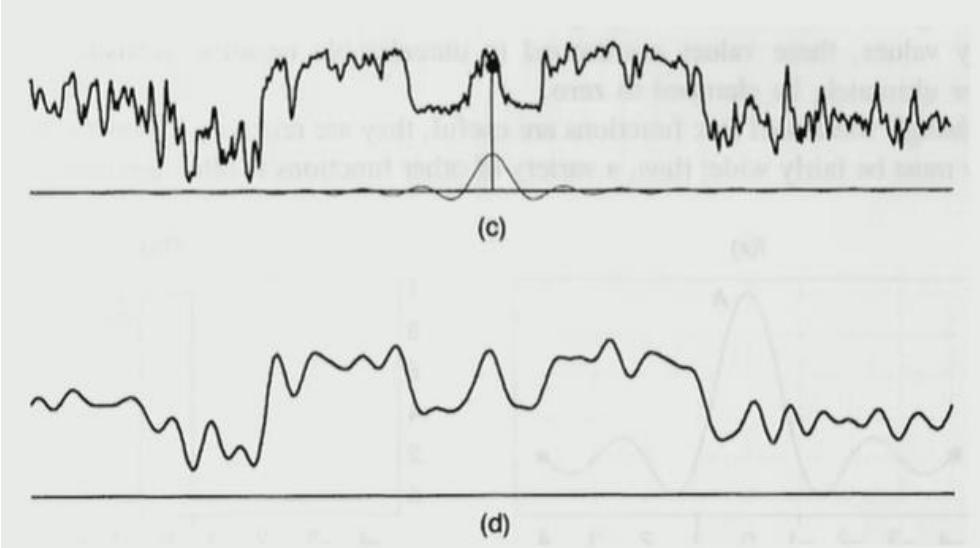
Low-Pass Filter

- Low-pass filtering performed on Mandrill scanline
- **Spatial domain**: convolution with sinc function
- **Frequency domain**: cutoff of high frequencies - multiplication with box filter
- *Sinc function corresponds to box function and vice versa!*

Im Frequenzraum multipliziert man mit einer Rechtecksfunktion. Alle Frequenzanteile im Rechteck bleiben erhalten, die anderen werden weggehaut. Im Ortsraum muss man demnach falten, mit der fouriertransformierten Funkton des Rechtecksfilter, weil, siehe oben Faltungstheorem.

Man bekommt dann eine sinc Fuktion für den Ortsraum raus:





Das Resultat ist dann ein geglättetes Signal.

Sampling

Man fährt mit einem „Kamm“ drüber und tastet in regelmäßigen Stellen ab. Das funktioniert durch das Multiplizieren mit einer Kammfunktion:

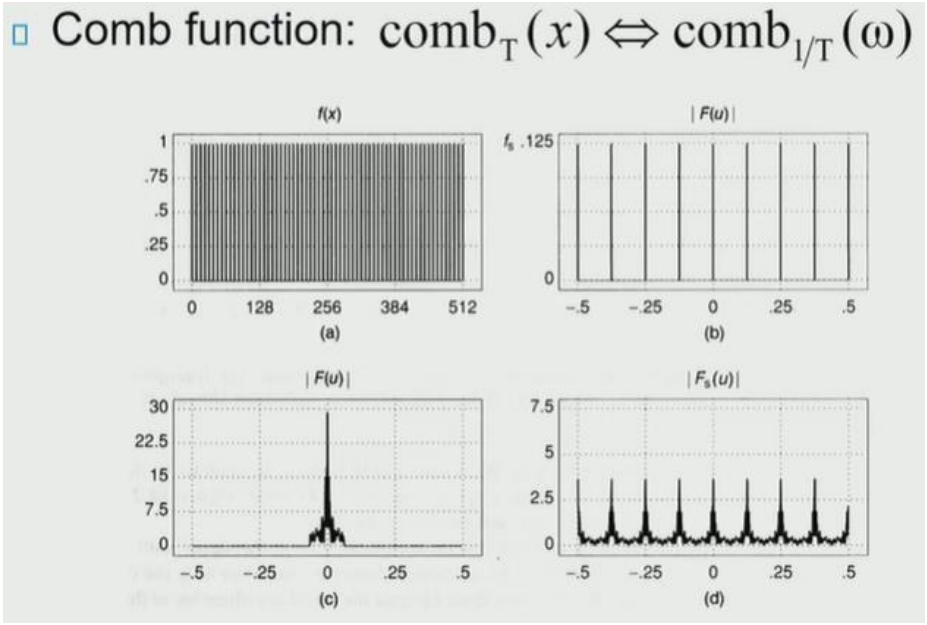
- The process of sampling is a multiplication of the signal with a comb function

$$f_s(x) = f(x) \cdot \mathrm{comb}_T(x)$$

- The frequency response is convolved with a transformed comb function.

$$F_s(\omega) = F(\omega) * \mathrm{comb}_{1/T}(\omega)$$

Das Frequenzspektrum der abgetasteten Funktion fs(x) bekommt man auch wieder durch eine Fouriertransformation. Die Fouriertransformation einer Kammfunktion ist wieder eine Kammfunktion. In der Transformierten ist der Abstand zwischen der Abtastung nicht mehr T wie vorher sondern 1/T.
Wenn man also beim Abtasten das T zu groß wählt, ist im Frequenzbereich dann das 1/T zu klein und man hat evtl. Probleme später.

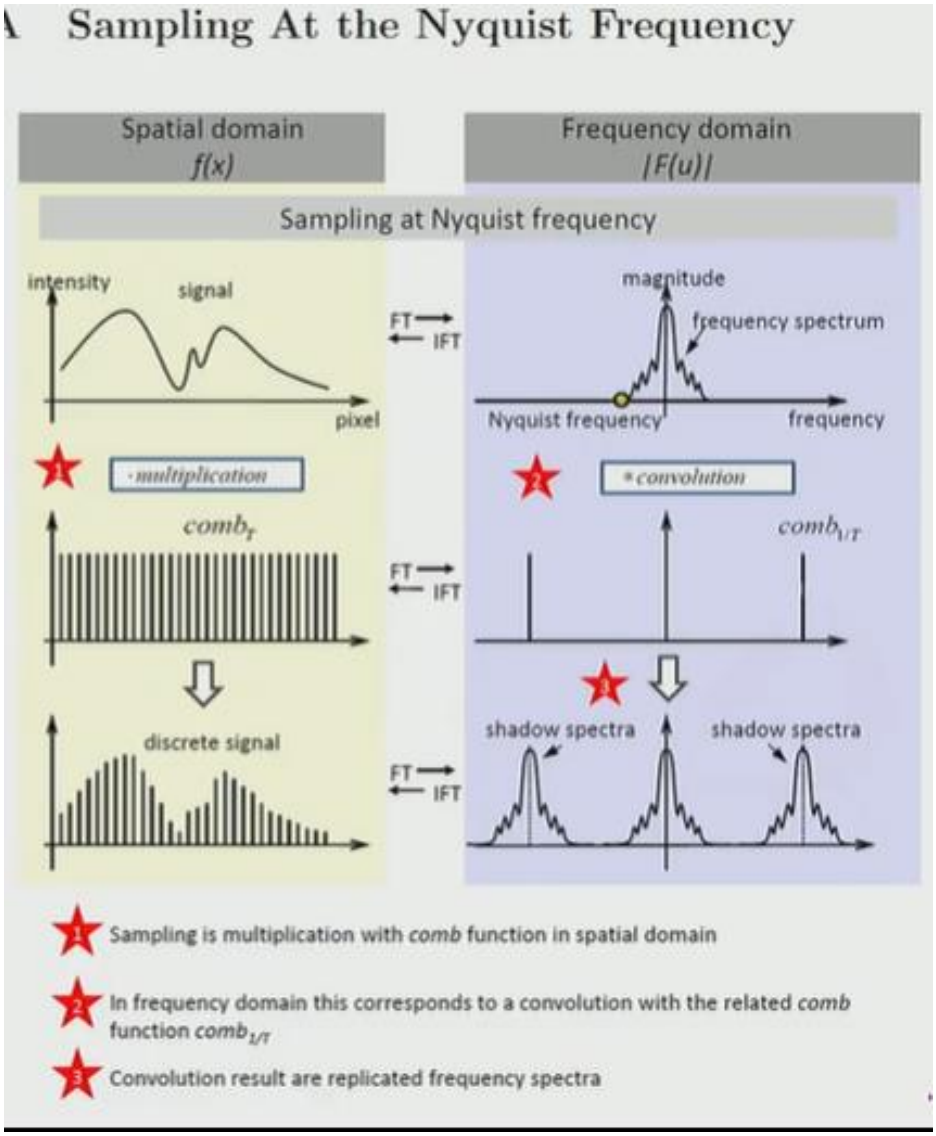


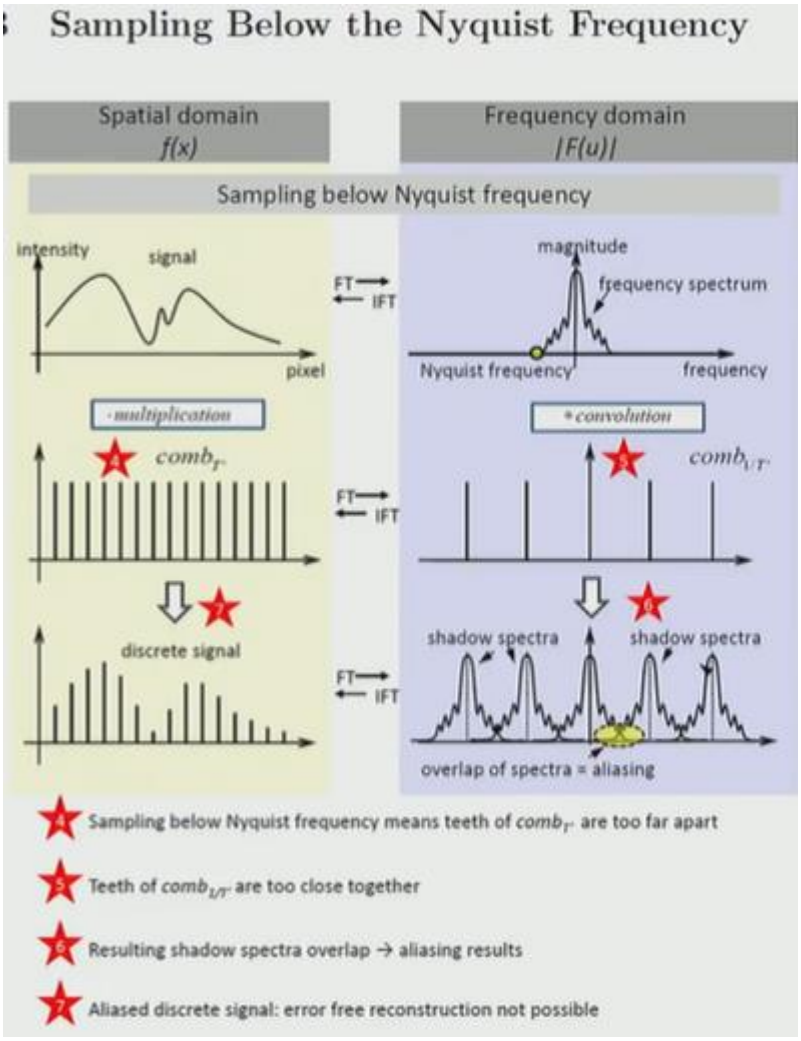
Im Frequenzraum passiert beim Sampling das hier. Die ursprüngliche Funktion wird für jeden Kamm repliziert. Wenn der Abstand der Kämme groß genug ist, gibt es keine Überlagerung, wenn der aber zu klein ist, überlagern die Frequenzen sich und werden vermischt. Wenn man also den Abstand im Ortsraum zu groß wählt, überlappen sich die Frequenzen im Frequenzraum dann und dadurch entsteht Aliasing. (Herr Gott das hat dauert, oida…)

Das gibt einem auch Auskunft, wie man seine Samplerate wählen muss. Sie müssen eben genug Abstand haben, dass die Frequenzen sich nicht überlagern, aber auch nicht unbedingt mehr.

Das Spektrum bei 0 in der Mitte heißt Originalspektrum, die kopierten darum heißen Schattenspektra.

Eine exakte Rekonstruktion bedeutet, dass man sein ursprüngliches Spektrum wieder bekommt. Man möchte also einfach alle Schattenspektra wegwerfen, also multipliziert man einfach mit einer Rechtecksfunktion die genau so groß ist, dass nur das Originalspektrum in der Mitte bleibt. Im Frequenzraum ist es eine Multiplikation mit der Rechtecksfunktion, im Ortsraum daher eine Faltung mit der Sincfunktion. Das ist gut, weil man damit mal eine Methode hat um es genau zu bekommen, schlecht, weil die Sincfunktion unendlich ist und das Rechnen damit natürlich eigl. nicht möglich. Man schaut also wie weit man davon weg ist mit einer praktikablen Funktion und kann sich damit den Fehler errechnen.


Sampling At the Nyquist Frequency
Spatial domain
f(x)
Frequency domain
|F(u)|
Sampling at Nyquist frequency
intensity
signal
pixel
FT
IFT
magnitude
frequency spectrum
Nyquist frequency
frequency
·multiplication
*convolution
comb_T
comb_1/T
discrete signal
shadow spectra
shadow spectra
1 Sampling is multiplication with comb function in spatial domain
2 In frequency domain this corresponds to a convolution with the related comb function comb_1/T
3 Convolution result are replicated frequency spectra

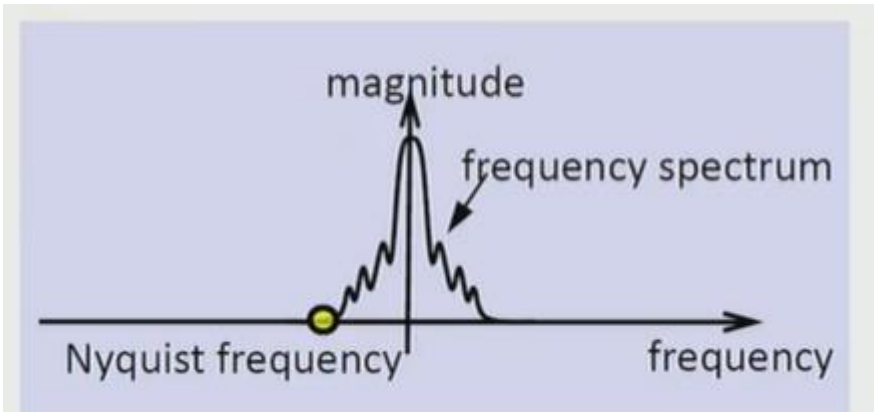


Hier ist der Abstand im Objektraum zu groß – im Frequenzraum überlagern sich die Frequenzen.

Bandlimitiert

Eine Funktion ist bandlimitiert, wenn sie keine Frequenzen außerhalb des Intervalls [-u, u] besitzt. Dabei ist u die Bandweite der Funktion.

Die Nyquist Frequenz einer Funktion ist ihre doppelte Bandweite, also w = 2u.

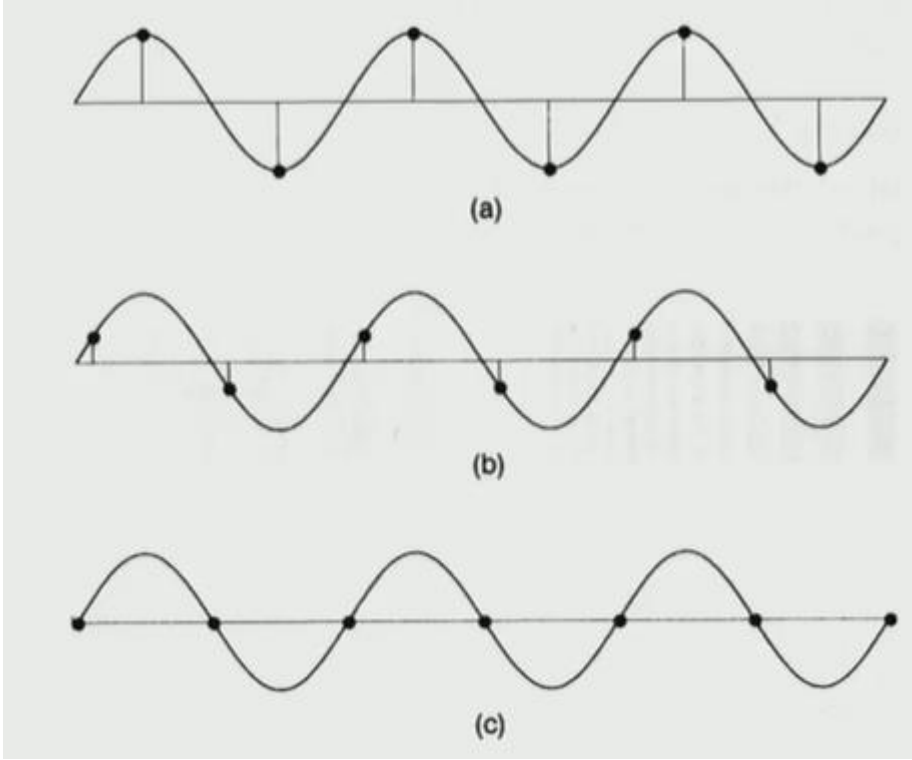


Konstantin Lackner 03.07.2020

Wenn man eine Funktion hat die bandlimitiert ist und die Abtastrate ist über dieser Nyquist Frequenz, dann kann man sie genau rekonstruieren.

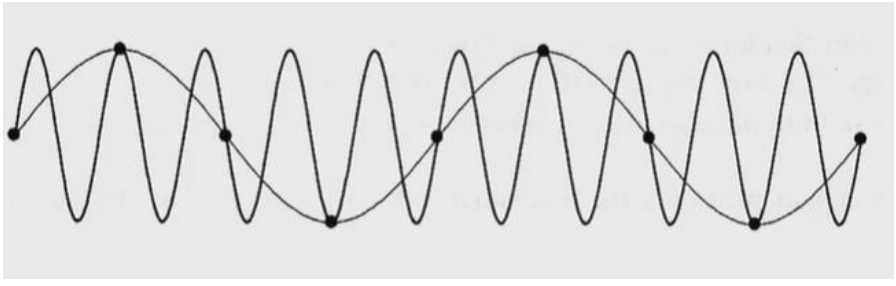
Sampling auf der Nyquist Frequenz

Man braucht bisschen mehr als die Nyquist Frequenz. Hat man genau die Nyquistfrequenz, nämlich könnte man folgendes erhalten:



a zufällig richtig, weil die Abtastpunkte genau auf den Maxima sind.
b könnte man schon einen Sinus mit kleinerer Amplitude drauflegen.
c könnte auch ein gerader Strich sein.

Sampling unter der Nyquist Frequenz

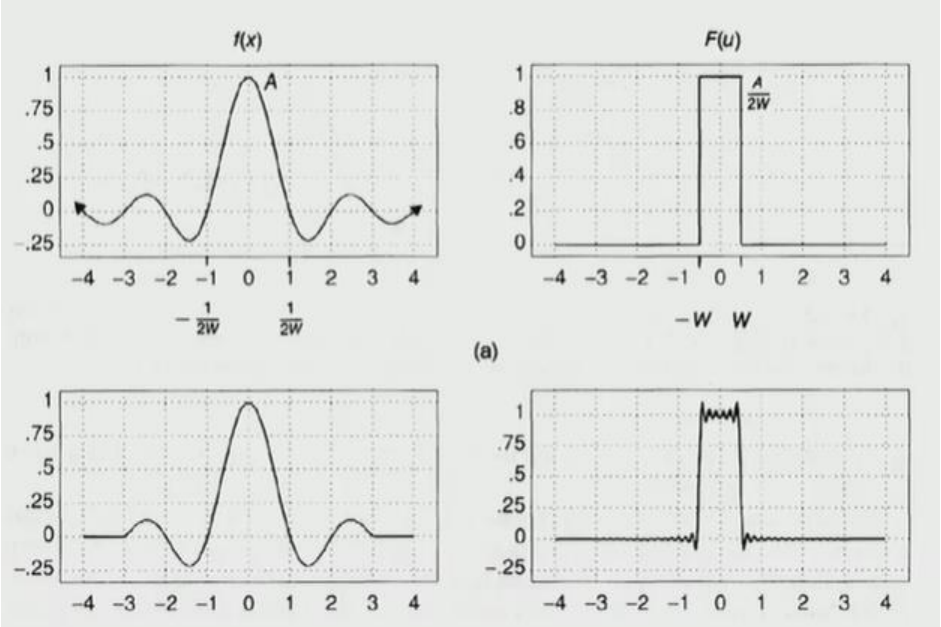


Daher kommt auch das Wort Alias. Der niedrigfrequente Sinus gibt sich hier für den hochfrequenten aus.

Konstantin Lackner 03.07.2020

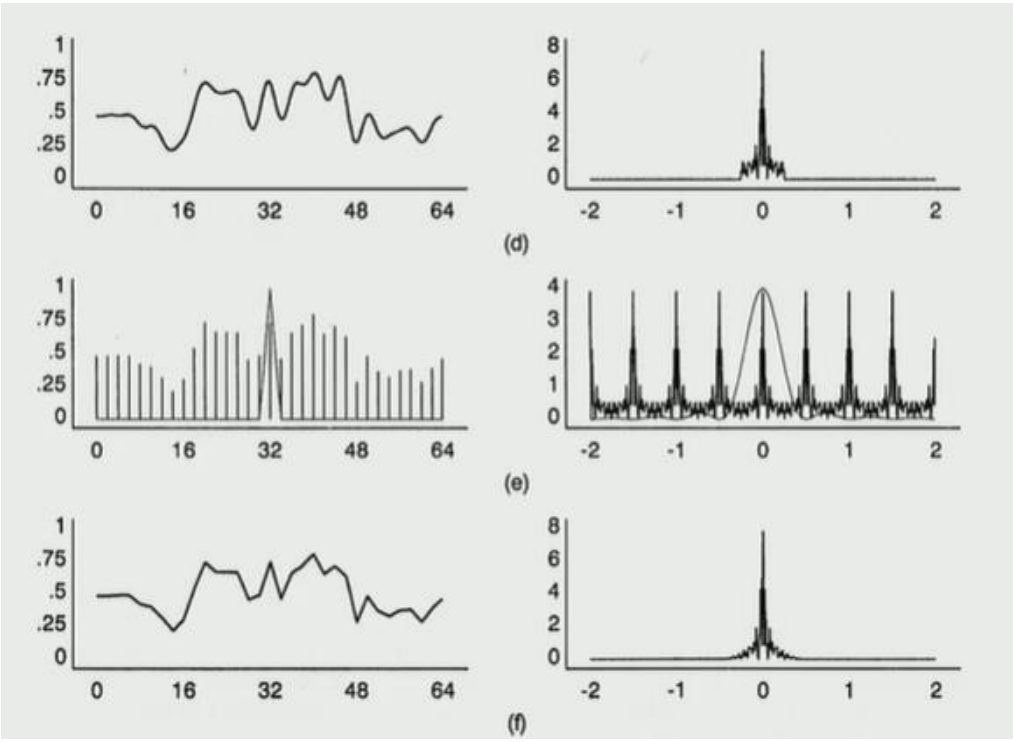
Sinc und truncated Sinc

Weil ja die Sinc Funktion beim Rekonstruieren leider unendlich ist und daher nicht praktikabel, muss man einen beschnittenen Sinc benutzen:



Rekonstruktion

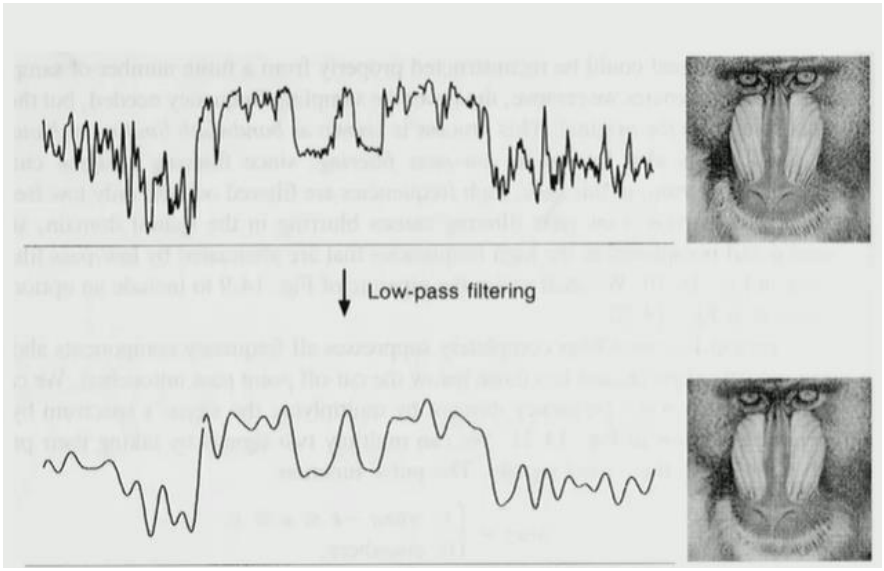
Bei der Rekonstruktion müsste man einen REchtecksfilter im Frequenzbereich multiplizieren, also im Ortsraum eine Faltung mit Sinc. Man kann dann entweder den Fehler beim Rekonstruieren machen und bspw. mit einer Dreiecksfunktion falten statt mit Sinc, oder der Fehler geschieht schon davor beim Sampling. (zu viele Schattenspektra zu nah aneinander)



Hier geschehen beide Fehler. Einerseits wurde zu grob gesampelt wodurch im Frequenzraum die Schattenspektra zu nahe am Originalspektrum sind. Andererseits wurde dann mit einer Dreiecksfunktion multipliziert anstatt mit einer Rechtecksfunktion, also mit Sinc^2 statt Sinc gefalten. Das Ergebnis weißt also einige Fehler auf.

Signale bandlimitieren

Man kann Signale bandlimitieren, also einfach bspw. die hohen Frequenzen wegschneiden. Das ginge bspw. mit einem Lowpassfilter.



Der Vorteil davon ist, dass man das Signal dann exakt abtasten und exakt rekonstruieren kann.

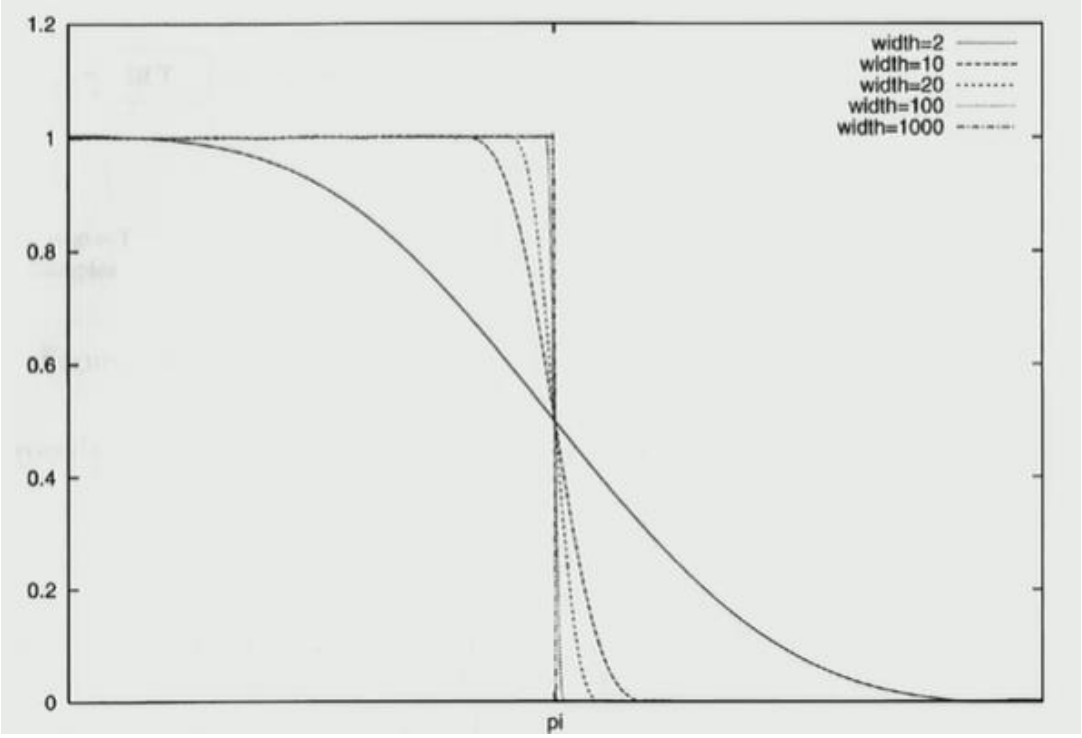
Rekonstruktion in der Praxis

Man kann jetzt also nicht Sinc nehmen weil unendlich und Truncated Sinc ist oft zu ungenau.

In der Praxis macht man nearest neighbour Interpolation. = Diskretes Sinal mit Rechtecksfunktion falten.

Lineare Interpolation = Faltung mit Dreiecksfunktion. Man interpoliert immer zwischen zwei Abtastpunkten.

Windowed Sinc = Abgeschnittene Sinc



Hier sieht man die Annäherung an die Sinc Funktion.

Sampling und Reconstruction Errors

Aliasing – Die Abtastrate ist zu gering, die Frequenzen im Frequenzraum überlappen sich.

Truncation Error – Da man beim Rekonstruieren nicht mit Sinc faltet, also einen Sinc Filter anwendet, sondern nur eine Annäherung, entstehen Fehler.

Non-Sinc Error – Weil man etwas benutzt was gar ned erst Sinc is, entstehen Fehler. Bspw. Lineare Interpolation.

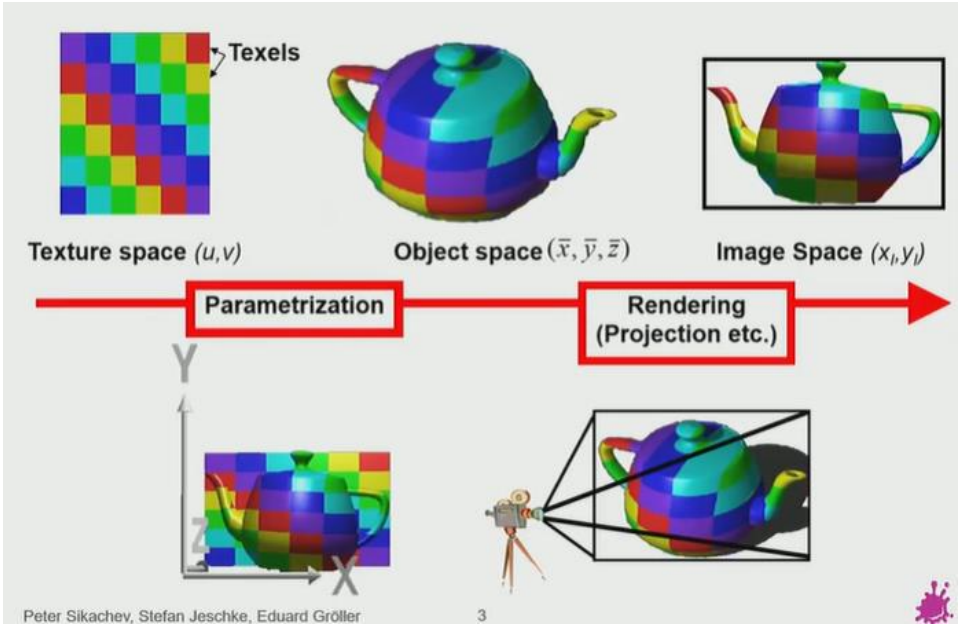
Interpolation – Zero Insertion

Ist eine Methode im Frequenzraum. Man hat ein diskretes Signal und will dieses rekonstruieren. Man hat ein Spektrum mit n Werten und fügt einfach ein nochmal n hinzu die alle 0 als Amplitude haben. Man bekommt dann ein Signal das ident ist, aber eben mehr Abtastpunkte hat.

Texturen

Mikro, Meso, Makro Strukturen

Mikro wird mit Texturen gemacht.



Prozedurale Texturen

Sind solche, wo Heuristiken für meist 3D Texturen erstellt werden. Bspw. gibt es das für Holz, wo einfach Jahresringe simuliert werden.

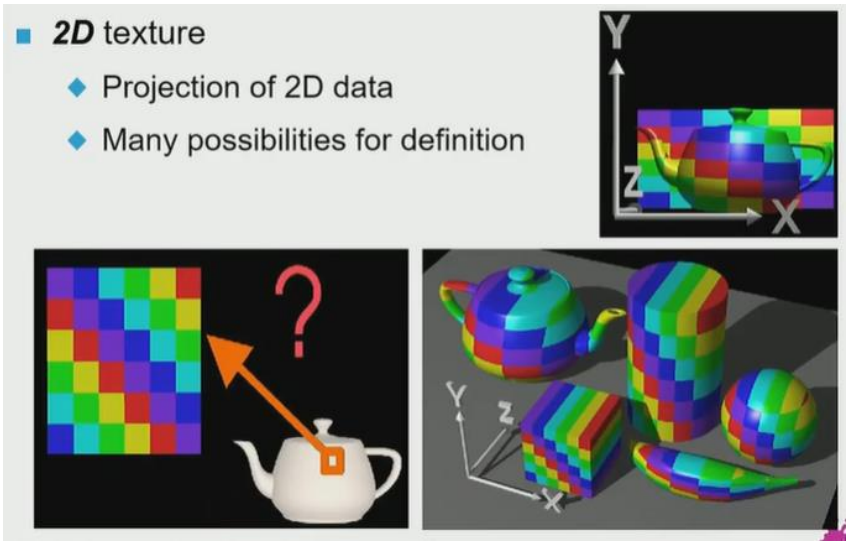
Sampled Textures

Einfach Rasterbilder die mit einer Kamera aufgenommen wurden. (oder auch gezeichnet) Diese werden dann wie im Bild oben mit UV Koordinaten auf die Objekte geklatscht.

Texturen können definiert werden als:

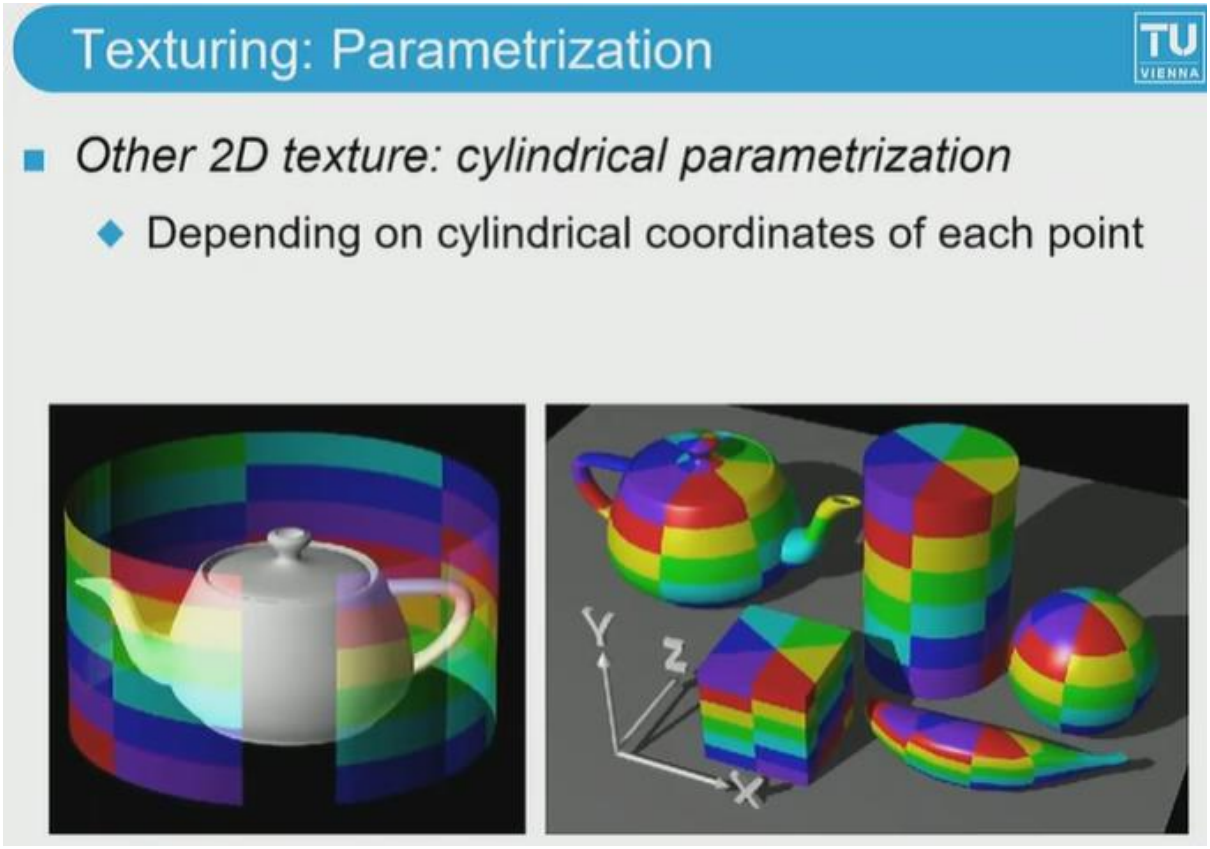
3D object space mit cube mapping bspw.

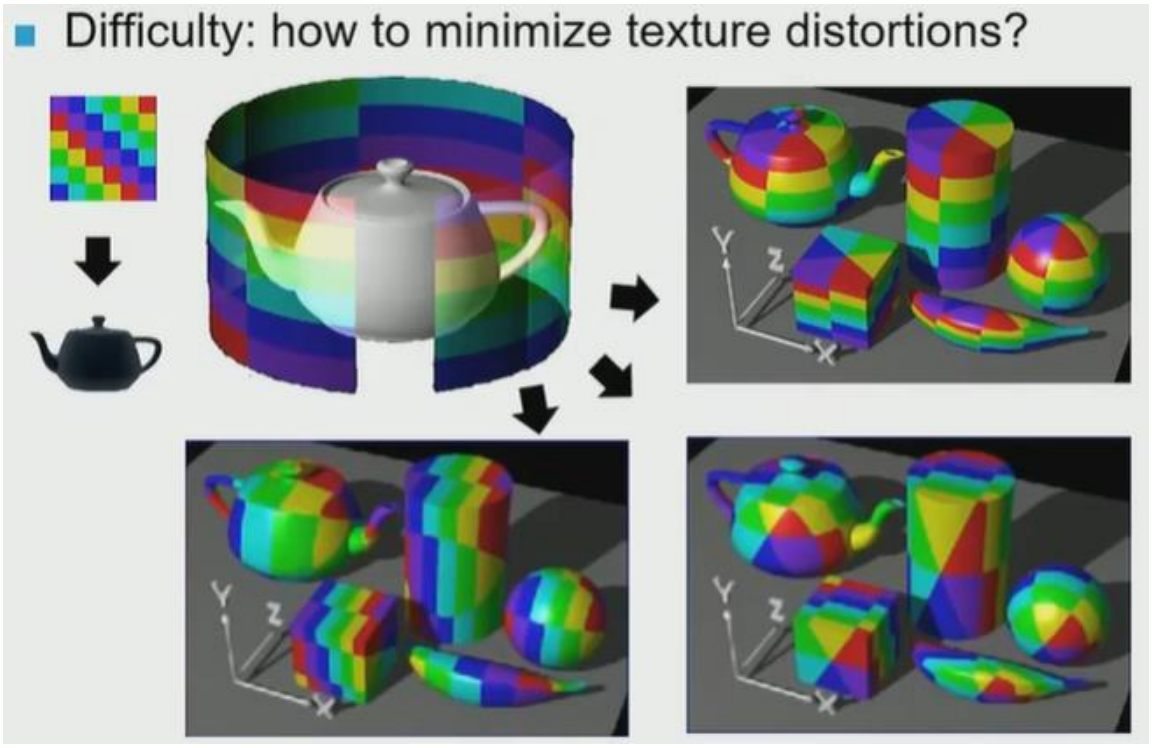
2D object surface mit texture mapping



Als würde man das Objekt durch die Textur schieben.

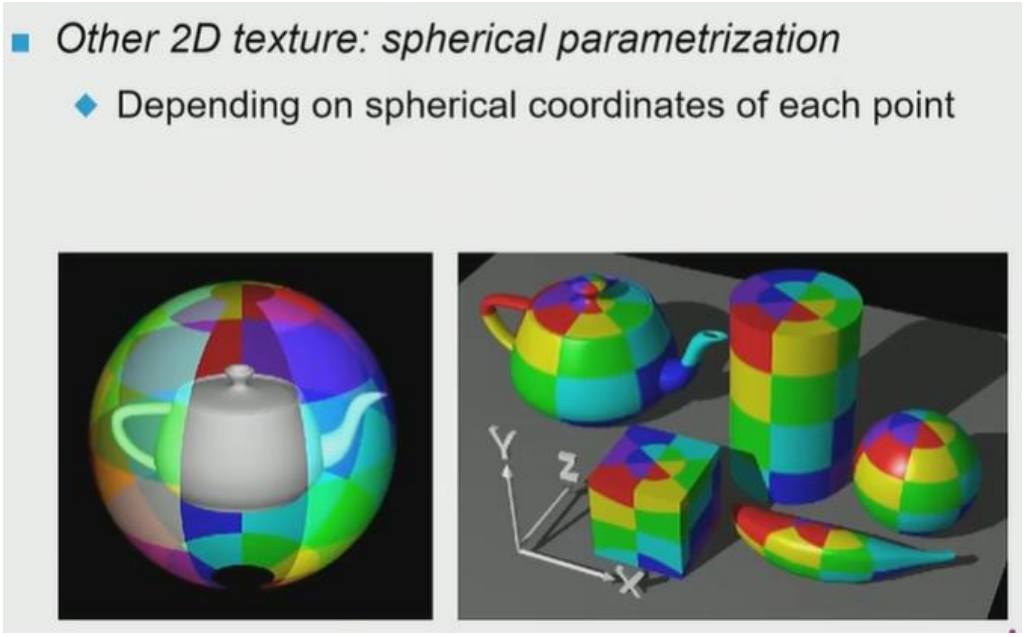
Man kann seine Objekte auch durch Zylinder umgeben und darauf dann die Textur aufklatschen.





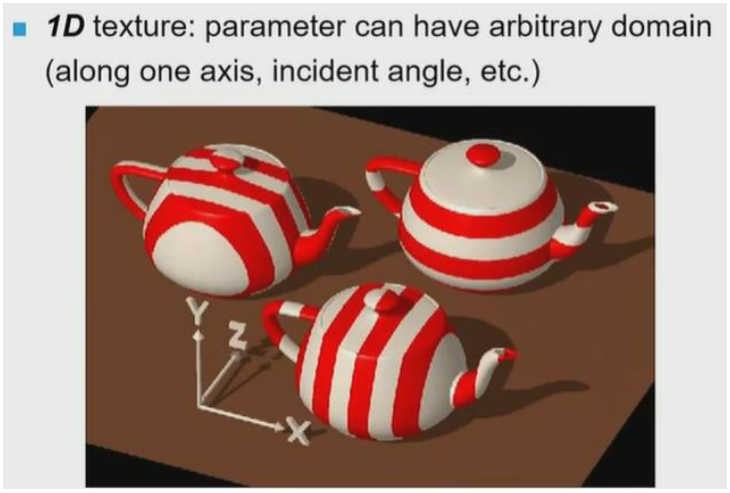
Zylinderpositionierung ist relevant.

Oder auch Spheren…



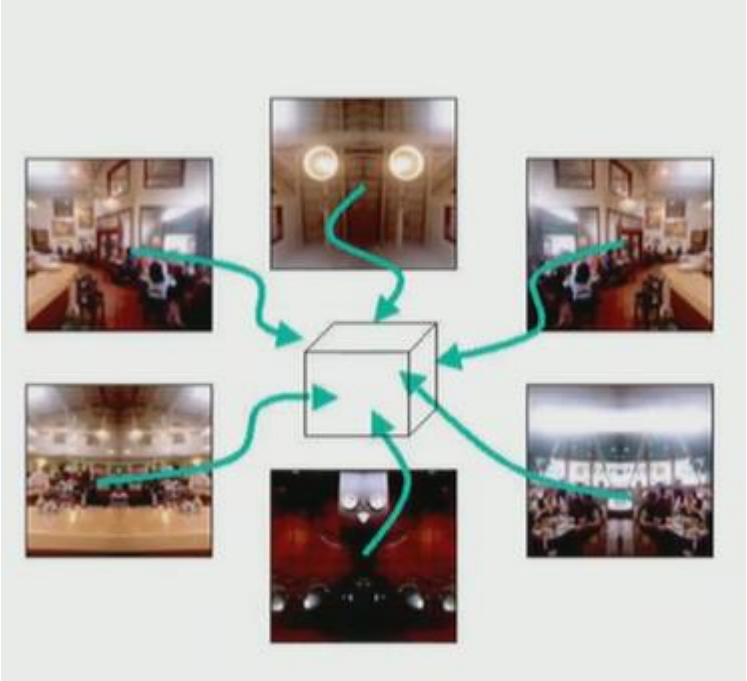
Polstellen sind da halt deppat.

1D Funktion als lookup table



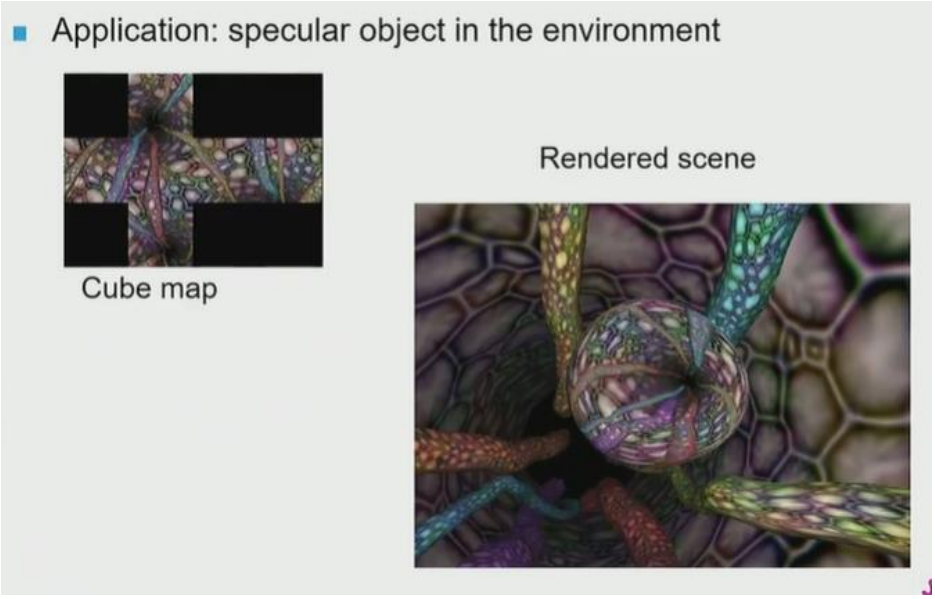
Cube mapping

Alles was weit weg is kann durch Bilder ersetzt werden. Man hat eine Szene die von außen Input bekommt, bspw. einen Theatersaal wo man in der Mitte sitzt.



Man umschließt also die Szene mit einer großen Kugel und klatscht auf die Innenseite der Kugel eben diese Raumbilder. Das wäre Sphere mapping, nur mit einem Cube umschlossen ist es eben Cube mapping.

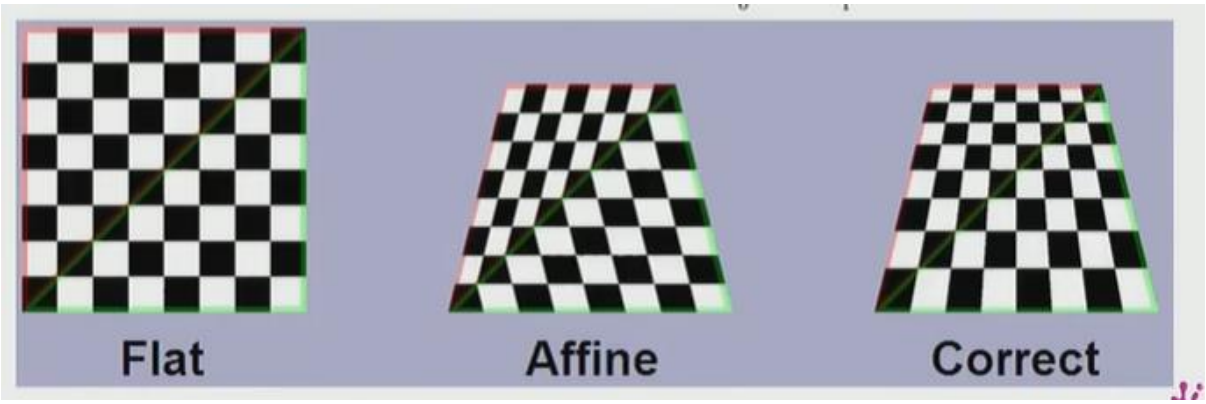
Hat man dann spiegelnde Objekte in der Szene, nimmt man einfach die Blickrichtung, geht in Spiegelungsrichtung weiter und nimmt einfach das Bild aus der Cubemap das gegenüber liegt.

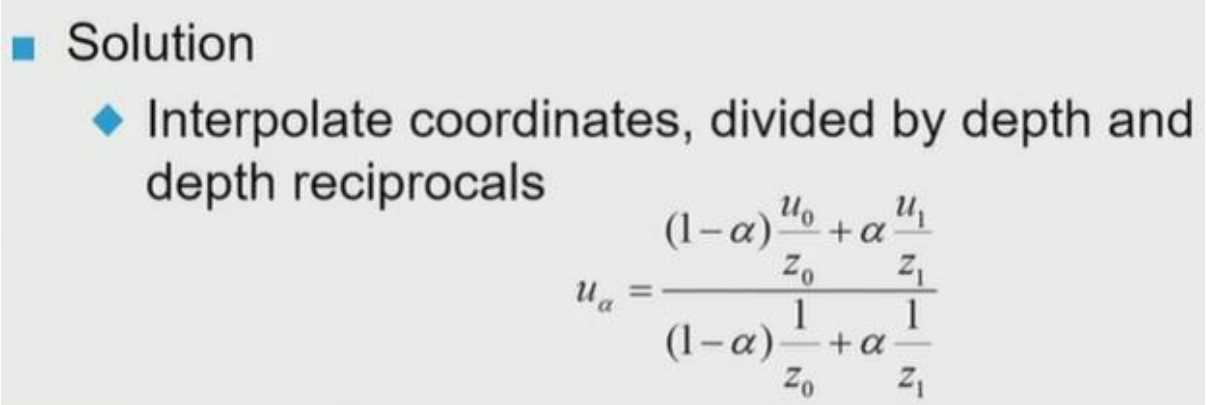


Texture mapping

Man bekommt beim Texturieren UV Koordinaten die für Positionen in der Textur stehen. Fürs Innere des Dreiecks kann man im Texturraum auch Werte errechnen. Bspw. durch Baryzentrische Koordinaten. Man muss die Tiefeninformation aber auch berücksichtigen,

sonst werden bei kleineren Dreiecken kleinere Texturen angezeigt und an den Übergängen schaut das dann so aus:

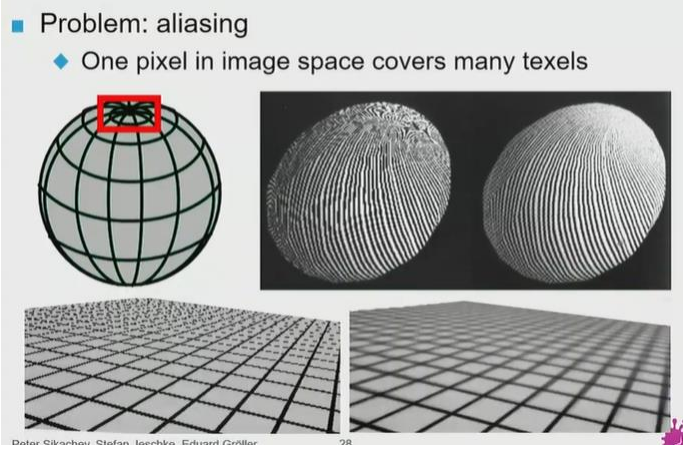




Aliasing

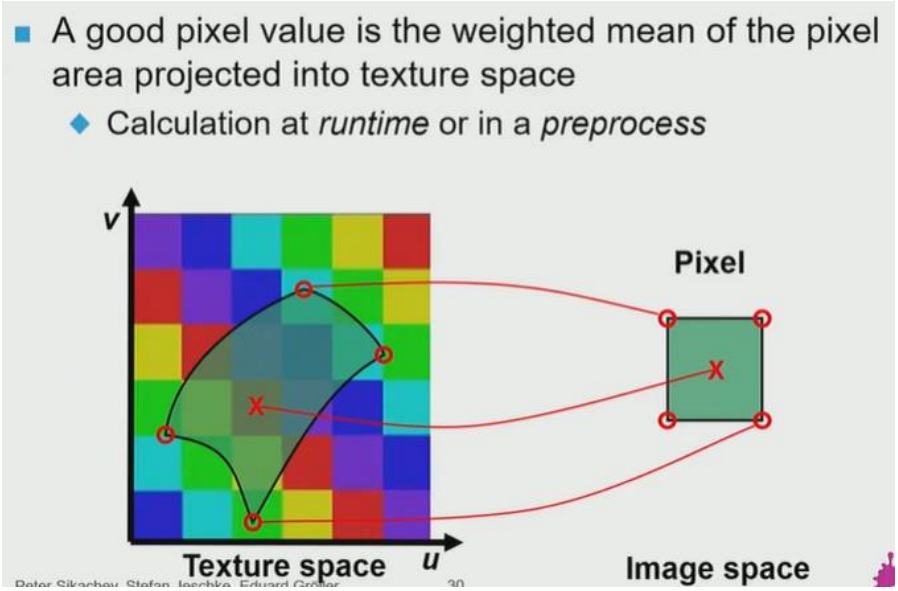
Hat man eine 2D Textur und eine Sphere, ist an den Polen natürlich alle Texturinformation in einem Punkt. Die oberste Zeile der Textur würde auf einem Punkt zusammengefasst. Nimmt man einfach nur den Mittelpunkt der obersten Zeile der Textur, is es auch falsch, richtig wäre also wenn man ein Mittel aller Pixel in der obersten Zeile nimmt.

Ein weiteres Problem entsteht mit der Perspektive. Wenn man durch die Verzerrung eine Verkleinerung des Horizonts bekommt, deckt ein Pixel am Schirm schon einen sehr großen Texturbereich ab und dadurch bekommt man Aliasingrtefakte.



Lösung

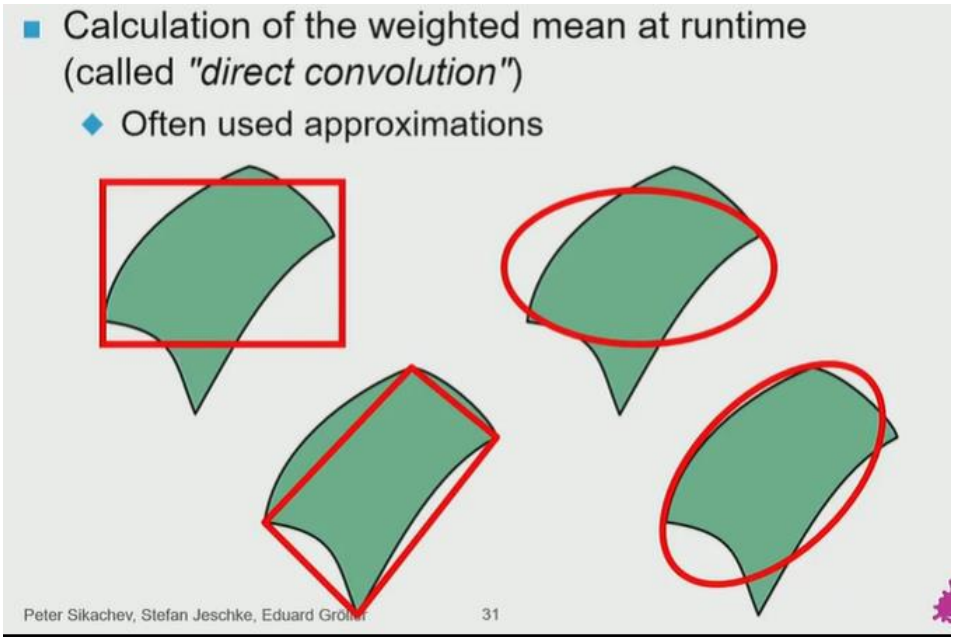
Man nimmt ein Pixel im Bildraum, das ist ein Quadrat, im Texturraum kann das aber verzogen sein usw. Man muss sich den gesamten Texturwert in dem Bereich anschauen, gewichtet diese und errechnet sich ein Mittel.



In diesem Fall wird einfach der Mittelpunkt genommen, das ist wenn die Fläche nicht zu groß ist eine mögliche Lösung, stößt aber sehr schnell an seine Grenzen.

Vereinfachungen

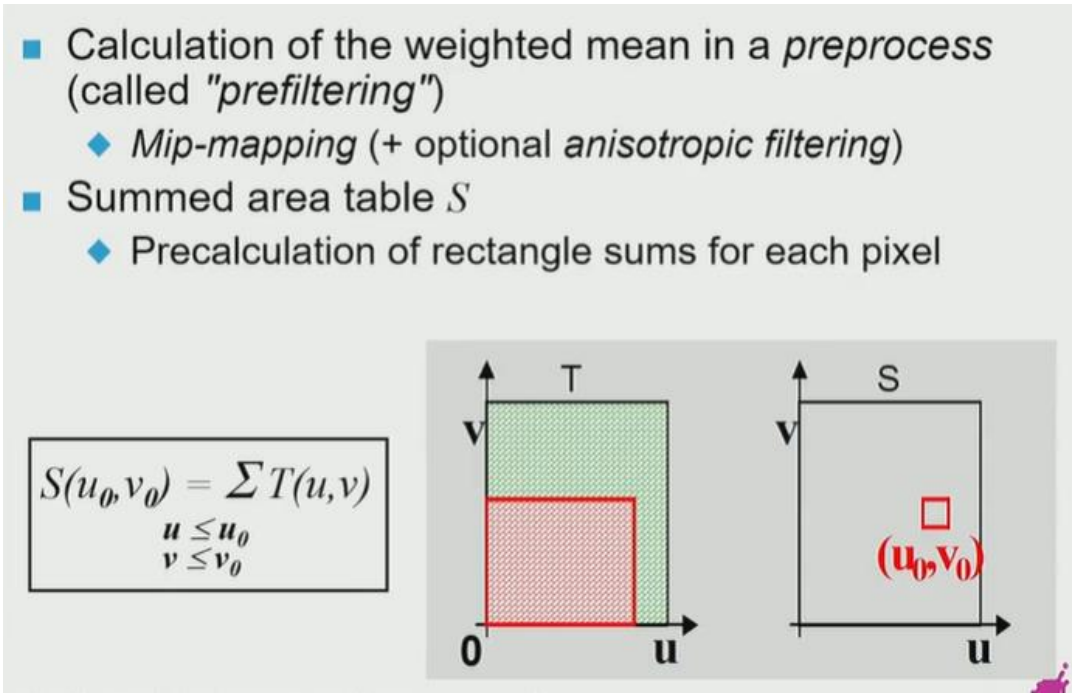
Hier nimmt man den gekrümmten Raum im Texturraum und approximiert diesen durch eine einfache Form:



Konstantin Lackner 03.07.2020

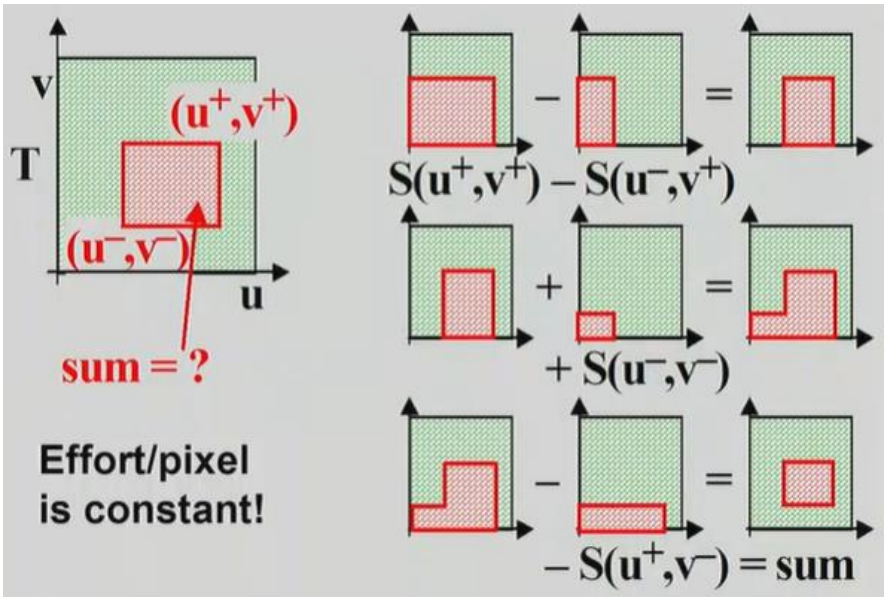
Summed Area Methode

Im Texturraum erstellt man eine Tabelle mit gleicher Auflösung wie die Textur. Wenn die Textur Pixel groß ist, so ist die Tablle eine 10x10 Tabelle. Die Quadrate der Tabelle sind dann jeweils so gefüllt, dass man einfach alle Werte die kleiner/gleich den u/v Koordinaten des Quadrats sind, nimmt und aufsummiert.



Je weiter man also nach oben recht kommt, desto größer der Wert.

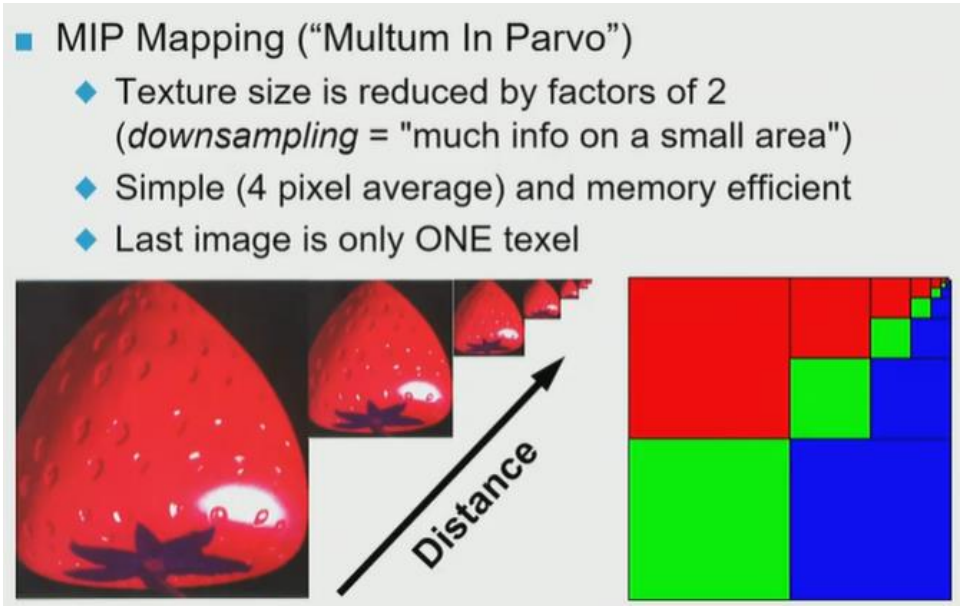
Man nimmt dann ein Pixel, schaut wie das approximiert wird im Texturraum. Da hat man dann u/v Werte. Der rechte obere Eckpunkt gibt amal die größte Fläche, dann zieht man die Fläche des linken oberen Punktes ab, dann fügt man den des linken unteren hinzu und dann zieht man den rechten unteren wieder ab.



Jetzt hat man ein Ergebnis übrig. Man geht hier davon aus, dass das Bild eines Pixels im Texturraum achsenparalleles Rechteck ist.

MIP Mapping

Was man hier macht ist, dass man davon ausgeht, dass ein Pixel im texturraum ein achsenparalleles Quadrat ist. Man halbiert dann immer die Achsen der Textur und macht ein Bild das ein Viertel so groß ist wie zuvor.

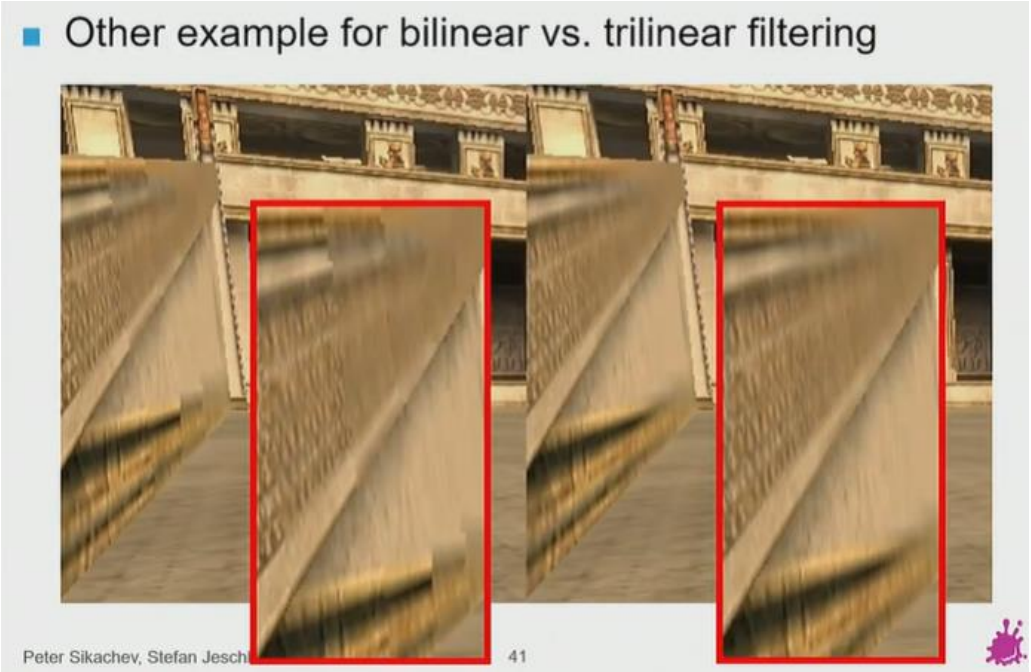


Was man dann macht ist: Wenn ein Pixel im Texturraum ungefähr einem Texel entspricht, benutzt man das ursprüngliche Bild. Wenn ein Pixel aber mehr überdeckt, dann geht man eine Ebene runter oder eben mehr Ebenen. Man schaut auf die Größe bspw. anhand der Diagonale. Dann nimmt man den log dualis davon und bekommt die entsprechende Textur. Die ursprüngliche Textur, hier wäre die Größe der Diagonale im Vergleich zum Texel 1, log2 davon ist 0 und damit hat man den Index der Textur, hier also das Original, also 0.

Dafür braucht man ein Drittel mehr Speicher, denn es handelt sich hier um eine geometrische Reihe.

Man bekommt damit aber Sprungstellen zwischen den verschiedenen Texturbereichen:

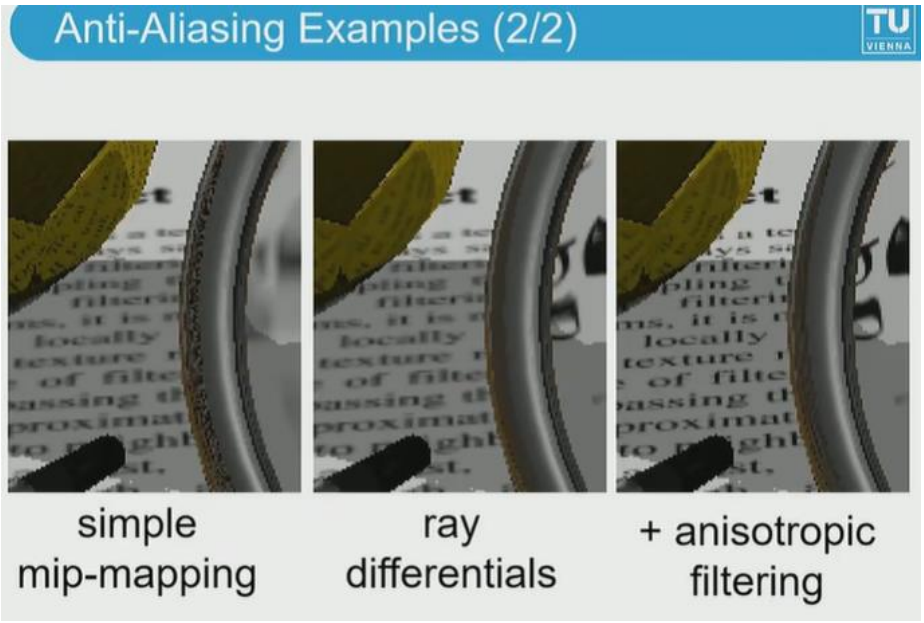
Wenn man dann aber Trilinear interpoliert, nimmt man einfach mehrere Ebenen und Gewichtet zwischen diesen einfach mit dem Gewicht zwischen Texel und Pixel.



Bekommt man einen Wert kleiner als 0 beim Log2, also das Texel ist größer als das Pixel. Was man dann machen kann ist, dass man Upsampling macht, also die vier umliegenden Texturwerte nimmt und dann wieder ein gewichtetes Mittel errechnet.

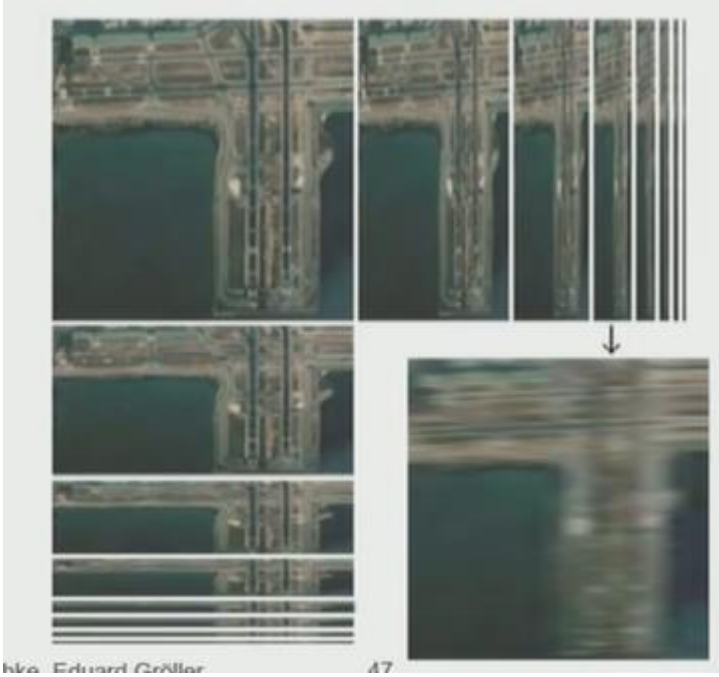
Anisotropic Filtering Methoden

Wenn die Annahme, dass man ein achsenparalleles Quadrat bekommt, doch nicht ganz richtig ist, dann kann das, je nachdem wie falsch, zu ziemlichen Problemen führen.



RIP Mapping

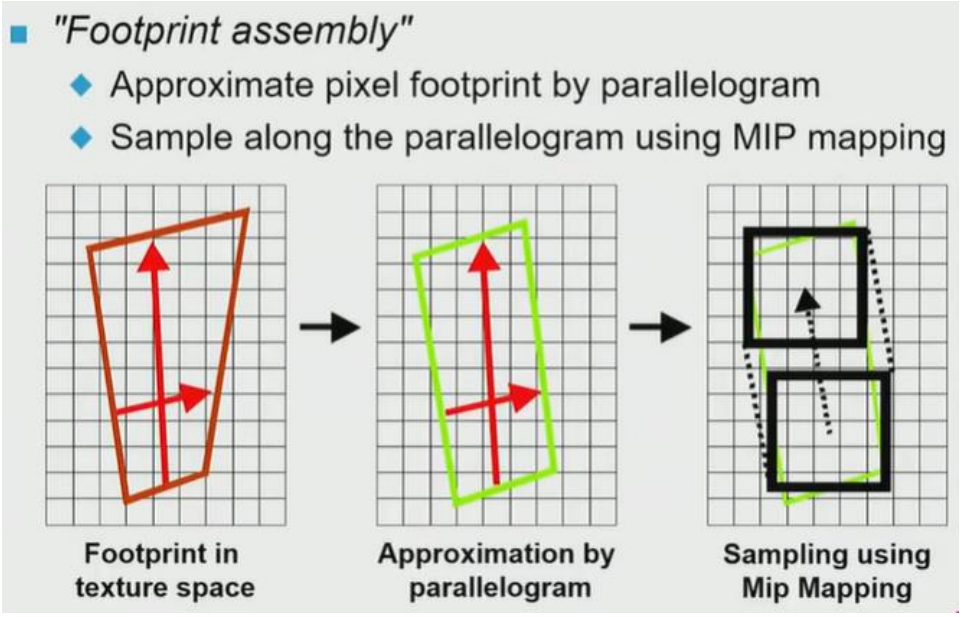
Man muss also berücksichtigen, dass es sich evtl. doch um was längliches handelt. Dabei geht man noch von achsen parallel aus, aber nicht mehr von Quadraten, sondern Rechtecken. Man macht dann einfach eine andere Map, das nennt man RIP Mapping.



Man schaut dann einfach einzeln in die Richtungen.

Footprint assembly

Man schaut einfach mehrmals nach und summiert dann auf:



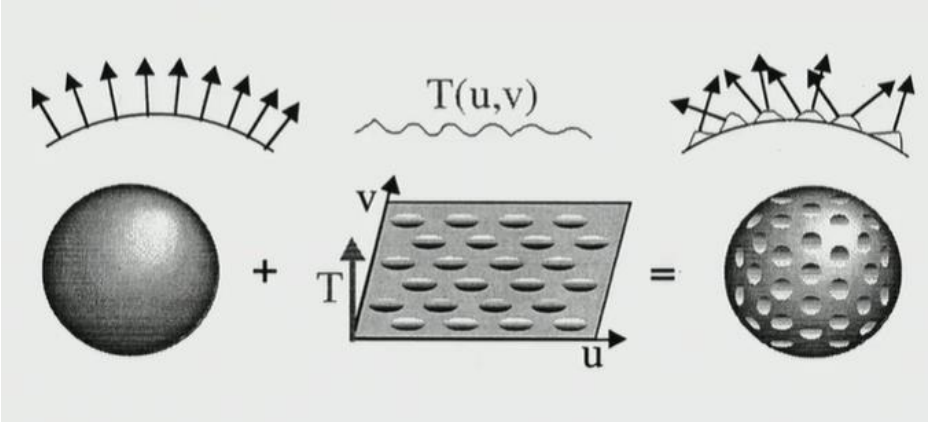
Die beiden kann man dann mitteln.

Konstantin Lackner 03.07.2020

Bump Mapping

Bei Bump Mapping wird die Textur nicht als Farbinfo sondern als Texturierung der Oberfläche interpretiert.

Was man also macht ist die Normalvektoren verdreht.



Normal mapping

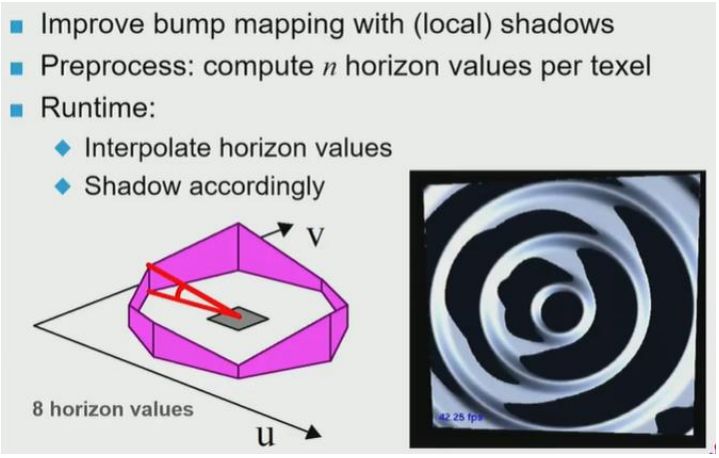
Da nimmt man ein Objekt her das aus 10000 Dreiecken besteht und speichere die Normalvektoren in einer Textur. Dann Reduziert man das Objekt auf 100 Dreiecke aber benutzt für die Schattierung die Textur mit den 10000 Normalvektoren. Das Ergebnis schaut dann sehr viel besser aus als mit nur 100 Normalvektoren.

Silhouette

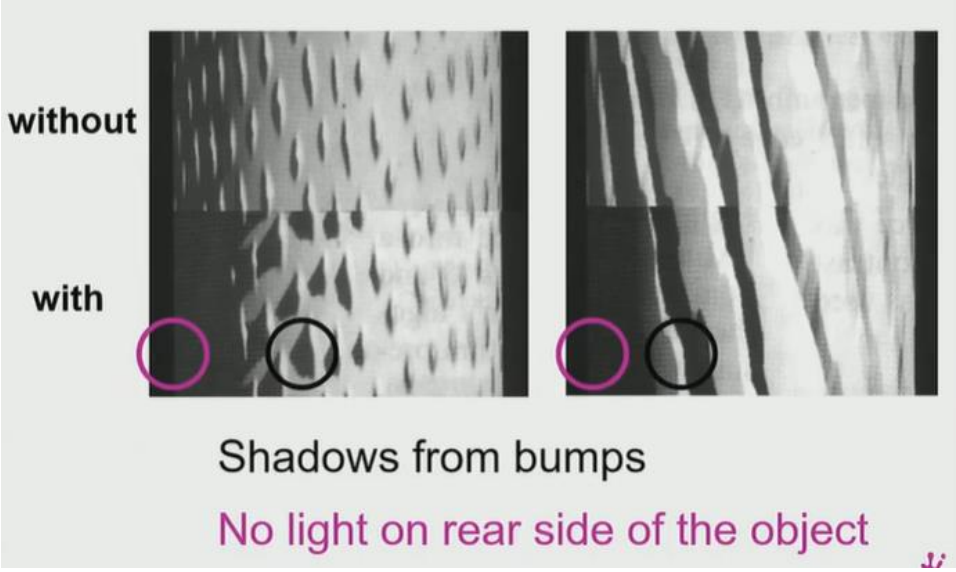
Bei beiden Verfahren sieht man bei der Silhouette die Fehler. Um das zu vermeiden kann man Horizon Mapping machen.

Horizon Mapping

Man merkt sich einfach in einer weiteren Textur zu jedem Texel das Höhenprofil. Damit kann man dann auch Schatten auf dem Objekt richtig darstellen.

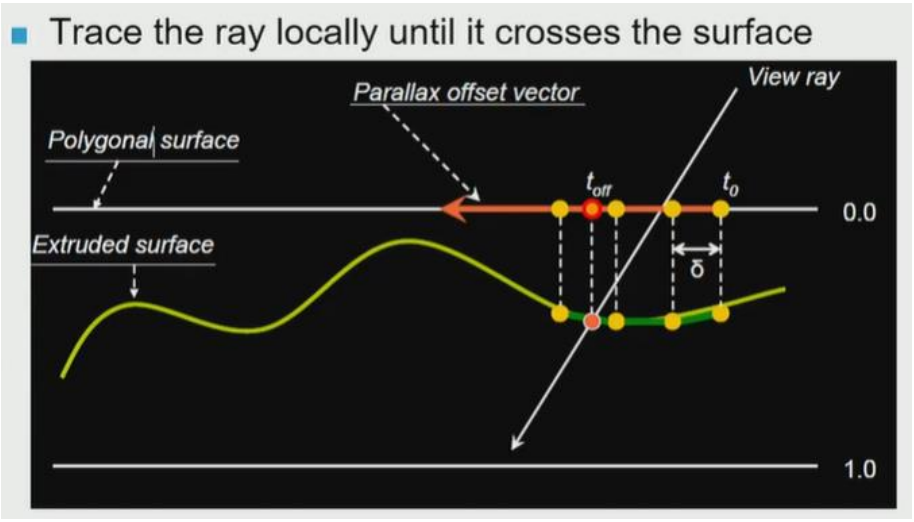


Konstantin Lackner 03.07.2020



Parallax Mapping

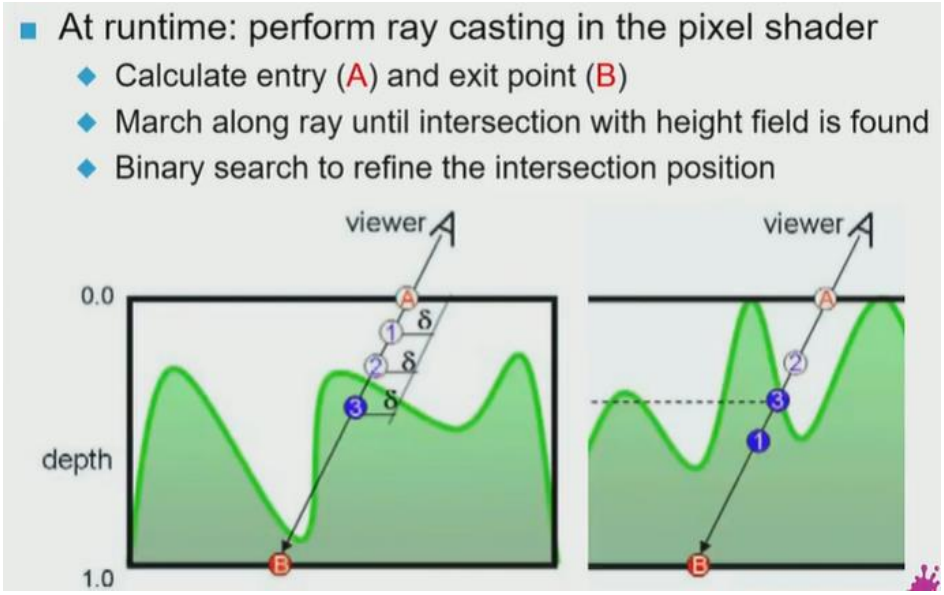
Man versucht Obenflächenunebenheiten genauer zu modellieren indem man nur Texturkoordinaten verändert. Wenn bspw. durch ein Höhenfeld verändert, müssen gewisse Texturwerte verschoben werden.



Für den Texturwert müsste man hier also nicht direkt den drunter nehmen, sondern den daneben, es ist also ein bisschen verschoben. Das hängt vom Einfall des Strahls und von der Höhe.

Relief Mapping

Wie Parallax Mapping aber genauer weil es ein richtiges Ray casting macht und Eintritts und Austrittspunkt zur Berechnung hernimmt.

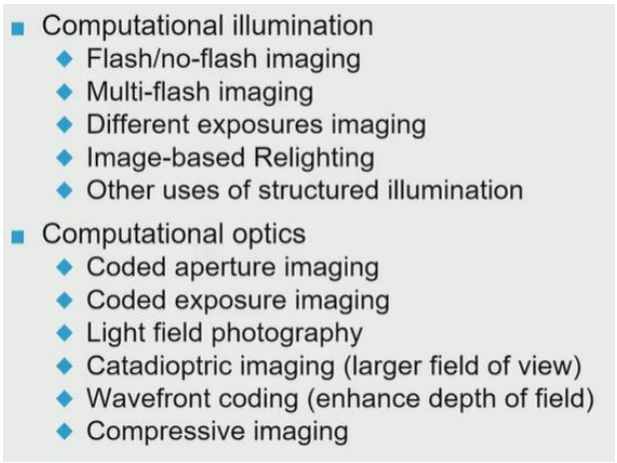


Computational Photography

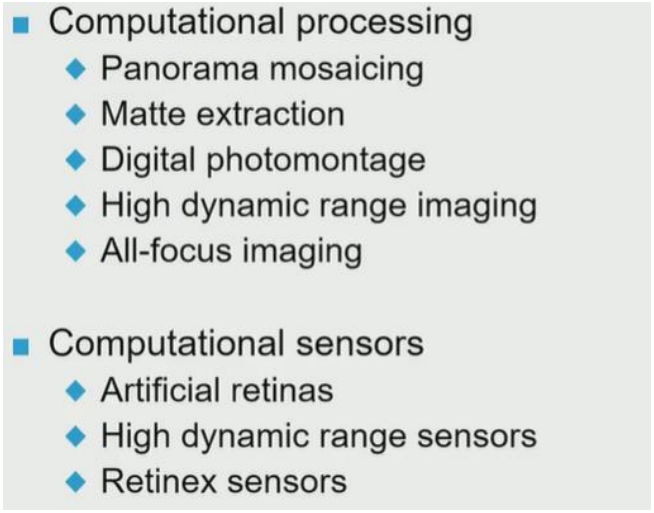
Sind Verfahren bei denen mit herkömmlichen Kameras nicht erreichbare Effekte durch Computer berechnet werden.

Bspw. 3D Rekonstruktion oder Beleuchtungserrechnung aus mehreren Bildern.

Arten der Computational Photography



Konstantin Lackner 03.07.2020



Defocus Mapping

Man hat mehrere Videoströme wo man mit einer den Vordergrund aufnimmt, mit einer den Hintergrund, usw.. Damit kann man verschiedene Schärfebereiche für verschiedene Bildebenen festlegen. Man kann damit auch Vorder und Hintergrund trennen durch die Schärfeinformation. Damit kann man dann den Hintergrund ersetzen.

Motion Magnification

Verstärken von kleinen Bewegungen. Man trennt Layer in Bewegungen und kann dann Layer hervorheben in denen Bewegungen stattfinden, die man sonst gar nicht gesehen hätte. Bspw. das Biegen eines Balkens beim Schaukeln.

High dynamic range imaging

Man hat eine Szene mit she hohem Kontrast, nimmt drei Bilder auf, unterbelichtet, überbelichtet und baut daraus ein Bild zusammen das nicht überbelichtet oder unterbelichtet ist.

Bilateral filtering

Kanten nicht verwischen aber Rauschen weichzeichnen.

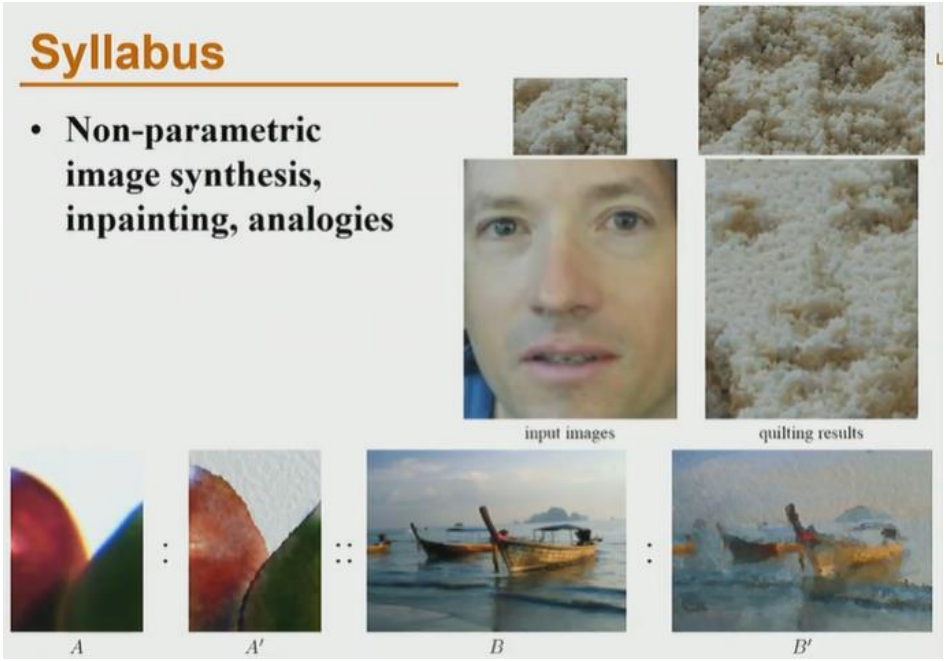
Grading image manipulation

Man nimmt die erste Ableitung. Schaut sich also die Gradienteninformation der einzufügenden Bilder und lässt dann den Gradienten des Hintergrunds reinfließen.

Konstantin Lackner 03.07.2020

Image/Textur synthesis

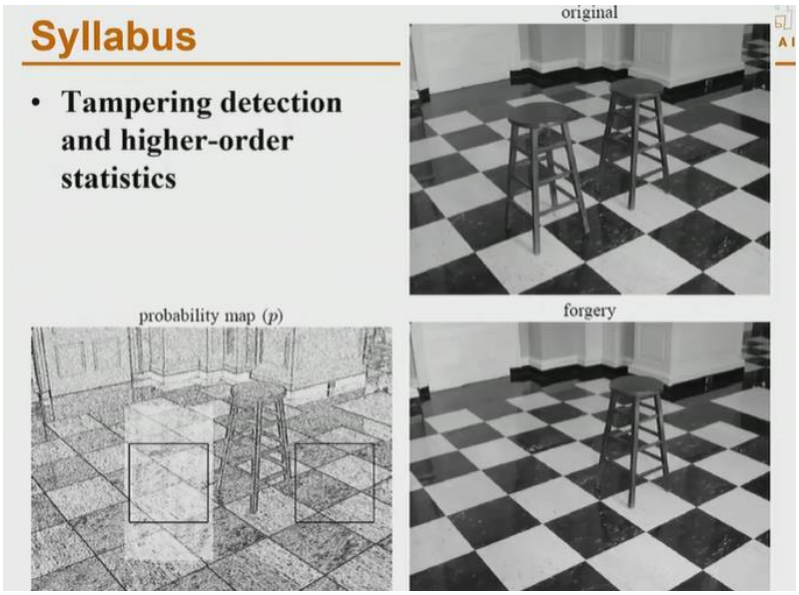
Man baut aus kleinen Stücken neue Sachen nach anderen Bildern zusammen. Man kann damit auch Style-transfers machen.



Konstantin Lackner 03.07.2020

Tampering detection

Ermitteln ob man da gephotoshoppt hat.



Panoramic Imaging

Imagestitching. Zuerst Kameraposition und Richtung errechnen. Dann Helligkeit usw. errechnen und die Überlappungsbereiche anpassen, bissl interpolieren. Bspw. wieder Gradientenverfahren.

Image Warping

Gewisse Bereiche werden markiert im Start- und Endbild und auf Grund dessen weiß man wie die Elemente dann platziert werden sollen. Man kann sich also daraus ein Koordinatensystem ableiten.

Konstantin Lackner 03.07.2020

Light Field

Man schaut in jedem Punkt des Raumes wie viel Licht sich wohin bewegt. Man nimmt also sehr viele Kameras auf einer Ebene und macht viele Aufnahmen derselben Szene. Dann kann man die Tiefenschärfe bspw. ändern und einfach herumrechnen wo man wie was sieht.

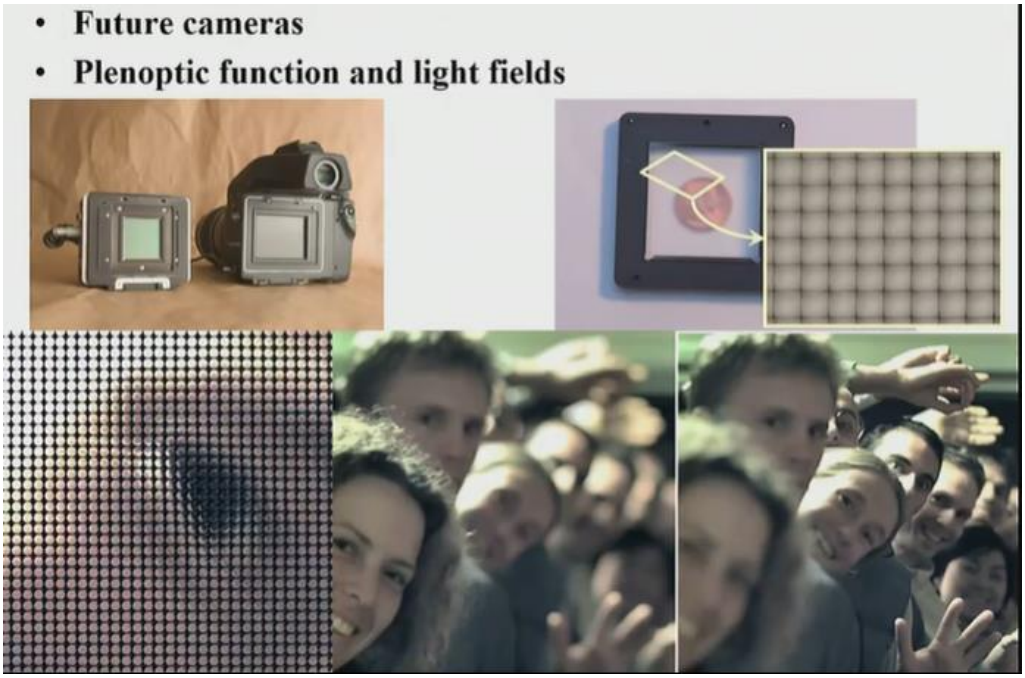
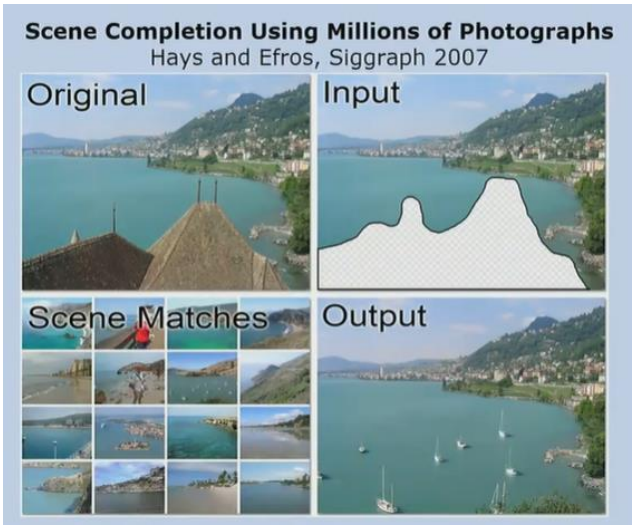


Image Stacks

Einfach mehrere Bilder bspw. einer Sehenswürdigkeit wo Leute vorbeilaufen und dann diese rausrechnen. Auch Bilder zamstecken wo Leute gut ausschauen auf Gruppenbildern.

Scene Completion

Man errechnet einfach für einen Bereich aus einer großen Datenbasis ein neues Bild.

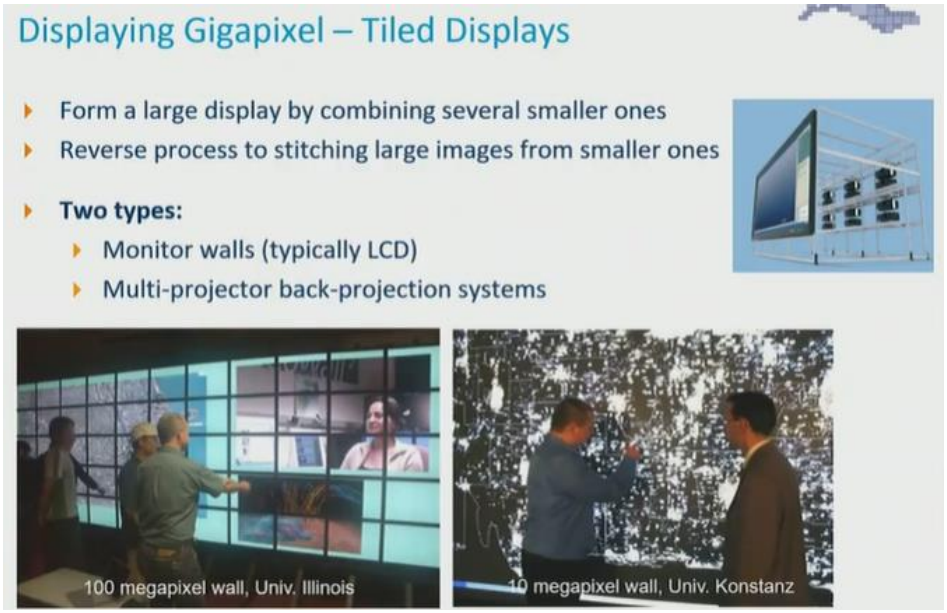


Konstantin Lackner 03.07.2020

Beautification

Einfach eine Datenbank die sich anschaut was schön is und dann mit machine learning einfach anpassen.

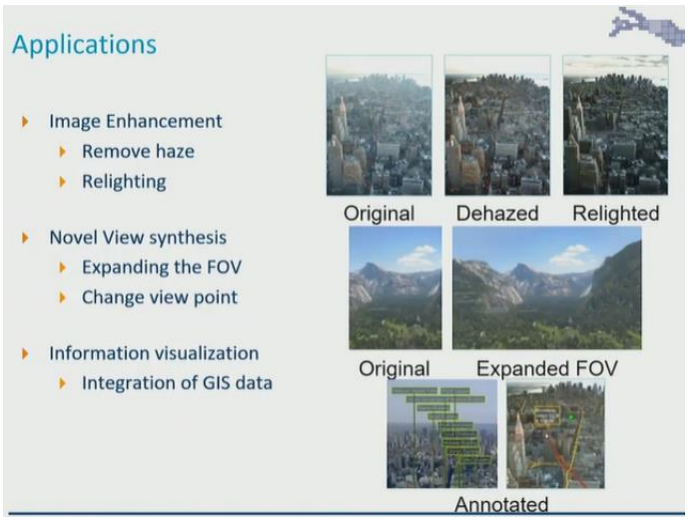
Displaying Gigapixel – Tiled Displays



Deep Photo

Nicht nur Bild sondern auch Tiefeninfo.

Damit kann man bspw. 3D Information abbilden also bspw. Google Maps Maps machen. Damit kann man bspw. Nebenschleier entfernen. Man weiß wie stark der Nebel etwas beeinflusst je nachdem wie weit das Objekt weg ist und das zurück rechnen.

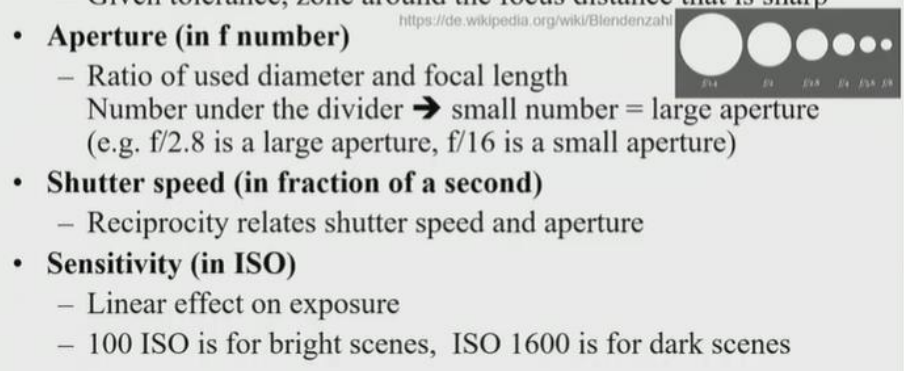


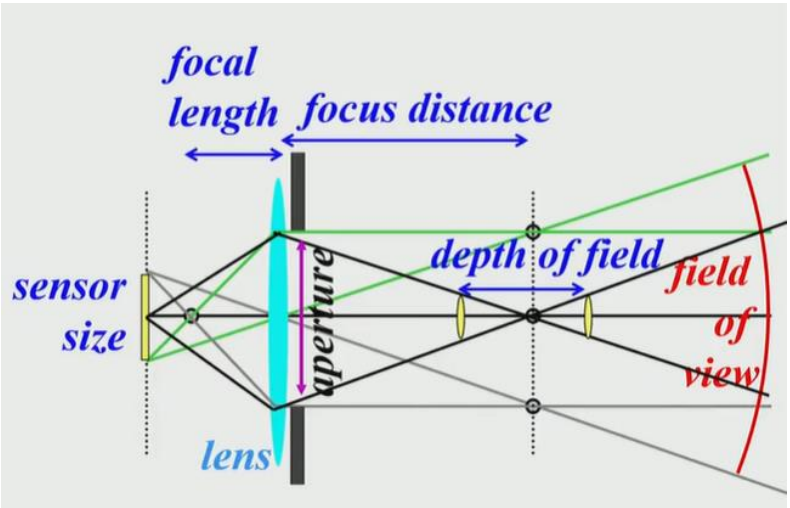
Konstantin Lackner 03.07.2020

Kameras

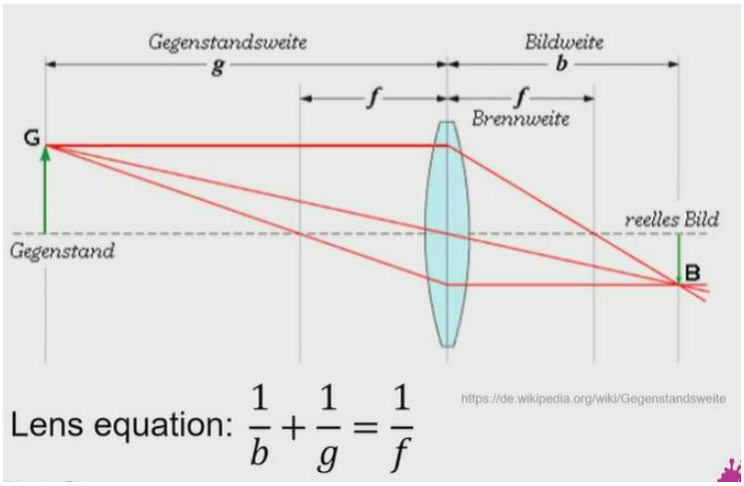
Durch die Blende tritt Bild ein und für am Sensor zur Beleuchtung. Je nachdem wie lang man den Verschluss aufmacht, wird der Sensor länger oder weniger lang beleuchtet. Länger in der Nacht um das gesamte Licht auszubeuten. Eine Sammellinse stellt sicher, dass das gesamte Licht aufgenommen wird, diese hat aber eine gewisse Brennweite, weswegen ein Teil unscharf ist und ein Teil scharf, der nämlich im Brennpunkt. Die Brennweite ist dabei die Distanz die scharf ist im Bild.

- **Focal length (in mm)**
  - Determines the field of view.
    wide angle (<30mm) to telephoto (>100mm)
- **Focusing distance**
  - Which distance in the scene is sharp
- **Depth of field**
  - Given tolerance, zone around the focus distance that is sharp
- **Aperture (in f number)**
  - Ratio of used diameter and focal length
    Number under the divider ➔ small number = large aperture
    (e.g. f/2.8 is a large aperture, f/16 is a small aperture)
- **Shutter speed (in fraction of a second)**
  - Reciprocity relates shutter speed and aperture
- **Sensitivity (in ISO)**
  - Linear effect on exposure
  - 100 ISO is for bright scenes, ISO 1600 is for dark scenes





Konstantin Lackner 03.07.2020



Focal length Abstand zwischen Brennpunkt und Linse. Wenn man mit der Blende spiel, je größer die Blende desto kleiner der Schärfebereich.

Depth of field – je größer die Blende, desto kleiner der Bereich der scharf dargestellt wird. Hyperfocal distance = largest depth of field das man erreichen kann.

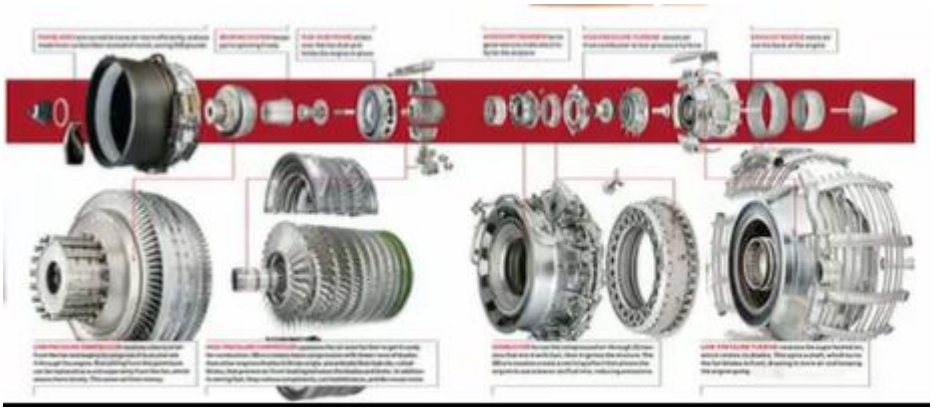
Illustrationen

Haben kommunikativen Intent.

Verpacken komplexe Strukturen in einfache verständliche Bilder.

Benutzen Abstraktion um visuellen overload zu vermeiden.

Benutzen gewisse stilistische Eigenheiten.



Focus + Context Principle

Important regions in great detail(focus), global view with reduced detail (context)

Abstraction

Introduces a distortion between visualization and underlying model. Different levels of abstraction are introduced at different levels.

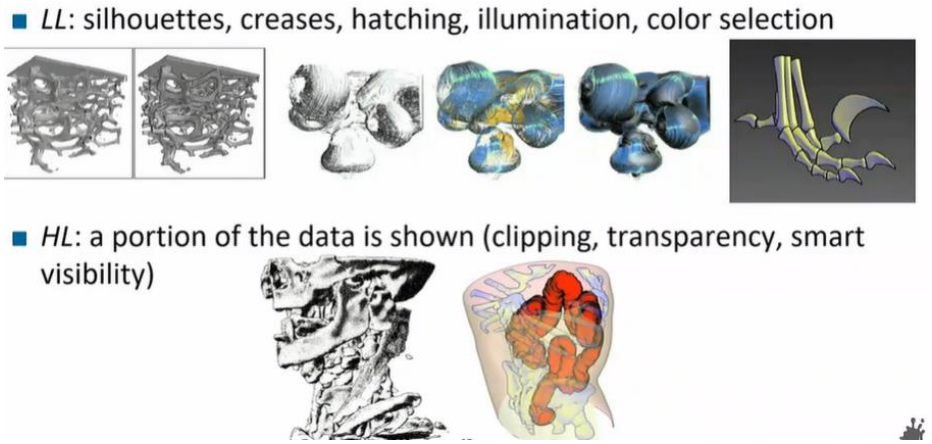
Ziele der Abstraktion:

- Communicate shape and structure.
- Emphasize or de-epmhasize
- Prevent visual overload
- Ensure visibility of important structures
- Provide context (mainly spatial context)

Low and High level techniques

LL – Silhouettes, creases, hatching, illumination, colour selection

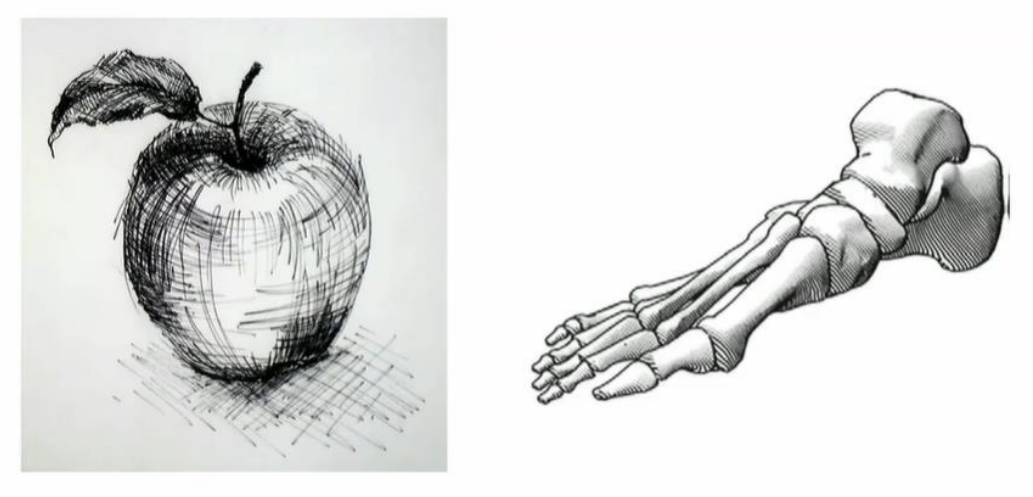
HL – clipping, transparency, smart visibility

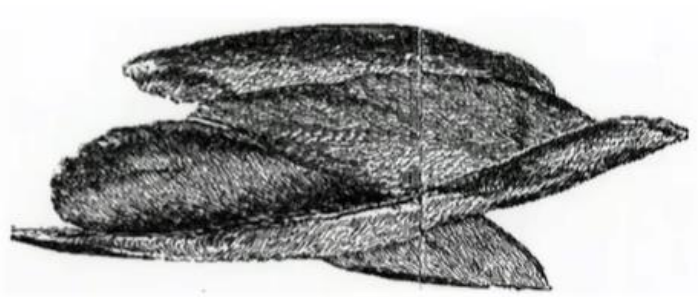


Konstantin Lackner 03.07.2020

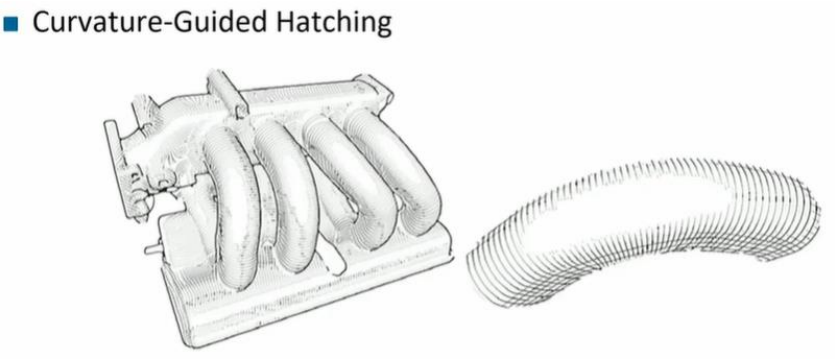
LL: Hatching

Einfach lines





Hat weniger Rules als Stippling. Das Hatching wird in Real time generiert.



Model-based Hatching

Man muss eine preferred direction angeben und Krümmungsinformaiton und daraus kann man dann das Hatching berechnen.

Konstantin Lackner 03.07.2020

LL: Stippling

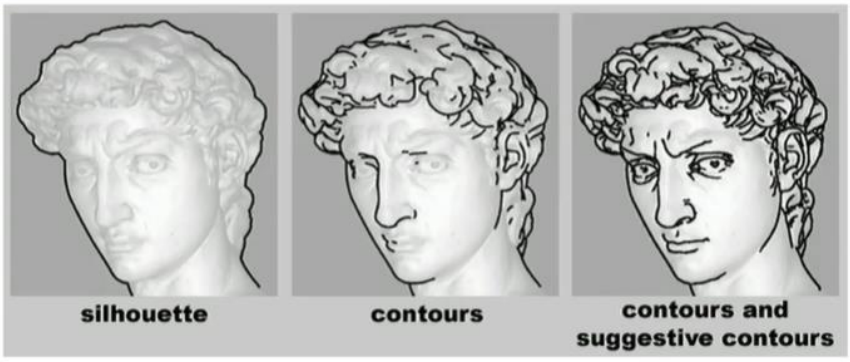
Fots statt lines

- *Dot spacing* : Dots should be placed such that no clear holes or clusters arise.
- *Dot size* : Dot size may be varied according to brightness, but the largest dots are no larger than twice the smallest ones.
- *Dot shape :* The individual dots shape may vary simulating the effect of a pen that is pushed on paper with a varying angle (elongated = / vs. circular = |)

Object based – die Puntke müssen inserted werden beim Reinzoomen und entfernt beim Rauszoomen.
Texture-based – Eine textur wird erstellt die beim Zoomen einfach durch eine andere ausgewechselt wird.

LL: Suggestive Contours

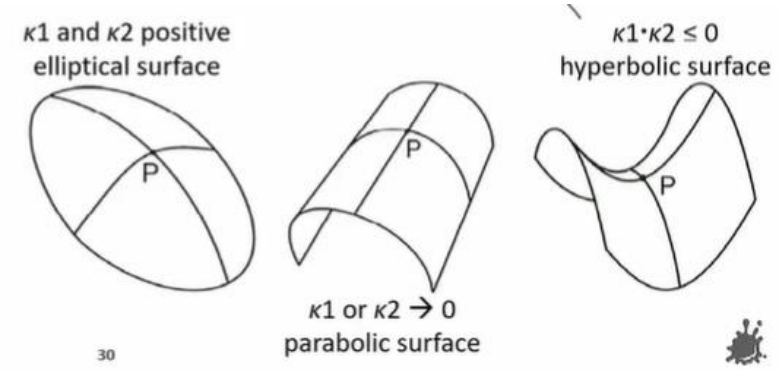
- Suggestive contours [DeCarlo et al. 2003]



LL: Curvature Approximation

Man macht hatching und feature lines according zu den surface normals.

Für die Krümmung gibt es zwei values. K1 and K2.



Man kann dann die feature lines berechnen mit:

- Gaussian curvature $K = \kappa 1 \cdot \kappa 2$ for hyperbolic surfaces
- Mean curvature $H = (\kappa 1 + \kappa 2)/2$ for flatness
- Absolute curvature $|\kappa 1| + |\kappa 2|$

Wenn man feature lines macht, muss man auch ein smoothing machen, damit man noise wegbekommt.

An muss dann natürlich schauen welche lines gebaucht werden und welche nicht. Dafür schaut man welche olygons mit welchen connectet sind. Da werden dann Überdeckungen klar, aber auch diverse andere Hindernisse.

Man muss auch auf Frame coherence achten. Wenn man zoom, müssen auch die featurelines mitangepasst werden. Auch hier gilt wieder, dass man immer bedenken muss was relevant ist und was nicht.

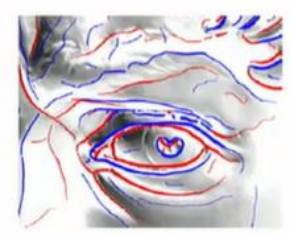
Silhouetten

Silhouetten werden gemacht indem man den gradient anschaut und dort wo hohe spike sind eben eine Linie zeichnet.

Crease Lines

Crease Lines sind Lines die Bumps in Objekten darstellen, also dort gemacht werden wo auf einem Objekt sich die normals sehr schnell ändern.

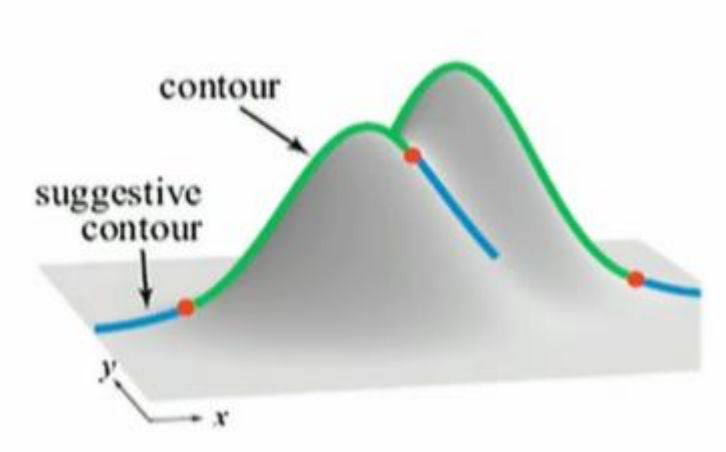
Ridge and Valleys



Sind einfach ein anderer Threshold

Konstantin Lackner 03.07.2020

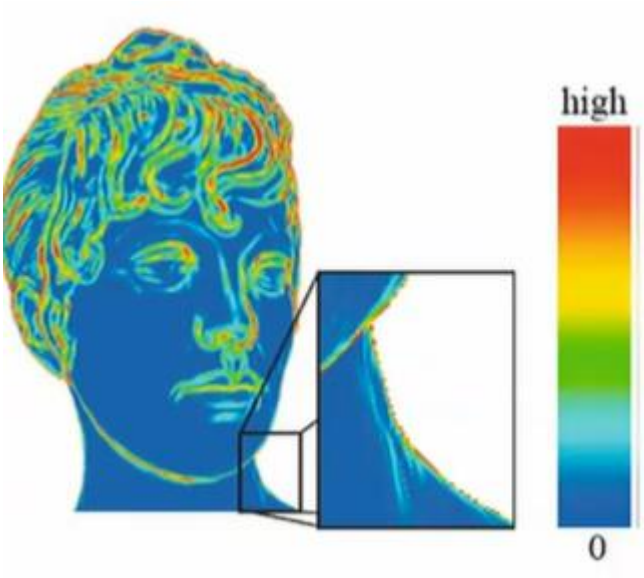
Suggestive Contours



Sind die Fortläufe von richtigen Konturen.

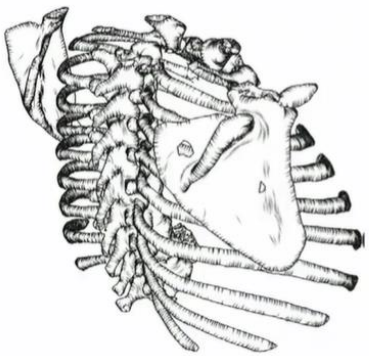
Apparent Ridges

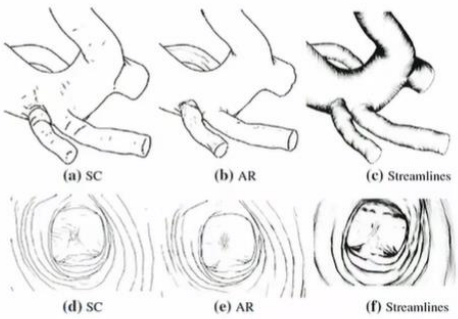
Sind einfach Markierungen von Maxima in der Krümmung



LL: Streamlines

Man sieht sich Vectorfields an und zieht diese mit Streamlines nach.





Konstantin Lackner 03.07.2020

HL: Generell ist bei High Level Sachen eher die Frage was zu sehen ist, nicht wie.

HL: Abstraction

Was soll sichtbar sein und was erkennbar.

HL: Smart Visibility

Unobstructed view of important parts, bspw. Durchsichtigkeit von Sachen oder das Fehlen. Auch oft smart lenses, also so kleine Linsen wo Bereiche angezoomt sind. Auch Cutaways.

HL: Transparency

Viewdependent Transparacny heißt, dass, je nach Winkel, andere Sachen andere Transparancy haben.

HL: Cutaways, Breakaways

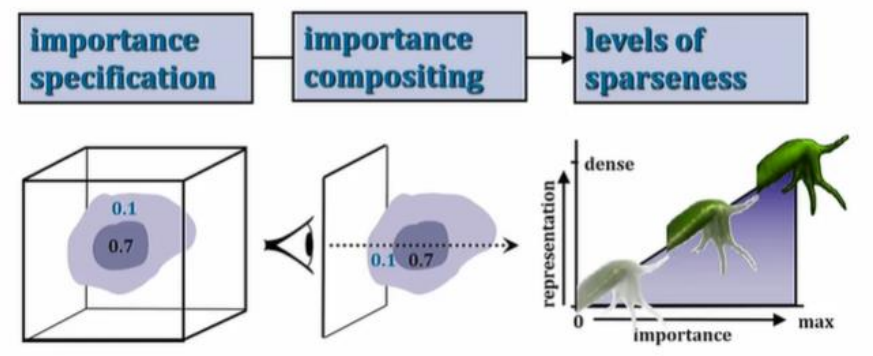
Eh kloa

HL: Colume Splitting

Man sieht hier das Öffnen.

HL: Importance-driven feature enhancement

Man legt am Anfang fest welcher Part wie important ist

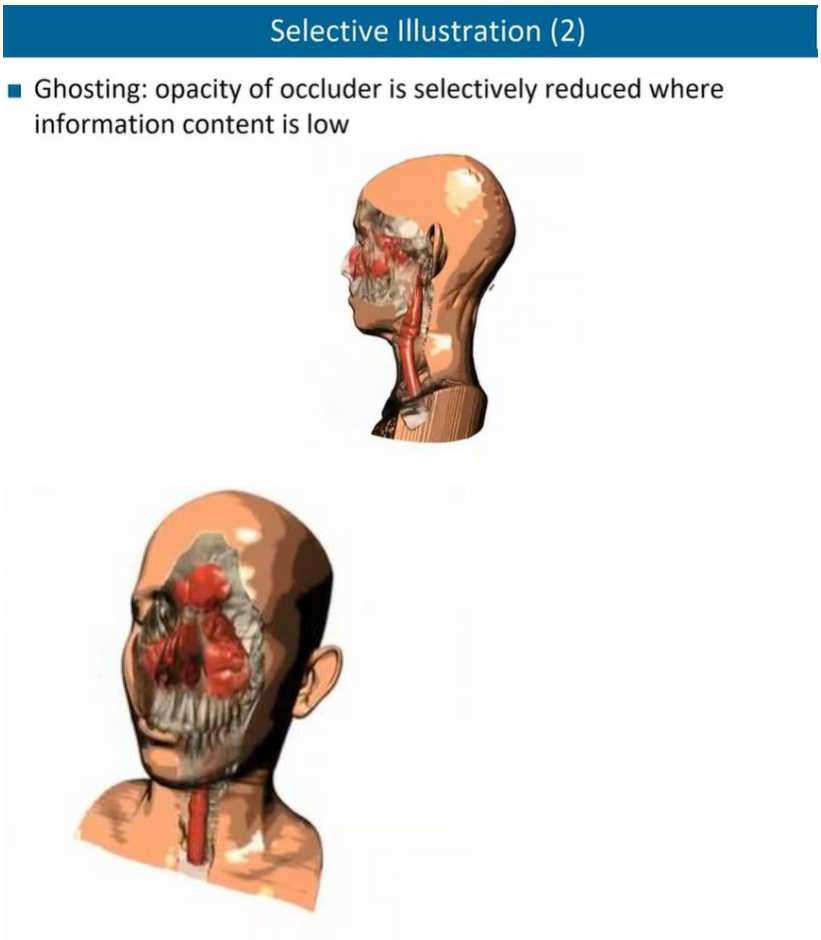


Man ordnet dann auch die Sachen an. Was sehr wichtig ist, kommt nach vorne und ist sehr opaque.

Konstantin Lackner 03.07.2020

Ghosting

Beim Rotieren wird dennoch der Part gezeigt der wichtig ist, nur von der anderen Seite.



Digital Fabrication

Der Prozess digital designte Objekte dann im echten Leben zu kreiern heißt digital Fabrication.

Application areas

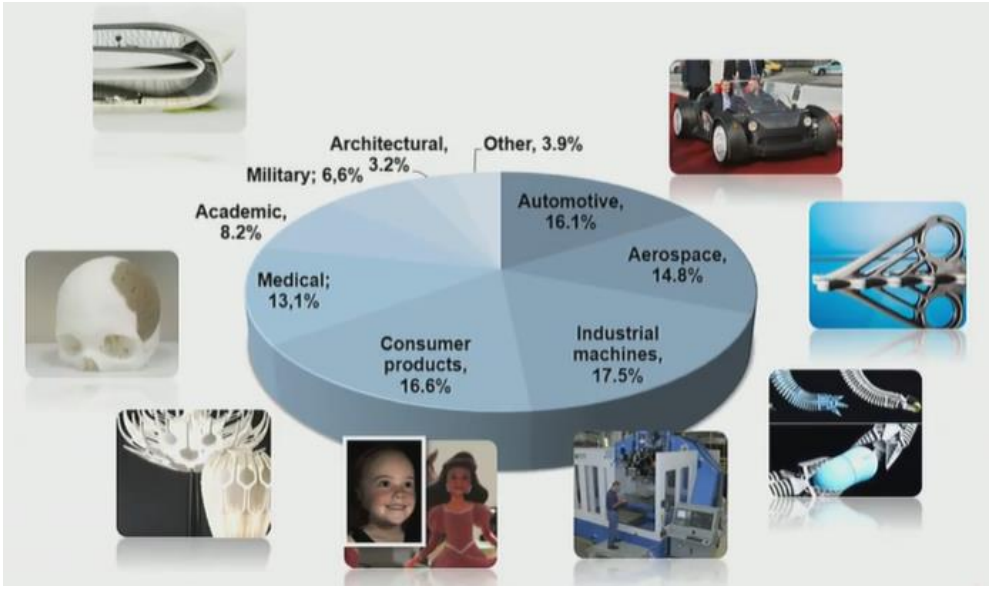
Industry – Military, Ventilation Beispiele

Aerospace

Medical domain

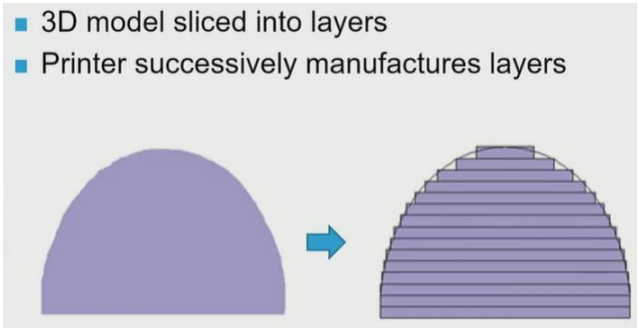
Socio-cultural

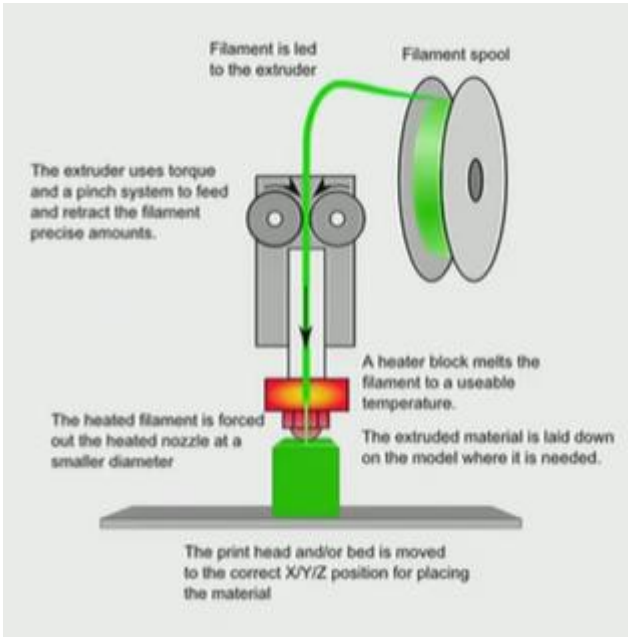
Konstantin Lackner 03.07.2020



3D Printing

Man printet in layern.





Konstantin Lackner 03.07.2020

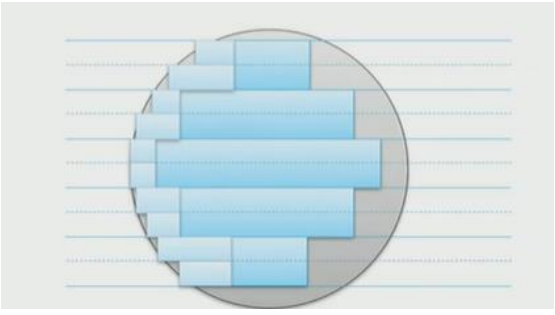
Steps beim 3D Printing

Input Model muss erstellt werden.

Orientierung und Positionierung für Materialeinsparung und Platz im Printer, außerdem kann der Printer vl. in eine Richtung besser arbeiten. Und die Stärke des fertigen Dings hängt davon ab wie die Strukturen geprintet wurden.

Support Structures weil es sonst eppa umknickt.

Slicing Objekte bestehen ja aus slyces. Indem man die lyerdicke verändert kann man die Genauigkeit des Models anpassen.



Path planning Man muss den path genau optimieren damit man erstens ned ewig braucht und tausend mal hin und her fährt. Außerdem muss ein layer coolen bevor der nächste aufgetragen wird aber ned zu sehr.

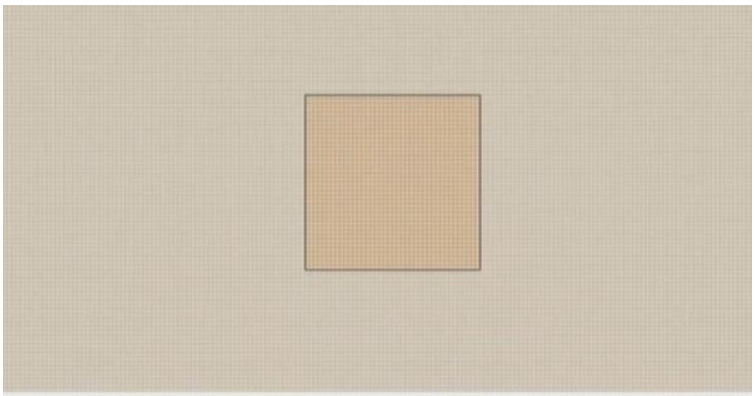
Machine instructions Der printer braucht instructions um das finale Produkt zu produzieren.

Konstantin Lackner 03.07.2020

Deformable Objects

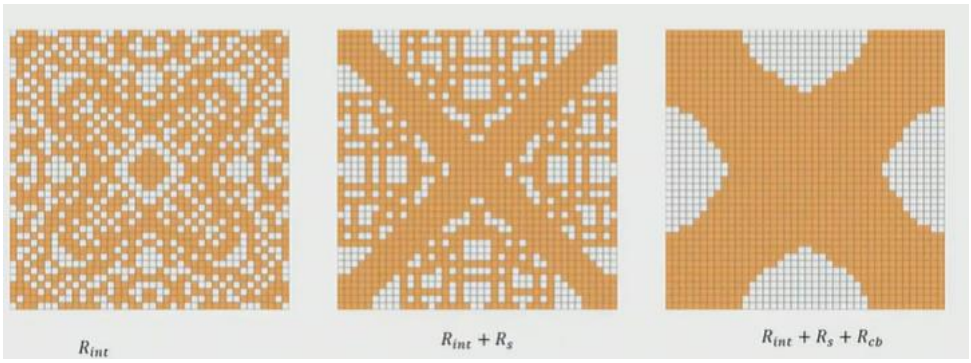
Stiffness und co können mit Materialmischungen oder auch mit Strukturen (meshes) angepasst werden.

Ein approach um die richtige Struktur zu finden, ist die Topology Optimization. Man Teilt eine Zelle in Voxels und kann damit die Stiffness testen.



$$O(\alpha) = \left\| C^{goal} - \tilde{C}(\alpha) \right\|_F^2 + R$$

- $R = w_{int}R_{int} + w_s R_s + w_{cb}R_{cb}$
- Integer
- Smooth
- Checkerboard



Beim Mischen von Materialien benutzt man subregions des Objekts und teilt ein wo was sein muss um ein deformation model zu erfüllen.

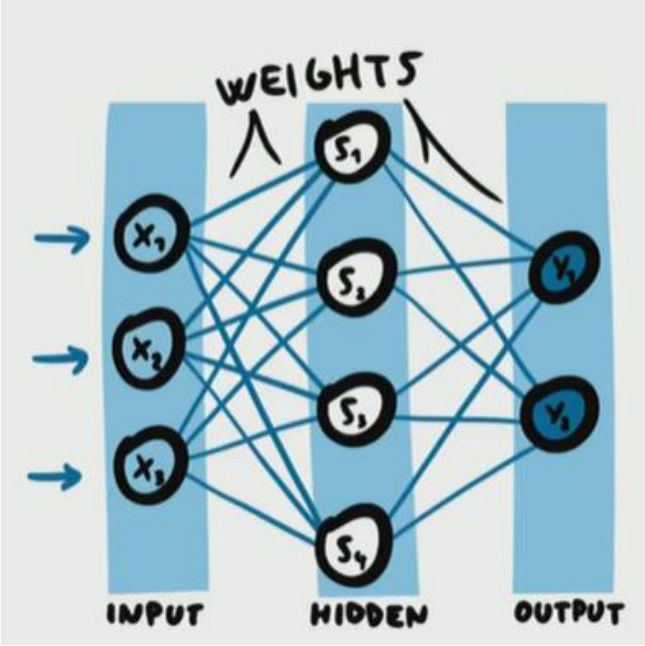
Konstantin Lackner 03.07.2020

Artificial Neural Networks

Machine learning is a set of algorithms that allow computer systems to perform tasks without explicit instructions.

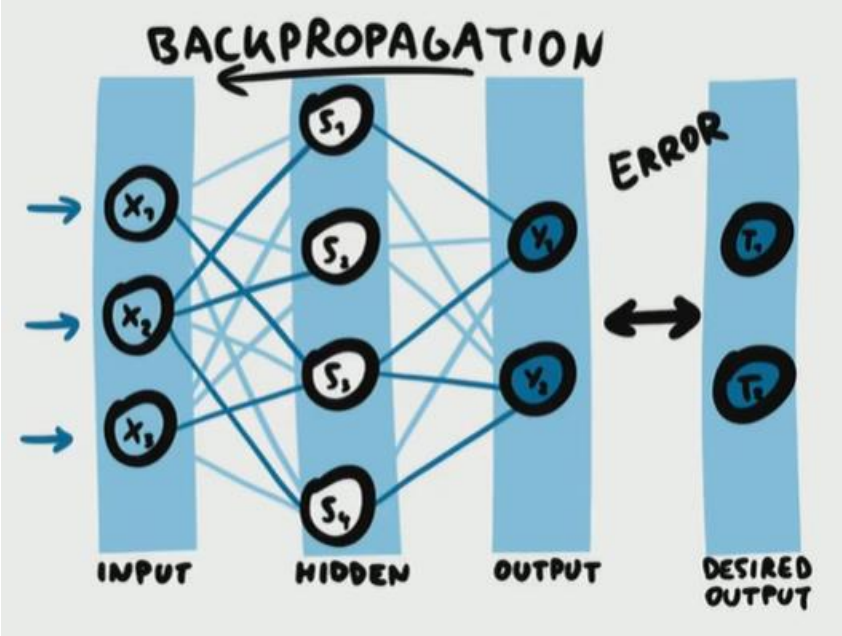
Sind Programme bei denen alle Inputs durch Parameter definiert ist. Die können damit auch alle Tasks lösen.

Dabei gibt es drei Layer. Den Inputlayer wo die Daten reingehen. Dann einen hidden layer bei dem iwelche Berechnungen stattfinden und einen output layer wo der output rauskommt duh.



Die weights sind die Werte der Parameter.

Das Zuteilen der Weights passiert durch Training. Man hat also eine richtige Antwort, haut den Input rein der das ergeben soll, wenn was falsches reinkommt, haut man von der anderen Seite den Error zurück und schaut was geändert werden muss.



Das nennt man supervised learning. Man trainiert das Network mit einem labled Dataset, also den richtigen Antworten.

Es gibt aber auch unsupervised learning. Da versuchen die Algortihmen patterns in den Datensätzen zu finden um diese dann zusammenzuclustern.

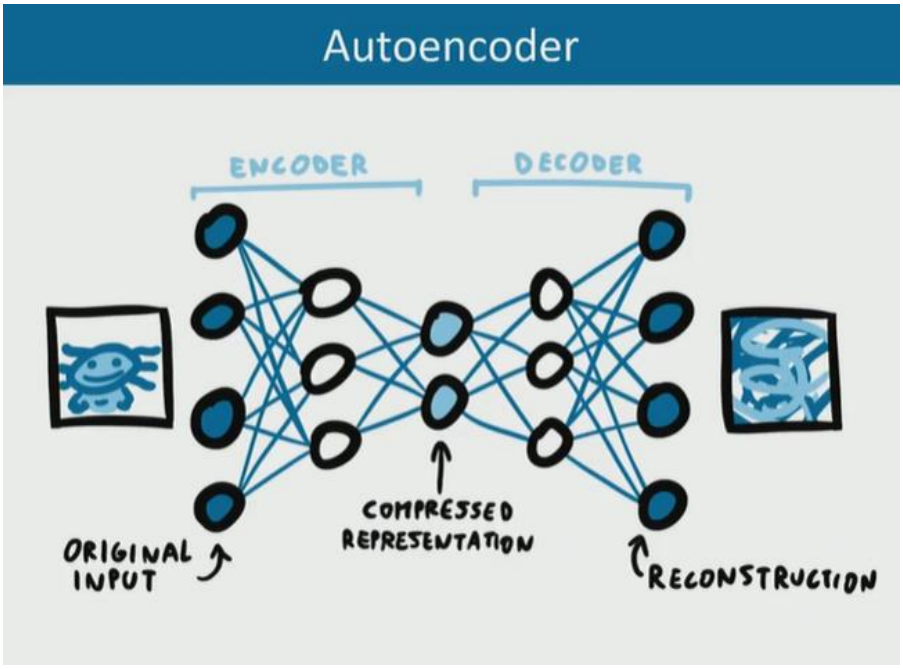
Dann gibt es noch reinforcement learning wo man das Network auf Sachen trainiert wo mehrere Schritte gemacht werden müssen (bspw. Schach) – da probiert die Maschine einfach random Sachen so lange bis sie zu einem Resultat kommt, ist dieses gut, wird sie belohnt, ist das schlecht, wird sie bestraft.

Unsupervised Learning

Autoencoder

Ein Autoencoder ist eine Maschine die einen Input bekommt und versucht exakt denselben Input wieder zurückzugeben. Der Trick dabei ist, dass man aber in den Mittellayern nicht

einfach dieselbe Anzahl an Neuronen hat wie im Input. Die Maschine kann also ned einfach die Daten weiterreichen.

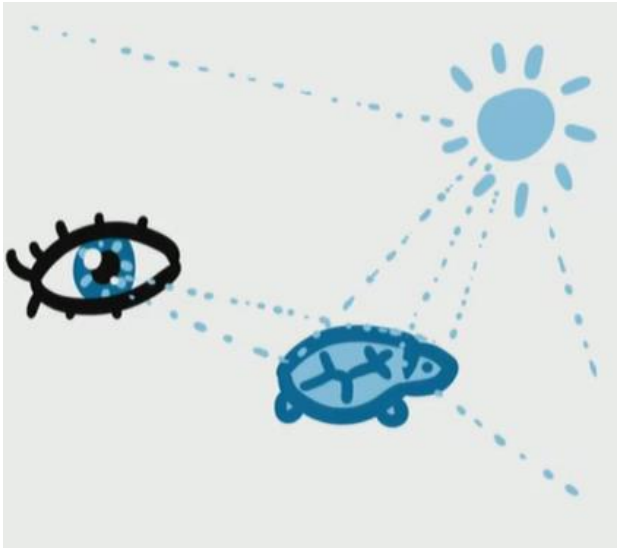


Deswegen muss der erste Part encoden, der zweite decoden.

Man compared einfach Input und Output am Ende und damit kann das Network sich selbst trainieren.

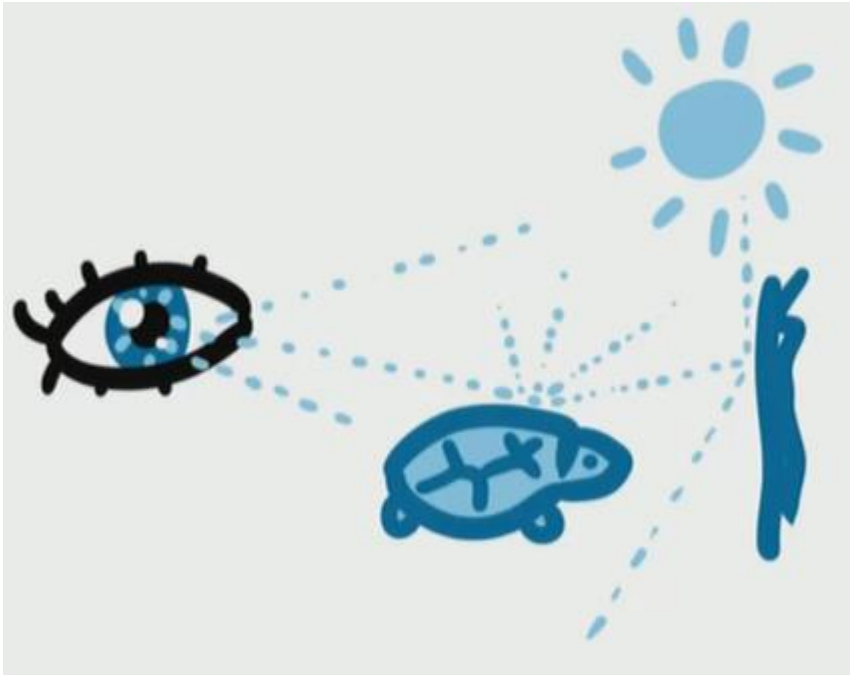
Supervised learning

Path tracing



Will man sowas im Computer nachbauen, schießt man Rays vom Aug aus anstatt von der Sonne, weil man ja nur die braucht die im Auge landen. Alles andere wäre nicht machbar. Dann schaut man wo die hinfliegen wo sie abprallen und berechnet daraus die Farbe, die im Auge ankommt. Wenn zuerst die Schildkröte dann Wand und dann Sonne getroffen würde, ist das resultierende Bild sicherlich dunkler als wenn zuerst die Schildkröte und dann direkt die Sonne getroffen wird.

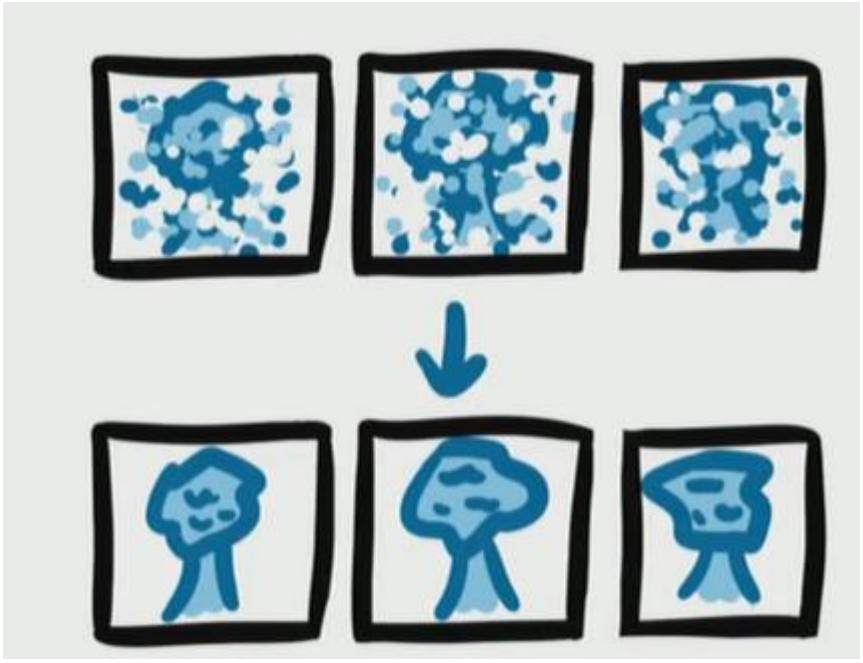
Konstantin Lackner 03.07.2020



Was man mit machine learning machen kann ist zuerst ein sehr low res Bild des Raytracing zu machen und dann das mit einem machine learning algorithm cleanen.

Dazu muss man ein Netzwerk trainieren, das macht man am besten indem man einmal sehr aufwändig ein richtiges Path Tracing Bild generiert und dann Anhand von diesem mit noisy images trainiert. Das ist sozusagen supervised learning.

Probleme gibt es dabei mit animierten Szenen, weil das Netzwerk sich natürlich nicht merkt was es im vorigen Frame gemacht hat. Wenn man also einen neuen Frame errechnet, muss das nicht heißen, dass man denselben Fehler, oder dieselben Artefakte, einbaut, die man auch im Frame davor eingebaut hat. Das kann zu blinking führen.

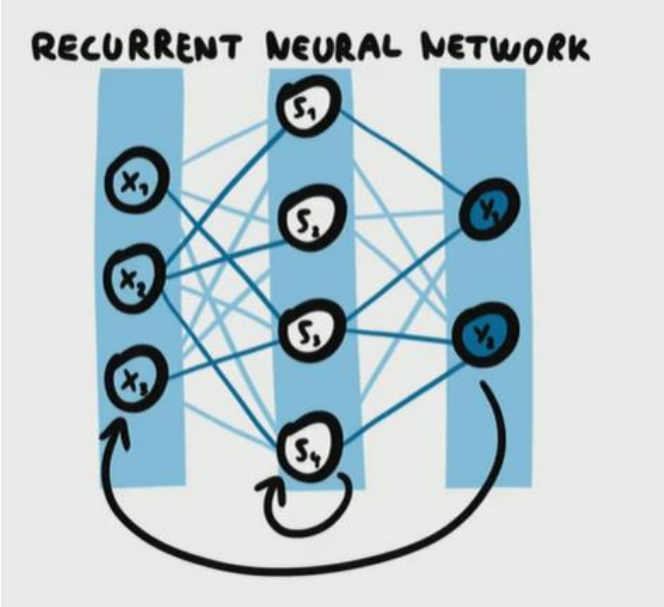


Um das zu lösen, benutzt man:

Konstantin Lackner 03.07.2020

Recurrent Neural Networks

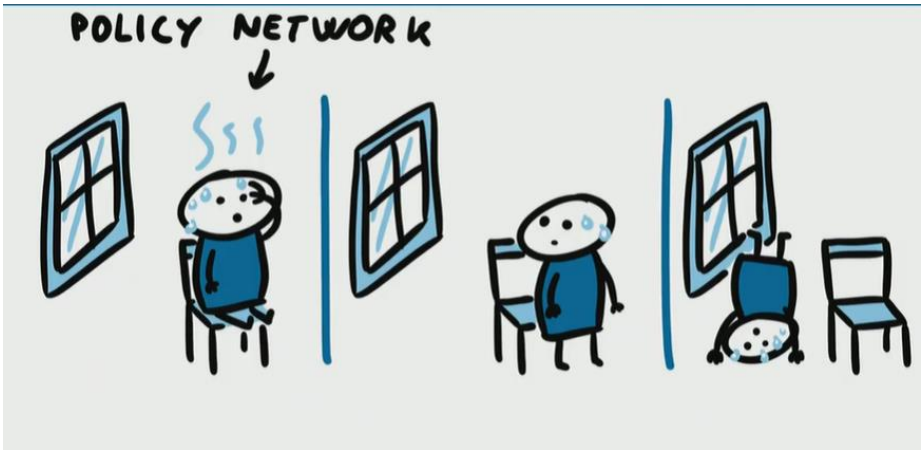
Dieses NN hat nicht nur den Input den man provided sondern zusätzlich auch noch das, was es letztes Mal gemacht hat.



Reinforcement learning

Policy Networks

Policy Networks haben Agents denen sie Anweisungen geben. Bspw. einen Strichmaxel, dass es aufsteht und das Fenster aufmacht.



Das macht es mit random Befehlen. Wenn es etwas richtig macht, wird es rewarded, wenn falsch, nicht. Das Problem hier ist, dass man auch mitunter unendlich lang brauchen könnte, weil das Network nie zufällig die richtigen Schritte macht.

Dafür gibt es ein actor critic Policy Network.

Hierbei gibt es zwei NNs. Eines ist der Actor, gibt also Befehle an das Strichmaxerl, das andere ist trainiert um jede einzelne Aktion zu sehen und versucht jede einzelne Aktion zu kritisieren. Am Anfang haben beide Networks keine Ahnung. Am Anfang könnte bspw. das

Critic Network sagen, dass vom Sessel aufstehen eine schlechte Idee ist. Das wird dann auch einfach mit der Zeit trainiert.

Reinforcement learning wird benutzt um bspw. Schach und co beizubringen. Bei Supervised Learning könnte das NN nur so gut sein wie die Person die supervised. Mit Reinforcement hat man keine Grenze.

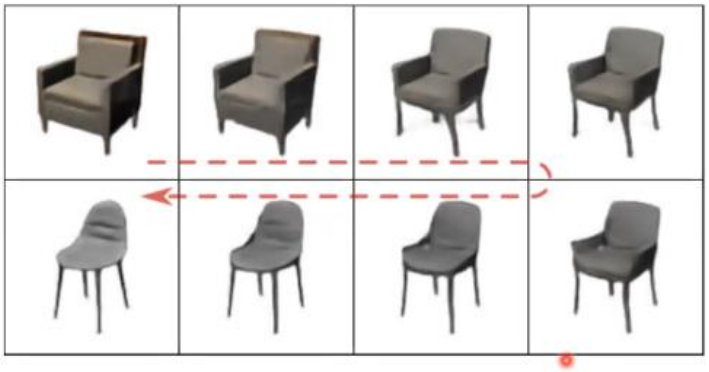
Anti-Aliasing

- Fast Approximate Anti-Aliasing (FXAA) – **fast but low quality**
  1. Find the edges based on the depth
  2. Apply per-pixel blur with the radius based on the detected edges

- Supersampling Anti-Aliasing – **high quality but slow**
  1. Render the image at higher resolution than desired
  2. Use several pixels from the rendered image to calculate one pixel in the target image

3D Modeling with NN

Man übergibt dem Network wenig Daten. Was für eine Art Objekt will man, welche Farbe und welcher viewpoint.

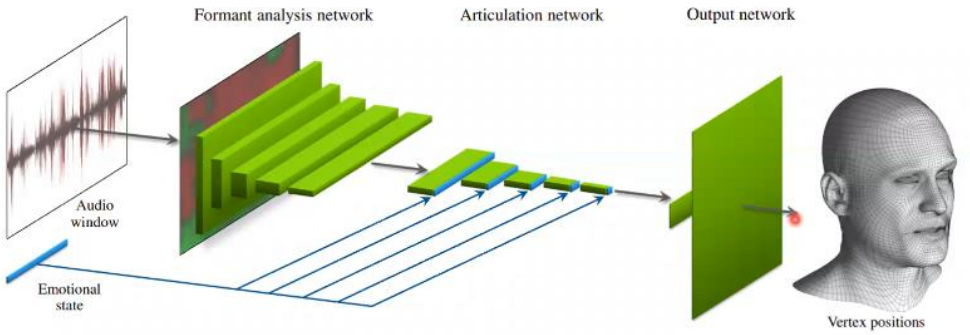


- The network learns a meaningful representation for a chair
- It can smoothly interpolate between objects

Der Algorithmus weiß actually wie chairs aussehen, das sieht man hier, weil es zwischen den beiden faden kann. Es macht ned einen Crossfade sondern iterpoliert wirklich.

Konstantin Lackner 03.07.2020

Generating Facial Axpressions



Generiert Gesichter zu gesagtem Text.