Aliasing und Antialiasing

Wesen des Phänomens **Aliasing** (*)

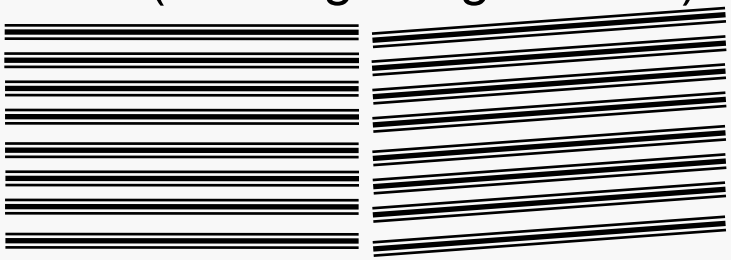
Wirkung der Technik Anti-**Aliasing**

(*) „éiliæsing“ (< alias < άλλως = anders): Veränderung, Entstellung

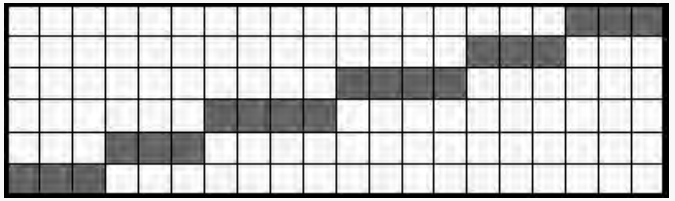
Aliasing und Antialiasing

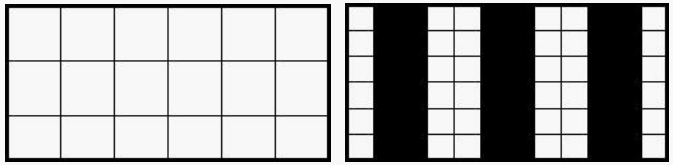
Aliasing-Effekte:

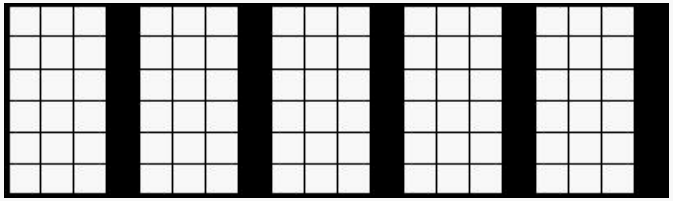
- Brüchige, gestufte, zackige Kanten und Linien („jaggies“)
- Schwingende („atmende“) Kanten animierter Objekte
- Ausblendung / Flimmern ein-pixel-breiter Strukturen
- „Postkutschen-Effekt“ (Objekt-bewegung gegen erwartete Richtung) – ohne Unschärfe!
- Moiré (Überlagerungsmuster)

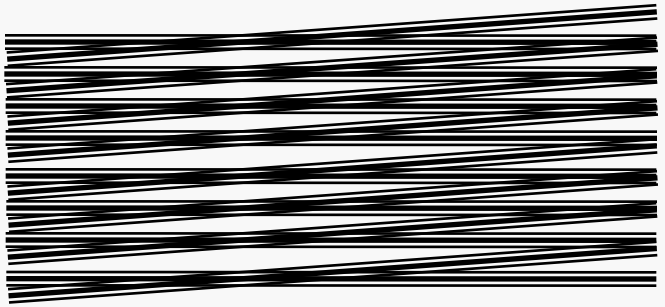


Reelle und synthetische Bilder gleichermaßen betroffen!





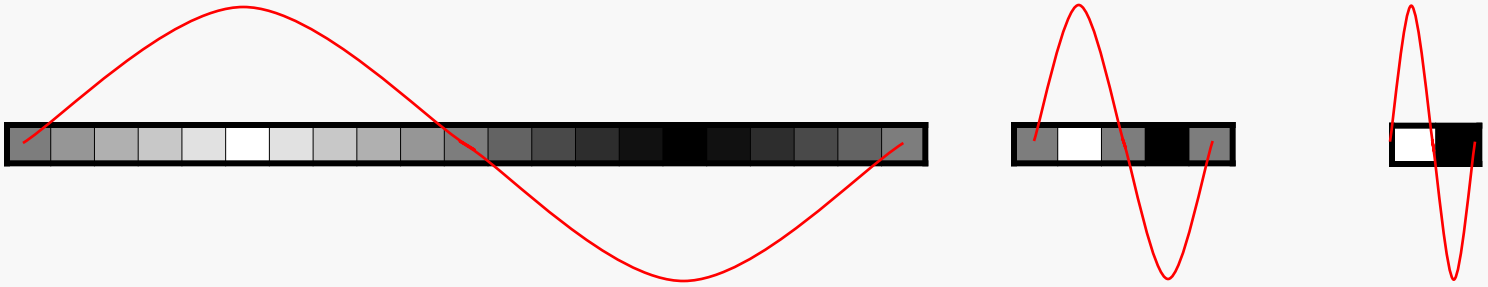




Aliasing und Antialiasing

Bild als Signal, Sehen als Signalverarbeitung:

- Signal: Funktion (Werteverlauf) als Informationsträger im Orts-bereich als f(x,y) (z.B. Bild) oder im Zeitbereich als f(t) (z.B.Ton)
- Werteverlauf beliebig ⇨ Abtastung (Darstellung kontinuierlicher Signale durch diskrete „Kacheln“ mit gemitteltem Wert) i.d.R. verlustbehaftet ⇨Wahrnehmung:(exakte) Signal-Rekonstruktion (~Spannungsverlauf an Kathodenstrahlröhre, an Lautsprecher)
- Darstellung der Information durch Wert-Schwankung (-Wechsel) (Ton-Wechsel ⇨ Sprache, Musik; Farb-Wechsel ⇨ Bild)
- Aliasing stärker in Mustern (Bildmotiven) mit hohen Frequenzen u.Kontrasten (d.h.: mit schnellem u. starkem Helligkeitswechsel)



- Periodische Muster (Gitter, Schachbrett): max.Frequenz = π/Pixel

Aliasing und Antialiasing

Signaltheorie / Abtasttheorie / Fourier-Analyse:

- Jede beliebige periodische, kontinuierliche Fkt. mit unendlichem Definitionsbereich ist auch „im Frequenzbereich“ darstellbar, als Summe über abzählbar viele Sinusoide („Komponenten ihres Frequenz-Spektrums“); deren Frequenzen sind ganzzahlige Vielfache („Harmonische“) einer Grundfrequenz (Fourier 1807).

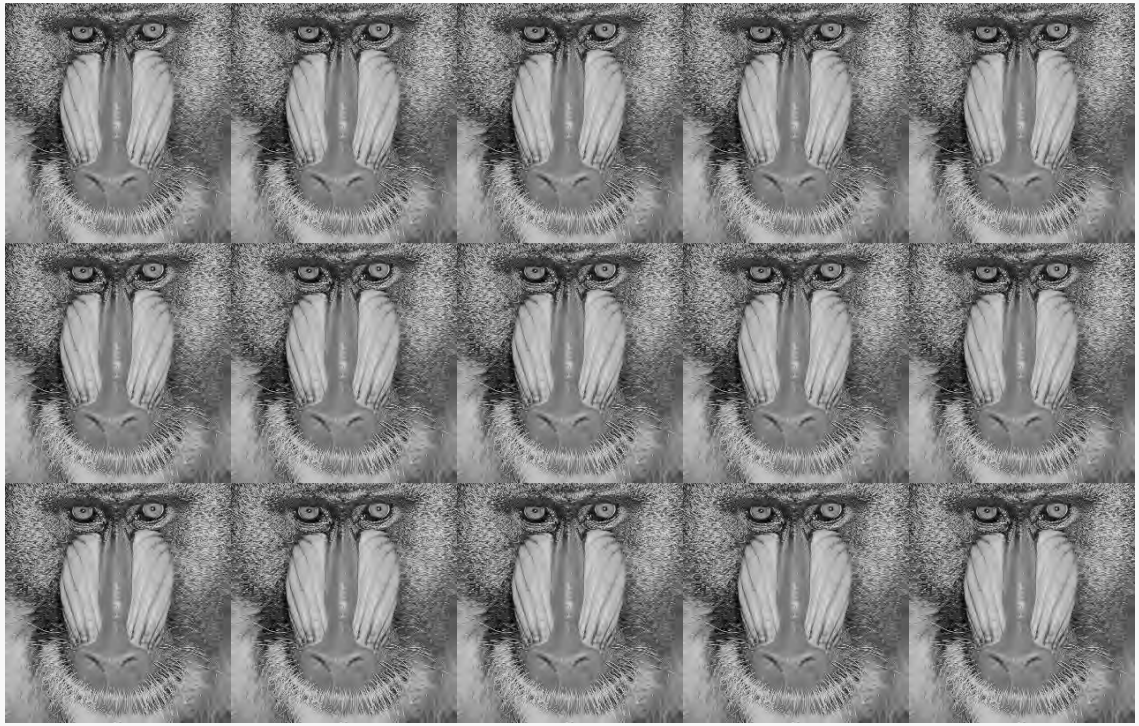
  Ermittlung der Sinusoide zur Darstellung eines konkreten Signals ist Aufgabe d. Fourier-Analyse (auch: Spektralanalyse).

  [(Der) Sinusoid: sinusförmige Funktion, entstanden aus sin(ωt) durch Skalierung (mit A) der Amplitude und (mit ω) der Frequenz und Verschiebung (um φ) der Phase: $f(t)=A\cdot\sin(\omega t+\varphi)+C$]
- (Nicht-periodische) Signale mit endlichem Def.-Bereich (Bilder) sind darstellbar als Sinusoid-Summen, deren Werte außerhalb des (endl.) Def.-Bereichs rasch gegen Null tendieren (<1/x). Ihr Spektrum hat keine Grundfrequenz; es kann alle Frequenzen zw. 0 und (+/-) ∞ enthalten: Frequenzen-Integral statt -Summe.

Aliasing und Antialiasing

- Abhilfe: Bild wird betrachtet als Einzel-Periode eines sich ins Unendliche wiederholenden, kontinuierlichen Musters.

(...)

(...)  (...)

(...)

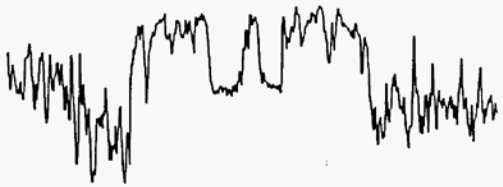
„Nahtstellen“ werden evtl. durch Umrahmung überwunden.

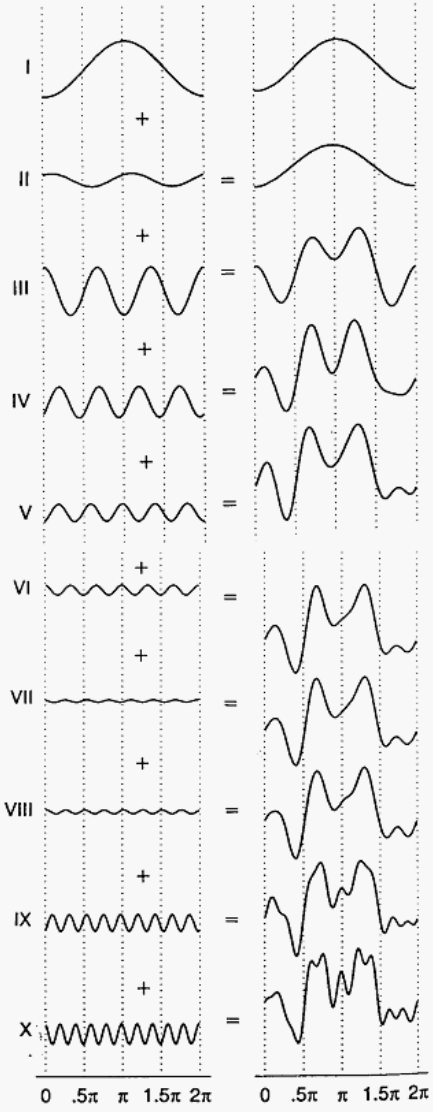
[Vorteil der Darstellungen im Orts- und Frequenzbereich:
Operationen oft leichter durchführbar in nur einem der beiden.]

Aliasing und Antialiasing

Abtasttheorem
(H.Nyquist 1928; Beweis: C.E.Shannon 1949):

Ein periodisches Signal läßt sich **aus einer endlichen Anzahl von Abtastwerten exakt** (d.h.: fehlerfrei) **rekonstruieren**. Dabei dürfen die Abtastpunkte nicht weiter auseinanderliegen als eine halbe Periode der höchsten Frequenz, die im Signal enthalten ist. Diese Grenzfrequenz wird „Nyquist-Frequenz“ genannt.

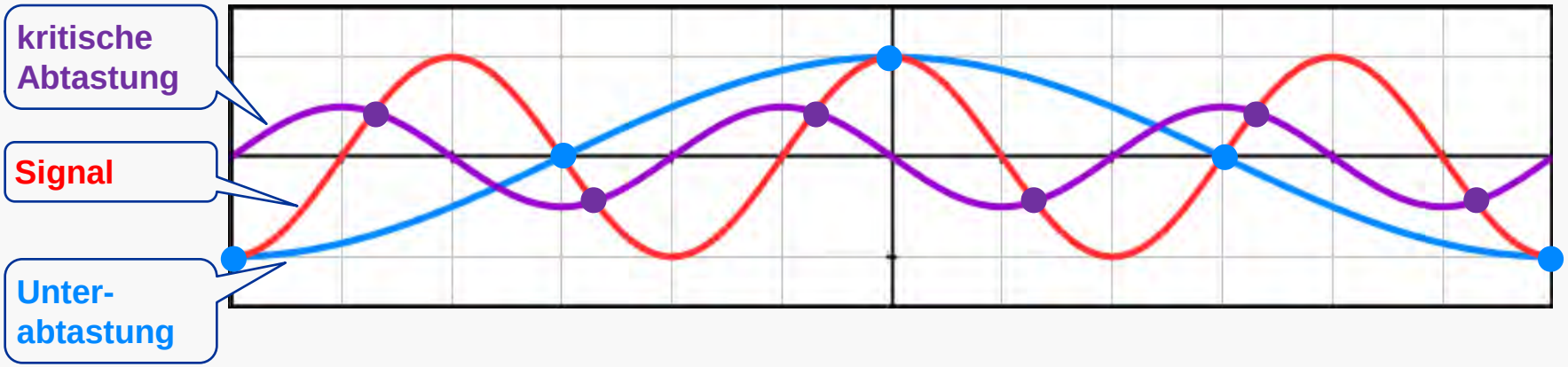




Grafiken: J.D.Foley, A.v.Dam et al.: „Computer Graphics: Principles and Practice in C“, 2/E, Addison-Wesley 1996

Operationen im Frequenzbereich

- Abtastung mit der „Nyquist-Frequenz“ ermöglicht Signal-Rekonstruktion nur, wenn Minima / Maxima erfaßt werden



- Abtastung unterhalb der Nyquist-Frequenz kann Abtastwerte ergeben, die auf eine niedrigere Frequenz schließen lassen. Diese „Maskierung“ hoher durch niedrige Frequenzen nennt man **Aliasing**.
- Linien, Objektkanten und sich perspektivisch verjüngende Strukturen sind hochfrequente, durch Aliasing gefährdete Bildregionen.

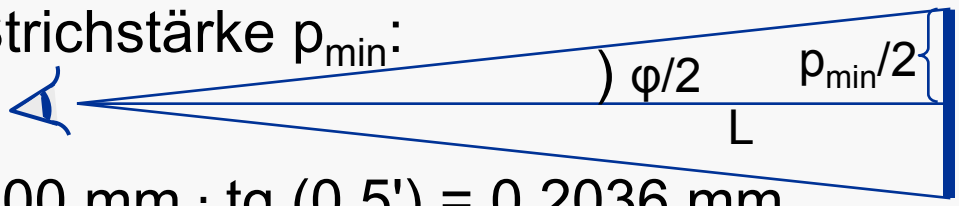
Aliasing und Antialiasing

- Höchste nutzbare Abtastfrequenz („Haar in der Suppe“):

  Auflösungsgrenze des menschlichen Auges: $\varphi \approx 1' (= 1°/60)$

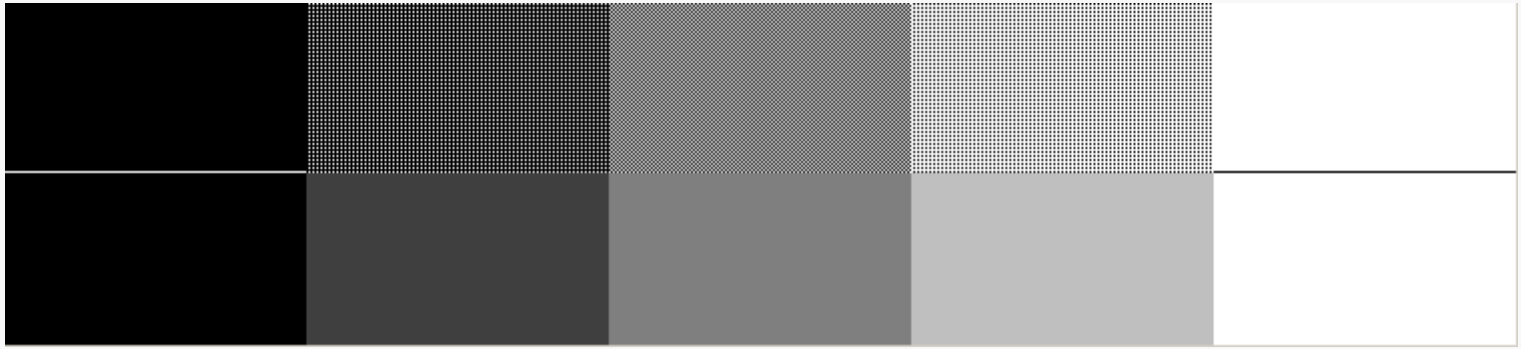
  ergibt bei L=70 cm Abstand vom Display d. minimale noch erkennbare Pixelkante / Strichstärke p_{min}:

  $tg(\varphi/2) = p_{min} / (2 \cdot L)$



  ⇨ $p_{min} = 2 \cdot L \cdot tg(\varphi/2) = 2 \cdot 700\ mm \cdot tg\ (0{,}5') = 0{,}2036\ mm$

  d.h.: eine digitale Darstellung mit einer Pixelgröße $p \leq p_{min}$ ist (beim o.a. Abstand) visuell von einer analogen nicht zu unterscheiden (gängige Pixelgrößen heute: ≤0,4 mm).



⇨ Antialiasing als Verzicht auf Helligkeitssprünge (Tiefpaß)

Aliasing und Antialiasing

Antialiasing-Ansätze: Prefiltering, Supersampling/Postfiltering

Prefiltering (Anwendung v.a. auf Polygonkanten u. Linien):

Farbmittelung zw. Fläche u. Hintergrund beim Setzen der Kantenpixel je nach Grad der Verdeckung jedes Pixels

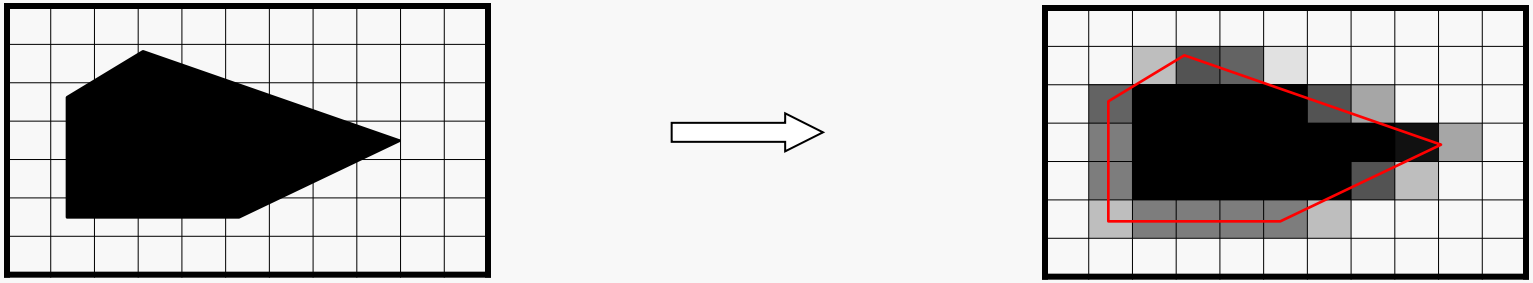
(+) Behandlung nur der Kanten als Problembereiche

(+) Schnell durch Optimierung mit Ganzzahl-Arithmetik (Pitteway / Watkinson 1980, Xiaolin Wu 1991)

Bild:http://en.wikipedia.org/wiki/Xiaolin_Wu's_line_algorithm

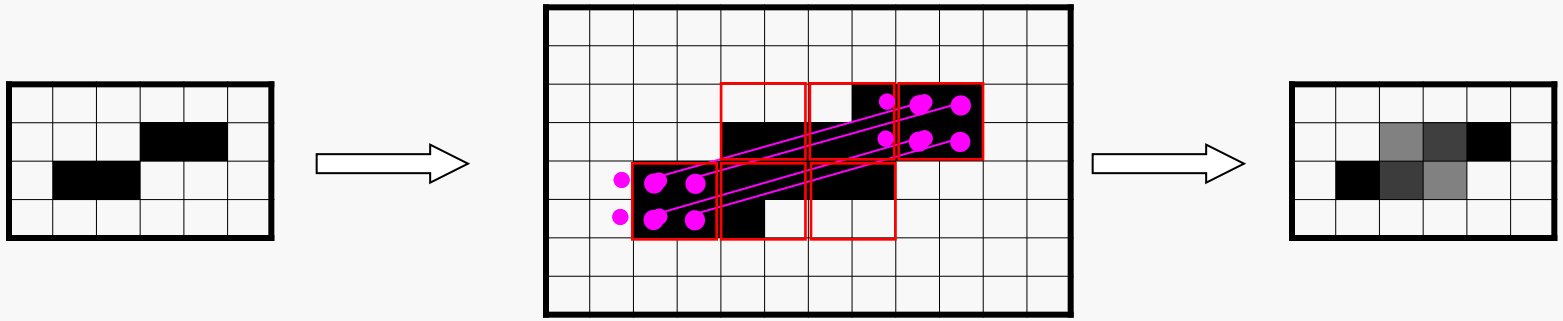
(–) Ungeeignet für texturierte Flächen

(–) Keine Berücksichtigung nachfolgender Zeichnung (evtl. Verdeckung behandelter Pixel/ Änderung d. Hintergrunds)



Aliasing und Antialiasing

Supersampling: Bilderstellung mehrfach, versetzt, in ‚Over-sampling', d.h. in größerer (meist 4- oder 16-facher, gespeicherter o.algorithmisch berücksichtigter) Auflösung, dann Verkleinerung auf d. Auflösung d. Ausgabemediums



Postfiltering: Ungleiche Gewichtung der Nachbarpixel bei der Verkleinerung (z.B. mit 2er-Potenzen):

1/16	1/16	1/16
1/16	1/2	1/16
1/16	1/16	1/16

(+) Überwindet Prefiltering-Schwächen

(–) Rechenintensiv

⇨ Ständige F&E (Gewichtungsmuster, Algorithmen, Textur-Behandlung etc.). Besonders interessant (s. Übung):

Multisampling: Nutzung v. Transparenz, ohne Oversampling

Aliasing und Antialiasing

OpenGL-Antialiasing für Punkte und Linien (RGBA-Modus):

- An-/ Abstellen: **`void glEnable/glDisable(GLenum cap);`**
  [für **`cap`**: **`GL_POINT_SMOOTH`** oder **`GL_LINE_SMOOTH`**]
- Dazu notwendig: Farbmischg anstellen: **`glEnable(GL_BLEND);`**
  ... und Mischungsfunktion wählen: source factor destination factor
  `void glBlendFunc(GLenum srcfct,GLenum destfct);`
  `srcfct`: Opazität („Undurchsichtigkeit“) hinzukommender Farbe
  `destfct`: Opazität der im Framebuffer vorhandenen Farbe
  Meistverwendete Mischungsfunktion:
  `glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);`
  (**`ALPHA`** über **`glColor*()`** gesetzt)
- Einstellbare Punkt- und Strichstärken (in Pixel):

```
void glPointSize(GLfloat size); /*(def.: 1.0)*/
void glLineWidth(GLfloat width);/*(def.: 1.0)*/
```

Aliasing und Antialiasing

Source and Destination Blending Factors (OpenGL 1.1):

Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	src or dest	(0, 0, 0, 0)
GL_ONE	src or dest	(1, 1, 1, 1)
GL_DST_COLOR	source	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	destination	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	source	(1,1,1,1)-(Rd,Gd,Bd,Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1,1,1,1)-(Rs,Gs,Bs,As)
GL_SRC_ALPHA	***src*** or dest	(As,As,As,As)
GL_ONE_MINUS_SRC_ALPHA	src or ***dest***	(1,1,1,1)-(As,As,As,As)
GL_DST_ALPHA	src or dest	(Ad,Ad,Ad,Ad)
GL_ONE_MINUS_DST_ALPHA	src or dest	(1,1,1,1)-(Ad,Ad,Ad,Ad)
GL_SRC_ALPHA_SATURATE	source	(f, f, f,1); f=min(As,1-Ad)

Source / Destination blend factors (r,g,b,a): Sr, Sg, Sb, Sa / Dr, Dg, Db, Da

Source / Destination RGBA (R,G,B,A): Rs, Gs, Bs, As / Rd, Gd, Bd, Ad

Blended RGBA: RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa

Aliasing und Antialiasing

OpenGL-Antialiasing für gefüllte Flächen (RGBA-Modus) durch Nutzung des „Akkumulationspuffers“. Grundsätzlich verfügbar:

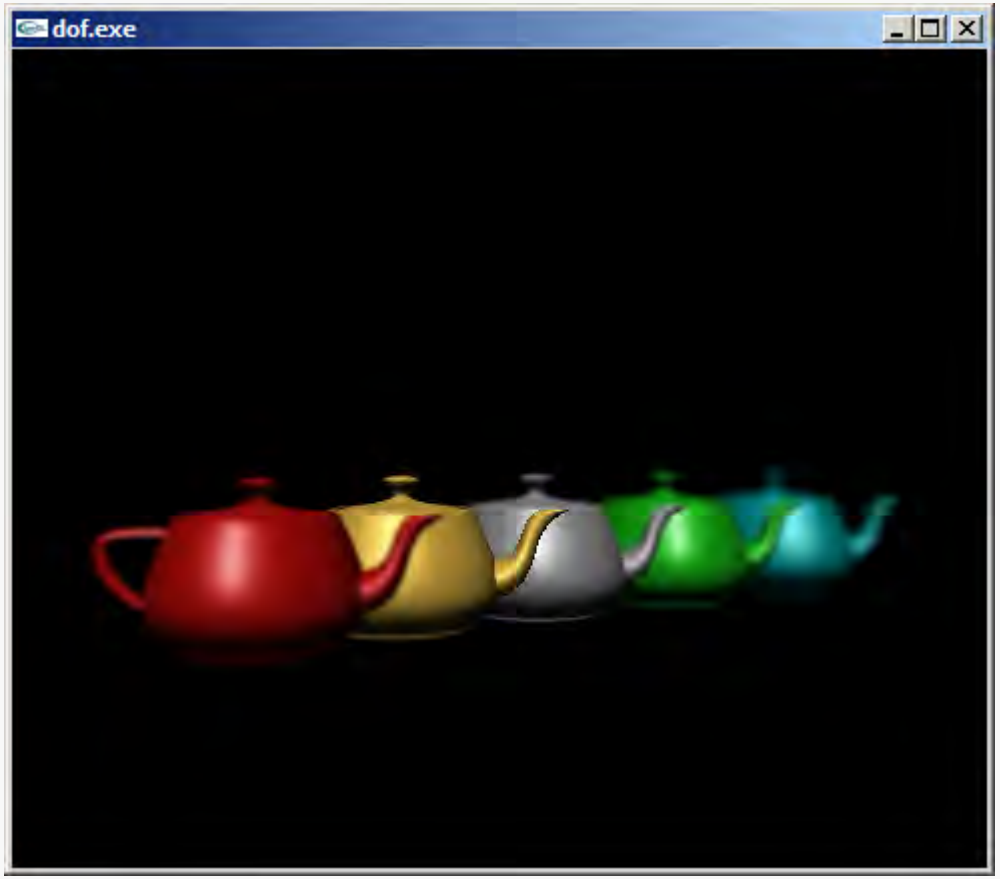
- **Color Buffers** (Farbpuffer): Speicher zum Zeichnen / Anzeigen; Mindest-Ausstattung: ein „Front-Left Color Buffer“ (Double Buffering: front + back; Stereoskopie: left + right); optional zusätzlich: nicht-anzeigbare Hilfspuffer (nondisplayable auxiliary color buffers) – z.B. für unveränderliche Kulissen
- **Depth Buffer** (z-Buffer): Je Pixel (Fragment) ein Tiefenwert
- **Stencil Buffer** (“Schablonenpuffer”): Begrenzt Bildausgabe auf Pixel, die in diesem Puffer nicht besetzt sind (z.B. im Simulator: Stencil Buffer für Konsole / Armatur).
- **Accumulation Buffer**: Hintergrund-Bildspeicher; erlaubt Verknüpfung von Kopien mehrerer Bilder / Bildblöcke aus dem Farbpuffer zu einem Ergebnisbild, das zur Anzeige zurück i.d. Farbpuffer kopiert wird (sog.‘Jittering’ – vgl.Mehrfachbelichtung)

  ⇨ Eignung für Tiefen- u. Bewegungsunschärfe / Stereoskopie / Antialiasing (Multisampling) /...

Aliasing und Antialiasing

- Anwendungsbeispiel Accumulation Buffer:

  Tiefenschärfen-Effekt (Depth Of Field) durch Überlagerung mehrerer Darstellungen mit versetztem Sichtvolumen.



Aliasing und Antialiasing

- Setzen des Löschwertes für den jeweiligen Puffer mit

```
void glClearColor(GLclampf red, GLclampf green,
       GLclampf blue, GLclampf alpha);
void glClearDepth (GLclampd depth); /*def.:1.*/
void glClearStencil (GLint s);
void glClearAccum (GLfloat red, GLfloat green,
       GLfloat blue, GLfloat alpha);
```

  Voreinstellung des Löschwertes für den Tiefenpuffer ist 1.0, für alle anderen 0.0. Werte der Datentypen **`GLclamp*`** werden auf das Intervall [0.0,1.0] begrenzt.

- Puffer-Löschung: **`void glClear(GLbitfield mask);`**
  Bit**`mask`**e: **`GL_COLOR_BUFFER_BIT`**, **`GL_DEPTH_BUFFER_BIT`**, **`GL_STENCIL_BUFFER_BIT`**, **`GL_ACCUM_BUFFER_BIT`** – ggf. mehrere, verknüpft mit bitweisem ODER (**`|`**).

  Löschungen sind teure (d.h. rechenintensive) Operationen!

Aliasing und Antialiasing

Accumulation Buffer zur Implementierung von Antialiasing:

- Initialisierung des entsprechenden Modus (und Speichers) im Fenstersystem:
  `glutInitDisplayMode(/*…|*/GLUT_ACCUM/*|GLUT_DOUBLE…*/);`
  (falls getrennte Aufrufe: **GLUT_ACCUM** vor **GLUT_DOUBLE** !)
- Vereinbarung des Löschwertes u. Löschung d. Puffers in OpenGL:
  `/*void*/ glClearAccum(0.0, 0.0, 0.0, 0.0);`
  `/*void*/ glClear(GL_ACCUM_BUFFER_BIT);`
  (bei Einsatz von **GL_LOAD** meist entbehrlich – s.u.)
- Erzwingung d. Ausführg v. Anweisgn („Weiter mit nächstem Bild!“):
  `void glFlush(void);/*erzwingt AusfuehrgsStart*/`
  Erzwingung der Fertigstellung („Warte auf Pixelbild!“):
  `void glFinish(void);/*wartetAusfuehrgsEnde ab*/`
  (Hintergrund: Manche Spezial-Hw (z.B. Netzwerk) sammelt mehrere Anweisungen, bevor sie sie verarbeitet.)

Aliasing und Antialiasing

- Steuerung des Akkumulationspuffers:

  `void glAccum(GLenum op, GLfloat value);`

  Werte für die Operation **`op`**:

`GL_ACCUM` liest Pixelwerte vom (aktuell lesbaren) Farbpuffer, multipliziert ihre R-,G-,B- und Alpha-Werte mit **`value`** und addiert das Ergebnis zum Inhalt des Akkumulationspuffers.

D.h.: Zur Überlagerung mehrerer Einstellungen (müssen bzw.) muß der Farbpuffer vor jedem Einzelbild gelöscht werden!

`GL_LOAD` berechnet dasselbe Ergebnis wie **`GL_ACCUM`** und überschreibt damit den Akk/puffer; kann zu Beginn einer Akkumulation **`glClear()`** (u.ggf. **`glClearAccum()`**) ersetzen.

`GL_RETURN` multipliziert die Werte im Akk/puffer mit **`value`** und setzt das Ergebnis in den (aktuell beschreibbaren) Farbpuffer.

`GL_ADD` / **`GL_MULT`** addiert / multipliziert die Pixelwerte im Akkumulationspuffer mit **`value`** (**`GL_MULT`** begrenzt die Ergebniswerte auf das Intervall [-1.0,1.0]).

Aliasing und Antialiasing

Antialiasing mit dem Akk/puffer bedeutet mehrfache, versetzte Überlagerung ein und derselben Szene bei individueller Bild-Rasterisierung nach jeder Versetzung; dazu wird oft in der Grafik-Pipeline eine abschließende Translation hinzugefügt.

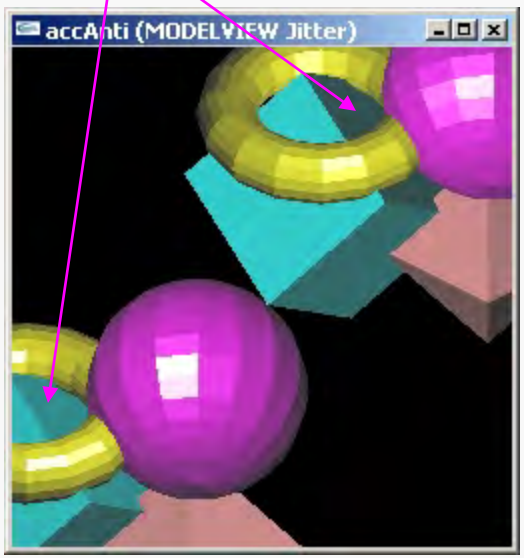
Grundsätzlich mögliche Versetzungen:

- Verschiebung der Szene:
  `glTranslatef()` im **`GL_MODELVIEW`**-Modus
  (interessant allenfalls für Szenen mit geringer Tiefe)
- Verschiebung des Augenpunkts:
  `glTranslatef()` im **`GL_PROJECTION`**-Modus
  (oft genutzt – evtl. mit Nebeneffekten: „Blick auf Rasierklinge“)
- Verschiebung des Viewports:
  Veränderung der Variablen **`left`**, **`right`**, **`bottom`**, **`top`** in **`glFrustum()`** bzw. **`glOrtho()`** im **`GL_PROJECTION`**-Modus
  – empfohlen!

Aliasing und Antialiasing

Die Unterschiede zwischen der Versetzung (i) der Szene (vgl. Stereoskopie – im **`GL_MODELVIEW`**-Modus) und (ii) des Viewports (im **`GL_PROJECTION`**-Modus) sind meist erst bei übergroßer Versetzung erkennbar

(z.B. anhand der Schattierung oder der Objekt-Ansichten):



(i)

(ii)

Übung:

Erweiterung des OpenGL-Programms zur Darstellung eines 3D-Modells um die Antialiasing-Ansätze (s. Übungsblatt).

vgl. `accanti.exe`, `accpersp.exe`

Aliasing und Antialiasing

Antialiasing-Ansätze durch Verschiebung des Augenpunkts und des Viewports liefern meist gleichwertige Ergebnisse

– wobei Verschiebung des Viewports keine Koordinaten-Transformation der Szene benötigt (Optimierungspotential)

kein Antialiasing

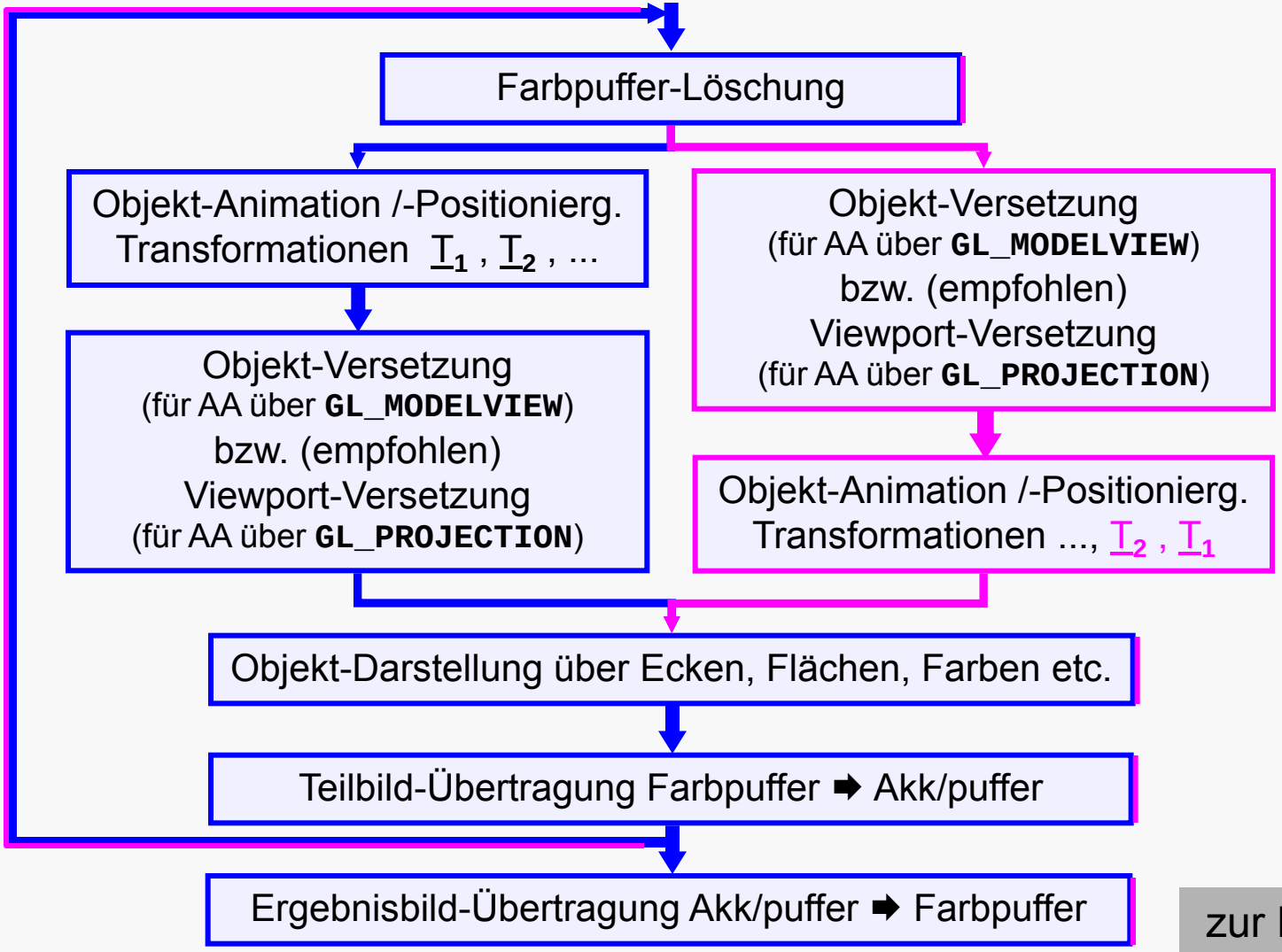
AA durch Augenpunkt

AA durch Viewport

Aliasing und Antialiasing

Logisch-mathematische Schritte:

OpenGL-Anweisungen:



zur Erinnerung:

OpenGL–Transformationen: Modeling

- Punkt-Trfn = Matrizen-Multiplikationen von links (s.o.):
  $\underline{v}_{neu} = \underline{T}_n \bullet (\dots) \bullet \underline{T}_2 \bullet \underline{T}_1 \bullet \underline{v}_{alt} = \underline{T}_{gesamt} \bullet \underline{v}_{alt}$
- OpenGL: Laden **`mat[16]`**: **`glLoadMatrix{fd}(mat)`**
  Matrizen-Multiplikation: **`glMultMatrix{fd}(mat)`**

⇨ eigene Matrizen (z.B.: OpenGL bietet keine Scherung)

trans-poniert!

Aber: OpenGL multipliziert von rechts (engl.: *postmultiply)*

⇨ Aufbau von $\underline{T}^T_{gesamt}$ in umgekehrter Reihenfolge:

$\underline{C}_0 = \underline{\mathbf{I}}$

$\underline{C}_1 = \underline{C}_0 \bullet \underline{T}^T_n$

(...)

$\underline{C}_i = \underline{C}_{i-1} \bullet \underline{T}^T_{n-i+1}$

(...)

$\underline{C}_n = \underline{C}_{n-1} \bullet \underline{T}^T_1 = \underline{T}^T_n \bullet (\dots) \bullet \underline{T}^T_2 \bullet \underline{T}^T_1 = \underline{T}^T_{gesamt}$

```
/*Betr.: Matrix Modell-Trf.:*/
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(TN); /*...*/
glMultMatrixf(T1);
```

$\underline{C}$: aktuelle (engl. *current*) Positionierungsmatrix; $\underline{\mathbf{I}}$: Einheitsmatrix