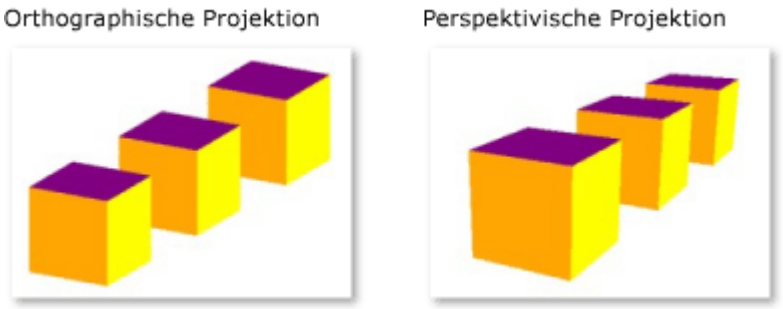
4.4. PROJEKTIONSTRANSFORMATIONEN

Projektions-Transformationen

- Abbildung der 3D-Szene auf den zweidimensionalen Raum
- Projektionsarten:
  - Orthografische Projektion (auch: Parallelprojektion)
  - Perspektivische Projektion (auch: Zentralprojektion)



- Wegfall einer Dimension, z.B. $(x, y, z) \rightarrow (x, y)$
- z-Werte werden aber weiterhin normiert gespeichert für spätere Verarbeitungsschritte (z.B. Verdeckungsberechnung)

https://msdn.microsoft.com/de-de/library/system.windows.media.media3d.orthographiccamera%28v=vs.110%29.aspx

Orthografische Projektion

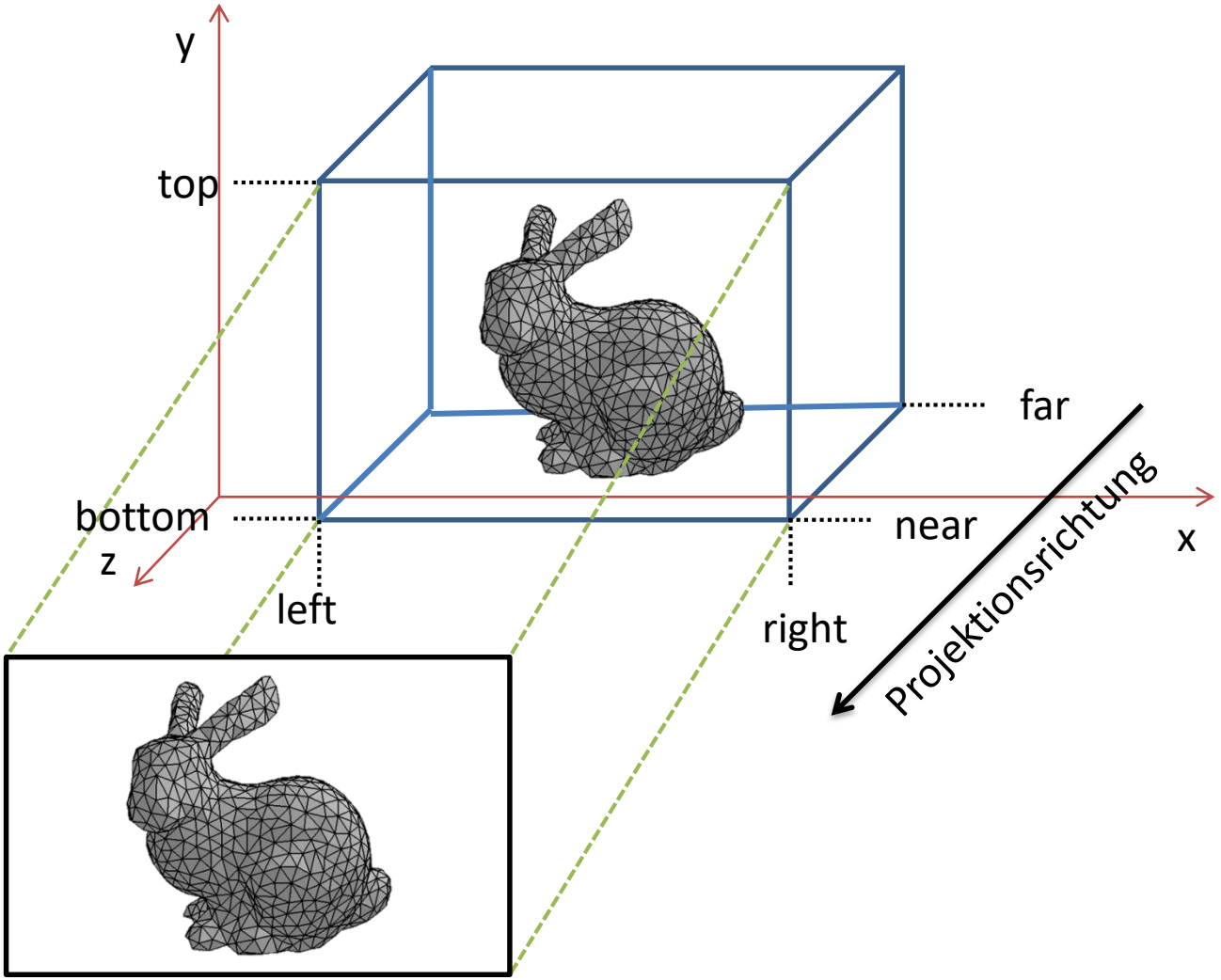
- Parallele Strahlen von Objekten zur Bildfläche
  - Größen und Winkel aller Objekte bleiben erhalten
- Ausschnitt aus der Szene durch *Clipping Planes.*
- In OpenGL:

```
glOrtho( GLdouble left, GLdouble right, Gldouble bottom,
         GLdouble top,  Gldouble near,  Gldouble far);
```

- Transformationsmatrix der orthografischen Projektion: Identitätsmatrix

$$\mathbf{P}_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthografische Projektion



Perspektivische Projektion

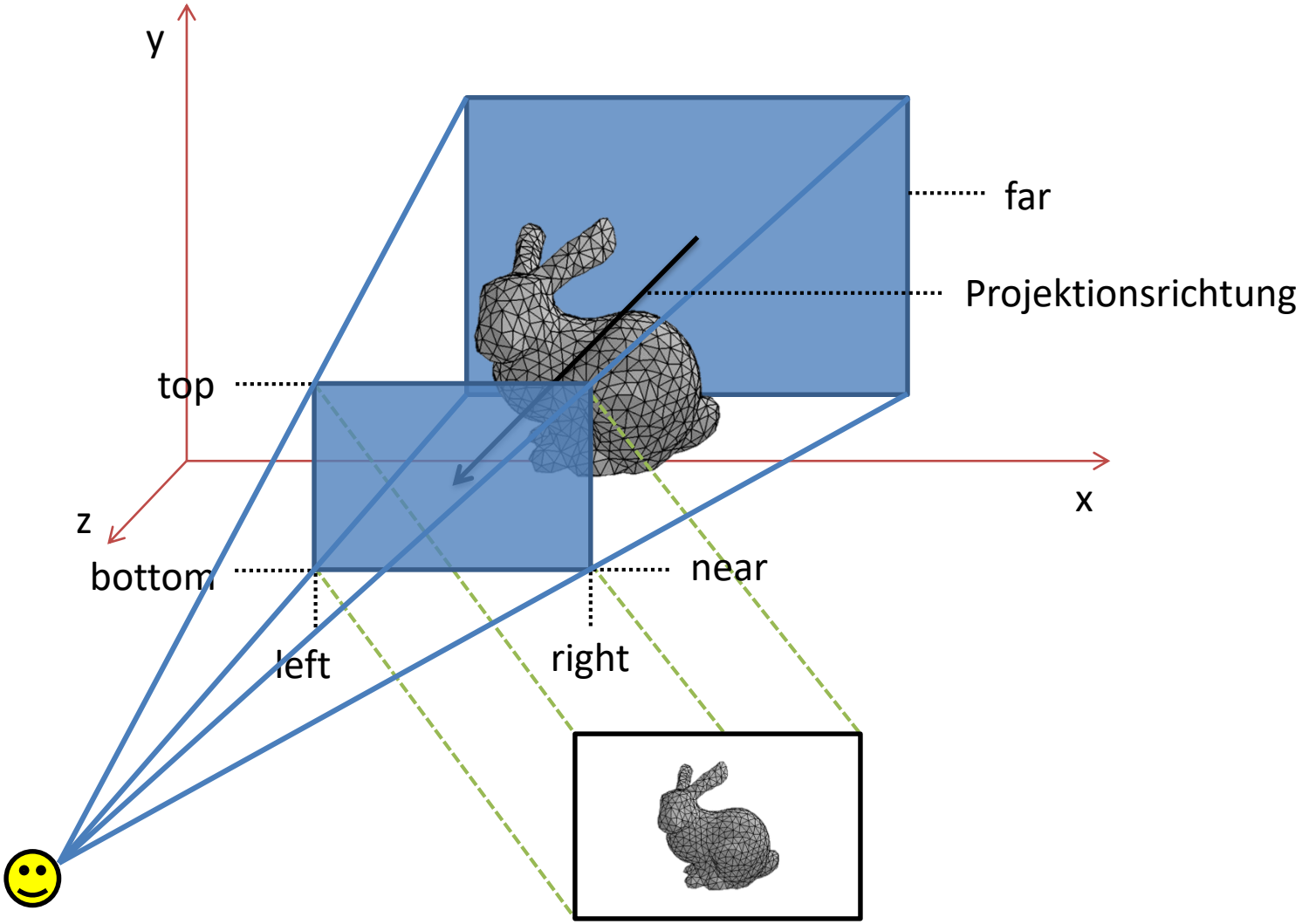
- Konvergierende Strahlen von allen sichtbaren Objekten zum Augpunkt
- Objekte nah am Augpunkt erscheinen größer als entfernte Objekte
- Ausschnitt aus der Szene durch einen Kegelstumpf, definiert durch *Clipping planes*
- In OpenGL:

```
glFrustum( GLdouble left, GLdouble right, Gldouble bottom,
           GLdouble top,  Gldouble near,  Gldouble far);
```

- Transformationsmatrix der perspektivischen Projektion:

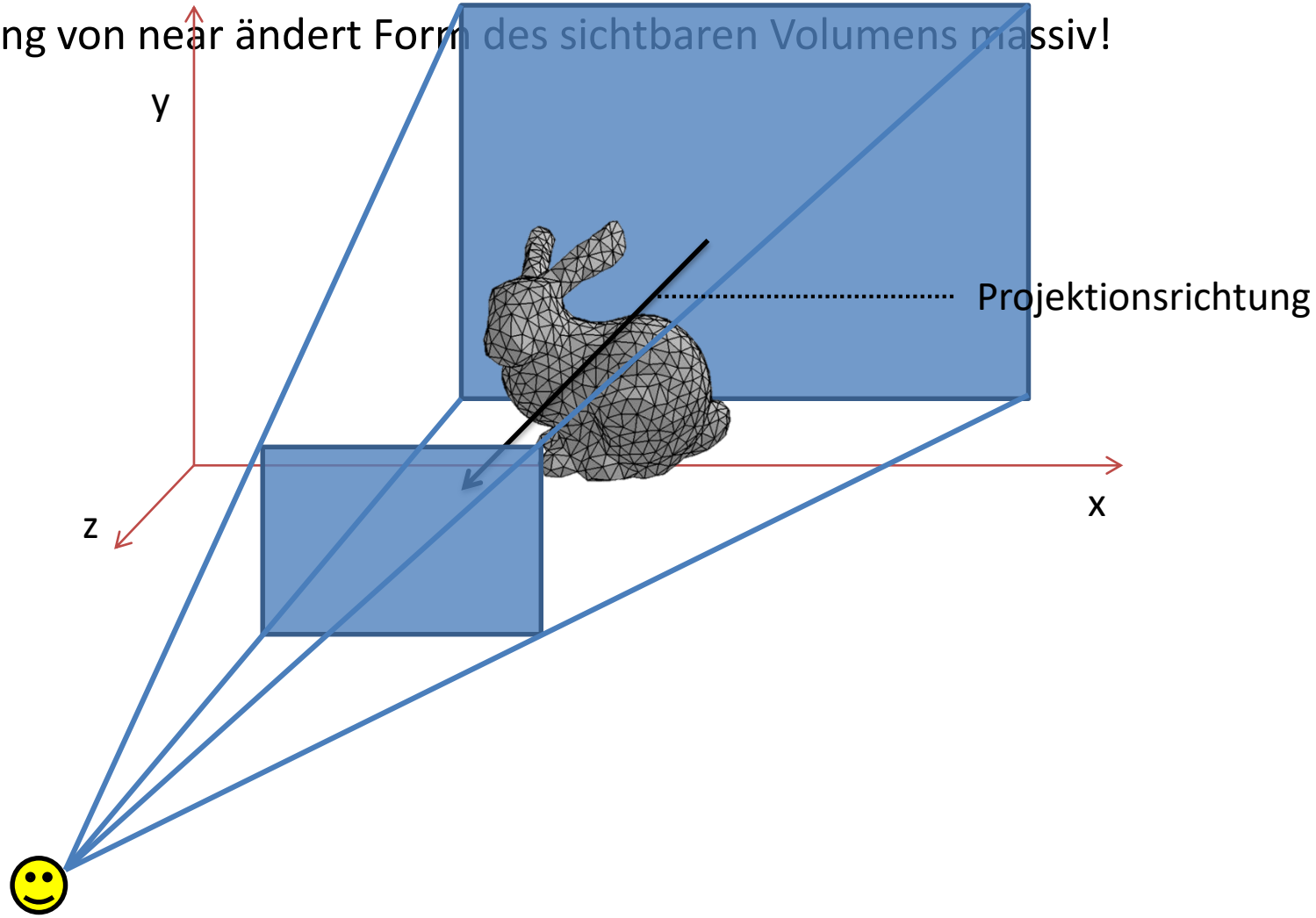
$$\mathbf{P}_{persp.} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{far}{near} & far \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix}$$

Perspektivische Projektion



Perspektivische Projektion

- Änderung von near ändert Form des sichtbaren Volumens massiv!



Perspektivische Projektion

- **`glFrustum`** oft etwas umständlich: left, right, bottom, top müssen z.B. erst aus Blickwinkeln berechnet werden.
- OpenGL Utility Library definiert bequemeren Befehl für solche Fälle:

```
gluPerspective(GLdouble alpha, GLdouble aspect,
               GLdouble near, GLdouble far);
```

  - α: Vertikaler Blickwinkel im Wertebereich $[0, 180]°$
  - aspect: Verhältnis zwischen vertikalem (α) und horizontalen (β) Blickwinkel: β/α.

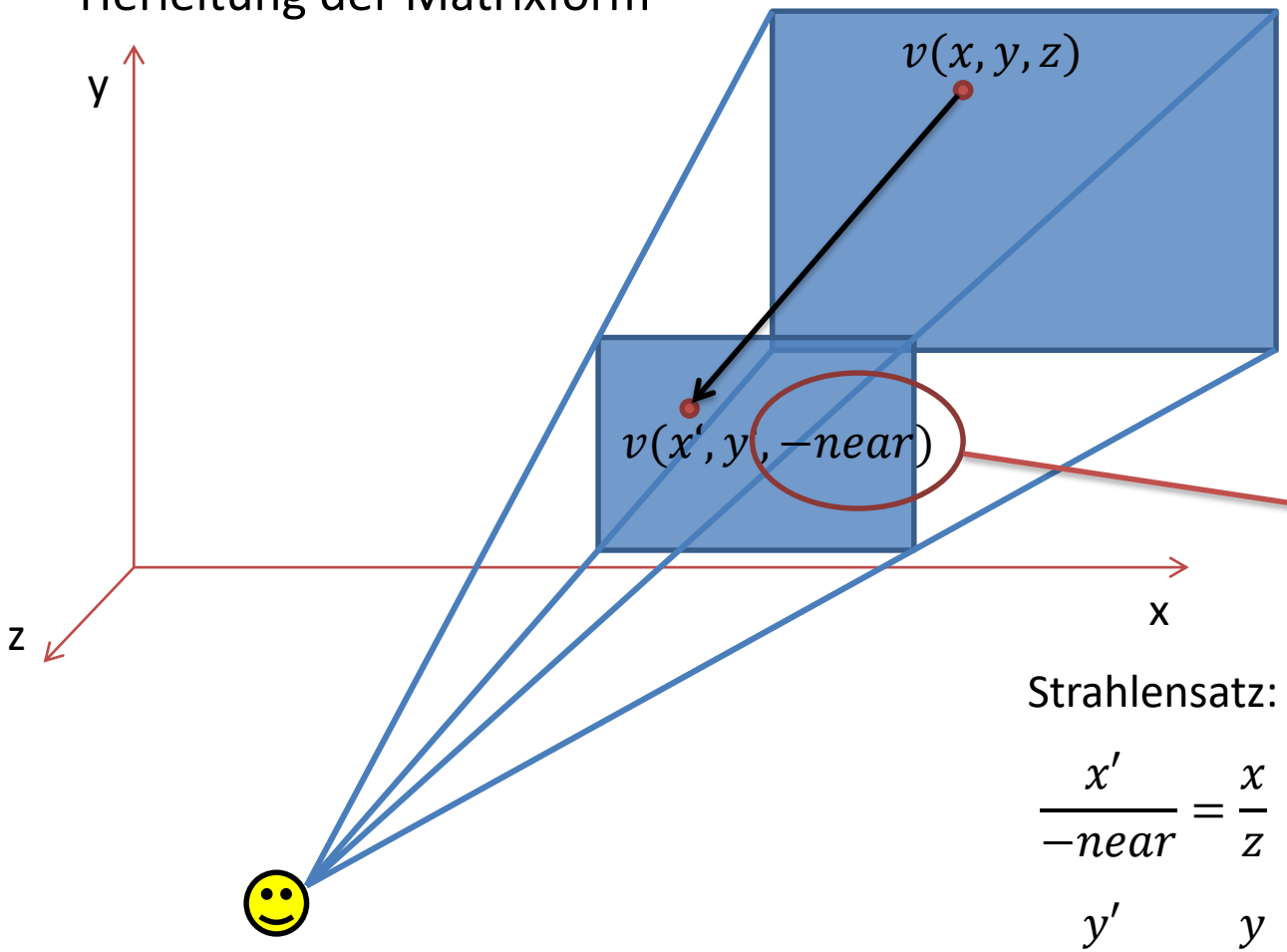
- Berechnung von left, right, top, bottom:

$$left = -near \cdot \tan\frac{\beta}{2}$$
$$right = near \cdot \tan\frac{\beta}{2}$$
$$bottom = -near \cdot \tan\frac{\alpha}{2}$$
$$top = near \cdot \tan\frac{\alpha}{2}$$

- Einschränkung: Blickwinkel müssen nach links/rechts, oben/unten symmetrisch sein!

Perspektivische Projektion

- Herleitung der Matrixform



Beachte hier Vorzeichen!
$near$ entspricht Abstand vom Augpunkt (entlang der negativen z-Richtung)

Strahlensatz:

$$\frac{x'}{-near} = \frac{x}{z} \Leftrightarrow x' = -\frac{near}{z}x$$

$$\frac{y'}{-near} = \frac{y}{z} \Leftrightarrow y' = -\frac{near}{z}y$$

Perspektivische Projektion

- Perspektivische Projektionsmatrix erfüllt die Strahlensatzgleichungen:

$$v' = \mathbf{P}v \Leftrightarrow \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 + \frac{far}{near} & far \\ 0 & 0 & -\frac{1}{near} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ \left(1 + \frac{far}{near}\right) z + far \cdot w \\ -\frac{z}{near} \end{pmatrix}$$

- Dividieren durch den inversen Streckungsfaktor ergibt die euklidischen Koordinaten:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} -\frac{near}{z} x \\ -\frac{near}{z} y \\ -\frac{near}{z} far \cdot w - (far + near) \end{pmatrix}$$

Siehe Strahlensatz auf der letzten Folie!

Normierung

- Abbildung der *near clipping plane* auf Bildschirmfenster: Zwischenschritt Normierung
  - Division durch halbe Ausdehnung der *near clipping plane*
  - Verschieben des Zentrums des Wertebereichs in den Ursprung

$$x' = \frac{2}{right - left}\, x - \frac{right + left}{right - left}$$

$$y' = \frac{2}{top - bottom}\, y - \frac{top + bottom}{top - bottom}$$

$$z' = \frac{-2}{far - near}\, z - \frac{far + near}{far - near}$$

$$w' = w$$

  - Wertebereich nach der Transformation: $[-w\ w]$
  - Division durch w bildet x, y, z-Werte auf Intervall $[-1\ 1]$ ab.

Normierung

- Erzielt Unabhängigkeit von Größe des sichtbaren Volumens
- In Matrix-Schreibweise:

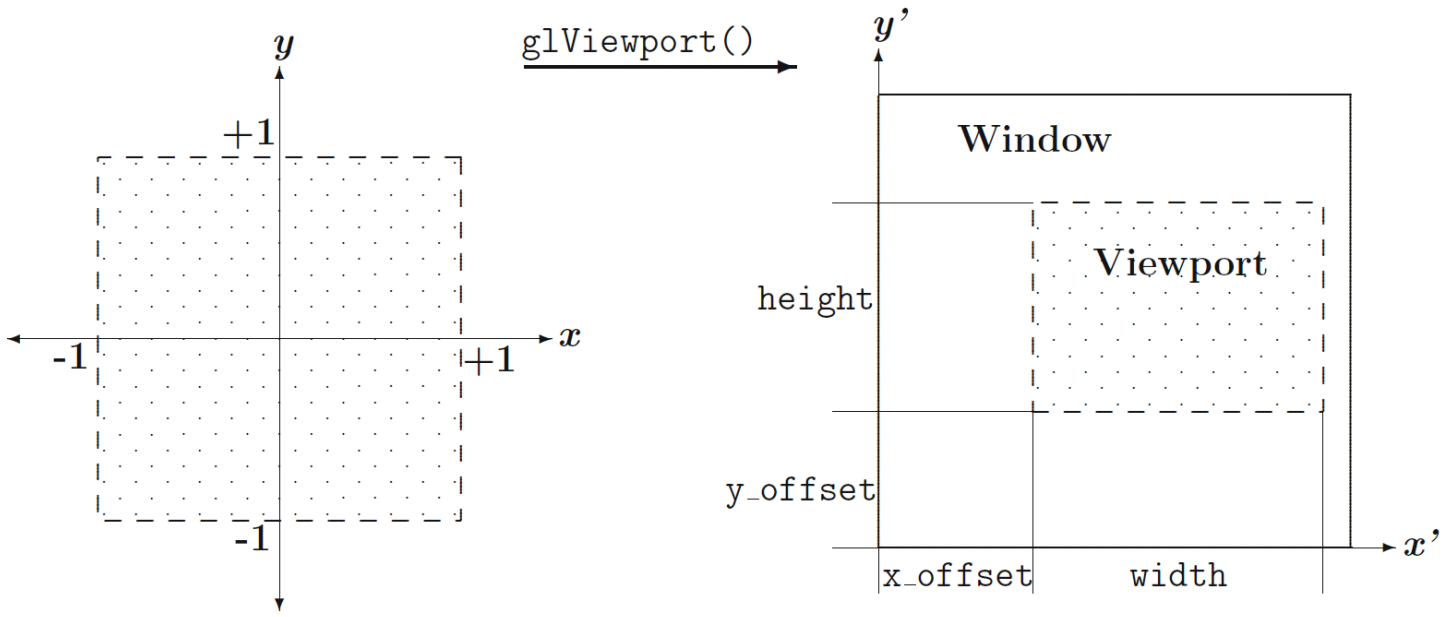
$$\mathbf{N} = \begin{pmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{-2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- **`glOrtho()`**, **`glFrustum()`** und **`gluPerspective()`** führen Projektion und Normierung gemeinsam durch!

4.5. VIEWPORT-TRANSFORMATIONEN

Viewport-Transformation

- Abbildung der Szene auf Ausschnitt des Bildschirms (*Viewport*)
- Viewport definiert in Pixeln
  - Startpunkt (x_{Offset}, y_{Offset})
  - Ausdehnung (width, height)



A. Nischwitz et al. „Computergrafik und Bildverarbeitung“, Band 1: Computergrafik“

Viewport-Transformation

- Umrechnung aus den normierten projizierten Daten (für $w = 1$):

$$x' = \frac{width}{2}x + \left(x_{\text{offset}} + \frac{width}{2}\right)$$
$$y' = \frac{height}{2}y + \left(y_{\text{offset}} + \frac{height}{2}\right)$$

- In Matrix-Schreibweise:

$$\mathbf{V} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & x_{\text{offset}} + \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & y_{\text{offset}} + \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

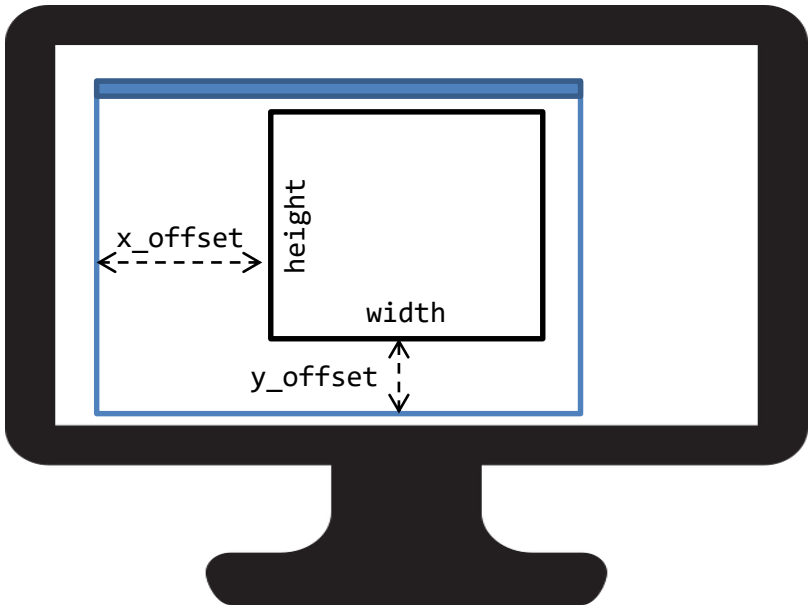
- Die z-Koordinate normiert auf den Wertebereich $[-1\ 1]$ wird erhalten um spätere Auswertungen durchführen zu können (z.B. Verdeckungsberechnung)

Viewport-Transformation

- Umsetzung in OpenGL:

```
glViewport(x_offset, y_offset, width, height);
```

- Größen in Pixel
- Offset bezieht sich auf Fenster
- Wenn nicht explizit angeben: Viewport = Window.
- Window kann mehrere Viewports enthalten
- Aspektverhältnis des Viewports muss dem des sichtbaren Volumens entsprechen, sonst werden Objekte gestaucht/gedehnt dargestellt.



4.6. MATRIZENSTAPEL

Matrizenstapel

- Abarbeitungsreihenfolge: Jeder Vertex durchläuft alle Transformationsstufen

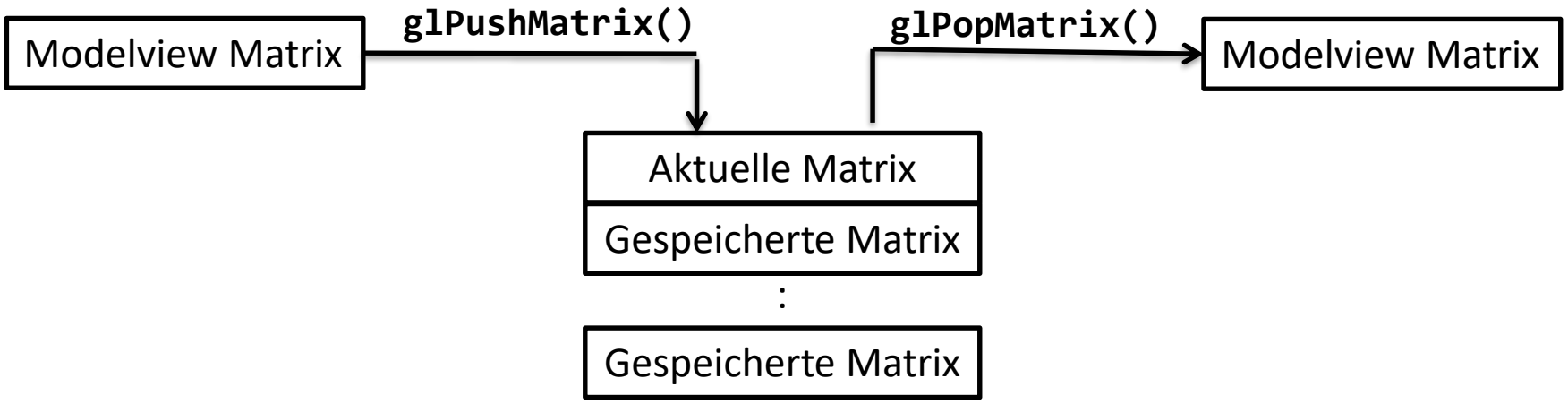
$$v' = \mathbf{V}(\mathbf{N}(\mathbf{P}(\mathbf{S}(\mathbf{R}(\mathbf{T} \cdot v)))))$$

  - Hoher Aufwand, da große Zwischenergebnisse gespeichert werden müssen.
  - Effizienterer Weg:

$$v' = (\mathbf{V} \cdot \mathbf{N} \cdot \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{T}) \cdot v$$

- Meist mehrere Matrizen vorhanden, z.B. zur Wiederverwendung von Objekten → Matrizenstapel zur Zwischenspeicherung von Matrizen
- In OpenGL:
  - „Modelview-Matrizen“: bis 32 Matrizen (**`glMatrixMode(GL_MODELVIEW)`**)
  - „Projektions-Matrizen“: bis 2 Matrizen (**`glMatrixMode(GL_PROJECTION)`**)

Matrizenstapel



- **`glPushMatrix:`**
  - Anfertigen einer Kopie der aktuellen Matrix + auf dem Stapel speichern.
  - Weitere Transformationen können hinzugefügt werden (Multiplikation auf bisher aktuelle Matrix).
- **`glPopMatrix:`**
  - Entfernt aktuelle Matrix und kehrt zur letzten auf dem Stapel zurück.

Matrizenstapel

- Beispielhafte Benutzung der Matrizenstapel:
  - Projektions-Matrizen-Stapel: Umschalten zwischen orthografischer und perspektivischer Projektion
  - Modelview-Matrizen-Stapel: Wiederverwendung von Objekten

```
glLoadIdentity();
glTranslatef(…);

for(int i=0; i<5; i++) {
    glPushMatrix()  // - ab hier arbeiten im „lokalen“ Koordinatensystem
    glRotatef(…);
    glTranslatef(…);
    glBegin(GL_TRIANGLES);  // Objektdefinition
    glVertexfv(v);
                :
    glEnd();
    glPopMatrix();
}
```

ZUSAMMENFASSUNG

Zusammenfassung

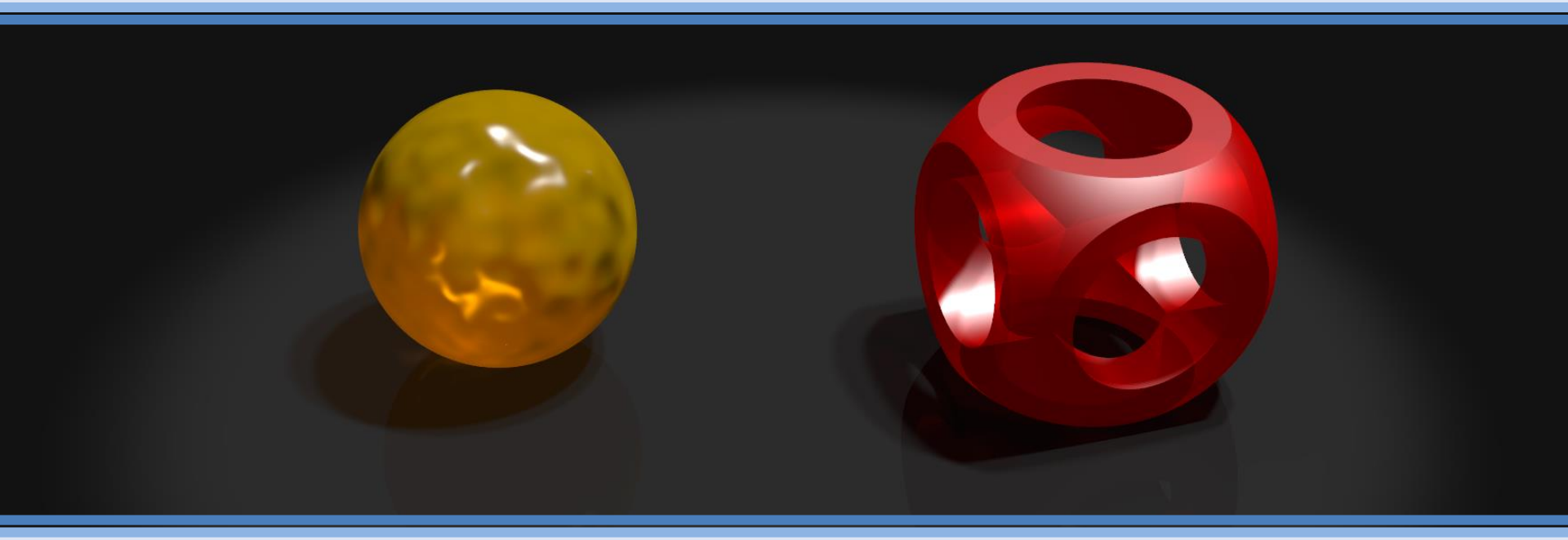
- Transformationskette von lokalen Koordinaten bis Bildschirmkoordinaten
- Verwendung von homogenen Koordinaten in allen Transformationen
  - Transformationsmatrizen immer 4x4.
- Modelltransformationen: Positionierung von Objekten in Weltkoordinaten
  - Translation, Rotation, Skalierung
  - Reihenfolge der Transformationen wichtig
- Augpunkttransformationen zur Positionierung des Beobachters
  - Lassen sich in Modelltransformationen überführen
- Projektionstransformation zur Abbildung der 3D-Szene in 2D
  - Orthografische oder perspektivische Projektion
  - Normierung der Koordinaten
- Viewport-Transformationen: Positionierung auf dem Bildschirm bzw. im Fenster
- Wiederverwendung von Matrizen/Transformationen durch Matrizenstapel

Übungsfragen Kapitel 4

- Welche Koordinatensysteme und Transformationen kommen in der Transformationskette vor? Geben Sie die Reihenfolge der Abarbeitung an.
- Warum werden Modelltransformation und Augpunkt-Transformation meist in einer gemeinsamen Modelview-Matrix zusammengefasst?
- Was ist der Matrizenstapel? Wofür wird er benutzt?

Computergrafik

T. Hopp

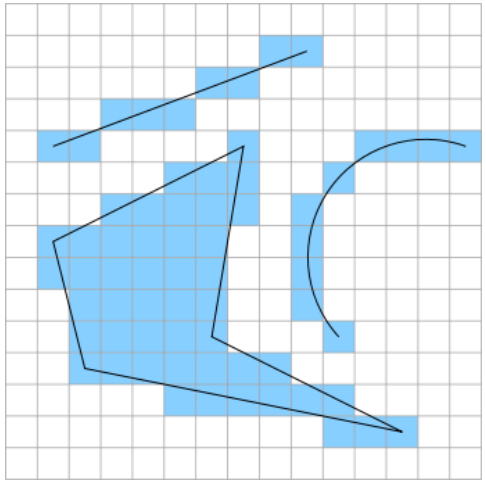


Themenübersicht

1. Einführung
2. Programmierbibliotheken / OpenGL
3. Geometrische Repräsentation von Objekten
4. Koordinatensysteme und Transformationen
5. **Zeichenalgorithmen**
6. Buffer-Konzepte
7. Farbe, Beleuchtung und Schattierung
8. Texturen
9. Animationen
10. Raytracing
11. Volumenvisualisierung

Zeichenalgorithmen

- Durch Objekte und Transformationen definierte Szene muss auf den Bildschirm gezeichnet werden.
- Zeichenalgorithmen übernehmen diesen Vorgang des sogenannten Rasterns
  - D.h. Umsetzung eines kontinuierlichen Objektes in diskrete Pixel

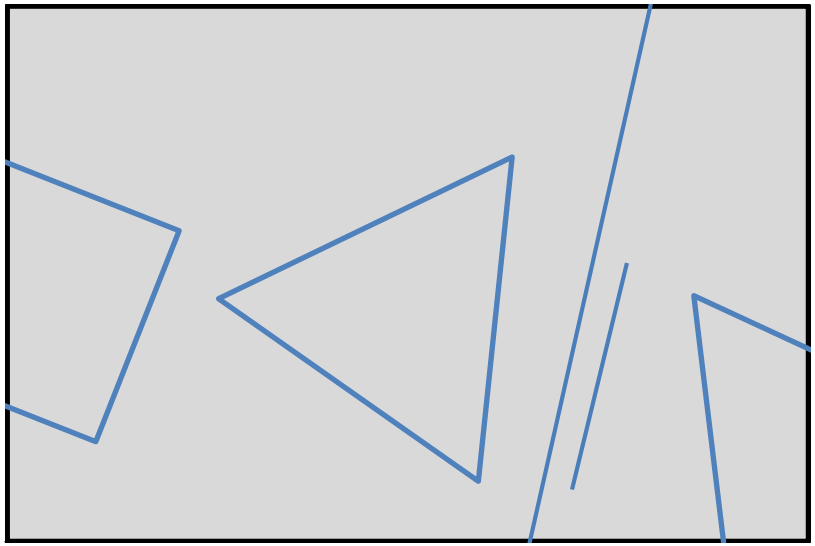


http://iloveshaders.blogspot.de/2011/05/how-rasterization-process-works.html

5.1. CLIPPING

Clipping

- Nach Durchlaufen der Transformationskette ist nur noch ein Ausschnitt der 3D Szene zu sehen.
- Alle Objekte die außerhalb dieses sichtbaren Bereichs (=Viewport) liegen müssen nicht gezeichnet werden.
- Clipping = Zuschneiden von Objekten an einem vorgegebenen Bereich.

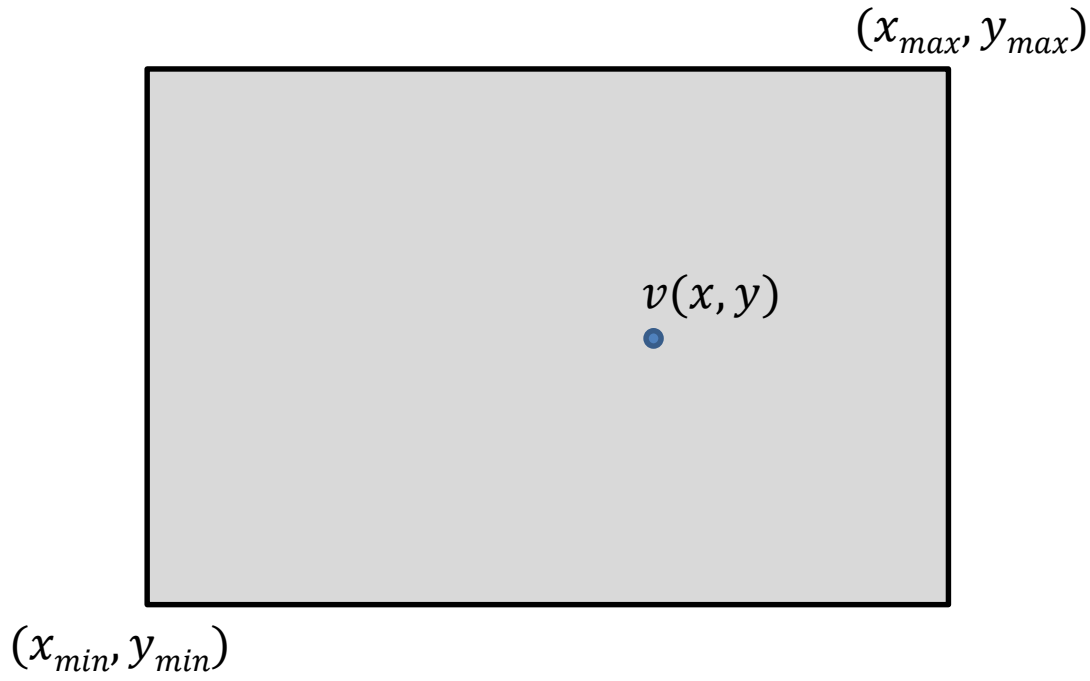


Clipping von Vertices

- Einfachste Form des Clippings
  - Test ob der Vertex innerhalb des Viewports liegt:

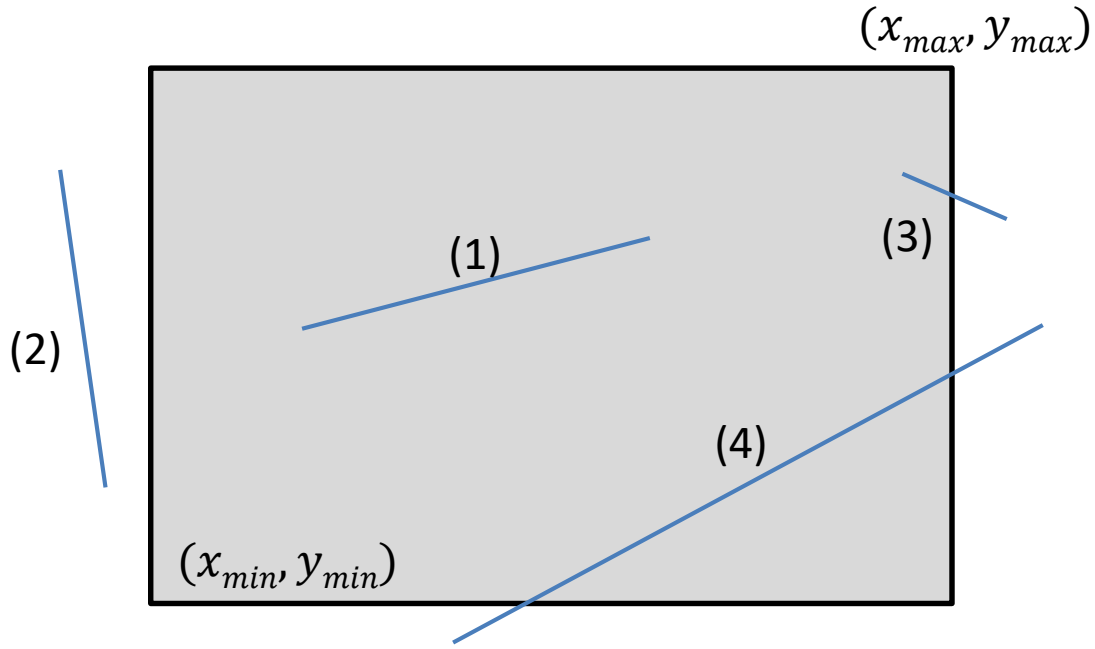
Es müssen folgende Bedingungen erfüllt sein:

$$x_{min} \leq x \leq x_{max}$$
$$y_{min} \leq y \leq y_{max}$$



Clipping von Linien

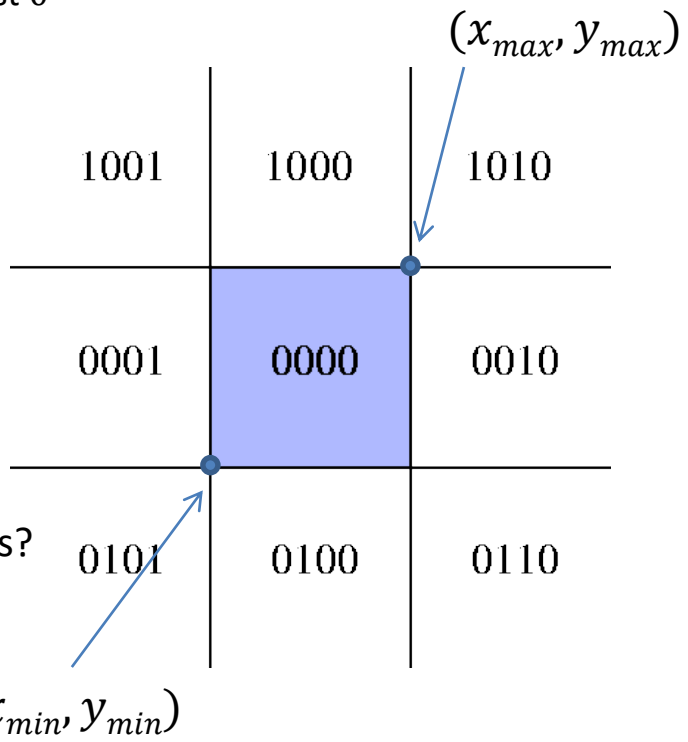
- 4 mögliche Lagebeziehungen von Linien zum Darstellungsbereich:
  (1) Linie befindet sich vollständig im Darstellungsbereich
  (2) Linie befindet sich vollständig außerhalb des Darstellungsbereichs
  (3) Ein Endpunkt innerhalb, ein Endpunkt außerhalb
  (4) Beide Endpunkte außerhalb, aber Linie schneidet den Darstellungsbereich



Clipping von Linien

- Algorithmus von Cohen und Sutherland:
  - Prinzip der Bereichsprüfung: Einteilung des Darstellungsbereichs in 9 Teile
  - Zuordnung eines Bereichscodes zu jedem Anfangs- und Endvertex
  - Bereichscodes aus binärer Darstellung der Lagebeziehung der Bereiche
    - Erstes Bit (hinten): 1 = links von Darstellungsbereich, sonst 0
    - usw.

$y_{max} - y < 0$: viertes Bit = 1
$y - y_{min} < 0$: drittes Bit = 1
$x_{max} - x < 0$: zweites Bit = 1
$x - x_{min} < 0$: erstes Bit = 1

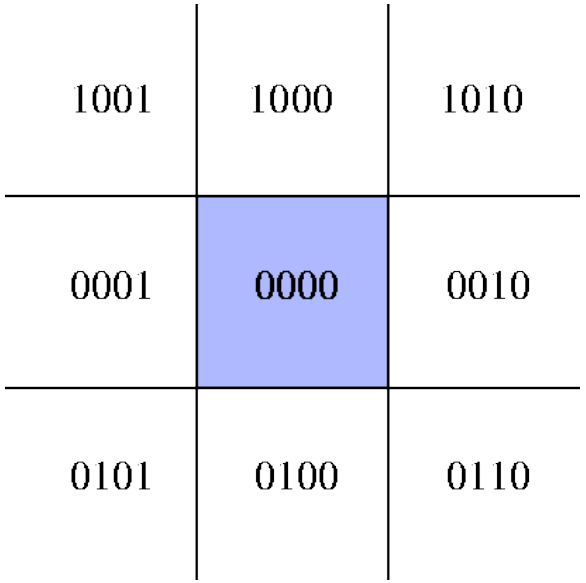


  - Beide Punkte innerhalb des Fensters?
    - Ja, wenn bitweise ODER-Verknüpfung von v_1 und $v_2 = 0$
  - Beide Punkte und gesamte Linie außerhalb des Fensters?
    - Ja, wenn bitweise UND-Verknüpfung von v_1 und v_2 an einer Stelle ungleich 0

http://www-lehre.inf.uos.de/~rkunze/flashweather/Diplomarbeit/node37.html

Clipping von Linien

- Ist keiner der beiden Tests erfolgreich, ist nicht auszuschließen dass die Linie das Fenster schneidet.
- In diesem Fall wird ein Schnitttest durchgeführt:
  - Berechnung des Schnittpunktes der Linie mit einer Seite des Fensters
  - Seite(n) kann (können) anhand von Bereichscode gewählt werden.
  - Für beide Teilsegmente der Linie wird erneut getestet ob Anfangs- oder Endpunkt außerhalb des Fensters liegen
  - Gegebenenfalls erneute Schnittpunktberechnung für Teilsegmente



Clipping von Linien

- Bestimmung des Schnittpunktes einer Linie mit dem Darstellungsbereich:
  - Parameterform (Punkt-Richtungsform) einer Geraden: $x = v_0 + r(v_1 - v_0)$
  - Eine Koordinate des Schnittpunktes ist durch den Algorithmus von Cohen und Sutherland i.d.R. bekannt, z.B. $s_y = y_{max}$
  - Gesucht wird in diesem Beispiel nun s_x.
  - Daraus ergibt sich folgendes Gleichungssystem:

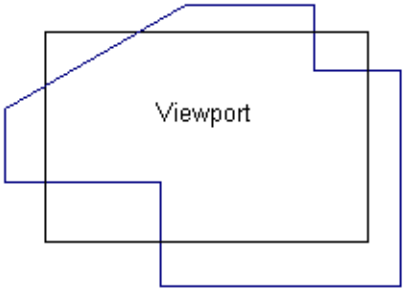
$$(1)\ s_x = v_{0,x} + r\left(v_{1,x} - v_{0,x}\right)$$
$$(2)\ s_y = v_{0,y} + r\left(v_{1,y} - v_{0,y}\right)$$

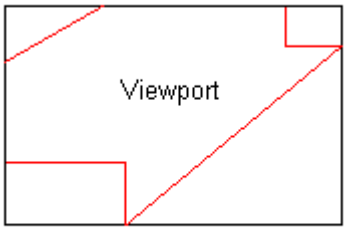
  - Aus (2) lässt sich nun r berechnen, da z.B. s_y bekannt ist.
  - Das Ergebnis wird in (1) eingesetzt und nach s_x aufgelöst:

$$s_x = v_{0,x} + \frac{s_y - v_{0,y}}{v_{1,y} - v_{0,y}}\left(v_{1,x} - v_{0,x}\right)$$

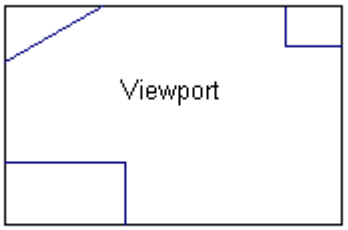
Clipping von Polygonen

- Linien-Clipping-Verfahren kann bei Anwendung auf Polygone zu falschen Ergebnissen führen: Topologie eines Polygons wird u.U. zerstört.





falsch



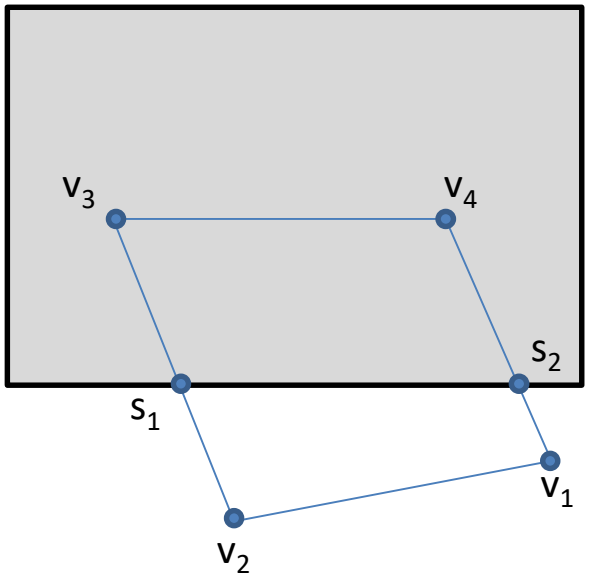
richtig

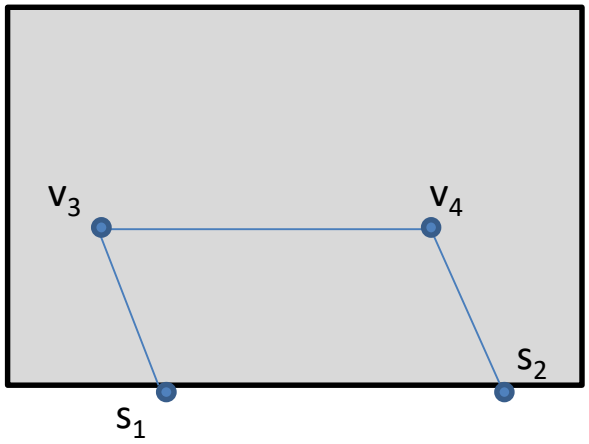
- Algorithmus von Sutherland und Hodgman:
  - Interpretation der Kanten des Darstellungsbereichs als Geraden ohne Begrenzung
  - Schrittweise Schneiden aller Kanten des Polygons an jeweils einer Geraden des Darstellungsbereichs

  - Entscheidung welche Vertices in Ausgabepolygon übernommen werden:
    (1) Beide Vertices der Kante außerhalb: keinen Vertex in Ausgabepolygon übernehmen
    (2) Gerichtete Kante von v_1 zu v_2 von außen nach innen: Schnittpunkt und v_2 übernehmen
    (3) Beide Vertices der Kante innerhalb: beide Vertices in das Ausgabepolygon übernehmen
    (4) Gerichtete Kante von v_1 zu v_2 von innen nach außen: Schnittpunkt und v_1 übernehmen

http://www.schulportal.bremerhaven.de/wypior/GraphDV/GrafDVTeil2.html

Clipping von Polygonen





Kante $v_1 \rightarrow v_2$: Beide außerhalb. Nicht in Ausgabepolygon übernehmen

Kante $v_2 \rightarrow v_3$: Gerichtete Kante von außen nach innen: s_1 und v_3 einfügen

Kante $v_3 \rightarrow v_4$: Beide innerhalb. v_4 einfügen (v_3 wurde schon)

Kante $v_4 \rightarrow v_1$: Gerichtete Kante von innen nach außen: s_2 einfügen (v_4 wurde schon)

5.2. ZEICHNEN VON LINIEN

Brute Force

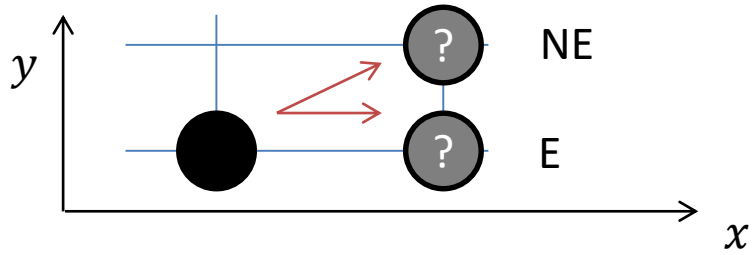
- Naiver Algorithmus basierend auf Steigungsform:
  - Berechnung der Steigung: $m = \frac{\Delta y}{\Delta x} = \frac{v_{2,y} - v_{1,y}}{v_{2,x} - v_{1,x}}$
  - Starte von links: $y_i = mx_i + B$
  - Zeichne den Pixel $(x_i, round(y_i))$
  - Inkrementelles Ermitteln des nächsten Pixels:

$$\begin{aligned} y_{i+1} &= mx_{i+1} + B \\ &= m(x_i + \Delta x) + B \\ &= m\Delta x + mx_i + B \\ &= m\Delta x + y_i \end{aligned}$$

- Nachteile: Speicherung als Gleitkommazahl + aufwendige Rundung

Midpoint Line Algorithmus

- = Bresenham Algorithmus, nach Jack Bresenham, 1965
- Grundidee: Nutzung der Scanline. Aufbau des Bildes von links nach rechts
- Vorteil: Beschränkung auf ausschließlich Ganzzahl-Arithmetik möglich
- Reduktion des Problems auf Linien, deren Steigungswinkel zwischen $0°$ und $45°$ liegt.
  - Alle anderen Linien lassen sich aus Symmetrieüberlegungen auf diesen Fall zurückführen.
- Ablauf:
  - X-Koordinate wird schrittweise um 1 erhöht
  - Für zugehörige y-Koordinate wird festgestellt, ob sie gleich bleibt (E) oder um 1 erhöht wird (NE)



Midpoint Line Algorithmus

Herleitung der y-Wert-Entscheidung: E oder NE?

- Konventionen
  - Zeichnen einer Linie von (x_0, y_0) zu (x_1, y_1)
  - (x_p, y_p) sei ein bereits ausgewählter Pixel auf dieser Linie
- Steigungsform einer Linie:

$$y = \frac{\Delta y}{\Delta x} x + B \quad \text{mit} \quad \Delta y = y_1 - y_0, \Delta x = x_1 - x_0$$

- Umgeschrieben in implizite Formulierung:

$$F(x, y) = \Delta y \cdot x - \Delta x \cdot y + B \cdot \Delta x$$

- $F(x, y)$ ist
  - $= 0$ für Punkte auf der Linie
  - > 0 für Punkte unterhalb der Linie
  - < 0 für Punkte oberhalb der Linie

Midpoint Line Algorithmus

- Berechnet wird nun der Funktionswert von Punkt M (Midpoint) als

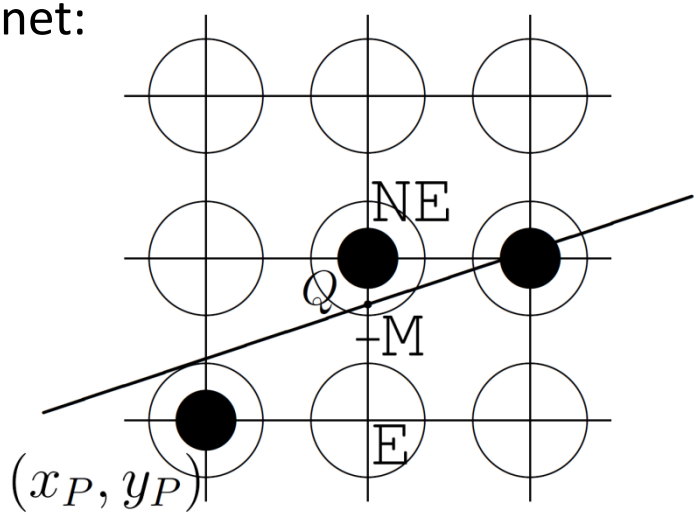
$$F(M) = F\left(x_p + 1, y_p + \tfrac{1}{2}\right) = d$$

- Das Vorzeichen der Entscheidungsvariablen d entscheidet nun darüber ob der y-Wert inkrementiert wird oder gleich bleibt:
  - $d > 0$: wähle NE *(entspr.: M liegt unterhalb der Geraden)*
  - $d \leq 0$: wähle E *(entspr.: M liegt oberhalb der Geraden)*
- Die neue Entscheidungsvariable d_{new} wird in Abhängigkeit von der Wahl NE oder E inkrementell aus der alten berechnet:
  - Wenn E gewählt wurde:

$$d_{new} = F\left(x_p + 2, y_p + \tfrac{1}{2}\right)$$
$$\Rightarrow\ d_{new} = d_{old} + \Delta y$$

  - Wenn NE gewählt wurde:

$$d_{new} = F\left(x_p + 2, y_p + \tfrac{3}{2}\right)$$
$$\Rightarrow\ d_{new} = d_{old} + (\Delta y - \Delta x)$$



S. Krömker: Computergrafik I, Universität Heidelberg

Midpoint Line Algorithmus

- Bei der Initialisierung wird der anfängliche Wert der Entscheidungsvariablen festgelegt:
  - Funktionswert bei (x_0, y_0)

$$d_{ini} = F\left(x_0 + 1, y_0 + \tfrac{1}{2}\right) = \underbrace{F(x_0, y_0)}_{= 0} + \Delta y - \tfrac{\Delta x}{2} = \Delta y - \tfrac{\Delta x}{2}$$

- Somit ergibt sich folgender Pseudocode:

```
// init
dx = x1-x0;
dy = y1-y0;
incrE = dy*2;
incrNE = (dy-dx)*2;
d = dy*2 - dx;
x = x0;
y = y0;
```

Inkremente entsprechend der Wahl E oder NE

→ Hier multipliziert mit 2 um den Bruch $(y + ½)$ zu sparen

```
writePixel(x,y);
while(x < x1) {
   if(d <= 0) {
      d += IncrE;     // choose E
      x++;
   } else {
      d += IncrNE;    // choose NE
      x++;
      y++;
   }
   writePixel(x,y)
}
```