

# Computergrafik

Universität Osnabrück, Henning Wenke, 2012-06-05

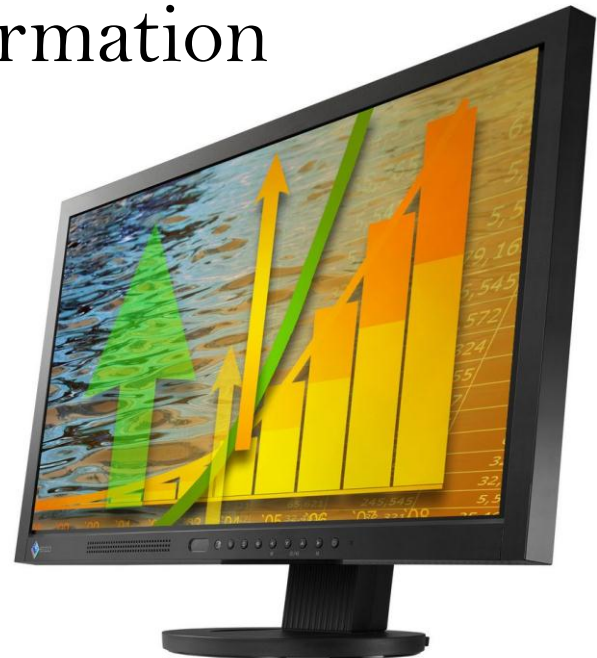
# Noch Kapitel VIII:

## **Per Primitive Operations**

## 8.6

---

# Viewport Transformation



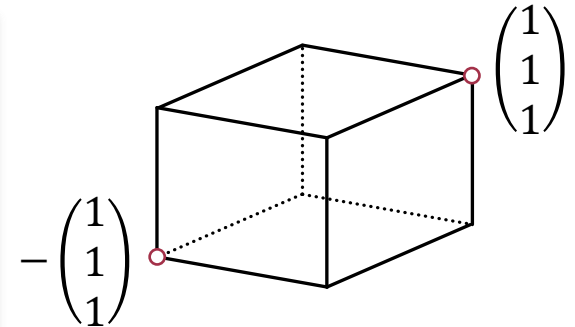
# Mapping auf Ausgabegerät

IN

n  
Vertices

## Normalized Device Coordinates (LH)

x	$-1 \leq x \leq 1$
y	$-1 \leq y \leq 1$
z	$-1 \leq z \leq 1$
w	<i>n. A.</i>

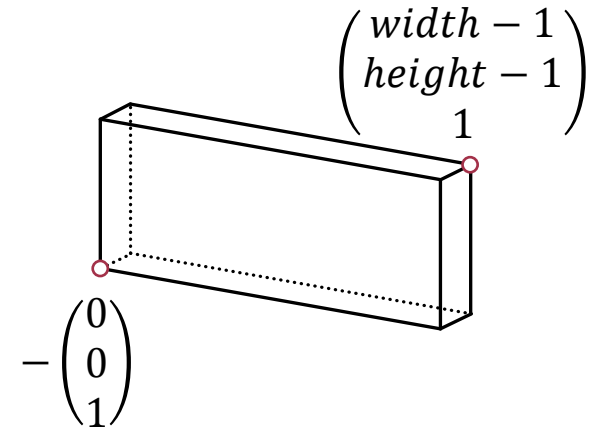


OUT

n  
Vertices

## Window Coordinates (LH)

x	$0 \leq x \leq \text{width} - 1$
y	$0 \leq y \leq \text{height} - 1$
z	$-1 \leq z \leq 1$
w	<i>n. A.</i>



width	Fensterbreite in Pixeln
height	Fensterhöhe in Pixeln

Hinweis: Es kann auch in Teile des Fensters gerendert werden.

# Koordinatentransformation

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} \Rightarrow \text{Viewport Mapping} \Rightarrow \begin{pmatrix} (1+x_{ndc}) \cdot \left(\frac{width-1}{2}\right) \\ (1+y_{ndc}) \cdot \left(\frac{height-1}{2}\right) \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_w \\ y_w \\ z_w \end{pmatrix}$$

➤ Mögl. Viewport Transform Matrix  $M_{VP} = S\left(\frac{width-1}{2}, \frac{height-1}{2}, 1\right) \cdot T(1,1,0)$

➤ U.a. zum Clippen andere KS nötig  $\Rightarrow$  Zusammenfassen mit anderen Matrizen unmöglich  $\Rightarrow$  Matrixdarstellung unsinnig

➤ Keine homogenen Koordinaten mehr  $\Rightarrow$  Matrixdarstellung unmöglich

# Viewport Transformation in OpenGL

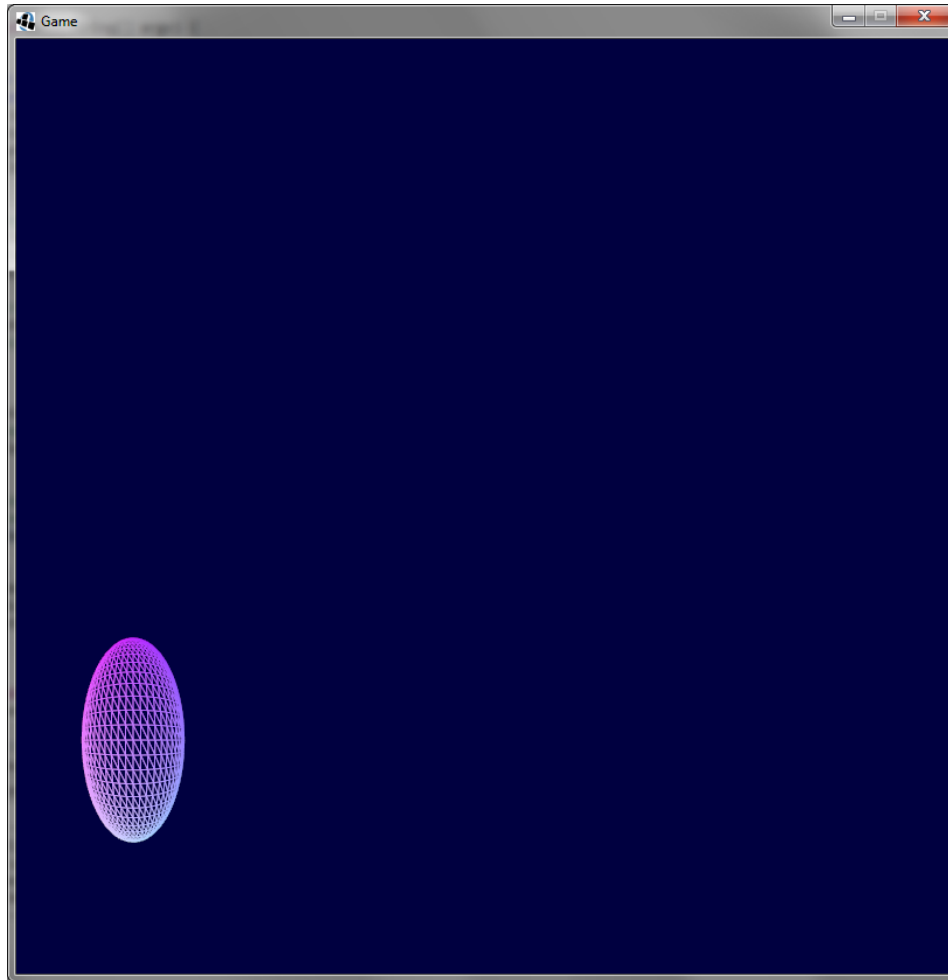
---

```
// Konfiguriert die Viewport Transformation so, dass in den
// angegebene Bereich des Ausgabegeräts gerendert wird
void glViewport(
    int x,          // Linke...
    int y,          // untere Ecke des Fensterausschnitts in Pixeln
    int width,      // Breite des Fensterausschnitts in Pixeln
    int height,     // Höhe des Fensterausschnitts in Pixeln
);
```

```
// Wir rendern immer in das ganze Fenster
glViewport(
    0,              // Linke...
    0,              // untere Ecke des Fensters. Bei uns immer (0, 0)
    1920,           // Breite unseres Fensters
    1200,           // Höhe unseres Fensters
);
```

# Vorführung: Viewport Transformation

---



8.7

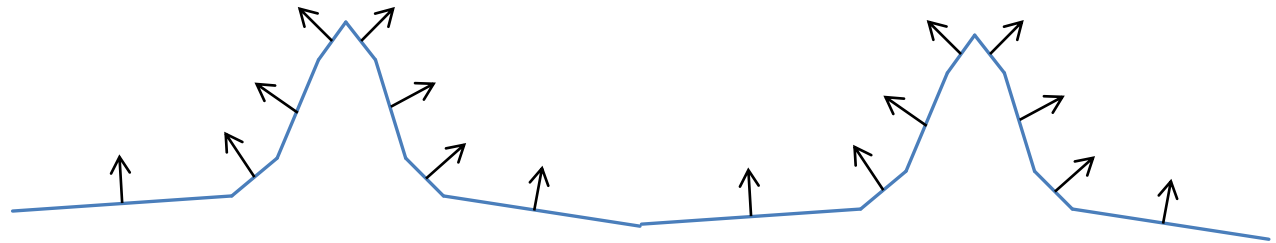
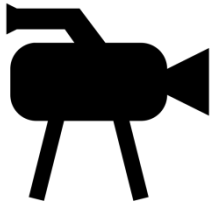
---

## Backface Culling



# Backface-Culling

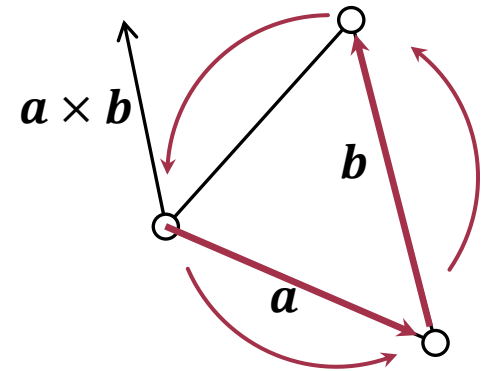
- Wird auf Polygone angewandt
- Ziel: Blende vom Betrachter abgewandte Flächen aus
- Vor Rasterizer und, in OpenGL, nach Clipping
- Bei Transparenz oft sinnvoll
  - Beispiel: Wellen



- Entlastet Pipeline-Stages ab Rasterizer
- Erhöht Geometrieverarbeitungsaufwand

# BFC mit Flächennormale

- Orientierung wird über Reihenfolge der Eckpunkte festgelegt
- Lege positiven Drehsinn fest
  - Wir wählen CCW (in Modelling Coordinates)
- Für jedes Dreieck:
  - Berechne Flächennormale
  - Zeigt zum Betrachter, bzw.
  - Ist z-Komponente  $< 0$ ? (in Window Coords)
    - Behalten
  - Sonst
    - Entfernen



# BFC in OpenGL

---

- Backface Culling (de-)aktivieren

```
glEnable (GL_CULL_FACE) ;  
glDisable (GL_CULL_FACE) ;
```

- Positiver Drehsinn

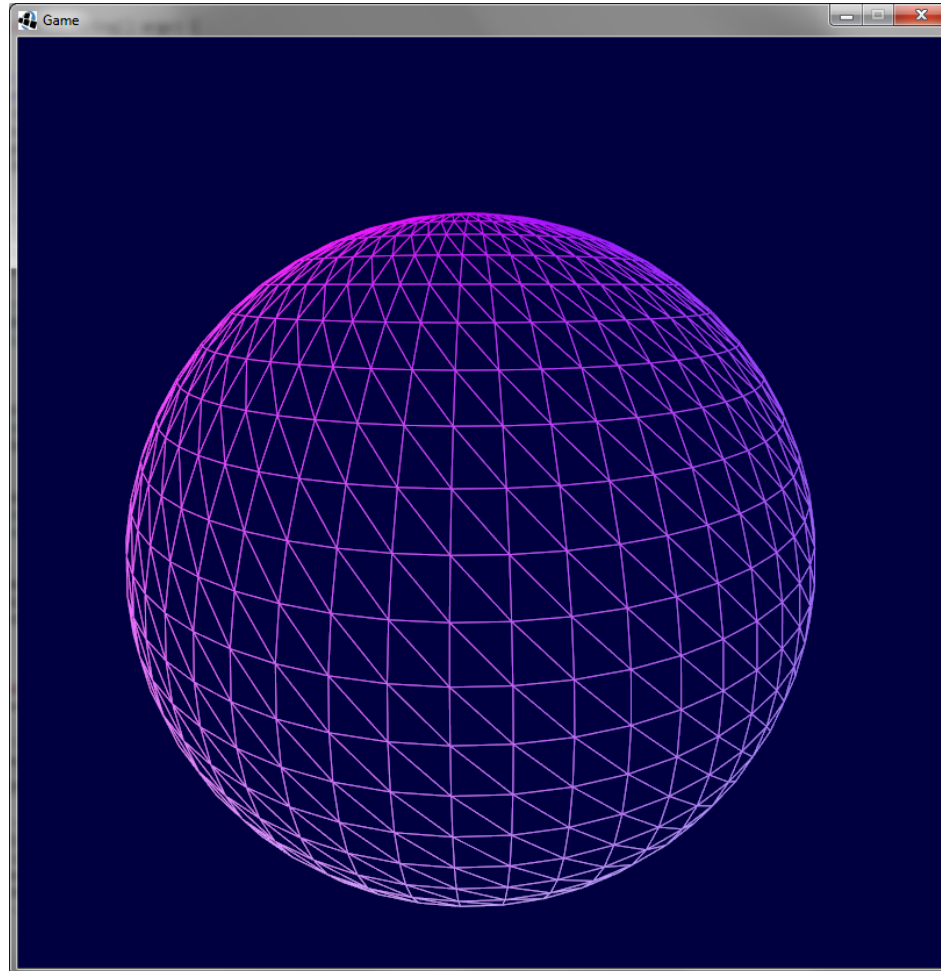
```
glFrontFace (GL_CCW) ;    // Gegen Urzeigersinn  
glFrontFace (GL_CW ) ;    // Im Urzeigersinn
```

- Zu entfernende Fläche

```
glCullFace (GL_BACK) ;           // Abgewandte Seite  
glCullFace (GL_FRONT) ;          // Zugewandte Seite  
glCullFace (GL_FRONT_AND_BACK) ; // Beide
```

# Vorführung: Culling

---



8.8

---

## Primitive Processing Stage

# Primitive Processing

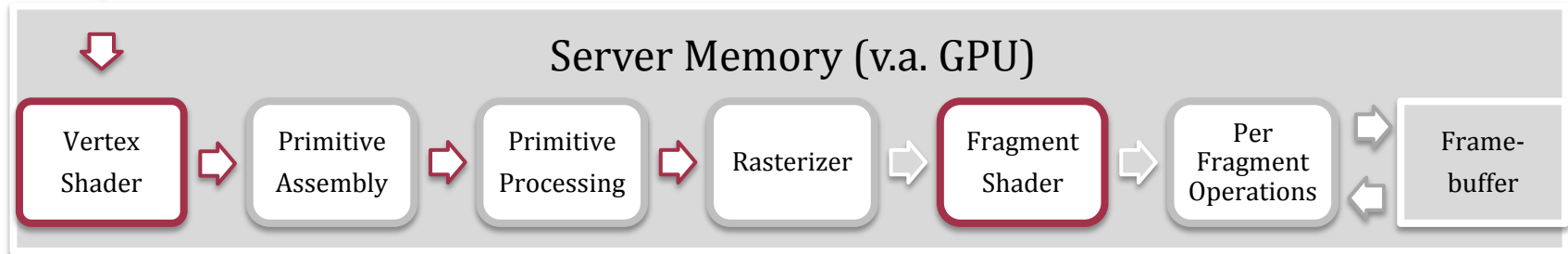
---

- Sammlung verschiedener fester aber (teilweise) konfigurierbarer Operationen, ...
- ...die Informationen über ganzes Primitive benötigen...
- ...und restliche Transformationen
- Reihenfolge:
  1. Clipping
  2. Perspective Division
  3. Viewport Transformation
  4. Culling

# OpenGL Graphics Pipeline

Client Memory (Unsere Java Applikation)

OpenGL Befehle





Legende


 Vertices

 Fragments

 Pixel Data

 Programmable Stage

 Fixed Stage

 Memory

## 8.9

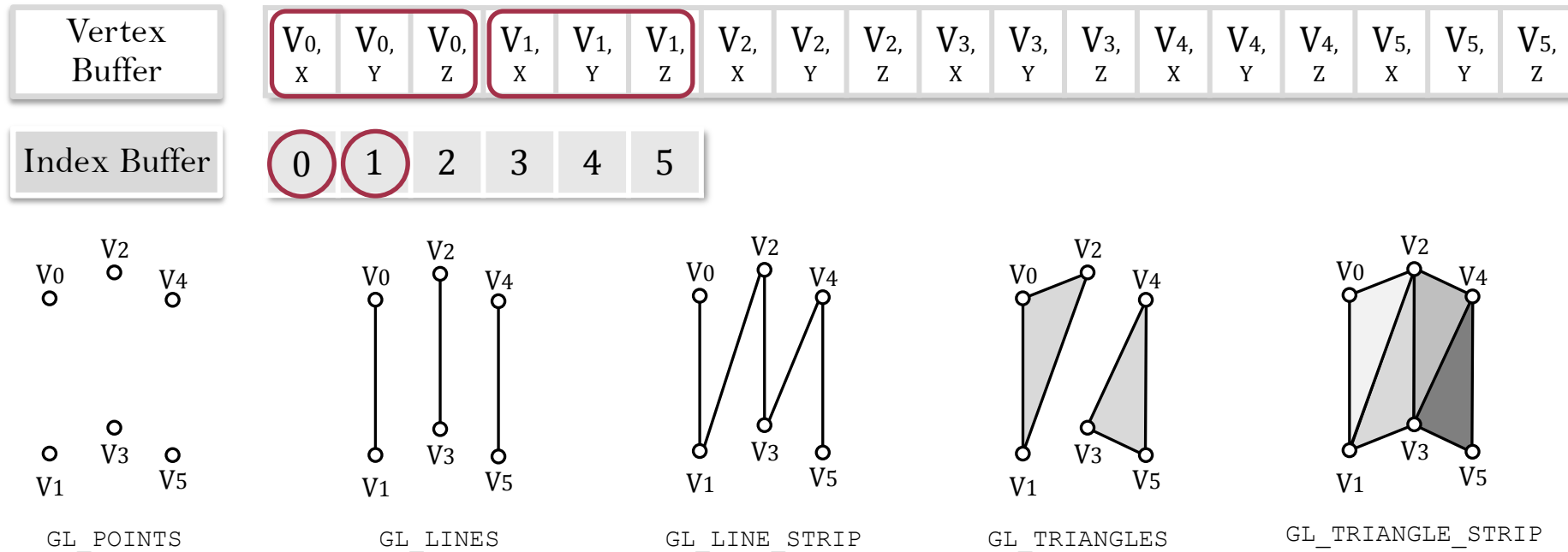
---

# Spezifikation der Primitives



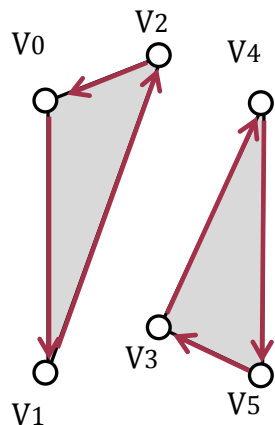
# Primitive Indizierung

- Verschiedene Indizierungstechniken für Folge von Vertices bzw. Indices möglich
- Gegeben:
  - VertexBuffer, enthält Daten jedes Vertex genau einmal. Hier: 6 Vertices
  - IndexBuffer legt Reihenfolge fest. Enthält Index eines Vertex oft mehrmals. Hier jedoch 6 Indices

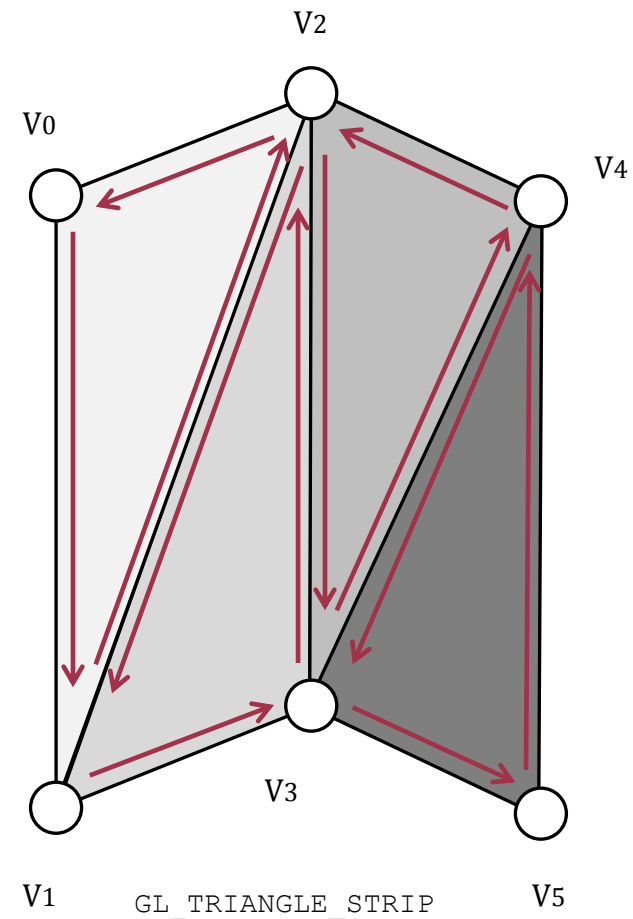


# Primitive Indizierung II

- Achtung: Bei Dreiecken Indexreihenfolge beachten
- GL\_TRIANGLES:
  - Erstes Dreieck:  $V_0, V_1, V_2$
  - Zweites:  $V_3, V_4, V_5$
  - ...
- GL\_TRIANGLE\_STRIP:
  - Erstes Dreieck:  $V_0, V_1, V_2$
  - Zweites:  $V_2, V_1, V_3$
  - Drittes:  $V_2, V_3, V_4$
  - ...



GL\_TRIANGLES



GL\_TRIANGLE\_STRIP

# Meshes I

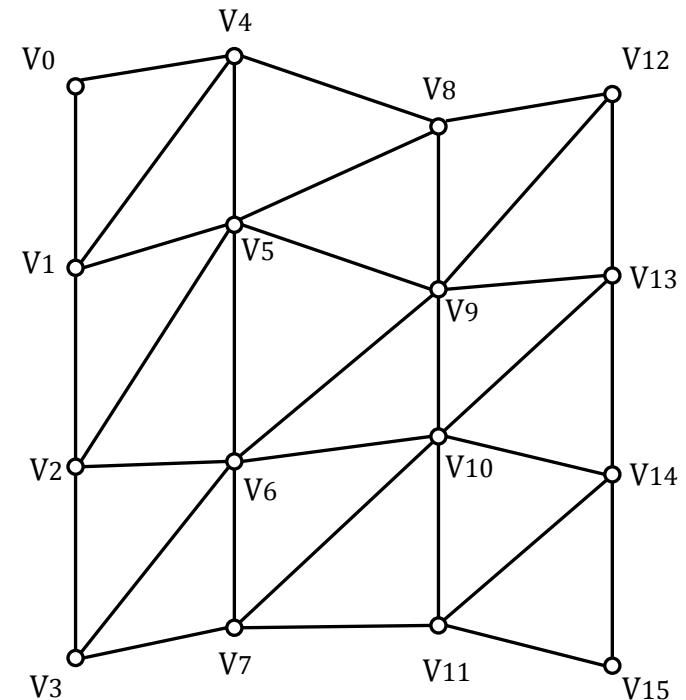
- Gegeben: 16 Vertices, angeordnet in 4 Zeilen und 4 Spalten
- Gesucht: Minimale Indexfolge welche diese in zusammenhängendes Dreiecknetz überführt
- Versuch 1: GL\_TRIANGLES Funktioniert, aber sehr viele Indices (54)

Vertex Buffer

V <sub>0</sub>	V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>	V <sub>6</sub>	V <sub>7</sub>	V <sub>8</sub>	V <sub>9</sub>	V <sub>10</sub>	V <sub>11</sub>	V <sub>12</sub>	V <sub>13</sub>	V <sub>14</sub>	V <sub>15</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

Index  
Buffer

0	1	4	4	1	5	8	4	5
8	5	9	12	8	9	12	9	13
1	2	5	5	2	6	9	5	6
9	6	10	13	9	10	13	10	14
2	3	6	6	3	7	10	6	7
10	7	11	14	10	11	14	11	15



# Meshes II

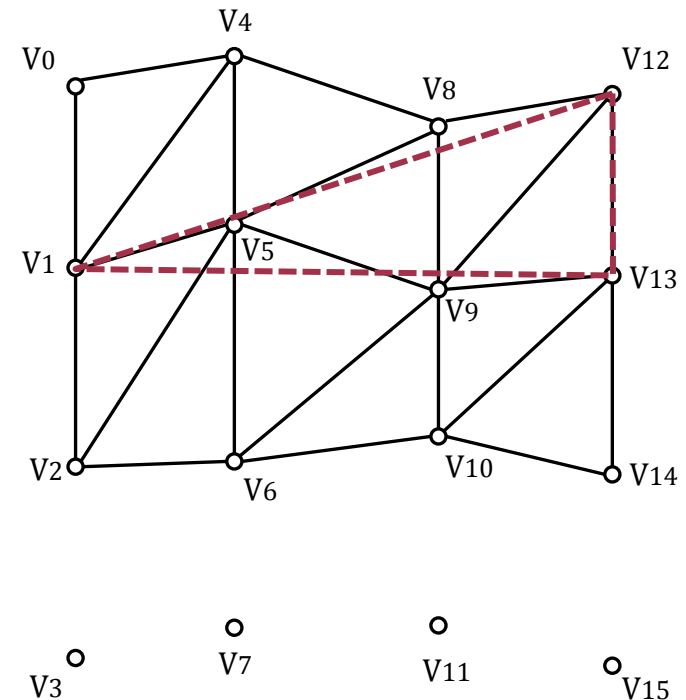
- Versuch 2: GL\_TRIANGLE\_STRIP
- Problem mit “Absetzen” beim Zeilenumbruch

Vertex Buffer

V<sub>0</sub> V<sub>1</sub> V<sub>2</sub> V<sub>3</sub> V<sub>4</sub> V<sub>5</sub> V<sub>6</sub> V<sub>7</sub> V<sub>8</sub> V<sub>9</sub> V<sub>10</sub> V<sub>11</sub> V<sub>12</sub> V<sub>13</sub> V<sub>14</sub> V<sub>15</sub>

Index Array

0 1 4 5 8 9 12 13 1 2 5 6 9 10 13 14



# Meshes III

- Versuch 3: GL\_TRIANGLE\_STRIP + Primitive Restart Index
- Beginnt neue Line- oder Triangle Folge
- Funktioniert mit weniger Indices (29 statt 54)

Vertex Buffer

V<sub>0</sub>

V<sub>1</sub>

V<sub>2</sub>

V<sub>3</sub>

V<sub>4</sub>

V<sub>5</sub>

V<sub>6</sub>

V<sub>7</sub>

V<sub>8</sub>

V<sub>9</sub>

V<sub>10</sub>

V<sub>11</sub>

V<sub>12</sub>

V<sub>13</sub>

V<sub>14</sub>

V<sub>15</sub>

```
glEnable(GL_PRIMITIVE_RESTART); // aktiviere PR
int RESTART_INDEX = -1;        // Definiere PR Index

// Setzt Primitive Restart Index auf -1.
// Anschließend bewirkt ein Eintrag -1
// im IndexBuffer ein Absetzen / Neustarten
// der aktuellen Primitiv-Kette
glPrimitiveRestartIndex(RESTART_INDEX);
```

Index Array

0

1

4

5

8

9

12

13

-1

1

2

5

6

9

10

13

14

-1

2

3

6

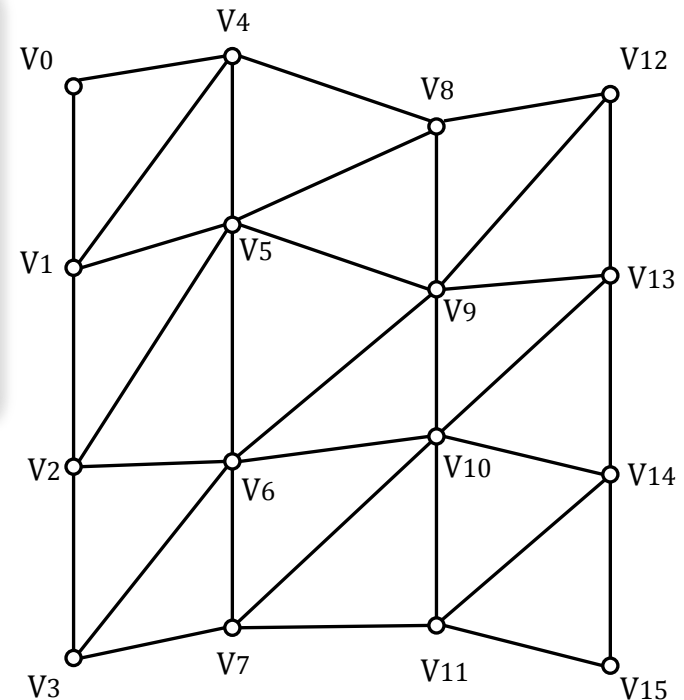
7

10

11

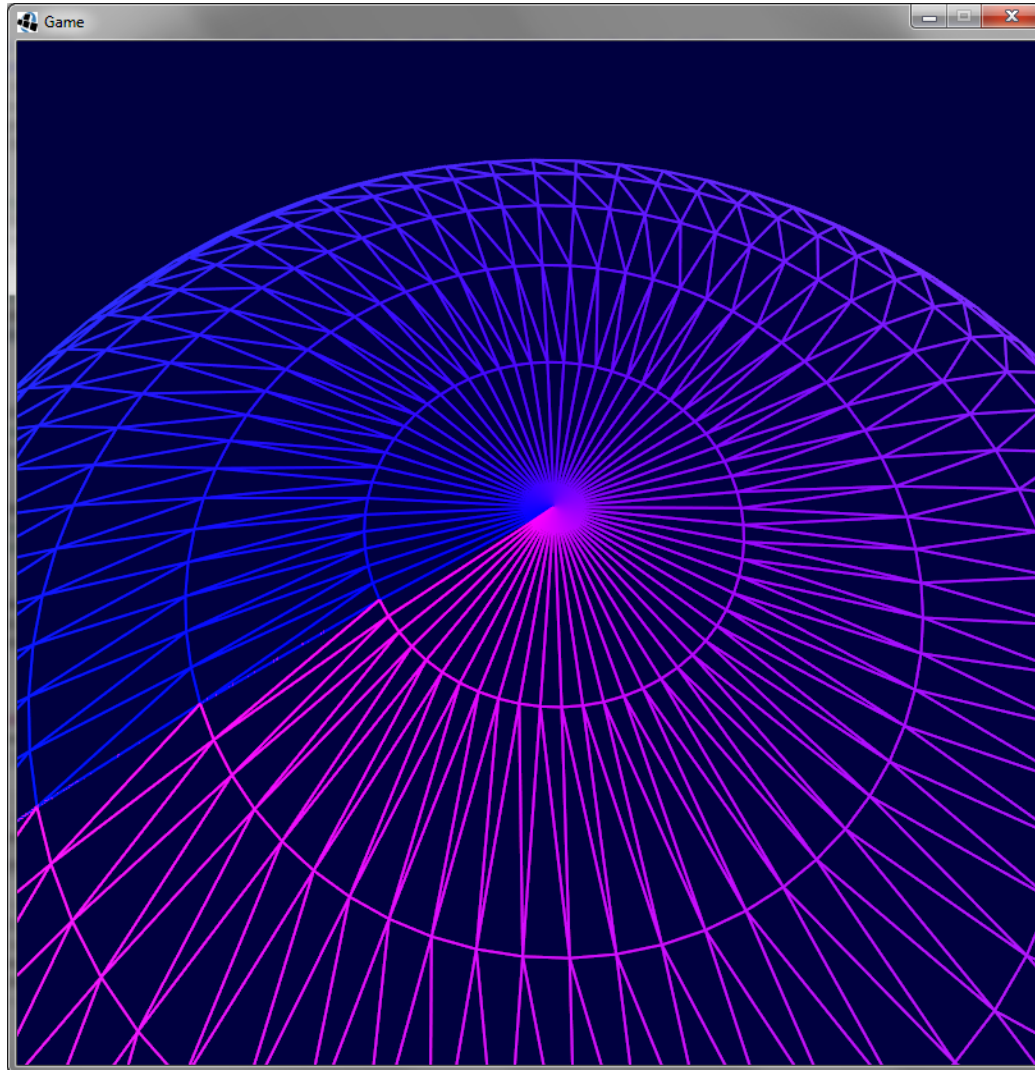
14

15



# Vorführung: Kugeltopologie

---



# Index Buffer Object

// Erinnerung: Weist einem Binding Point ein Buffer Object zu.

```
void glBindBuffer(  
    int target,        // Binding Point, etwa  
                        //    GL_ARRAY_BUFFER für Vertexdaten oder  
                        //    GL_ELEMENT_ARRAY_BUFFER für Indexdaten  
    int buffer         // id des anzubindenden Buffer Objects  
);
```

// Beispiel: Erzeugen eines Index Buffer Objects "iBO"

```
int vao = glGenVertexArrays();  
int iBO = glGenBuffer(); // Erzeuge Buffer Object  
glBindVertexArray(vao);
```

// Verwende iBO als Index Buffer Object.

// Dadurch sind dessen Daten die aktuell zu verwendenden

// Topologischen Informationen.

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, iBO);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, <Indexdaten>, GL_STATIC_DRAW);
```

8.10

---

Zeichnen der Primitives



# Draw

```
// Löst das Zeichnen aus
void glDrawElements(
    int mode,                                // Art der Primitive Indizierungstechnik, etwa
                                            // GL_POINTS, GL_LINES, GL_LINE_STRIP
                                            // GL_TRIANGLES, GL_TRIANGLE_STRIP, ...
    int indices_count,                       // Anzahl der Indices
    int type,                               // Datentyp
    int indices_buffer_offset,              // Offset des ersten Index in byte
);
```

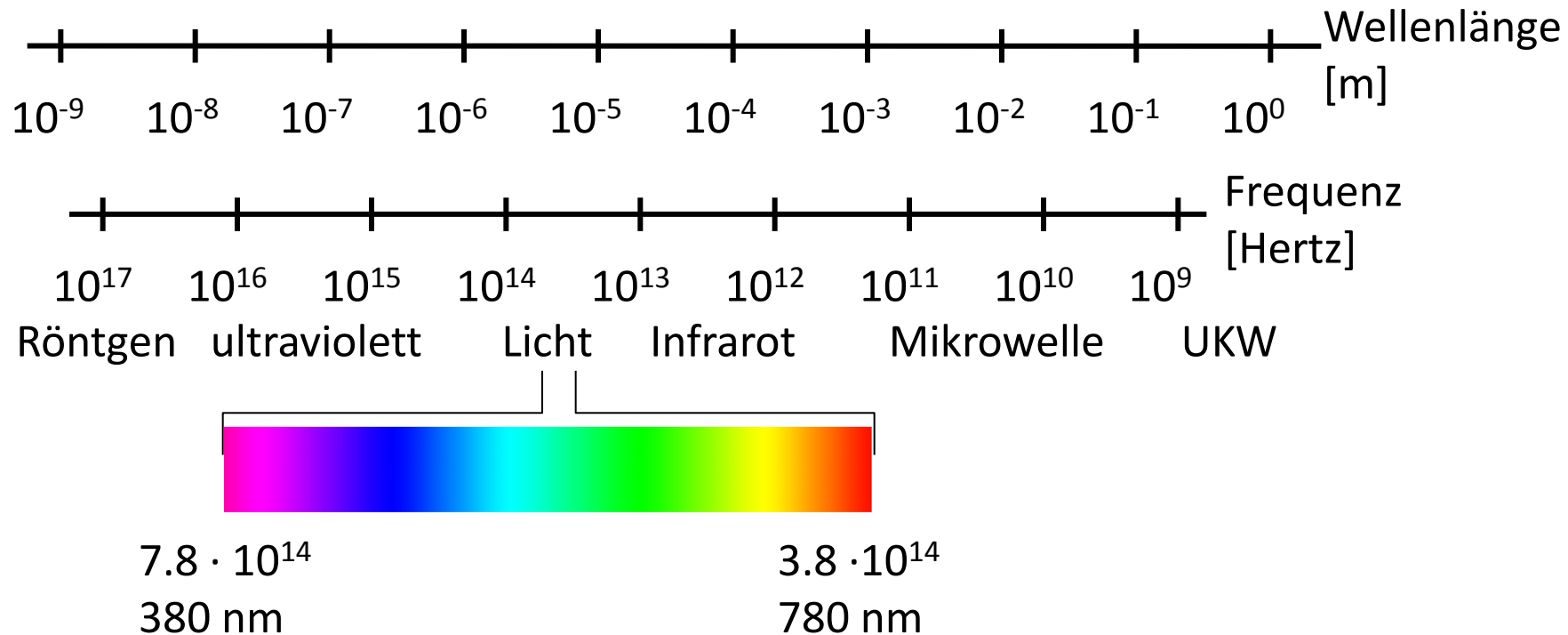
```
// Voraussetzungen:
// 1) aktiviertes Vertex Array Object mit gekapseltem State
//     des / der Vertex Buffer Objects
//     des / der Vertex Arrays
//     des Index Buffer Objects (Folie 23)
// 2) Aktives Program Object
// 3) in / uniform Daten übergeben (nicht zwingend)
// 4) Hier: Primitive Restart aktiviert (Folie 21)
```

```
// Zeichne unser Mesh von Folie 21
glDrawElements(GL_TRIANGLE_STRIP, 29, GL_UNSIGNED_INT, 0);
```

# Kapitel IX:

## Farbe

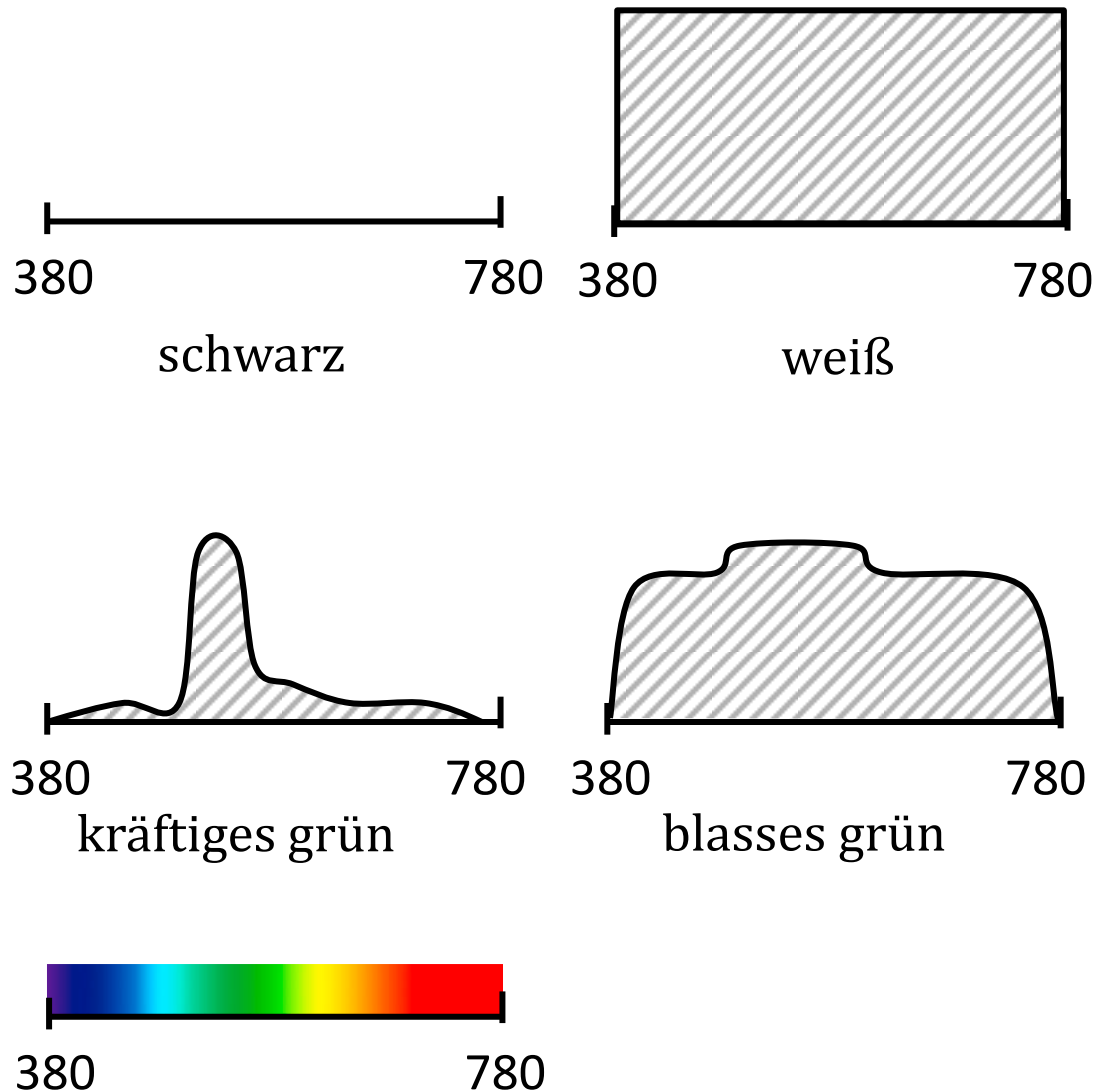
# Elektromagnetische Schwingungen



$$\text{Wellenlänge} \cdot \text{Frequenz} = \text{Lichtgeschwindigkeit} = 2.998 \cdot 10^8 \text{ m/s}$$

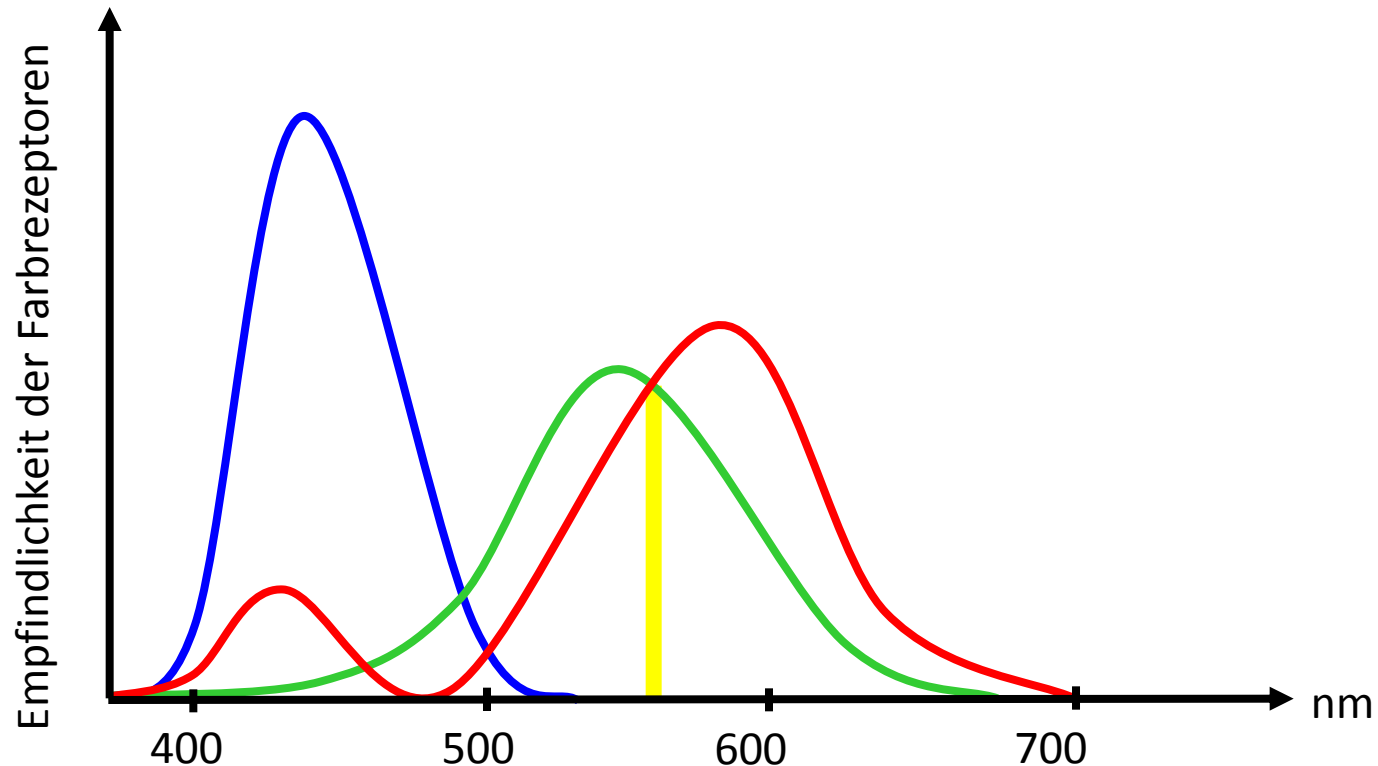
# Spektrum

- Spektralfarben haben genau eine Wellenlänge bzw. Frequenz
- natürliches Licht enthält Mix von Frequenzen
- Verteilung von Frequenzen heißt Spektrum



# Menschliches Sehen

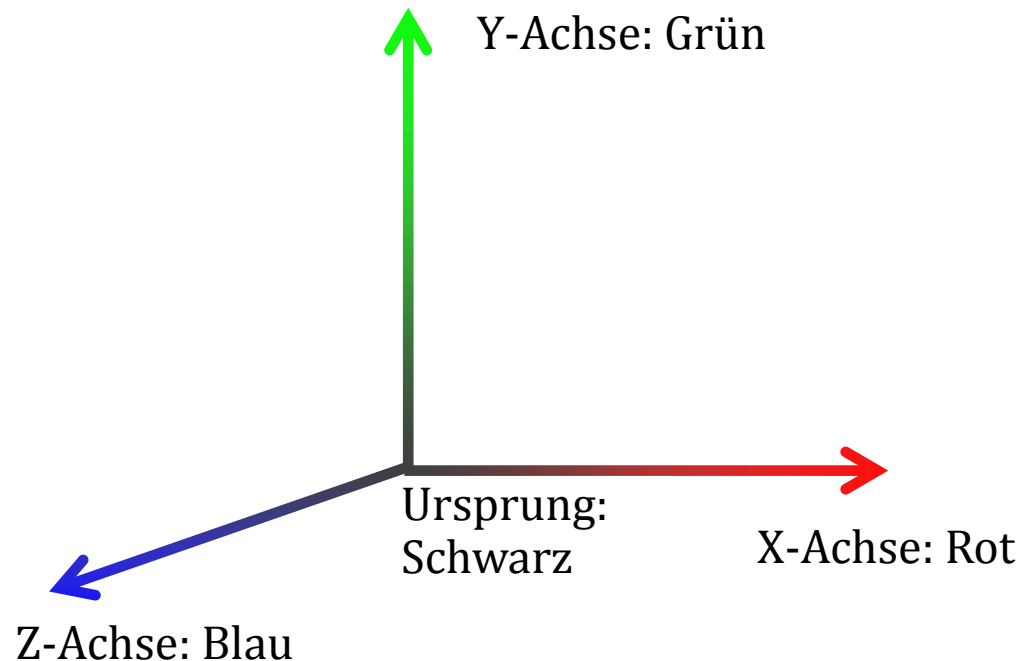
	Typ	Anzahl	Schwelle
S/W:	Stäbchen	125.000.000	1 Photon
Farbe:	Zäpfchen	5.000.000	100 Photonen



# RGB Modell

---


- Idee: Stelle jede Farbe als Linearkombination von Rot, Grün und Blau dar (Additives Farbmodell)
- Dann ist jede Farbe ein Punkt im RGB-Koordinatensystem
- Passend für “lichtemittierende Ausgabegeräte” wie Monitore




# Mischen im RGB-Modell

---


  $(1,0,0)$  Rot

  $(0,1,0)$  Grün


---

  $(1,1,0)$


  $(0,1,0)$  Grün

  $(0,0,1)$  Blau


---

  $(0,1,1)$

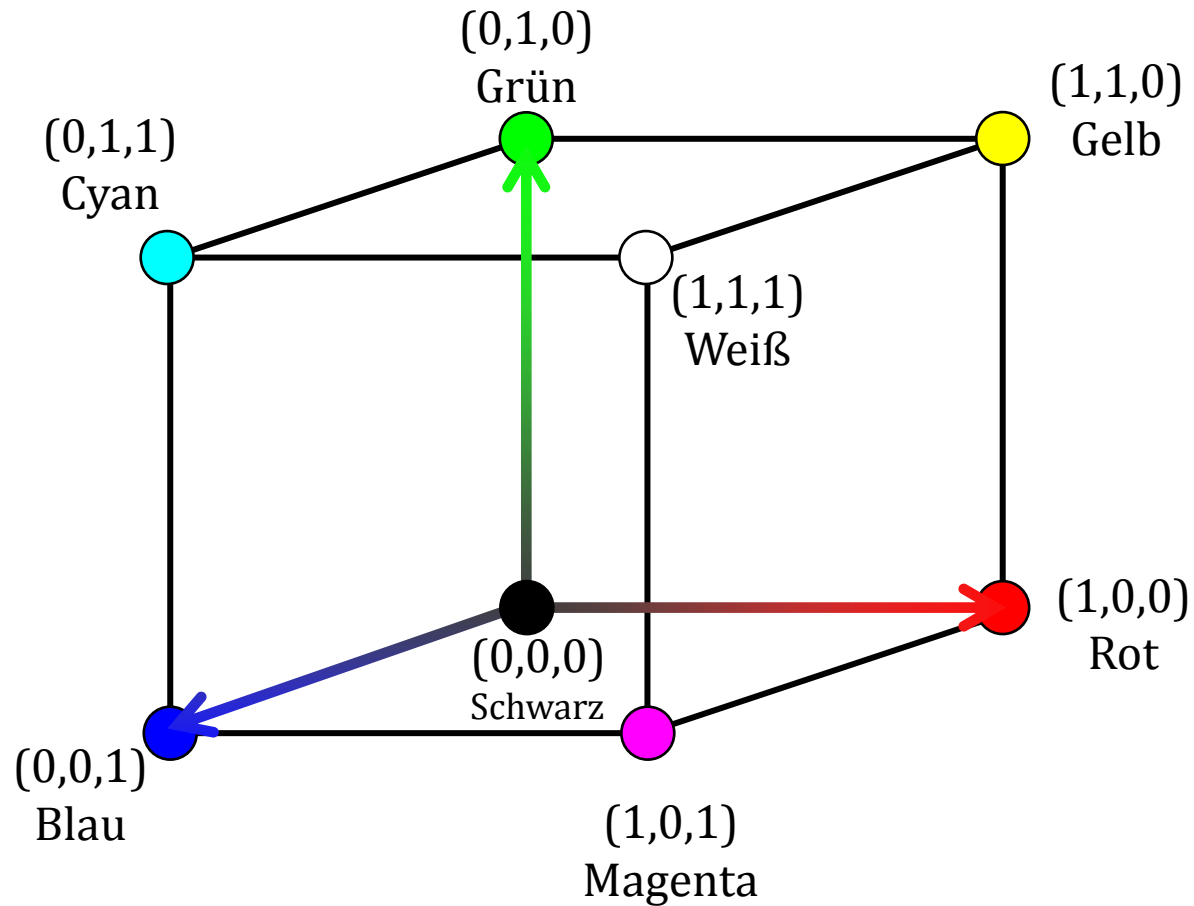
  $(1,0,0)$  Rot

  $(0,0,1)$  Blau

---

  $(1,0,1)$

# RGB Würfel

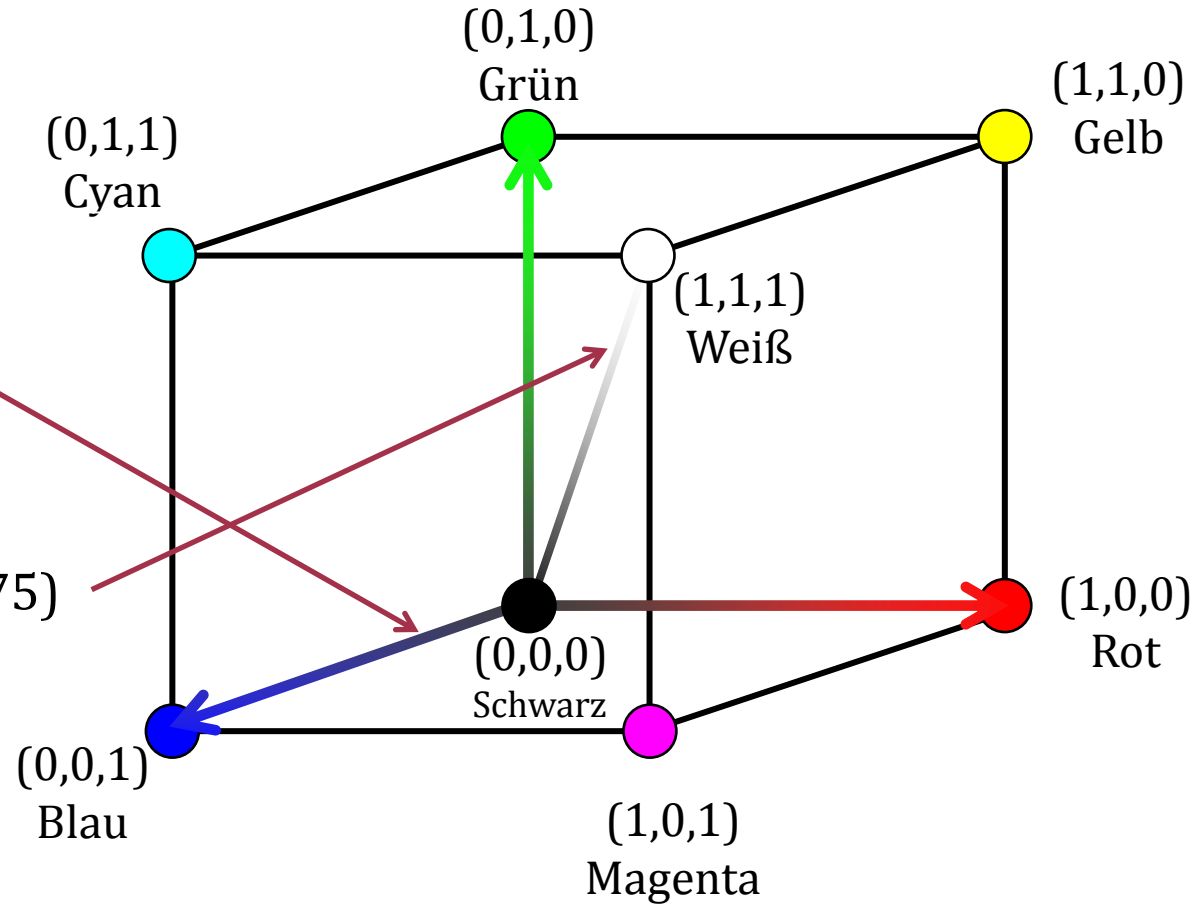




# RGB Würfel II

➤ Dunkelblau?  
• Z.B. RGB(0, 0, 0.25)

➤ Hellgrau?  
• Z.B. RGB(0.75, 0.75, 0.75)



# Vorführung: Vertexfarbe

---

