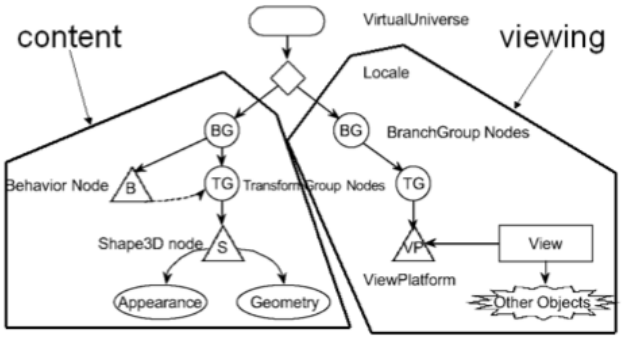
# Java 3D

**was ist das allgemein, wie wird geometrie gespeichert (--> Geometrie- und Appereance klassen), aufbau, inhalt, knotenarten**

java3d dient zur darstellung von 3d graphik, die durch einen szenegraphen definiert wird. der szene graph besteht aus dem viewing-zweig (def. d. darstellung und projektion d. parameter) und dem content-zweig (geometrie, transformationen,...).
einzelne knoten in dem graphen können detached (erzeugt aber noch nicht eingefügt), live (eingefügt aber noch bearbeitbar) und compiled (nicht mehr veränderbar) sein.

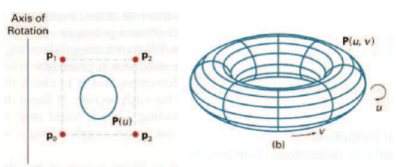



virtualUniverse: Wurzelknoten
locale: enthält ursprung des darunterliegenden teils (es ist die root für welt- und koordinatensystem)
group: vorfahre aller zwischenknoten
transformGroup: transformierung für alle nachfolger
branchGroup: gruppiert nachfolger, teilbaum kann komplett entfernt werden
shape3d: definiert geometrie (info z. renderingmodus, materialeigenschaften, textur,...)
appearence klasse: materialeigenschaften, renderingattribute
geometrie klasse: def. d. geometriedaten (koordinaten, normalvektoren,...)
light: lichtquellen in der szene
behavior: eventbehandlung, ausführung von code mit einfluß auf den szenengraphen
+ zusätzliche helperklassen (vecmath, transform3d, mouseBehavior,...)

# Advanced Modelling

## Sweeps

ein 2d objekt wird verwendet um ein translationales, rotationales oder symmetrisches 3d objekt herzustellen. dabei wird das 2d objekt anhand einer achse oder kurve bewegt und an bestimmten intervallen auf seinem pfad dupliziert. die duplizierten objekte werden mit verbindungslinien verbunden. ergebnis ist eine wireframe darstellung des objektes.
+ erzeugt objekte, die normalerweise schwer zu generieren sind
- schwer zu rendern, schwer zu modellieren



**Blobby Objects (Gauss) formel sehr ins detail gefragt, wie berechnet man es genau für eine gegebene Koordinate**

blobby objects sind objekte, die ihre form verändern (zb flüssigkeit) aber die masse bleibt gleich. ohne kontakt zu anderen sind sie meist kugeln, je näher sie anderen gleichen objekten kommen desto eher ziehen sie sich an und verschmelzen zu einem.
zur berechnung von blobby objects wird eine gauss'sche dichtefunktion herangezogen, bei der T ein grenzwert ist und a,b für die blobbiness zuständig sind (bei negativen werten von b entstehen einsenkungen anstatt erhöhungen). r^2 gibt den abstand zwischen einem punkt im raum und einem bestimmten kontrollpunkt an. a ist die standardabweichung der kurve, b ist die höhe der gausskurve, k steht für die kontrollpunkte. eine oberflächenfunktion kann demnach wie folgt definiert werden:

$$f(x,y,z) = \sum_k b_k e^{-a_k r_k^2} - T = 0 \qquad \text{where} \quad r_k^2 = x_k^2 + y_k^2 + z_k^2$$

durch die berechnung entstehen punkte auf höhenschichtlinien, die dadurch zustande kommen, dass alle punkte für einen bestimmten grenzwert berechnet werden.

**soft objects: formel, wie bekomm ich ein dreidimensionales objekt aus einer eindimensionalen dichtefunktion**

soft objects sind wie blobby objects nur mit anderer dichtefunktion (geht nach 0 in einem endlichen intervall, nicht exponentiell). d gibt dabei den abstand vom kontrollpunkt an, in dem dieser einen einfluss ausübt.

$$f(r) = \begin{cases} 1 - \frac{22r^2}{9d^2} + \frac{17r^4}{9d^4} + \frac{4r^6}{9d^6} & if 0 < r \leq d \\ 0, & if r > d \end{cases}$$

eine andere möglichkeit der berechnung von blobby objects ist das metaball-modell mit der quadratischen dichte funktion. b ist der skalierungsfaktor, d gibt auch hier den abstand vom kontrollpunkt an, in dem dieser einen einfluss ausübt.

$$f(r) = \begin{cases} b(1 - 3r^2/d^2), & if 0 < r \leq d/3 \\ \frac{3}{2}b(1 - r/d)^2, & if d/3 < r \leq d \\ 0, & if r > d \end{cases}$$

um ein dreidimensionales objekt aus einer eindimensionalen dichtefunktion zu bekommen, muss man sich vor augen halten, dass die dichtefunktione für alle punkte auf einer bestimmten fläche errechnet wird, die, als einheit gesehen eine art hohenschichtlinie ergibt. viele solche höhenschichtlinien zusammen gesehen (eine pro treshold) ergeben ein 3d objekt.

**Superquadrics (was ist das, wie schaut sowas aus, wo muss der parameter hin)**

eine quadrik ist die menge der punkte, deren koordinaten durch nullsetzen einer quadratischen funktion entsteht. superquadriken stellen eine verallgemeinerung der quadriken dar. mithilfe der einführung weiterer besonders gewählter parameter lässt sich

die ursprüngliche form der quadrik modifizieren. für kurven wird ein zusätzlicher parameter eingefügt, bei oberflächen zwei. der parameter muss dabei in den exponenten geschrieben werden. (bsp. superellipsen (2d) und superellipsoids (3d))

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

**Structure-deforming Transformations inkl. der Twist Matrix.**

strukturverändernde transformationen sind nicht-lineare transformationen

tapering: nicht lineare skalierung (zb zylinder wird zu kegel transformiert), $f_1$, $f_2$, $f_3$ sind skalierungen in x, y, und z richtung

$$X = \begin{pmatrix} f_1(s) & 0 & 0 \\ 0 & f_2(s) & 0 \\ 0 & 0 & f_3(s) \end{pmatrix} \cdot x$$

twist, bend: nicht lineare rotation

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

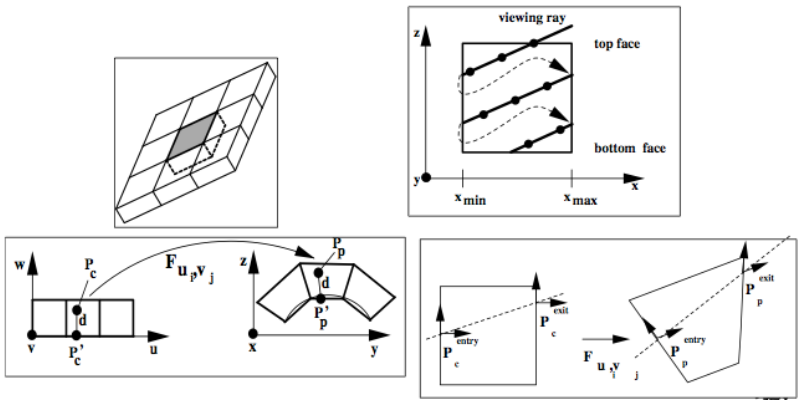
**Particle systeme: Cellular Texture Generation + Knitware + Terrain Specification/ Simulation**

partikelsysteme werden zb zur darstellung von naturphänomenen verwendet. sie bestehen aus einer größeren anzahl von objekten, die anhand von vorgegebenen parametern (farbänderung, abprallen) verändert werden können und eine bestimmte lebenslänge haben. dadurch können wasserfälle, explosionen, bienenschwärme.... modelliert werden.

cellular texture generation: ein zellpartikel system verändert die oberfläche von objekten anhand von “cell state” (position, orientation, form,...), “cell programs” (geh von der oberfläche weg, stirb nach einer bestimmten länge, pass dich an deine umgebenden zellen an,...) und “extra cell environment” (richtung der nachbarn, konzentration an einem ort,...). eine zelle ist eine gruppe von polygonen. je weiter weg das darzustellende objekt ist, desto weniger polygone müssen verwendet werden (level of detail). durch die parametrisierung der zellen ist es einfach auch komplizierte objekte zu texturieren. (bsp fell, stacheln,...)

knitware: strickmuster werden durch die verwendung von basis elementen der mikrostruktur (“instance volume elements”) erzeugt. dabei gibt es basis objekte für rechts und links maschen. die volumen elemente bestehen aus 2 dimensional gedrehten und rotierten kurven um den faden darzustellen. um die strickmuster zu rendern wird raycasting verwendet. bei einfachen strickmustern (nur rechtsmaschen, oder nur linksmaschen), wird nicht das fertige strickmuster gerendert, sondern nur das basis

element. dabei wird davon ausgegangen, dass die beleuchtung regelmäßig ist und parallele projektion verwendet wird. ein strahl wird durch das element gelegt und zyklisch, am wert der ausgangsstelle, wieder von der anderen seite durch das basiselement gelegt. dieser vorgang wird sooft wiederholt, bis der strahl am boden des basiselements austritt. für das rendern von verschiedenen maschentypen kann dieser zyklische neueintritt nicht gemacht werden.




terrain simulation: die simulation von gelände erfolgt durch das zusammensetzen mehrerer elemente. wichtig bei der erstellung einer solchen umgebung ist der realismus (fraktale, geographische daten/gegebenheiten, ...), nicht die physikalische genauigkeit. zur erstellung von objekten wird häufig auf fraktale (initiator, generator) zurückgegriffen, da objekte wie zb berge, bäume,... eine gewisse selbstähnlichkeit aufweisen. auch das “level of detail” ist zu beachten, da zb. bei einer flugsimulation die einzelnen grashalme an wichtigkeit verlieren. um die simulation realistisch wirken zu lassen sind komplexe modelle notwendig. dazu gehören die pflanzenverteilung des ecosystems, die reduktion geometrischer komplexität (annäherung an ähnliche pflanzen,...), parametrisierung der modelle von pflanzen (man speichert nur parameter (zb winkel) und wendet diese auf generierte strukturen an). schatten müssen bei vielen simulationen auch nicht korrekt berechnet werden sondern können approximiert werden (baumkrone -> kreis,...).
terrain spezifikation kann durch höhenlinien (zb berge), farbliche veränderungen (zb flüsse) oder noise (zb hügel) angegeben werden. pflanzen verteilungen in terrains können auf ähnliche art und weise spezifiziert werden (verschieden farbene punktwolken).

# Curves and surfaces in CAGD

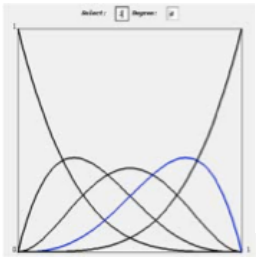
auszug aus cg1:
folgende eigenschaften charakterisieren die verschiedenen kurventypen:
interpolierend (kurve verläuft durch ihre stützpunkte) versus approximierend (kontrollpunkte liegen neben der kurve, diese art von kurven wird hier genauer besprochen)
stetigkeitsgrad an verbindungen
globaler einfluss (alle punkte beeinflussen jeden kurvenpunkt) versus lokaler einfluss (punkte beeinflussen nur nahe kurventeile)
achsenabhängige darstellung (drehung des koordinatensystems verändert kurve) versus achsenunabhängige darstellung (drehung des koordinatensystems verändert kurve nicht)
tendenz zur dämpfung an ecken versus tendenz zum überschwingen
mögliche kurvenformen, einschränkungen, doppelte punkte, geschlossene kurven usw.
kurven, die durch stütz- oder kontrollpunkte definiert sind, nennt man auch splines.

eine parametrische kurve ist definiert durch:

$$c: \qquad \boldsymbol{c}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}; \quad t \in I = [a,b] \subset \mathbb{R}$$

**Bezier Kurve, De Casteljau-Algorithmus (Eigenschaften, wie der Algorithmus funktioniert, anwenden, zeichnen ...)**

die bezier kurve ist eine *approximierende* kurve, bei der die sogenannten bernstein-polynome ($B_{n,i}$) als gewichtsfunktionen für die kontrollpunkte verwendet werden. jeder kurvenpunkt ist das gewichtete mittel aller kontrollpunkte. die summe aller bernsteinpolynome für einen gegebenen punkt ist 1. $b_i$ in der berechnungsformel bezieht sich auf die kontrollpunkte.

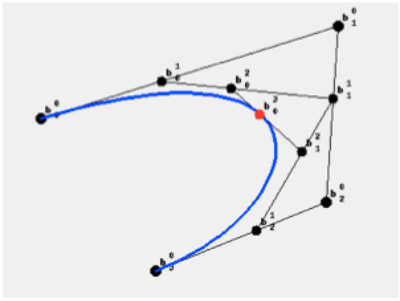


$$\boldsymbol{b}(t) = \sum_{i=0}^{n} \boldsymbol{b}_i B_i^n(t)$$

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

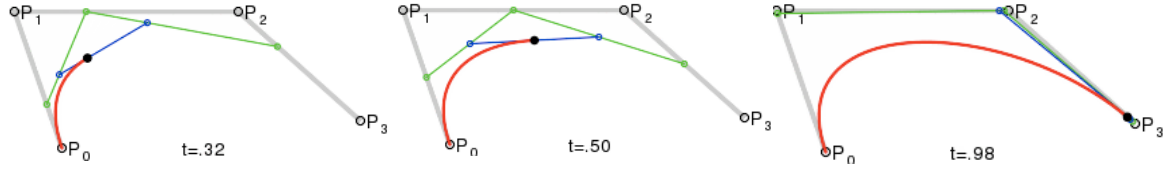
der algorithmus von de casteljau beruht darauf, dass eine bézierkurve geteilt und durch zwei aneinandergesetzte bézierkurven (mit jeweils 4 kontrollpunkten) dargestellt werden kann. diese unterteilung kann rekursiv fortgesetzt werden. das kontrollpolygon der zusammengesetzten bézierkurve nähert sich dabei der originalkurve an. nach ausreichend vielen verfeinerungsschritten kann der entstandene polygonzug als näherung für die originalkurve verwendet werden.

$$\begin{array}{ccccc} b_0^0 & & & & \\ & b_0^1 & & & \\ b_1^0 & & b_0^2 & & \\ & b_1^1 & & b_0^3 & \\ b_2^0 & & b_1^2 & & \\ & b_2^1 & & & \\ b_3^0 & & & & \end{array}$$



$$\boldsymbol{b}_i^r := (1-t)\boldsymbol{b}_i^{r-1} + t\boldsymbol{b}_{i+1}^{r-1}, \qquad i = r, \ldots, n$$

in der eben erwähnten formel ist r die iterationsstufe, i der kontrollpunkt und t eine zahl zwischen 0 und 1, die angibt auf wieviel prozent die verbindungslinie getrennt wird.



eigenschaften von bezier-kurven:
affine invariance: affine transformationen der bezier punkte transformieren die kurve
bei n+1 kontrollpunkten ist der grad der bezier kurve n. durch viele kontrollpunkte ist die kurve zwar flexibler, aber die berechnungen sind kostspieliger und die kontrolle über die kurve niedriger
pseudo local control: jeder kontrollpunkt zieht die kurve wie mit einem gummiband an, globaler einfluss (gewichtsfunktion fast überall >0) bedeutet, wenn ein punkt verändert wird hat das zwar am meisten einfluss in der umgebung des punktes, nimmt aber auf die gesamte kurve einfluss
endpoint interpolation: $p_0$ (erster punkt) und $p_n$ (letzer punkt) liegen auf der kurve
linear precision: liegen alle bezier punkte auf einer linie ist die bezier kurve diese linie
die tangenten in $p_0$ und $p_n$ sind die verbindung zu den nächsten punkten $p_1$ und $p_{n-1}$
convex hull property: die kurve liegt zur gänze in der konvexen hülle der kontrollpunkte
variation diminishing property: eine bezier kurve hat nicht mehr schnittpunkte mit einer linie als ihr bezier polygon (siehe bild bei b-spline)
die tangente von $b_{n\ 0}(t)$ ist die verbindung der punkte $b_{n-1\ 0}$ und $b_{n-1\ 1}$

algorithmen:
degree elevation: erzeugen neuer kontrollpunkte ohne die form der kurve zu verändern
subdivision: unterteilung der bezier kurve (wird auch durch de casteljau erreicht)

zur evaluierung (darstellung) von bezier kurven kann auch das horner schema statt dem de casteljau algorithmus herangezogen werden

**b-splines: eigenschaften, in welchem raum liegen die knoten, welche dimension haben kontrollpunkte und knoten. was is affine invarianz, was ist projektionsinvarianz, trifft das zu? auf welche kurven trifft das zu? kontrollpunkte vs. knotenpunkte**

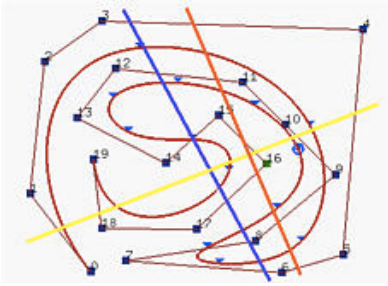
die sogenannten b-splines sind ebenso wie die bezier kurven approximierende kurven, jedoch sind die bernsteinpolynome durch b-spline-polynome $N_{k,i}(u)$ ersetzt (cox-deboor). diese beschränken die anzahl der kontrollpunkte, die einen kurvenpunkt beeinflussen, auf k. die berechnung der $N_{k,i}(u)$ ist etwas komplexer und erfolgt rekursiv. jede kurve der b-spline- polynome ist nur in einem begrenzten bereich ungleich null, jeder punkt hat über weite bereiche der kurve keinen einfluss. wenn k des b-spline-polynoms auf n+1 gesetzt wird erhält man eine bezier kurve (dann haben wieder alle punkte einfluss auf alle anderen). s(u) ist die formel zur berechnung von koordinaten auf der gegebenen kurve. in der formel ist $d_i$ ein eingabeset von n+1 kontrollpunkten.

$$s(u) = \sum_{i=0}^{n} d_i N_i^k(u)$$

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

$$N_i^r(u) = \frac{u - u_i}{u_{i+r-1} - u_i} N_i^{r-1}(u) + \frac{u_{i+r} - u}{u_{i+r} - u_{i+1}} N_{i+1}^{r-1}(u); \qquad r = 1, ..., k$$

eigenschaften von b-splines:
affine invariance: affine transformationen der b-spline punkte transformieren die kurve
strong convex hull property: das kurvensegment liegt in der konvexen hülle seiner kontrollpunkte
variation diminishing property: eine b-spline kurve hat nicht mehr schnittpunkte mit einer linie als ihr bezier polygon



local support: wenn ein punkt geändert wird, werden nicht alle anderen beeinflußt
knotenpunkte mit multiplen k stimmen mit einem kontrollpunkt überein. (kontrollpunkte n = anzahl der punkte anhand derer die kurve bestimmt wird, dimension k = grad der gewichte (wieviele gewichtskurven für die berechnung der kurve benötigt werden), knotenvektor U = $\{u_0,....,u_{n+k}\}$)
eine b-spline der ordnung k die nur knoten mit multiplen k hat ist eine bezier kurve
differenzierbarkeit

es gibt verschiedene arten von b-splines: offene (endpunkte sind von einander entfernt), geschlossene (endpunkte fallen zusammen), uniform (der abstand zwischen den knoten ist konstant, dadurch haben uniform b-splines eine periodische basisfunktion und algorithmen werden vereinfacht)

zur evaluierung von b-splines kann der de boor algorithmus (ähnlich dem de casteljau, de casteljau ist im endeffekt ein sonderfall von de boor) herangezogen werden

**NURBS, warum wird normalisiert bei rationalen kurven? warum sind NURBS perspektivisch invariant, B-splines aber nicht**

rationale kurven:
der unterschied zwischen splines und rationalen kurven ist der, dass es bei rationalen kurven möglich ist auf die gewichtung von kontrollpunkten einfluss zu nehmen. bei splines können kontrollpunkte mehrfach genommen werden, aber der einfluss wird schnell stärker (1facher kontrollpunkt -> 1facher einfluss, 2facher kontrollpunkt -> 2facher einfluss). bei rationalen kurven können kontrollpunkte den einfluss von zb 1.6 auf die kurve ausüben. bei rationalen kurven wird normalisiert damit die summe der gewichte über alle kontrollpunkte 1 ergibt, da sonst die konvexe-hülle-eigenschaft verletzt wird. es werden homogene koordinaten verwendet, bei denen das gewicht als 4te koordinate angegeben ist. die homogenen koordinaten können durch division durch x(u) wieder in euklidische koordinaten umgewandelt werden.

$$\mathbf{c}(u) = \begin{pmatrix} w(u) \\ x(u) \\ y(u) \\ z(u) \end{pmatrix}$$

eigenschaften rationale bezier kurven:

$$\boldsymbol{b}(u) = \frac{\sum_{i=0}^{n} w_i \boldsymbol{b}_i B_i^n(u)}{\sum_{i=0}^{n} w_i B_i^n(u)}$$

dieselben von bezier kurven

müssen nicht innerhalb des kontroll polygons liegen, wenn negative gewichte erlaubt sind

projective invariant: die projektion einer rationalen bezier kurve ist wieder eine rationale bezier kurve. das funktioniert bei anderen kurven (zb b-splines nicht), da man zusätzlich gewichte verändern muss damit die lage der kontrollpunkte zueinander nicht verzerrt wird.. bei der bezier kurve funktioniert das nicht, da die gewichte immer 1 sind.

gewichte sind zusätzliche design parameter: das gewicht zu verändern zieht die kurve näher an den kontrollpunkt

es ist möglich alle algorithmen von polynomiale bezier kurven auch auf rational bezier kurven, ohne änderung der homogenen koordinaten, anzuwenden

nurbs (non uniform rational b-splines):
bezier kurven (polynomial und rational) und b-spline kurven sind untergruppen von nurbs.

$$\boldsymbol{n}(u) = \frac{\sum_{i=0}^{n} w_i \boldsymbol{d}_i N_i^k(u)}{\sum_{i=0}^{n} w_i N_i^k(u)}$$

non uniform bezieht sich darauf, wie der knotenvektor definiert wird, also wenn man knotenpunkte nicht uniform (mit dem selben abstand zueinander) anordnet.

eigenschaften von nurbs:

dieselben wie b-splines

projective invariant

müssen nicht innerhalb des kontroll polygons liegen, wenn negative gewichte erlaubt sind

gewichte sind zusätzliche design parameter: das gewicht zu verändern bedeutet nur das gewicht in einem bestimmten intervall zu ändern

**flächen**

intuitiv kann man eine fläche als kurve auffassen, die sich durch den raum bewegt und dabei ihre form verändert.
eine kurve in einem 3d raum wird durch folgende formal beschrieben, wobei $F_i(u)$ die gewählte basisfunktion ist (bernsteinpolynom, b-spline basis funktion) und $c_i$ die kontrollpunkte beschriebt.

$$\boldsymbol{f}(u) = \sum_{i=0}^{n} \boldsymbol{c}_i F_i(u)$$

wird f durch den raum bewegt und gleichzeitig deformiert entspricht das einer anhand der kontrollpunkte ständig geänderten kurve.

$$\boldsymbol{c}_i(v) = \sum_{j=0}^{m} \boldsymbol{a}_{ij} G_j(v)$$

werden die beiden oberen gleichungen kombiniert erhält man eine tensorproduktfläche.

$$\boldsymbol{s}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \boldsymbol{a}_{ij} F_i(u) G_j(v)$$

# 3D Datenstrukturen

geometrische daten müssen in einer datenstruktur abgespeichert werden. es gibt mehrere mögliche datenstrukturen, die verschiedene vor- und nachteile haben.

### point cloud

bei der punktwolke existiert keine direkte information über das objekt. die punkte werden einfach anhand ihrer koordinaten dargestellt. um die sichtbarkeit zu überprüfen wird ein z-buffer eingesetzt. rendern erfolgt entweder über polygone, die zwischen benachbarten punkten aufgezogen werden, oder bei einer großen anzahl von punkten einfach durch deren darstellung. transformationen sind bei punktewolken sehr einfach, da sie über transformation der einzelpunkte erfolgen, kombinationen sind dafür nicht ohne zusätzliche annahmen möglich, da keine information über innen und außen des objekts vorhanden ist. man kann lediglich die punkte zusammenfassen.

### wire frame model

das wire frame model stellt objekte durch ihre kanten dar. als datenstruktur dient eine punkt -> punktliste -> kante -> kantenliste -> objekt. um gekrümmte oberflächen zu repräsentieren wird eine große anzahl von kanten benötigt. wie bei der point cloud sind transformationen sehr einfach, kombinationen aber nur durch kombination der objekte möglich, was aber zu sehr verwirrenden ansichten führen kann.

### B-Reps (Beschreibung, Vereinigung) wie sieht die struktur aus, kombination, was gibt es noch für datenstrukturen?

boundary representations sind ähnlich wie das wireframe modell, haben aber zusätzlich zu den punkten und kanten auch polygone mit zusätzlicher information (ebenengleichung, normalvektor,...) gespeichert. bei transformationen muss darauf geachtet werden, dass zusätzlich zu den punkten auch die zusatzinformationen geändert werden. b-reps müssen geschlossene, konvexe objekte sein und dürfen keine doppelten punkte enthalten.
durch die zusätzliche information können bei der b-rep auch einfache kombinationen durchgeführt werden. kombinationen werden schrittweise durchgeführt:

die polygone von objekt a werden mit objekt b geschnitten und umgekehrt (um herauszufinden bei welchen das notwendig ist wird zunächst für jedes polygon eine quadratumgebung erzeugt und anhand derer leicht überprüft, ob sich die polygone schneiden)

im nächsten schritt werden die polygone beider objekte klassifiziert als “im anderen objekt”, “außerhalb des anderen objekts” oder “an der oberfläche des anderen objekts” (dafür wird ein strahl in normalvektorrichtung des zu testenden polygons gelegt und mit allen polygonen des anderen objekts geschnitten. das vorderste polygon des anderen objekts wird zum vergleich herangezogen. stehen die normalvektoren in die selbe richtung ich das testende objekt innerhalb, wenn nicht außerhalb).
verbessert werden kann der algorithmus durch mitspeichern von randpunkten (punkte von a, die auf der oberfläche von b liegen und umgekehrt). dadurch wird die klassifikation von nebeneinanderliegenden polygonen einfacher

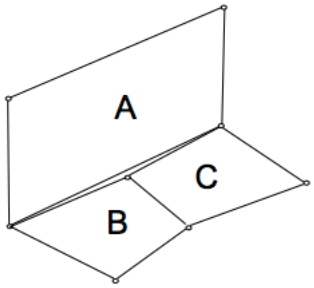
im letzten schritt werden die polygone entsprechend einer entscheidungstabelle entfernt

| Für Polygone von A | | | | |
|---|---|---|---|---|
| | in B | außerhalb B | auf B (coplanar) NV gleich | NV verschieden |
| A or B | ja | nein | nein | ja |
| A and B | nein | ja | nein | ja |
| A minus B | ja | nein | ja | nein |

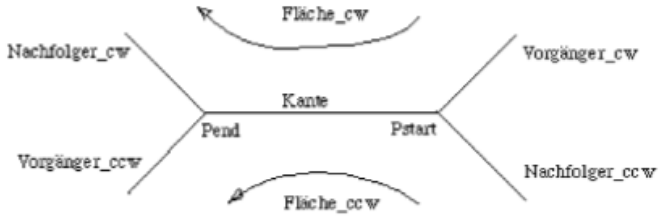
| Für Polygone von B | | | | |
|---|---|---|---|---|
| | in A | außerhalb A | auf A (coplanar) NV gleich | NV verschieden |
| A or B | ja | nein | ja | ja |
| A and B | nein | ja | ja | ja |
| A minus B | nein | ja | ja | ja |

**Winged-Edge: was ist der Vorteil von Winged Edge Datenstrukturen, was passiert wenn eine kante 3 flächen hat**

bei b-reps kann es vorkommen, dass ein sogenannter t-crack auftritt, dh zwei aneinandergrenzende polygone haben nicht dieselbe kante. dadurch können probleme beim IN-OUT-verfahren auftreten (strahl kann zwischen den polygonen durchgehen). es werden bei b-reps keine nachbarschaftsbeziehungen dargestellt.



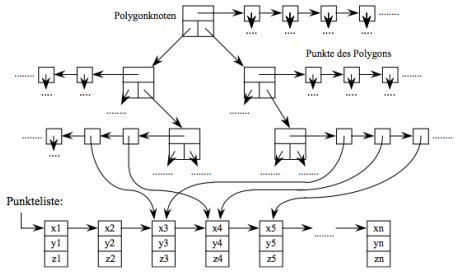

die winged-edge datenstruktur speichert zu jeder kante eines objekts die vorgänger, nachfolger und benachbarten flächen. dadurch ist sicher gestellt, dass jedes polygon, dass an eine kante grenzt berücksichtigt wird.




tesselation: wird dazu verwendet polygone in kleinere polygone(dreiecke) zu teilen um gekrümmte ebenen so exakt wie möglich zu approximieren. wenn normalvektoren von nebeneinanderliegenden flächen sehr ähnlich sind muss die fläche nicht mehr weiter unterteilt werden. $N_1 * N_2 >= 1 - \varepsilon$

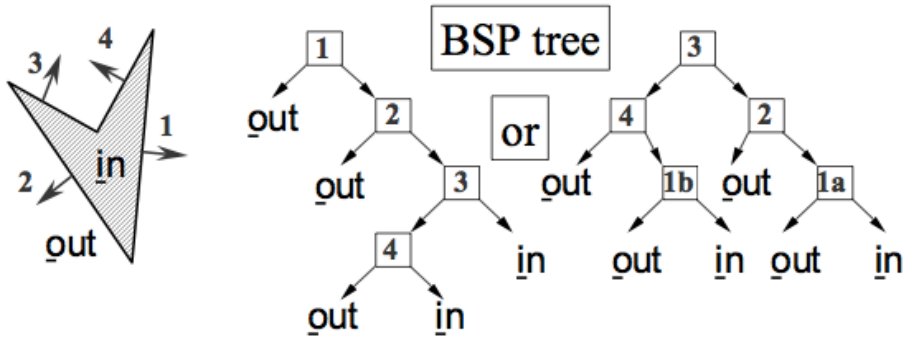
**BSP-Trees (Aufbau und Rendering aufzeichnen, wofür verwendet, wie funktioniert kombination) wie funktioniert das, Warum Baum?, Algorithmus erklären der ein Bsp Tree erzeugt (ziel: Halbierung der Objekte,...) (aufbau, wie wird geometrie gespeichert, was sind die vorteile, wie erfolgt die traversierung), interne Struktur, Rendering, Painters algorithmus, Schnitt mit Objekt, Wie gibt man im BSP-Baum die Sichtbarkeit an?**

binary space partitioning trees sind eine spezielle art von b-reps mit speicherung der sichtbarkeit anhand eines baumes. der rechte teilbaum beschreibt die objekte hinter dem polygon, der linke die davor.




algorithmus für die generierung:
nimm das polygon das am wenigsten andere zerschneidet und zerschneide die anderen polygone entsprechend dessen trägerebene (die ebene, in der das polygon liegt)
teile die polygon liste in vor und hinter dem wurzelpolygon
füge die polygone rekursiv an das wurzelpolygon

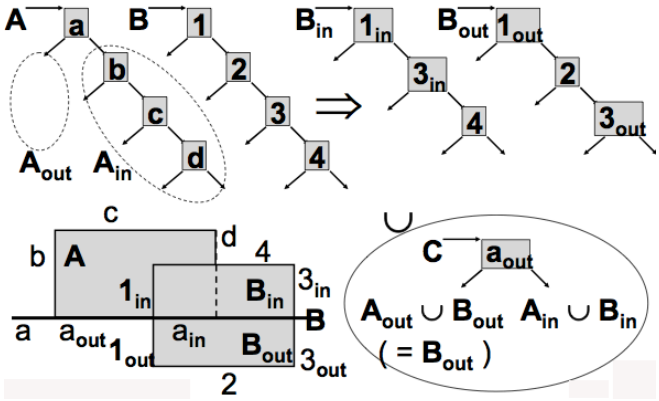
es gibt keine eindeutigen bsp bäume, wenn ein anderer wurzelknoten ausgewählt wird schaut der resultierende baum anders aus.




transformationen sind wie bei den b-reps anhand von punkten sowie ebenengleichungen und normalvektoren.

kombinationen können entweder anhand der b-reps durchgeführt werden, oder anhand der bsp bäume, was schneller geht:
das polgon a des objekts A wird mit dem objekt B geschnitten und man erhält 2 teile ($a_{in}$ und $a_{out}$)
desweiteren wird das objekt B mit der trägerebene von a geschnitten und man erhält 2 teile von B (B+ und B- der Trägerebene von a)
je nach operator wird $a_{in}$ oder $a_{out}$ als wurzelknoten von C (ergebnisbaum) gewählt und der baum wird rekursiv mit den unterbäumen $C_{left} = A_{out}$ op $B_{out}$ und $C_{right} = A_{in}$ op $B_{in}$ gefüllt

| Einfache Kombinationsregeln | | | |
|---|---|---|---|
| op | A | B | A op B |
| or | inhomogen | voll | voll |
| | inhomogen | leer | A |
| | voll | inhomogen | voll |
| | leer | inhomogen | B |
| and | inhomogen | voll | A |
| | inhomogen | leer | leer |
| | voll | inhomogen | B |
| | leer | inhomogen | leer |
| minus | inhomogen | voll | leer |
| | inhomogen | leer | A |
| | voll | inhomogen | ¬B |
| | leer | inhomogen | leer |

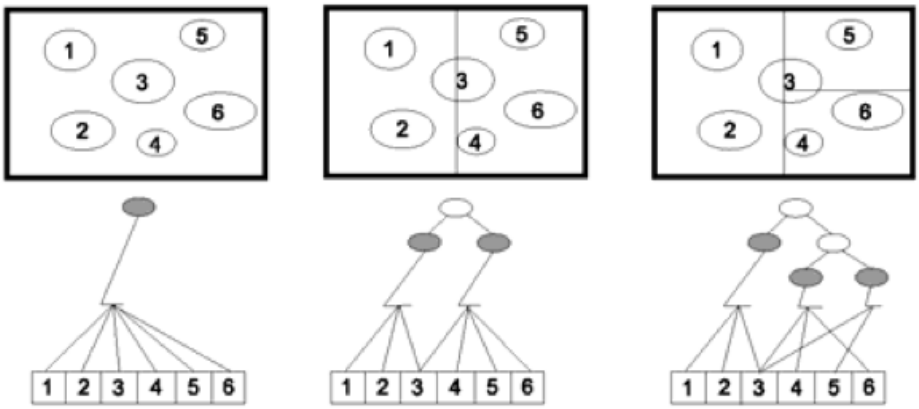



für das rendering wird ein painters algorithmus eingesetzt, dh unsichtbare polygone werden durch sichtbare übermalt.

wenn der augpunkt vor a liegt (in A+): zeichne alle A-, dann a, dann A+
wenn der augpunkt hinter a liegt (in A-): zeichne alle A+, dann a, dann A-

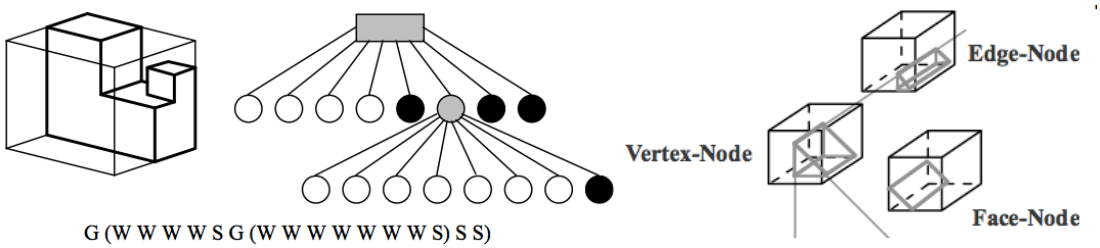
backfaceculling: bei geschlossenen objekten brauchen polygone mit nach hinten schauendem normalvektor nicht ausgegeben werden (das gilt auch für die boundary-representation)

spezialfälle von bsp bäumen sind k-D bäume die meist als index struktur verwendet werden und zeigen ob ein punkt innerhalb oder außerhalb einer gewissen fläche liegt




## Octrees

octrees sind eine datenstruktur, die aus verschieden großen würfeln besteht. jeder würfel kann schwarz (voll), weiß (leer) oder grau (weiter unterteilt) sein. transformationen sind schwer zu implementieren bis auf einige wenige (drehung um 90 grad, spiegelung,...), kombinationen hingegen sind duch ein einfache boolsche Regeln durchführbar.




erweiterte octrees haben neue knotenkanten: face nodes, edge nodes, vertex nodes

es wird eine flächen- und punkteliste erzeugt (b-rep)
die listen werden auf 8 punkte/flächenlisten aufgeteilt (flächen durch clipping an den teilungsebenen)
wenn eine der 8 doppellisten leer/voll ist s/w, wenn nicht und es ist nur ein punkt enthalten ist es ein vertex node, bei 2 flächen ein edge node, bei 1 fläche ein face node

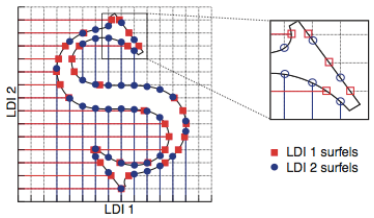
octrees eigenen sich gut als suchstruktur in einer szene (schon bei grober unterteilung).

**bintrees** sind wie octrees aber mit ebenen statt würfel. ein bintree kann für die suche optimiert werden, da die ebenen an eine beliebige stelle der szene gesetzt werden können.

**grids** aufteilung des raums in einem 3d gitter. es sollte darauf geachtet werden die zellanzahl gut zu wählen, da bei vielen zellen der speicheraufwand sehr groß wird, bei wenigen zellen aber viele objekte in einer zelle liegen können. abhilfe kann durch eine hierarchische aufteilung stattfinden: wenige zellen, aber die mit vielen objekten nocheinmal aufteilen.

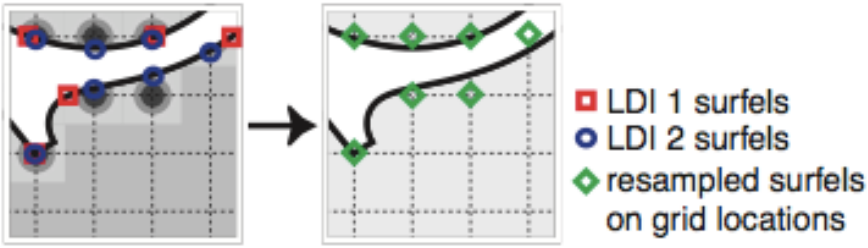
**constructive solid geometry (csg)** alle objekte werden durch kombinationen von elementarobjekten (einheitsobjekte mit seitenlänge, höhe oder radius von 1) erzeugt. die objekte werden in einem binären baum gespeichert, dessen zwischenknoten kombinationen repräsentieren und dessen endknoten elementarobjekte sind

**surfels**
surfels (simple surface element, surface voxel) werden zum rendern complexer 3d objekte verwendet.
ein surfel ist ein null-dimesionales n-tuple mit form- und schattierungsattributen, die lokal eine objektoberfläche approximieren.
sie werden in einem pre-prozess durch raytracing in drei orthographischen richtungen erstellt. die surfels speichern neben ihren koordinaten auch informationen zu tiefe, textur farbe, normalen, bump oder displacement mapping...




sampling und texture prefiltering: es werden strahlen in einem bestimmten intervall durch die objekte gelegt und an jenen punkten, an denen die strahlen das objekt schneiden werdern surfels erzeugt. die strahlen, die in die gleiche richtung durch das bild gelegt werden, werden in soganannten LDIs (layered depth images) abgespeichert. die drei LDIs in x, y, und z richtung werden gemeinsam in einem LDC (layered depth cube) oder auch block gespeichert. zum rendern muss eine effiziente hierarchische darstellung verwendet werden --> octree (benachbarte blöcke sind verschieden gefärbt, leere blöcke sind weiß und werden nicht mitgespeichert).
um eine sampling distanz zu finden, die das objekt optimal annähert ober nicht zu viel speicher benötigt muss pro ausgabepixel mindestens ein surfel gespeichert werden. im normalfall wird ein sampling intervall gewählt das den gerade erwähnten mindestabstand etwas überschreitet.

ein vorteil von surfels ist, dass texturen schon vor dem rendern vorgefiltert und auf den objektbereich gemappt werden können. um das zu bewerkstelligen werden tangent disks (ein kreis mit surfel als mittelpunkt) über jedes surfel gelegt und dessen ellipse im texturbereich (texture space) errechnet. anhand eines filters wird die farbe der texture in dem surfel errechnet (surfel mipmap). die ellipsen in textur ebene müssen sich überlappen um eine adequate rekonstruktion der textur auf dem objekt zu erlauben.
im letzten sampling schritt kann eine 3-zu-1 reduktion durchgeführt werden. dabei werden die surfels anhand eines rastes ausgerichtet. das erleichtert die speicherung, aber reduziert die qualität von form und schattierung des gesamtobjekts. meist jedoch auf nicht sehr auffällige art und weise, wodurch die speichervorteile oft überwiegen.




surface rendering: das rendering wird in mehreren schritte durchgeführt:
block culling: der ldc baum wird von oben (niedrigste auflösung) nach unten (höchste auflösung) traversiert. für jeden block wird zuerst view frustrum culling (welche oberflächen werden am bildschirm dargestellt) durchgeführt, danach werden visibility cones verwendet um backface culling anhand der normalvektoren der surfels durchzuführen.
block warping: zuerst wird die anzahl von surfels pro pixel festgestellt (1 surfel pro pixel bei schnellem rendern, für höhere qualität werden mehrere surfels pro pixel benötigt). um eine rekonstruktion der oberfläche zu ermöglichen muss für jeden block ein z-buffer herangezogen werden, der niedrig genug sein muss, dass die punkte keine löcher lassen.

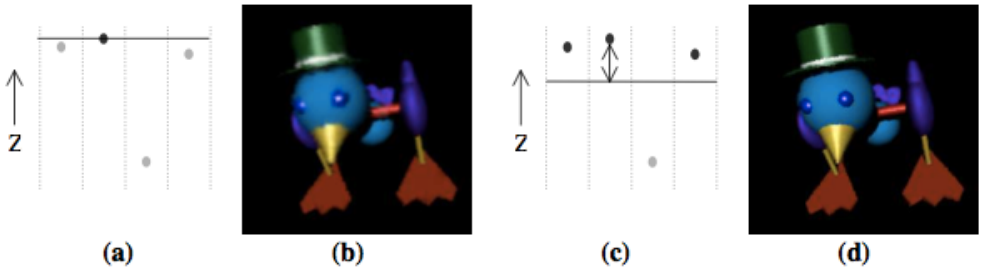



**Figure 10:** Four points are mapped to different image pixels contained in a single depth map pixel. **(a)** No depth threshold - only the nearest point is retained. **(b)** Corresponding image. **(c)** The use of a small threshold prevents points on the same surface as the nearest point from being discarded. **(d)** Corresponding image.

visibility splatting: trennt sichtbarkeitsberechnungen und rekonstruktion eines bildes
texture filtering: die im surfel gespeicherten texturwerte werden herangezogen und zwischen ihnen lineare interpoliert. um das richtige surfel mipmap level zu finden wird view-dependent texture filtering verwendet. ein punkt im bildbereich (image space) wird als ellipse in die tangent plane projeziert.
shading: das shading sollte nach dem visibility splatting vorgenommen werden um unnötige arbeit zu vermeiden. mit den schon transformierten normalen können “cube reflectance” und umgebungstexturen eingesetzt werden (phong beleuchtung). die normalen pro surfel heranzuziehen erzeugt eine gute qualität an highlights.
image reconstruction und antialiasing: um durchgehende objekte aus einzelnen surfels zu erzeugen, muss zwischen den surfels interpoliert werden. die surfels werden in die

pixel mittelpunkte gelegt (nearest neighbour). die größe das ausgabepixels entspricht der größe des z-buffers (beim supersampling wird der z-buffer multipliziert). um fehlende pixel zu interpolieren wird ein gauss filter angewendet.

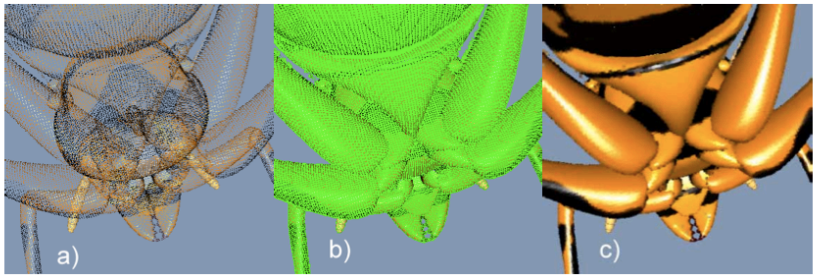
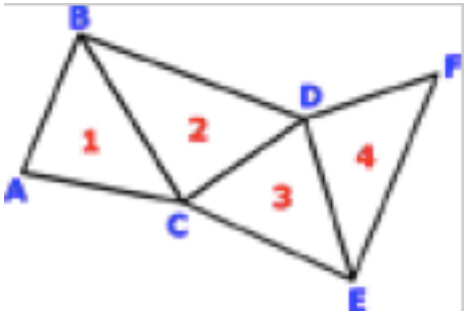



Figure 12: *Hole detection and image reconstruction. a) Surfel object with holes. b) Hole detection (hole pixels in green). c) Image reconstruction with a Gaussian filter.*

**was ist ein trianglestrip**

ein triangle strip ist eine sehr effiziente form der datenspeicherung. dabei wird ausgehend von einem dreieck mit jedem weiteren punkt ein neues dreieck definiert, dass eine kante (zwischen den letzten beiden punkten aufgespannt) mit dem vorherigen teilt.




# Uniform Sampling and Reconstruction

**Sampling und Reconstruction allgemein**

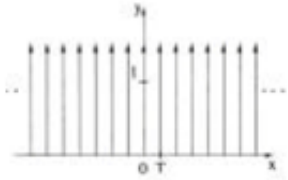
es müssen signale und funktionen von analog nach digital verwandelt werden und umgekehrt, um damit arbeiten zu können, sie zu verändern und wieder auszugeben. solche signale sind meistens kontinuierlich und unendlich, wodurch bei der digitalisierung probleme auftreten.

**was ist sampling**

um eine funktion, ein signal zu verarbeiten, werden an bestimmten abständen samples genommen. wenn die abstände regelmäßig sind nennt man das uniform sampling, wenn die abstände zwischen den samples variiren nennt man es non-uniform sampling. beim sampling muss darauf geachtet werden, dass der abstand zwischen den samples nicht zu groß oder zu klein gewählt wird, da sonst das signal nicht korrekt rekonstruiert werden kann, oder die datenmengen zu groß werden.
die beste sampling frequenz ist die nyquist frequenz $F_{N(f)} = 2u$, dh das sampling wird mit der doppelten höchsten frequenz durchgeführt.

da sich die frequenz eines signals verändern kann, kann es sein, dass nur ein kleiner teil des signals eine sehr hohe frequenz hat. in diesem fall ist die nyquist frequnz eine sehr aufwendige abtastrate, da der großteil des signals mit einer weitaus geringeren frequenz abgetastet hätte werde können. in diesem fall kann auf non-uniform sampling zurückgegriffen werden, oder ein tiefpassfilter (limitiert die frequenzhöhe auf ein intervall zwischen [-u, u], wobei u die höchste frequenz in der funktion ist) vorher angewand werden.

der prozess des samplings ist eine multiplikation des signals mit einer kammfunktion.




$$\mathrm{comb}_T(x) \Leftrightarrow \mathrm{comb}_{1/T}(?)$$

die fehler, die bei einer zu geringen abtastfrequenz auftreten können nennt man aliasing

**Wann ist eine Funktion ideal rekonstruierbar (Sampling Theorem)**

eine funktion ist ideal rekonstruierbar, wenn sie bandlimitiert ist, mit der nyquist frequenz abgetastet wurde und mit einem sinc filter rekonstruiert wird.

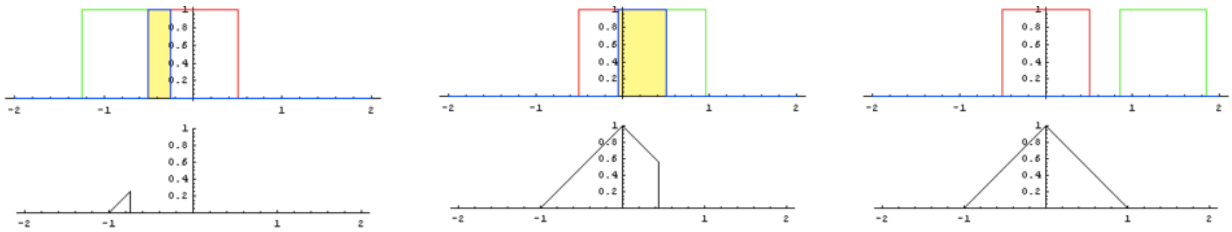
**aliasing**

alialising tritt bei einer zu niedrigen abtastrate auf. kontinuierliche information wird reduziert auf weniger information, was zur folge haben kann, dass notwendige information verloren geht. mit dem nyquist theorem kann aliasing großteils vermieden werden. als beispiel für aliasing können räder in filmen genannt werden, die sich rückwärts drehen.

**low pass filter**

low pass filter können durch faltung oder fourier transformationen durchgeführt werden.

eine gefilterte funktion ist das gewichtete mittel einer funktion, wobei eine 2te funktion die gewichte liefert. (faltung einer rechtecksfunktion mit sich selber ergibt eine dreiecksfunktion)



**Faltungstheorem**

das faltungstheorem besagt, dass das spektrum der faltung zweier funktionen gleich dem produkt der transformationen beider inputsignale ist und vice versa.
dh multiplikation im ortsbereich = faltung im frequenzbereich

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

**Fourier Transformation**

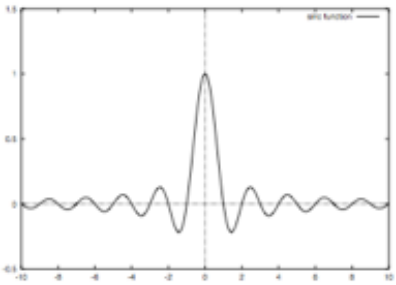
mit der fourier transformationen können signale von ortsraum zu frequenzraum überführt werden. im frequenzraum sind die signale eine summe von sinus und cosinus wellen. das signal kann auch wieder zurück verwandelt werden.

$$\mathrm{f}(x) = \int_{-\infty}^{\infty} \mathrm{F}(?)e^{2\mathrm{pj}?x}d\omega$$
$$\mathrm{F}(?) = \int_{-\infty}^{\infty} \mathrm{f}(x)e^{-2\mathrm{pj}?\mathrm{x}}dx$$

als alternative kann auch die hartley transformation verwendet werden.

**wieso ist der sinc (sinus cardinalis) filter am besten für rekonstruction? welche filter verwendet man in der praxis?**

eine bandlimitierte funktion, die mit der nyquist frequenz abgetastet wurde kann mit einem sinc filter perfekt rekonstruiert werden (anhand von glättung).



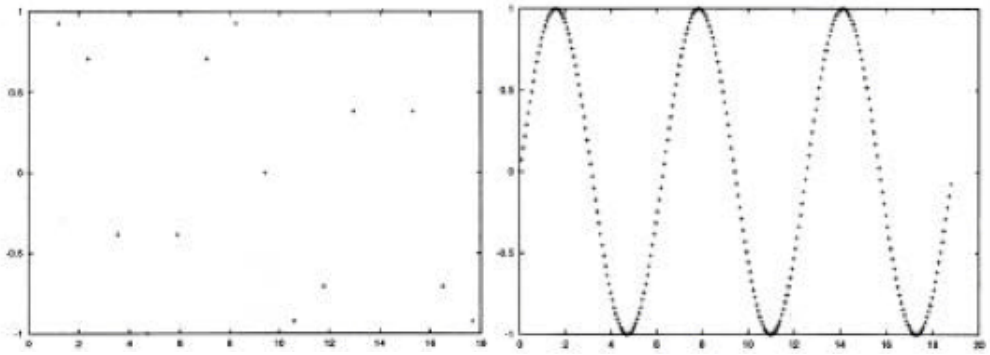
$$\mathrm{sinc}(x) = \begin{cases} \frac{\sin \pi x}{\mathrm{px}} & if\ x \neq 0 \\ 1 & if\ x = 0 \end{cases}$$

das hauptproblem des sinc filters ist, dass er ins unendliche geht. bei einfachem abschneiden kann es zu großen rekonstruktionsfehlern kommen (truncation error). das problem kann mit einem fenster des sinc filters umgangen werde, dh es wird nur der teil um den ursprung herum zur rekonstruktion herangezogen. ein weiterer vorteil des sinc filters ist, dass er an allen sampling positionen 0 ist, bis auf die ursprungsposition.

in der praxis werden oft andere filter zur rekonstruktion eingesetzt (dabei entsteht häufig ein non-sinc error, abweichungen vom orginalen signal):
nearest neighbour: hat eine stufenfunktion als ergebnis, da jede distanz nur eine bestimmte höhe einnehmen kann (siehe legosteine mit gewisser höhe). diese art der rekonstruktion wird mit einem boxfilter durchgeführt

lineare interpolation
symmetrischer boxfilter
windowed sinc

eine interpolation mittels “zero insertion” kann im frequenzraum angewendet werden. das prinzig beruht darauf, dass bei anhängen von nullen zur diskreten fourier transformation einer funktion, die daraufhin in den ortsraum umgerechnet wird, die neue funktion kleinere zeitschritte aufweist, aber dieselben frequenzen. das heißt, dass die neue funktion feiner gesamplet ist als die alte.



# Texturing

**Transformation Bildraum -> Objektraum -> Texturraum**

um eine textur (meist 2d) auf ein objekt zu mappen müssen die 2d koordinaten aus dem texturraum (u,v) in 3d koordinaten des objektraums ($x_0$, $y_0$, $z_0$) umgewandelt werden, und diese wiederrum in 2d bildkoordinaten (x,y). die umwandlung von texture space zu object space nennt man parametrisierung. die umwandlung von object space zu image space wird projection genannt.
bei der parametrisierung wird die textur auf das objekt projeziert. alle punkte in einem dreieck können anhand ihrer baryzentrischen koordinaten beschrieben werden:

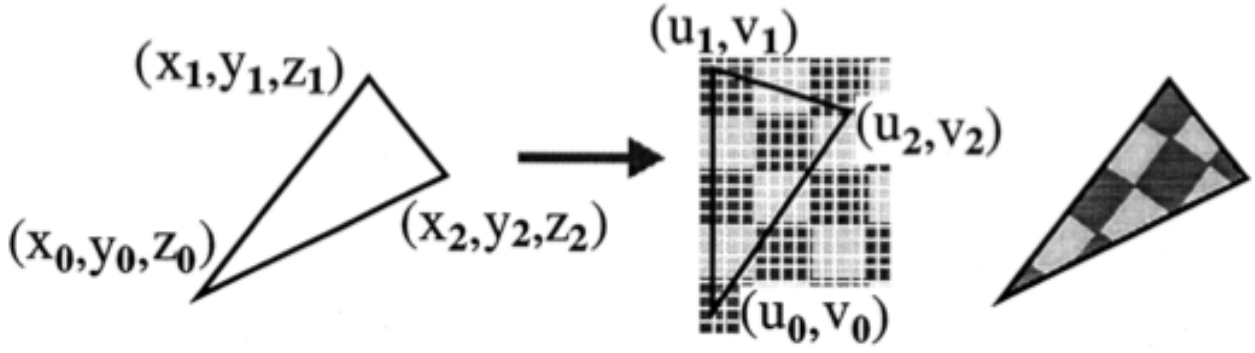
$(x, y, z) = a_0 * (x_0, y_0, z_0) + a_1 * (x_1, y_1, z_1) + a_2 * (x_2, y_2, z_2)$

wobei $(a_0, a_1, a_2)$ die baryzentrischen koordinaten sind

die zugehörigen textur koordinaten (u,v) können jetzt durch die folgende formel bestimmt werden:

$(u, v) = a_0 * (u_0, v_0) + a_1 * (u_1, v_1) + a_2 * (u_2, v_2)$

das mapping wird im normalfall in die andere richtung vorgenommen, dh von image- zu texturekoordinaten.



**Wie schaut ein Pixel im Texturraum aus?**

im texturraum heißen die kleinsten einheiten texel. meist wird eine textur als 2d gitter gespeichert (wie ein bitmap). es gibt aber auch mehrdimensionale texturen (zb solid texturing, bombing, environmental mapping,...).

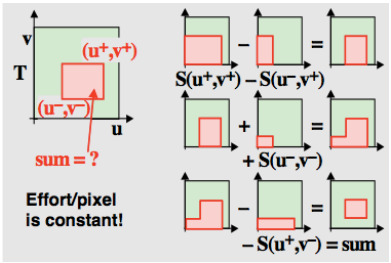
bump mapping: wird verwendet um eine oberfläche unregelmäßig wirken zu lassen. dafür werden die normalvektoren der oberfläche manipuliert. die geometrie des mit bump mapping texturierten objekts bleibt aber unverändert.

horizon mapping: versucht echte unebenheiten zu simulieren, in dem pro texel 8 horizontwerte berechnet und vor der anwendung interpoliert werden. dadurch kann direktes licht nur ab einem gewissen winkel auf das texel treffen (dunkel auf der vom licht abgewandten seite eines objekts).

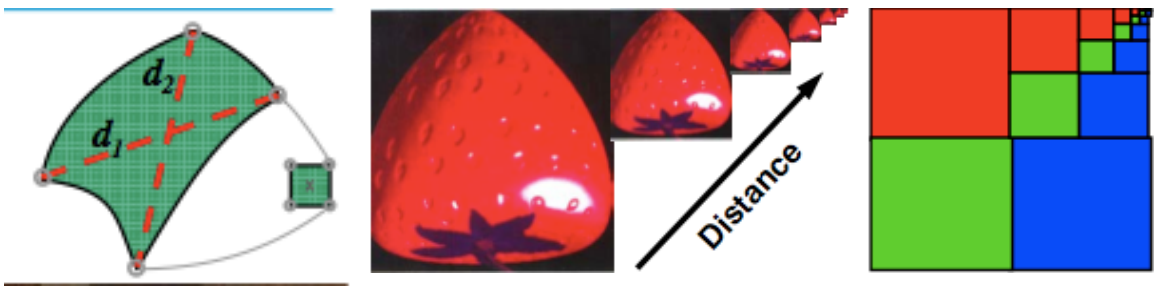
**Welche Annahme macht man bei der Summed Area Table Methode damit das Verfahren funktioniert? Aufzeichnen eines Tabellenzugriffs?**

die summed area table methode wird zu antialiasing zwecken verwendet (vor allem bei verzerrungen und weit entfernten objekten) und ist ein gewichteter mittelwert, der in einem vorprozess ermittelt wird. es wird angenommen, dass die texturintensität jedes pixels durch einen durchschnitt eines rechtecks der textur dargestellt werden kann.
das texel (x,y) im summed-area table ist die summe aller texel im rechteck (0:x, 0:y) des texture bildes.

es ist auch möglich die summe einer bestimmten region eines bildes zu berechnen.




**was ist mipmapping, wie funktionierts? wieviel mehr speicher braucht man bei mipmaps? Berechnungsbeispiel wie man auf das Mip map level kommt, was ist dieses d1, d2, Speicherung, Auswahl der richtigen Mipmap, Formel bzw Schema für Bilineare Filterung, warum ld(max(diagonale)), was passiert bei D<0 -> upsampling -> bilineare filterung, ist sie linear? warum logarithmus dualis?**




mip-mapping (multum in parvo) ist eine form von texture anti aliasing. die texturen werden immer um den faktor 2 verkleinert abgespeichert, bis die letzte verkleinerung die größe eines texels hat. der speicherbedarf für mipmaps wird um ein drittel zum ursprünglichen

erhöht. es werden oft die rgb kanäle getrennt voneinander gespeichert. um eine mip-mapping textur auf ein objekt zu mappen muss erst D (level der mipmap) berechnet werden. für die berechnung werden die maxima der projezierten diagonalen herangezogen: D = ld(max(ld$_1$l,ld$_2$l)). w0 und w1 sind die werte, die durch lineare interpolation auf D0 und D1 herausgekommen sind.

$$colorofpixel = (D_1 - D) * W_0 + (D - D_0) * W_1 = (D_1 - D) * (W_0 - W_1) + W1$$

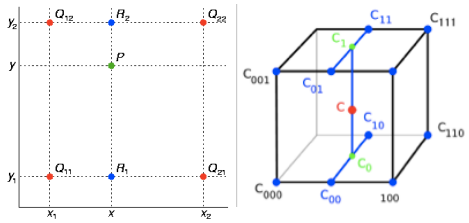
da D selten eine ganze zahl ist muss zwischen den beiden in frage kommenden maps interpoliert werden.

wenn die orginalgröße der textur zu klein ist (D<0), muss die textur aufgeblasen werden (upsampling). das kann zu schlechter qualität führen.

der unterschied zwischen bilinear und trilinear liegt in der detailstufe. es wird zuerst an jeder der 2 in frage kommenden mipmaps eine bilineare interpolation durchgeführt, und zwischen den beiden ergebnissen erneut interpoliert, wodurch es eine trilinear interpolation wird.

**wann kommt bilineares filtering zum einsatz? ist bilineares filtering linear?**

bei bilinearer filterung wird die textur pro achse je einmal linear interpoliert. dazu werden zunächst die beiden werte zwischen den oberen und den unteren texeln entlang der u-achse linear interpoliert. zwischen diesen beiden werten wird anschließend entlang der v-achse linear interpoliert. diese zweistufige lineare interpolation gibt der bilinearen interpolation ihren namen. die anteile sind dabei nur von der abtastposition abhängig, nicht jedoch von der größe der projektion des pixels in den texturraum. bei einer trilinearen interpolation handelt es sich um eine interpolation in einer weiteren ebene. das kann zb ein 3d würfel sein, oder im bezug auf mipmaps, eine interpolation zwischen 2 verschiedenen detailstufen von mipmaps.



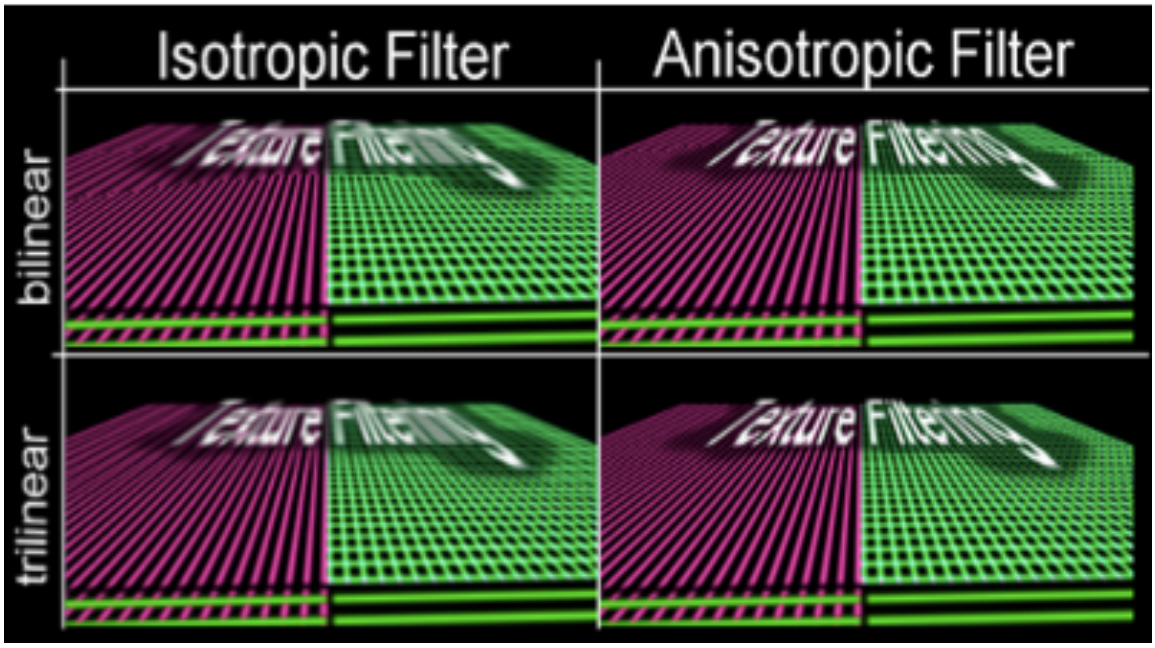

**aliasing beim texturing (allgemein ... speziell z.b. wozu logarithmus dualis bei mip mapping)**

wenn texturen auf objekte gelegt werden kommt es oft zu aliasing artefakten, da die objekte meist nicht plan sind und dadurch oft zb 1 pixel mehrere texel umfasst (oder umgekehrt). es kann natürlich nicht einfach irgendeines dieser texel genommen werden und die übrigen verworfen, da sonst sichtbare fehler entstehen. es muss ein mittelwert errechnet werden. dafür können verschiedene verfahren eingesetzt werden, wie zb direkte faltung, mipmapping oder die summed area table methode

logarithmus dualis: das konzept dahinter ist, dass zusammengehörende punkte in den verschiedenen hierarchieebenen einfach durch einen binary shift von einem input (u,v) addressiert werden kann. wegen 2er potenz, die beim mipmapping auch verwendet wird.

**anisotropic filtering**

anisotropische filterung wird bei texturierung von verzerrungen verwendet.

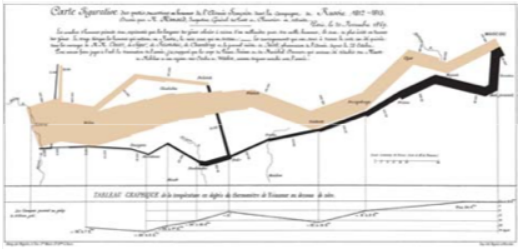



rip mapping: wie beim mipmapping wird eine textur öfters abgespeichert, aber hier werden nur entweder x werte oder y werte halbiert. dadurch ist die texturierung zwar schnell, es wird aber ein größerer speicher für die textur gebraucht.
footprint assembly: statt einem verzerrten viereck wird ein vereinfachtes parallelogramm angenommen, die textur wird mittels mipmaps angebracht.

# Visualization

**Visualisierung, allgemein warum macht man das? wofür wird das verwendet, was für methoden?**

die drei grundziele von visualisierung sind
- erforschen (nichts ist bekannt, daten werden durch visualisierung erforscht)
- analysieren (es gibt hypothesen, visualisierung dient zur verifikation oder falsifizierung)
- präsentieren (resultate werden durch visualisierung kommuniziert)



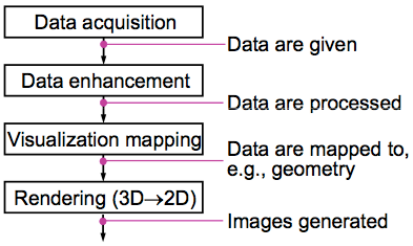
visualisierung wird dazu verwendet neue Informationen zu gewinnen, zusammenhänge herzustellen (zb cholera) und bekanntens wissen zu visualisieren. ‘it is a tool to enable a user insight into data’. visualisierung soll dem user helfen unter die oberfläche von

objekten zu schauen und die wichtigen, interessanteren eigenschaften wahrzunehmen. daten sollen veranschaulicht dargestellt werden.

bei visualisierungen werden oft abstraktionstechniken verwendet. diese dienen der vereinfachung und übersichtlichkeit der visualisierung. es soll form und struktur präsentiert werden, hervorstreichen und in den hintergrund stellen von objekten soll möglich sein, die künstlichkeit soll dargestellt werden, die wichtigen teile der objekte sollen sichtbar sein. ‘so detailliert wie nötig, so einfach wie möglich’.

es gibt bei visualisierungen eine einteilung in low-level techniken (stilisierte darstellung) und high-level techniken (smart visibility). es gibt mehrere smart visibility methoden, darunter importance driven feature enhancement, cut-away, volume splitting uvm. der mensch kann unvollständige objekte leicht vervollständigen, besonders symmetrien sind unproblematisch. diese eigenschaft kann von smart visibility methoden ausgenutzt werden, um wichtige teile vom objekt sichtbar zu machen.

**Visualisierung + Pipeline, Aus welchen Schritten besteht die Visualisierungspipeline?**




data acquisition: daten werden erhoben (aus messungen, simulationen, modellierung,...)
data enhancement: daten werden aufbereitet (mittles glättung, resampling, ergänzung, interpolation,....)
visualization mapping: daten werden darstellbar gemacht (auf flächen gelegt, 3d aufbereitet)
rendering: daten werden von 3d objekten zu 2d bildern gemacht (sichtbarkeitsberechnung, beleuchtung, komposition, animation,....)

**teilbereiche von visualisierung**

volume visualization: medizinische daten, mittels extraktion von isoflächen, abtasten des volumens, kombination,...
flow visualization: strömungsdaten, darstellbar durch direkte visualisierung (richtungspfeile), integrationsbasierte strömungsvisualisierung (strömungslinien), texturbasierte strömungsvisualisierung (textur verschmieren)
information visualization: abstrakte daten, sinnvolle abbildung hochdimensionaler daten, es sollte gleichzeitig überblick und detailinformation dargestellt werden, verknüpfung von ansichen
weitere visualisierungsbereiche:
- archäologie: historische daten
- molekularphysik: mikroskopische daten
- astronomie: makroskopische daten
- extreme große datenmengen

# Illustrative Visualisierung

**was ist importance driven feature enhancement (Erfinder Ivan Viola), was gibt es noch?**

beim importance driven feature enhancement werden zusätzlich zu daten wie farbe, durchsichtigkeit,... auch die wichtigkeit pro merkmal mitgespeichert. anhand dieser daten können wichtige merkmale hervorgehoben werden und unwichtige spärlicher (durchsichtig,...) dargestellt werden.

im gegensatz zum important driven feature enhancement, bei dem die wichtigkeit explizit angegeben werden muss, gibt es auch das illustrative context-preserving exploration verfahren, dass eine implizite betonung von relevanten features macht.

es gibt auch ghosting um gleichzeitig innen- und außenleben darzustellen

**context preserving: welche 4 parameter gibt es bei content preserving (was ist das, unterschied zu visualisierung allgemein, arten (low freq, high freq), beispiele für beides)**

der kontext in dem ein objekt liegt darf nicht verloren gehen. das kann sehr gut an medizinischen visualisierungen gezeigt werden. will man sich zb den adernverlauf in der hand anschauen, ist es wichtig nicht nur die adern zu sehen, sondern sie im kontext der hand betrachten zu können.
die context preserving methode zielt darauf ab den kontext zu belassen, und dennoch, ohne im vorfeld festgelegte auswahlen treffen zu müssen, das innere von objekten anzeigen zu können.

um die durchsichtigkeit eines objekts zu bestimmen werden 4 parameter eingesetzt:
- shading intensity
- gradient magnitude
- abstand zum augpunkt
- im vorfeld festgestellte durchsichtigkeit (opacity)

um dem/der benutzerIn die nutzung zu erleichtern werden 2 parameter eingesetzt. der/die benutzerIn muss nur konstante durchsichtigkeit der struktur bestimmen, um festzulegen, welche objekte durchsichtig gemacht werden sollen und welche ständig sichtbar sein sollen.

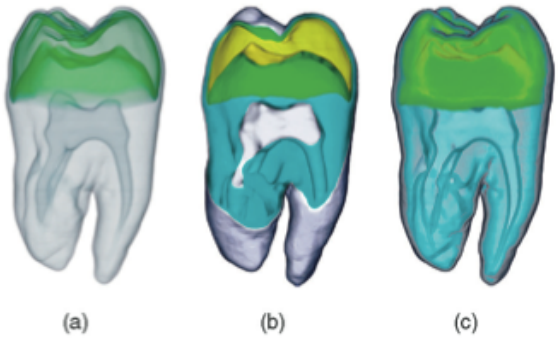



Fig. 7. CT scan of a tooth rendered with three different techniques. (a) Gradient-magnitude opacity-modulation. (b) Direct volume rendering with one clipping plane. (c) Context-preserving volume rendering.

# Computational photography

digitale photographie ist photographieren mit digitalen mitteln und sehr einfache bildverarbeitung.

computational photography ist digitale photographie mit elaborierterer bildmanipulation (mehr berechnungen). beispiele sind panoramas, 3d bilder, bilder miteinander verschmelzen, bilder rekonstruieren, morphing (mittels interpolation über die zeit), mehrere (fast gleiche) photos verbinden, “beautification” via warping (wie verändert sich ein pixel mit bestimmten abstand zu den strukturen) etc...
schon beim design der kamera muss die verwendung für computational fotography berücksichtigt werden.

taxonomie
- computational illumination
- computational optics
- computational processing
- computational sensors

image stacks: mehrere bilder der gleichen szene werden zu einem gemacht. zb gruppenphotos, bilder von plätzen,....

durch computational photography kann ein photo mehrere informationen dazubekommen: es wird ein photo gemacht und dessen gps koordinaten mitgespeichert. online gibt es viele daten zu der gps umgebung wodurch auch tiefendaten mit dem photo verbunden werden können. dadurch ergeben sich neue möglichkeiten, die mit dem photo gemacht werden

können, zb blau schleier aus der luft kann entfernt werden, andere beleuchtung sind möglich (vomittag, nachmittag),...

**photographie grundlagen**

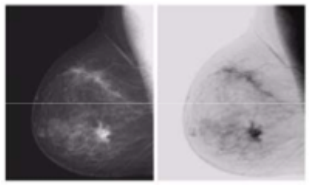
linsensystem, blende vor der linse, wird aufgemacht -> licht tritt auf lichtempfindliche ebene, verschlussmechanismus, der sicherstellt wie lange belichtet wird
focal lenght (brennweite): durch variation kann man steuern, welchen teil der szene man sieht
focusing distance: entfernung in der die szene scharf dargestellt wird
depth of field (tiefenschärfe): welcher teil soll scharf erscheinen, tiefenschärfe wird durch verschiedene faktoren beeinflusst
exposure (beleuchtung, lichteinfall): durch 3 parameter beeinflusst
aperture (in f number): welcher teil der linse ist dem lichteinfall ausgesetzt (kleine blendenöffnung, größere belichtungszeit)
shutter speed (in fractions of a second)
sensitivity (in iso): zb 100 iso für helle umgebung
mit diesen eigenschaften wird bei der computational photography gespielt.

**bild grundlagen**

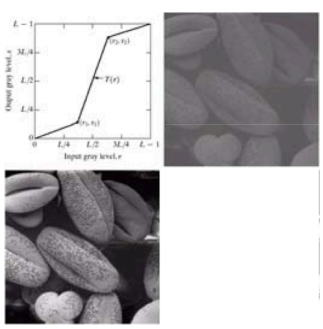
bilder können als funktionen von einem 3d raum in einen 2d raum dargestellt werden, wobei f(x,y) die intensität an position (x,y) zurückliefert. manchmal reichen repräsentationen von farbinformationen jedoch nicht aus (zb. bei hdr aufnahmen).

image processing: es kann image filtering (änderung der range (frequenz)) oder image warping (änderung der domain (bereich)) gemacht werden


point processing: einfachste situation: unabhängig von der position im raum den grauwert verändern. jedes pixel wird für sich betrachtet, pos im raum wird ignoriert.



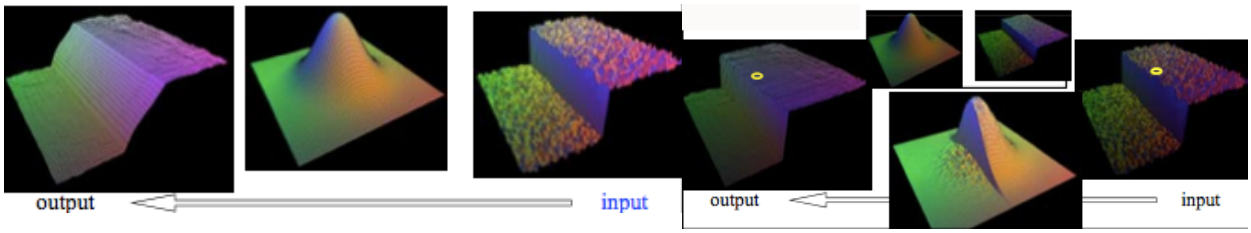
contrast stretching: histogrammoperationen: wenn ich einen eingangsgrauwert hab bekomm ich den jeweils anderen heraus



um die range eines images zu ändern kann gauss oder bilaterale filterung verwendet werden.

**bilaterale filterung**

auf ein verrauschtes bild wird in jedem pixel eine gauss glocke gelegt. dadurch wird das rauschen reduziert aber auch die kanten abgerundet. um kanten zu behalten muss der intensitätsunterschied auch miteinberechnet werden (zb anhand einer 2ten gauss kurve). alles in der nähe des zu berechnenden pixels (räumlich und intensität) hat einen großen einfluss. die stark unterschiedlichen pixel fließen nicht in die berechnung ein. filterung wird nur im homogenen intensitätsbereich durchgeführt.

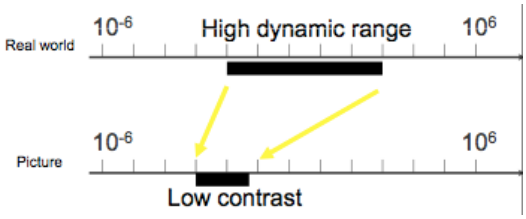



**high dynamic range**

dynamic range ist der kontrast in der szene, also der unterschied zwischen hellstem und dunkelsten bereich.

es kann bei photographien zu problemen kommen, wie zb dass der hintergrund zu hell und der vordergrund zu dunkel ist. in dem fall kann tone mapping eingesetzt werden, der eine farbpalette auf eine andere mappt (das reale bild wird in dem fall auf einen kleineren wert abgebildet). eine zweite lösung ist das bild mit variierender belichtungszeit zu machen. aus mehreren bildern kann man ein bild zusammenfügen, wo der kontrast entsprechend passt. bei der ausgabe von hdr bilder kommt wieder tone mapping zum einsatz, oder sie werden auf einem geeigneter monitor angezeigt.

contrast reduction: kontrast wird, wie beim tone mapping auf einen kleinere bereich im bild abgebildet.



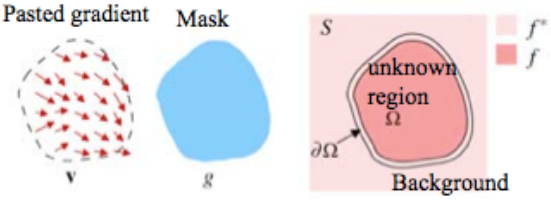

hdr display: zb. 2 lcd bildschirme hintereinander, vorderer ist durchsichtig, stellt bild dar. hinterer schirm für beleuchtung (gefiltertes bild)

gradient manipulation:
"in welche richtung ändert sich der grauwert am schnellsten?" = gradient
informationen von einem pixel werden an die benachbarten weitergegeben, so können verschiedene bilder zu einem zusammengelegt werden. man rechnet mit dem gradienten. gradient ist am übergang besonders stark

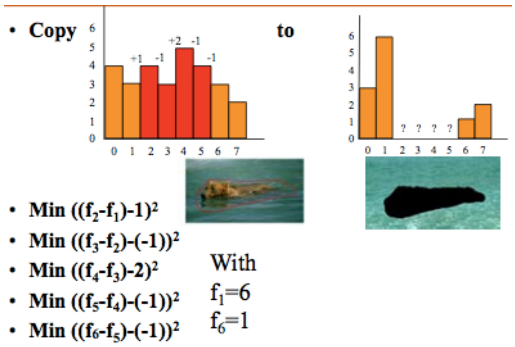
**seamless poisson cloning**

es werden nur die gradienten zur berechnung verwendet. der randbereich dient als anfangsbedingung, dh der vektor am rand ist der gradient, der angibt wie sich die farbe ändert. die farbinformation selber kommt vom hintergrund und wird hineindiffundiert.

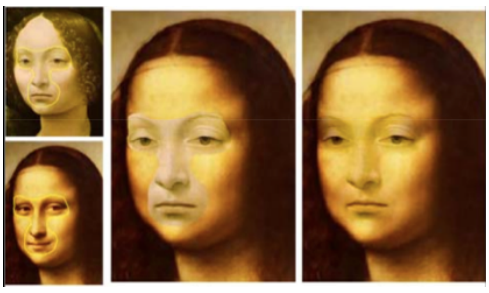
$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$




in der formel entspricht das delta f = gradient, with ... entspricht der nebenbedingung und das minimum wird verwendet, weil man ein optimierungsproblem formulieren muss.
discrete 1D example:
die zahlen in den formeln entsprechen dem v (gewünschten gradienten) aus der formel
man nennt dann die funktion, die man minimieren möchte Q




-> ableiten und null setzen
matrix entspricht dem faltungskern (im wesentlichen einem krümmungsfilter), weil sie eine 2te ableitung ist.



poisson matting:
beim poisson matting wird vordergrund und hintergrundgetrennt und ein neuer hintergrund eingefügt.

$$I = \alpha F + (1-\alpha)B$$
$$\nabla I = (F-B)\nabla\alpha + \alpha\nabla F + (1-\alpha)\nabla B$$
$$\nabla\alpha \approx \frac{1}{F-B}\nabla I$$

der alpha wert liegt zwischen 0 (hintergrund) und 1 (vordergrund). die formel wird abgeleitet und es wird davon ausgegangen, dass die terme f und b ignoriert werden können.
es werden die bereiche geteilt in sicher vordergrund, sicher hintergrund und dazwischen ein nicht definiertes “band”. opacity ändert sich durch die formel zwischen 1 und 0, schätzungen von f und b in der formel werden gebrauch (durch nächstgelegenen hinter- und vordergrund)
poisson-ish mesh editing: das pferd wird durch das kamel ersetzt