Computergrafik 1 Zusammenfassung

Bild->Bild: Bildbe-/-verarbeitung
Bild->Beschreibung: Bildanalyse (Mustererkennung)
Beschreibung->Bild: Generative Computergrafik

Einsatz von Computergrafik:
- Unterhaltung
- Fertigung
- Ausbildung
- Medizin

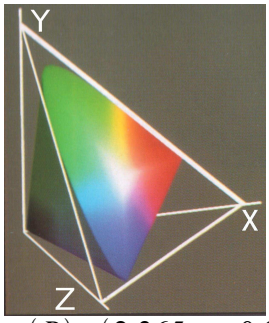
Komponenten:
- Input-Daten
- Geometrie
- Material & Beleuchtung
- Kamera
- Bewegungsparameter
- Generative Computergrafik
- Animation/Simulation

Mathematische Grundlagen:
- Vektorrechnung (siehe Lineare Algebra)
- Differentialgleichungen (siehe Mathe2)

Verschiebung: Addition mit Vektor
Punkte: P-Q=Vektor, P+v=Punkt

Farbe:
Additives Farbmodell (RGB, Red-Green-Blue):
- verwendet bei Monitoren
- 3 Farben (Blau 436nm, Grün 546nm, Rot 700nm)
- Addition der Farbfrequenzen ergibt neue Farben (manche Farben benötigen negative Gewichte)

CIE-XYZ-Modell: Y=Helligkeit, X,Z=Koordinaten der Farbvalenzen



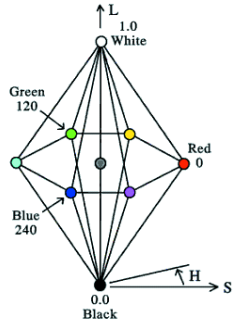
$$\begin{pmatrix} R \end{pmatrix} \begin{pmatrix} 2.365 & -0.875 & -0.468 \end{pmatrix} \begin{pmatrix} X \end{pmatrix}$$
$$\begin{pmatrix} G \end{pmatrix} = \begin{pmatrix} -0.503 & 1.392 & 0.086 \end{pmatrix} * \begin{pmatrix} Y \end{pmatrix}$$
$$\begin{pmatrix} B \end{pmatrix} \begin{pmatrix} 0.005 & -0.014 & 1.009 \end{pmatrix} \begin{pmatrix} Z \end{pmatrix}$$

Subtraktives Farbmodell (CMY, Cyan-Magenta-Yellow):
- verwendet bei Druckern
- Frequenzen werden aus weißem Licht absorbiert
- C=G+B, M=R+B, Y=R+G

$$\begin{pmatrix} C \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} R \end{pmatrix}$$
$$\begin{pmatrix} M \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix} - \begin{pmatrix} G \end{pmatrix}$$
$$\begin{pmatrix} Y \end{pmatrix} \begin{pmatrix} 1 \end{pmatrix} \begin{pmatrix} B \end{pmatrix}$$

Computergrafik 1 Zusammenfassung

HLS-Farbmodell (Hue-Luminance-Saturation, Farbe-Helligkeit-Sättigung):
$H \in [0°, 360°], S, L \in [0,1]$
H=0°: rot, H=120°: grün, H=240°: blau
S=0: Graustufe, S=1: reine Farbe
L=0: schwarz, L=1: weiß



Gamut: Bereich der mit einem Medium erzeugbaren Farbvalenzen

<u>Grundformate:</u>
Vektorformat: Beschreibung durch math. Primitive (Punkt, Linie, Kreisbogen...)
+: Bildschärfe auch bei Skalierung, wenig Speicher bei einfachen Bildern
-: Speicherung von Details aufwendig oder unmöglich
geeignet für technische Zeichnungen oder Diagramme
Formate: DXF, IGES, PS, CGM, WMF, ...

Rasterformat (meistens verwendet): Pixelmatrix
+: punktgenaue Darstellung, Transparenz möglich
-: bedingt skalierbar, speicherintensiv bei einfachen Bildern
geeignet für detailreiche Bilder (z.B. Fotos)
Formate: JPEG, TIFF, GIF, PNG, PNM, XPM, ...

Komprimierung:
verlustfrei: ursprüngliche Daten wiederherstellbar (z.B. PNG, ...)
verlustbehaftet: möglichst geringe visuelle Artefakte (z.B. GIF, JPEG, ...)
Farben: HighColor: 16 Bit (5R, 6G, 5B) => 65.536 Farben
TrueColor: 24 Bit (8R, 8G, 8B) => 16.777.216 Farben oder 32 Bit mit Transparenz
Quantisierung: (Verlustbehaftete) Reduktion der Farbanzahl oder -tiefe
Runlength-Encoding: Anzahl Wiederholungen mit Escape-Symbol speichern
Difference-Coding (Videos): Differenzwerte benachbarter Symbole speichern (bei Frames)
Fehler addieren sich => nach einigen Frames Vollbild speichern
Huffman-Codierung: häufiges Symbol -> kleiner Binärwert (siehe Info1)
Frequenzraumverfahren: Entfernung unwichtiger Teile (z.B. hohe Frequenzen)

<u>Raytracing (genaueres siehe unten):</u>
Szenenbeschreibung:
Objekte/Modelle: Geometrie (z.B. Polygone), Materialeigenschaften
Lichtquellen
Kamera
Szenenbeschreibung -> Renderer -> Bild
Strecke, Streckenzug: Direkte Verbindung mehrerer Punkte
Polygon: Fläche innerhalb eines geschlossenen, ebenen Streckenzuges

Polygonnetz, Polyeder: Zusammenhängende Polygone mit gemeinsamen Eckpunkten
Lichtinteraktion mit Materialien:
- Absorption
- Reflexion (spiegelnd oder diffus)
- Transmission (Transparenz)
- Mehrere Interaktionen gleichzeitig sind möglich

Lichtbrechung:

Brechungsindex $n = \frac{\text{Lichtgeschw. im Vakuum } c}{\text{Lichtgeschw. in Materie } v}$

Übergang zw. Materialien: $\sin(\theta_1)\, n_1 = \sin(\theta_2)\, n_2$ (θ = Winkel zur y-Achse)

Vereinfachtes Lichtmodell:
- Licht geht nur von einem Punkt aus

$$L = \begin{pmatrix} r_l \\ g_l \\ b_l \end{pmatrix}$$

Lichtquellen:
- Punktlicht: gleiche Intensität in alle Richtungen
- Richtungslicht: paralleles Licht aus Richtung l (Quelle unendlich weit entfernt)
- Spotlicht: Punktlicht mit Richtungskonus und cutoff-Winkel (wird evtl. nach außen schwächer)
- Lichtintensität kann mit steigender Entfernung abnehmen

Cutoff-Abschwächung: $L(d) = \begin{cases} <d_0, d>^n L_0 & \text{, wenn } \arccos(<d_0, d>) < cutoff \\ 0 & \text{, sonst} \end{cases}$

Einfache Linse:
- Brennpunkt F, F'
- Brennweite f: Abstand Brennpunkt zur Linsenmitte
- Bildebene: Aufnahmeebene (P in Objektebene -> P' in Bildebene)
- Unschärfe, falls P' vor/hinter Bildebene liegt

Physikalische Lochkamera: Bild hinter Lochblende, gespiegelt
Virtuelle Lochkamera: Bild vor Lochblende, keine Spiegelung

<u>Raycasting:</u>
- Kamera/Augpunkt hinter Bildebene
- Strahlen vom Augpunkt durch jedes Bildpixel in den Raum
- Schnittpunktberechnung mit allen Objekten
- kein Schnittpunkt: Hintergrundfarbe
- Schnittpunkte: nächsten ermitteln, Farbe mittels Beleuchtungsmodell ermitteln
- Die Schnittpunktberechnung benötigt ca. 95% der Rechenzeit

Schnittpunktberechnung:
- R(a) = V + a*r
- gesucht: kleinstes a mit: R(a) liegt auf Objektoberfläche
- $F : \mathbb{R}^3 \to \mathbb{R} : F(P) = 0$ (Objektbeschreibung)
- Schnittpunkte: $R(a) : \{ a > 0 : F(R(a)) = 0 \}$

- Vorgehen: F(R(a)) nach a umstellen, a in R(a) einsetzen

Schnittpunkt mit Polygon:
- S = Schnittpunkt mit Polygonebene
- beliebigen Strahl von S in Polygonebene wählen
- Anzahl Schnitte mit Polygonkanten gerade/ungerade <=> S liegt außen/innen

Umsetzung von Point-In-Polygon:
gegeben: ebenes Polygon, Schnittpunkt von R(a) mit Polygonebene
Eck-/Schnittpunkte auf R2 abbilden (z-Komponente entfällt, =>P', S')
S' := Ursprung => P''=P'-S', S''=(0,0)
Trivial Reject (TR): kein Schnitt bei: $x_i'' < 0 \wedge x_{i+1} < 0$, $y_i'' < 0 \wedge y_{i+1}'' < 0$, $y_i'' > 0 \wedge y_{i+1}'' > 0$
Gesucht: $a_i \in [0,1] : (1-a_i) P_i'' + a_i P_{i+1}'' = (*,0) \Rightarrow$ y-Komp: $a_i = \frac{-y_i''}{y_{i+1}'' - y_i''}$ =>Prüfe auf pos. x-Wert
Sonderfall: Punkt auf x-Achse: Schnittpunkt wenn $y_{i-1}'' * y_{i+1}'' < 0$

Phong Beleuchtungsmodell:
ambienter Anteil (Hintergrundlicht, Objektform) $I_a = k_a * L_a$ (k=Materialkonstante)
diffuser Anteil (diffuses Reflexionsverhalten) $I_d = max\{<n,l>,0\} * k_d * L_d$
spiegelnder Anteil (spiegelndes Reflexionsverhalten) $I_s = max\{<l,r_v>,0\}^n * k_s * L_s$
weitere Größen:
n = Normalenvektor d. Oberfl, l = Lichtvektor, v = Viewvektor, r_v = refl. Viewv.,
$\Theta, \alpha = \sphericalangle(n,l), \sphericalangle(l,r_v)$

Berechnung von Normalenvektoren:
gegeben: Polygon (P_1,...,P_k)
$\vec{n} = (P_2 - P_1) \times (P_3 - P_1)$, $n = \vec{n} / |\vec{n}|$
bei Flächen F(u,v):
$\vec{n} = \frac{\partial F}{\partial u}(u_0, v_0) \times \frac{\partial F}{\partial v}(u_0, v_0)$, $n = \vec{n} / |\vec{n}|$

<u>Raytracing:</u>
Generierung realitätsnaher Bilder mit Reflexionen und Schatten
Verwendung von Strahlenbäumen (Auge->Objekt->...->BG, je mit Licht, Hintergrund, Material)
$I = \rho(P_1, L)(I_d^1 + I_s^1) + k_s^1 * I(r_{v,1}) + k_t^1 * I(t_1)$
$I(r_{v,1}) = \rho(P_2, L)(I_d^2 + I_s^2) + k_s^2 * I(r_{v,2}) + k_t^2 * I(t_2)$
$I_{d,s,t}^i$: diff., spieg., transm. Anteil bei P_i, $k_{d,s,t}^i$: Reflexionskoeff.
$\rho(P,L) = \begin{cases} 0 & \text{L von P verdeckt} \\ 1 & \text{L von P sichtbar} \end{cases}$

Anti-Aliasing:
Problem: Pixelige Objektkanten
Lösung: Strahlen an Pixelecken statt in der Mitte
=>4 Strahlen in Objekt=Farbe kräftig, weniger Strahlen=Farbe verblasst (Interpolation)

Beschleunigungen:
Bounding-Objekt: komplexe Obj. in einfache Formen packen
Schnitt mit einfacher Form=weiter testen, sonst nicht
Achsenparallele Boundingbox: Schnitt mit R(a):
$a_{max}^y < 0$, $max\{a_{min}^i\} > min\{a_{max}^i\} \Rightarrow$ kein Schnitt, Schnitt bei $max\{a_{min}^i\} < min\{a_{max}^i\}$
Octree: Szene in Quader unterteilen, Objektzahl pro Quader (ganz oder teilweise) gleich
pro Quader speichern: Nachbarquader, enthaltene Objekte, P_{min}, P_{max}
vom Strahl durchlaufene Quader ermitteln, Schnitttest mit enthaltenen Objekten
Beam-Tracing: Strahlenbündel mit gleichem Stahlenbaum, statt einzelne Strahlen

<u>Rasterbasierte Grafik:</u>
keine Strahlen vom Beobachter
Jedes Objekt wird direkt auf die Bildebene projiziert

Keine indirekten Effekte wie Schatten möglich
Blickwinkel bestimmen->Rundungen linearisieren->Blickwinkel zu Einheitsquader->
->verdeckte Objektteile entfernen
Rasterisierung: Approximation exakter Polygone durch Pixel
Schattierung: Fortsetzung lokaler Farbwerte auf Pixel (Fragmente)
Fragment-Operationen:
Visibilität: Überprüfen der Fragmente auf Verdeckung
Alpha-Blending(Transparenz), Stencil- und Alpha-Test(Verwerfen von Pix/Fragm für spez. Eff.)

OpenGL: Open Graphics Library, 3D rasterbasierter Grafikstandard
Definition von Primitiven/Materialeigenschaften/Lichtquellen
Hierarchische Anordnung von Objekten zu einer Szene (inkl. Kamera)
Generierung eines Pixelbildes anhand der Szeneinformationen
meist in C/C++ geschrieben
keine eigene GUI, Anbindung: GLUT, Qt
Primitive (beginnen immer mit GL_):
POINTS, LINES, LINE_STRIP, LINE_LOOP (Punkte, Linien, Linienzug, geschl. Linienzug)
TRIANGLES, TRIANGLE_STRIP, TRIANGLE_FAN (Dreieck (-streifen/-fächer))
QUADS, QUAD_STRIP (Viereck (-streifen))
POLYGON (n-eckig)
Definition:
glBegin(*Primitiv*);
glVertex3f(x,y,z); (für jeden Eckpunkt wiederholt, von vorne gegen den Uhrzeigersinn)
glEnd();
Weitere Funktionen:
glEnable(GL_LIGHTING) – aktiviert Beleuchtungsberechnung
glEnable(GL_LIGHT0) – aktiviert Lichtquelle 0/8 (min)
glLightf() - setzen verschiedener Lichtparameter
glNormal3f() - Normalenvektor für Vertex setzen (für Beleuchtungsberechnung)
glMaterialf() - Materialkonstanten setzen
glutInitDisplayMode – Displaymodus initialisieren (siehe Framebuffer)
drawScene() - Szene aus Backbuffer zeichnen
swapBuffers() - Front-Backbuffer vertauschen
glTranslatef(x,y,z) – Verschieben (siehe Translation)
glScalef(sx,sy,sz) – Skalierung
glRotatef(angle,x,y,z) – Rotation
Transformationen werden nach Modellkoordinaten ausgeführt
gluLookAt(V,L,u) – Beobachter festlegen(Nullpunkt, LookAt, up-V. spannt m. L y-z-Ebene auf)
glOrtho – orthographische Transformation
glFrustum, gluPerspective – perspektivische Transformation

Hardwarebeschleunigung:
Teile des Bildgen.-prozesses je nach Rechnerausstattung in Hardware durchgeführt
Grafik-Programmierung muss Hardware-unabhängig sein
Grafik-Bibliothek: Init erkennt vorh. Hardware-Features, Umsetzung autom. (Hard-/Software)

Framebuffer:
verwaltet Informationen pro Pix: Farbe, Tiefe, weitere Buffer (Spiegel, Schatten, Unschärfe...)
Bitplanes: Color(Front), Color(Back), Depth (z-Buffer, Verdeckungs-Berechnung)
Displaymodus: bestimmt Framebuffer-Resourcen für Anwendung

Doublebuffer: getrennte Buffer für Generierung und Anzeige (Anzeigerate const./Erstellrate var.)

Transformationen und Modellhierarchien:

Aufgaben für Szeneerstellung:
- Bewegung v. Objekten (Verschieben, Drehen, Skalieren)
- Bewegung zusammengesetzter Objekte (z.B. Charaktere)
- flexible Arten der Modellerstellung (CG2)
- anwendungsspezifische Modellierungstechniken (CG2)

Affine Transformation:

Grundlage für Positionierung von Objekten und Wechsel des Koordinatensystems

A. Kombination: Punkte $P_{1,\dots}, P_k \in A$, Skalare $s_{1,\dots}, s_k$, $\sum_{i=1}^{k} s_i = 1 \Rightarrow \sum_{i=1}^{k} s_i P_i \in A$ ebenfalls ein Punkt

Konvexe Hülle: kleinste, umf. konv. Menge: $Q = \sum_{i=1}^{k} s_i P_i,\ \sum_{i=1}^{k} s_i = 1,\ s_i \geq 0$

Affine Transformationen bestehen aus einer linearen Abbildung und einer Translation
=> T(Vektor P) = nxn-Matrix*Vektor P+Vektor t

Homogene Koordinaten: Hinzunahme einer vierten Koordinate

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \rightarrow \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix},\quad v = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \rightarrow \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix},\quad M \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \rightarrow \begin{bmatrix} & & & t_x \\ & M & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

Translation (Verschiebung): P -> P+T

Skalierung: P -> S*P

Rotation: $R_{Achse,Winkel}$*P (R=Matrix, je nach Achse anders, Rot-Achse auf Diagonale = 1)

Affine Transformationen sind nicht kommutativ

Unterscheidung zw. Weltkoordinaten (WC) und Modellkoordinaten (MC)

MC->WC: $P^{WC} = (T_{(2,0,0)}(R_{z,-30^\circ}(S_{(1,2,1)} P^{MC})))$

Viewing Transformation:

Einbetten des Beobachters durch eigenes Beobachterkoordinatensystem

Objekte in WC

Rechtshändiges, orthonorm. Beobachterkoordinatensystem(V,x,y,z) (-z=Richtung)

WC->VC: $T_V = \begin{bmatrix} & A^T & & -A^T V \\ 0 & 0 & 0 & 1 \end{bmatrix},\quad A = (v_x, v_y, v_z)$

Projektionstransformation:

legt Sichtbereich und Transformationsart(orthographisch oder perspektivisch) fest

Sichtbereich nach NDC: $[-1,1]^3$ linkshändig (z=Abstand zum Beobachter)

Orthographische Transformation: orthogonale Projektion auf Ebene

Perspektivische Transformation: Pyramidenstumpf, Spitze=Beobachter, Dach=Bild

für Raytracing ungeeignet, da nicht winkel-erhaltend

Geraden auf Geraden abgebildet, Punkteordnung erhalten

Geraden durch Beobachterpunkt werden z-achsenparallel => Tiefenabstände verzerrt
kleine Abstände größer, große Abstände kleiner

Hierarchische Modelle:

ermöglichen Mehrfachverwendung eines Objekts

Datenstruktur als gerichteter azyklischer Graph

innere Knoten = Transformationen
Wurzel = Viewingtransformation V
Blätter = Geometriebeschreibung der Primitive
Kanten = gerichtete Verbindungen zwischen Knoten
Pfad Wurzel->Blatt = gezeichnetes Primitiv, mehrere Vorgänger erlaubt, keine Zyklen
Matrixstack = speichert (wiederverwendbare) Transformationsmatrizen
oberste Matrix = Reihenfolge der letzten Transformationen ab V
Probleme bei mehreren Vorgängern
Display-Listen = legen statische Befehle unter einem Namen ab, löst Vorgänger-Probleme

<u>Algorithmen der Rastergrafik:</u>
Teile der Grafik-Pipeline:
Geometrie-Subsystem: Clipping, Window-to-Viewport Mapping
Raster-Subsystem: Rasterisierung, Schattierung & Texturierung
Fragment-Operationen: Blending, Stencil- und α-Test

Window-to-Viewport Mapping:
Transformation von Window (NDC) in Viewport (Rasterkoordinaten, RC)
Längenverhältnisse bleiben erhalten

$$p'_x = x^V_{min} + \frac{x^V_{max} - x^V_{min}}{x^W_{max} - x^W_{min}} (p_x - x^W_{min}) \quad \text{(Analog zu y)}$$

Clipping:
Abschneiden von Geometrie außerhalb des Sichtbereichs
Angabe in Window-Edge-Coordinates (WEC):

$$WEC(P) = (wec_L(P), wec_R(P), wec_B(P), wec_T(P)) = (1 + p_x, 1 - p_x, 1 + p_y, 1 - p_y)$$

$sgn(wec_i(P))$ = Innen/Außenlage zur Kante, $|wec_i(P)|$ = Abstand zur Kante

$$outcode(P) = (b_L, b_R, b_B, b_T), \text{ wobei } b_i = \begin{cases} 1 & \text{falls } wec_i(P) < 0 \\ 0 & \text{falls } wec_i(P) \geq 0 \end{cases}$$

P liegt innerhalb des Fensters, wenn alle outcode-Elemente = 0
Trivial Accept/Reject (TA/TR):
TA: Strecke komplett im Fenster, wen $outcode(P_1) \vee outcode(P_2) = 0$
TR: $outcode(P_1) \wedge outcode(P_2) \neq 0$ (nicht alle Fälle erfassbar, wenn Strecke Diag. Schneidet)

α-Clipping:
Schnittpunktermittlung mit der erweiterten Fensterkante

$$\alpha_L = \frac{wec_L(P_1)}{wec_L(P_1) - wec_L(P_2)}$$

$$P(\alpha) = P_1 + \alpha(P_2 - P_1)$$

Polygon-Clipping:
Clipping aller Kanten gegen erweiterte Fensterkanten (wenn Polygon konvex)
Clipping gegen konvexe Polygone:
Abstand der Punkte zur Kante mit äußerer Normalen bestimmen

$wec(P) = -(\vec{n} * (P - Q_i))$ > 0 (innnen), < 0 (außen), = 0 (auf der Geraden)

Clipping in 3D:
WEC(P) = (w+x, w-x, w+y, w-y, w+z, w-z)

Rasterisierung (???, cgAlgo):
Approximation einer Strecke durch Pixel

Schattierungsmodelle:
Flat-Shading: eine Farbintensität pro Pixel => deutliche Kanten, keine glatten Flächen
Gouraud-/Smooth-Shading: eine Farbintensität pro Polygonecke, lineare Interpolation (Ortsabh.)
Lineare Interpolation:
$P_{ab}=(1-\alpha_{ab})P_a+\alpha_{ab}P_b \Rightarrow I_{ab}=(1-\alpha_{ab})I_a+\alpha_{ab}I_b$ (Fehler bei spiegelnder Reflexion)

Texturen:
Darstellung von zu komplexer Geometrie durch 2D-Bilder (Texturen) auf 3D-Objekten
Texturen sind diskrete Bilder aus n x m Texturpunkten B_{ij} (Texels)
z.T. Verzerrung durch generierte Textur-Koordinaten
Lineares Mapping:
Texturkoordinate (s,t) = Abstand des Vertex P zu einer Ebene
$E_s: a_s*x+b_s*y+c_s*z+d_s=0$ (entsprechend E_t)
$s=a_s*p_x+b_s*p_y+c_s*p_z+d_s$ (entsprechend t)
Bezugsystem: MC (relativ zum Objekt) oder VC (relativ zum Beobachter)
Sphärisches Mapping:
$\vec{r}=\vec{r_v}, v=(0,0,1)^T$
$\begin{pmatrix}(s)\\(t)\end{pmatrix}(r)=\frac{1}{2|r+z|}\begin{pmatrix}(r_x)\\(r_y)\end{pmatrix}+\begin{pmatrix}(1/2)\\(1/2)\end{pmatrix}$, $z=(0,0,1)^T$, $r=(r_x,r_y,r_z)^T$
ungleichmäßige Auflösung, Singularität in -z-Richtung
sehr einfach realisierbar
Interpolation von Texturkoordinaten:
Perspektivische Texturierung:
$Q_i^T=(1-\beta_i)P_1^T+\beta_i P_2^T$, $\beta_i=i*\Delta_o\in[0,1]$, Punkte in Bildebene, Δ_o=Schrittweite (const.)
$\alpha(\beta)=\frac{\beta z_1}{(1-\beta)z_2+\beta z_1}$
$S_i=(s_i,t_i)=(1-\alpha(\beta))S_1+\alpha(\beta)S_2$ (Texturkoordinaten)
Filterung:
(s,t) stimmen nicht immer mit Texelmittelpunkt überein
Wähle Intensität des nächstliegenden Texel (einfache Rechnung, evtl. Pixel zu sehen) oder
Affin-Kombination aus umliegenden 4 Texel-Intens. (Rechenintensiv, Unschärfe statt Pixel)
$I=(1-\alpha)(1-\beta)B_{i,j}+\alpha(1-\beta)B_{i+1,j}+(1-\alpha)\beta B_{i,j+1}+\alpha\beta B_{i+1,j+1}$
Probleme bei Filterung:
Ungültige (s,t)-Koordinaten
Lösungen: Textur wiederholen oder abschneiden
Unterschiedliche Auflösungen
Minification: Mehrere Texel auf 1 Bildpixel (Aliasing-Effekte, Moire-Effekt)
Verwendung von vorgefilterten Texturverkleinerungen (Mip-Mapping), Pixelgr. ca. Texelgr.
Verbesserung durch Lineare Interpolation zw. benachbarten Mip-Map-Leveln
Magnification: 1 Texel auf mehrere Bildpixel (Textur-Pixel stark sichtbar, besser mit lin. Interp.)
Textur-Funktionen: zur Kombination von Textur- und Beleuchtungsintensität
GL_REPLACE: $I(x,y) = I_T(x,y)$
GL_MODULATE: $I(x,y) = I_B(x,y)*I_T(x,y)$
GL_DECAL: $I(x,y)=(1-\alpha_T(x,y))I_B(x,y)+\alpha_T(x,y)I_T(x,y)$ (Textur hat α-Kanal)
GL_BLEND: $I(x,y) = (1-I_C)*I_B(x,y)+I_C*I_T(x,y)$ (mit zusätzlicher Blend-Farbe)

Spezielle Aspekte der Computergrafik:
OpenGL-Vertex-Arrays: zur effizienten Verwaltung großer Polygonnetze

Mehrere Arrays mit Blöcken: Vertex(3), Color(3), Normal(3), TexCoord(2), Index
Index-Array gibt Blockreihenfolge zum Zeichnen an
GL_VERTEX_ARRAY / _COLOR_ / _NORMAL_ / _TEXTURE_COORD_
Pointer für Arrays: glVertexPointer(), glNormalPointer(), ...

Blending und Transparenz:
opaque: getrübt (Angabe für Lichtundurchlässigkeit)
Transparenzinformationen im α-Kanal
Destination = (1- S_α)*Destination+S_α*Source (Destination=Hintergrund, auch mit α-Kanal)

Billboarding:
Darstellung von Objekten mit unscharfen Kanten (z.B. Bäume)
Textur auf Polygone auftragen -> Polygone zum Beobachter ausrichten (auffälliges Mitdrehen)

Erweiterte Framebuffer-Funktionalität:
weitere Buffer für Schatten, Spiegeleffekte, etc
Stencil-Buffer: bestimmte Eigenschaften in bestimmten Bildbereichen
Zeichnen nur in bestimmten Bildbereichen (Masken)
Spiegelung über Stencil-Maske
Accumulation-Buffer: Anti-Aliasing, Bewegungsunschärfe, Tiefenunschärfe
Antialiasing: Jittern: zufällige kleine Verschiebung in Bildebene, umrechnen auf Ursprungswert
Unschärfe: mehrfaches Zeichnen und Überlagern der Teilergebnisse

<u>Objektbeschreibungen:</u>
Kugel: $F(P)=<(P-M),(P-M)>-s^2=0$