

Computergrafik

Übung 0 – JavaScript Einführung

Steffen Hinderink / Ingmar Ludwig

9. / 11. April 2024

Organisatorisches

- Ablauf der Woche i :

Montag	Dienstag	Donnerstag	Sonntag
Ausgabe Blatt i bis 12 Uhr	1. Übung i (Steffen) 14 – 16 Uhr c.t.	2. Übung i (Ingmar) 10 – 12 Uhr c.t.	späteste Abgabe Blatt i bis 24 Uhr
	Nachbesprechung Blatt $i-1$ Vorbesprechung Blatt i	Gleicher Inhalt wie 1. Übung i	

- Abgabe in 2er Gruppen über Vips
- 10 Blätter mit je 10 Theorie und 10 Praxis Punkten
- Prüfungszulassung bei 50% erreichten Punkten

JavaScript

Beispiel: Sudoku Löser von C++ nach JavaScript

7				5			9	
						6		
				2	9		5	
						7		
					5			
	7	3			8			9
			8				6	
2		8	9		3	4		5
6				7	4	8		

- JavaScript in den Übungen immer in einer HTML-Datei
 - Einbindung von Canvas, Slidern, etc.
 - Plattformunabhängig in fast jedem Browser offenbar
- Spracheigenschaften:
 - Ausgaben über die Konsole mit `console.log()`
 - Schwache Typisierung, Deklarationen mit `let`
 - Arrays mit `[]`, Länge mit `.length`
 - Vergleiche mit `==` und `!=`
 - Call by value, aber Wert bei Arrays und Objekten ist Referenz
 - Sonstige Syntax ähnlich wie in C++

Computergrafik

Übung 1 – 2D Transformationen

Steffen Hinderink / Ingmar Ludwig

16. / 18. April 2024

Lineare Transformationen

- Darstellbar durch Matrizen
 - Skalierung um Faktor s :

$$S_s = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

- Rotation um Winkel α :

$$R_\alpha = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix}$$

(- Scherung, Projektion, ...)

- Hintereinanderausführung durch Matrixmultiplikation
- Inverse Transformation durch inverse Matrix

Affine Transformationen

- Lineare Transformation und Translation: $M \cdot x + t$
- Auch darstellbar durch Matrizen mit erweiterten Koordinaten

■ Punkt $\begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix}$

■ Vektor $\begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix}$

Skalierung um Faktor s

$$S_s = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Rotation um Winkel α

$$R_\alpha = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Translation um Vektor t

$$T_t = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Vorbesprechung Blatt 1

- Matrixmultiplikation ist nicht kommutativ
→ Reihenfolge beachten

Theorie

- Mehrere Möglichkeiten
→ Möglichst einfachen Punkt auf $(0, 0)^T$ schieben

Praxis

- Objekterzeugung mit `new`
- Matrixmultiplikation ist bereits implementiert
- Rotationswinkel sind in Grad gegeben

Computergrafik

Übung 2 – 3D Transformationen

Steffen Hinderink / Ingmar Ludwig

23. / 25. April 2024

Nachbesprechung Blatt 1

a) $T_2 SRT_1 = \begin{pmatrix} 1 & 0 & 4 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -8 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$
 $= \begin{pmatrix} 0 & -2 & 12 \\ 2 & 0 & -9 \\ 0 & 0 & 1 \end{pmatrix} = M$

b) $M \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}$ $M \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \\ 0 \end{pmatrix}$ $M \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 12 \\ -9 \\ 1 \end{pmatrix}$

Nachbesprechung Blatt 1

```
a) let points = [];  
let n = 41;  
for (let i = 0; i < n; i++) {  
    let x = (i - 20) * 10;  
    let y = Math.sin(x * Math.PI / 200) * 100;  
    let point = new Point(x, y);  
    points.push(point);  
}  
return points;
```

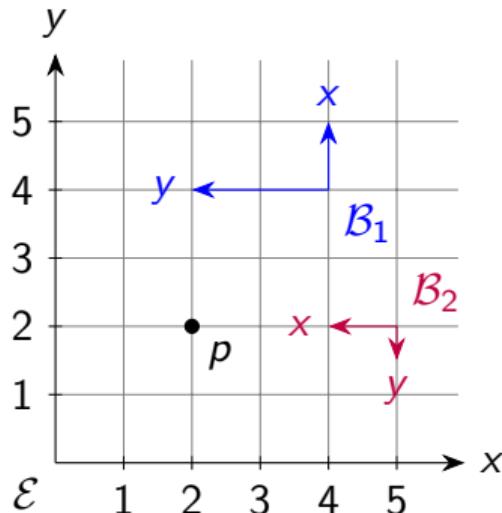
Beliebige Punkte möglich

Nachbesprechung Blatt 1

b)

```
let transMat = new Matrix(
    1, 0, transX,
    0, 1, transY,
    0, 0, 1
);
let scaleMat = new Matrix(
    scaleX, 0, 0,
    0, scaleY, 0,
    0, 0, 1
);
let rotRadians = rot * Math.PI / 180;
let rotMat = new Matrix(
    Math.cos(rotRadians), -Math.sin(rotRadians), 0,
    Math.sin(rotRadians), Math.cos(rotRadians), 0,
    0, 0, 1
);
return rotMat.mulMat(scaleMat).mulMat(transMat);
```

Koordinatensystemwechsel



- $p_{\mathcal{B}_1} = \begin{pmatrix} -2 \\ 1 \\ 1 \end{pmatrix}$, $p_{\mathcal{E}} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$, $p_{\mathcal{B}_2} = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}$
- $M_{\mathcal{B}_1 \rightarrow \mathcal{B}_2} = M_{\mathcal{B}_2 \rightarrow \mathcal{E}}^{-1} \cdot M_{\mathcal{B}_1 \rightarrow \mathcal{E}}$
$$= \begin{pmatrix} -1 & 0 & 5 \\ 0 & -0.5 & 2 \\ 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 0 & -2 & 4 \\ 1 & 0 & 4 \\ 0 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & 2 & 1 \\ -2 & 0 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$
- Probe: $M_{\mathcal{B}_1 \rightarrow \mathcal{B}_2} \cdot p_{\mathcal{B}_1} = p_{\mathcal{B}_2}$ ✓

Vorbesprechung Blatt 2

Theorie: Projektion, LookAt, Kreuz- und Skalarprodukt

- Perspektivische Projektion

$$P_{\text{std}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Danach Dehomogenisieren: Alle Koordinaten durch w teilen

- LookAt ist nichts anderes als eine Basiswechselmatrix

$$M_{\text{LookAt}} = M_{\mathcal{E} \rightarrow \mathcal{B}_{\text{Kamera}}} = \begin{pmatrix} r_x & r_y & r_z & -r^T c \\ u_x & u_y & u_z & -u^T c \\ -d_x & -d_y & -d_z & d^T c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Mit Kameraposition c , Blickrichtung d , right-Vektor r und up-Vektor u

- Kreuz- und Skalarprodukt für Längen und Winkel

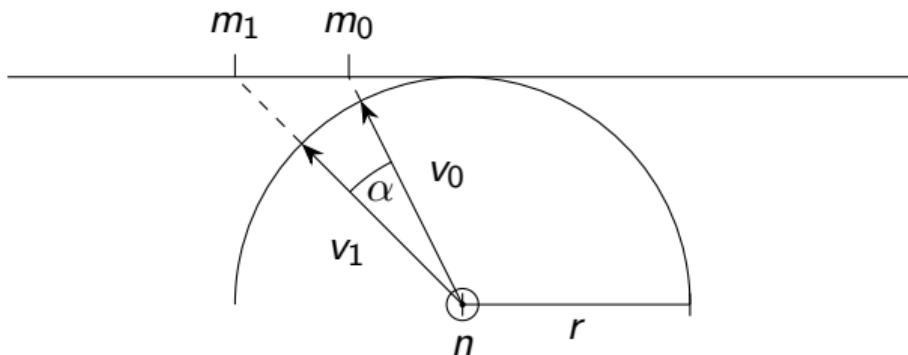
Vorbesprechung Blatt 2

Praxis: 3D Rotation, LookAt und Projektion

- Ähnlich wie Blatt 1 nur jetzt in 3D
- Punktzeugung
- Rotation um die Hauptachsen, keine Translation oder Skalierung
- LookAt von der z-Achse (entspricht Translation)
- Perspektivische Projektion und Dehomogenisierung
- Bonus: Virtueller Trackball . . .

Virtueller Trackball

- Bei jeder Mausbewegung von m_0 nach m_1 wird eine Rotationsmatrix R hinzugefügt



$$v_i = \begin{pmatrix} m_{ix} \\ m_{iy} \\ r \end{pmatrix}$$

$$n = \frac{v_0 \times v_1}{\| \cdot \|}$$

$$\alpha = \arccos(v_0^T v_1) = \arcsin \|v_0 \times v_1\|$$

- Allgemeine Rotationsmatrix mit Rotationsachse n und Winkel α
$$R = \cos \alpha I + (1 - \cos \alpha) nn^T + \sin \alpha X_n$$

Computergrafik

Übung 3 – Rasterisierung

Steffen Hinderink / Ingmar Ludwig

30. April / 2. Mai 2024

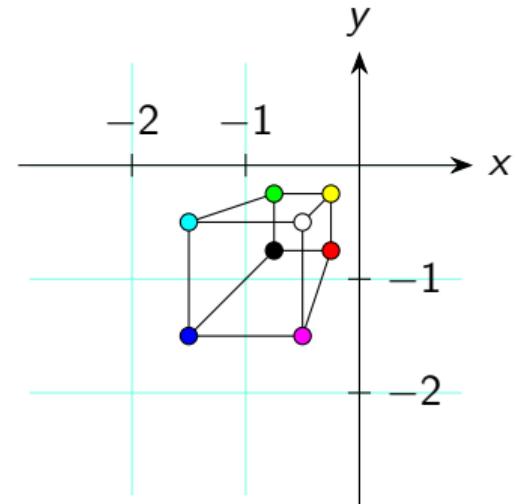
Nachbesprechung Blatt 2

a) $x \in \{-24, -8\}, y \in \{-24, -8\}, z \in \{-32, -16\}$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

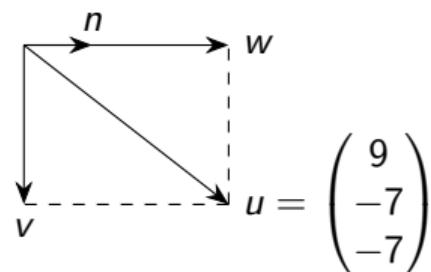
b) $r = d \times u$

$$M_{\text{LookAt}} = \begin{pmatrix} -0.6 & 0 & -0.8 & 3 \\ 0 & 1 & 0 & -2 \\ 0.8 & 0 & -0.6 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Nachbesprechung Blatt 2

- c)
- $\|v\| = \sqrt{v^T v} = 21$
 - $\angle vw = \arccos \left(\frac{v^T w}{\|v\| \|w\|} \right) = \frac{3}{4}\pi = 135^\circ$
 - $u = v \times w = \begin{pmatrix} 40 \\ -10 \\ -30 \end{pmatrix}$
 - w zum Beispiel als Projektion von p auf $n = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$:
 $w = n \cdot n^T u = \begin{pmatrix} 9 \\ 0 \\ 0 \end{pmatrix}$ $v = u - w = \begin{pmatrix} 0 \\ -7 \\ -7 \end{pmatrix}$



Nachbesprechung Blatt 2

```
a) let points = [];

function line(a, b) {
    for (let alpha = 0; alpha <= 1; alpha += 0.1) {
        let beta = 1 - alpha;
        points.push(new Point(
            alpha * a.x + beta * b.x,
            alpha * a.y + beta * b.y,
            alpha * a.z + beta * b.z
        ));
    }
}

let a = new Point(1, 1, 1);
let b = new Point(1, -1, -1);
let c = new Point(-1, 1, -1);
let d = new Point(-1, -1, 1);
line(a, b);
line(a, c);
line(a, d);
line(b, c);
line(b, d);
line(c, d);
return points;
```

Nachbesprechung Blatt 2

b)

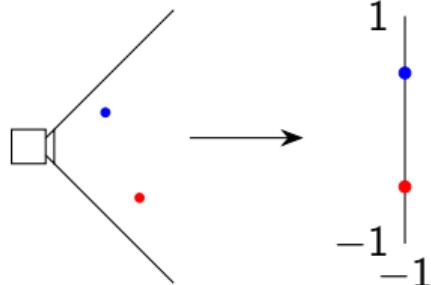
```
let rotXRad = rotX * Math.PI / 180;
let rotXMat = new Matrix(
    1, 0, 0, 0,
    0, Math.cos(rotXRad), -Math.sin(rotXRad), 0,
    0, Math.sin(rotXRad), Math.cos(rotXRad), 0,
    0, 0, 0, 1
);
let rotYRad = rotY * Math.PI / 180;
let rotYMat = new Matrix(
    Math.cos(rotYRad), 0, Math.sin(rotYRad), 0,
    0, 1, 0, 0,
    -Math.sin(rotYRad), 0, Math.cos(rotYRad), 0,
    0, 0, 0, 1
);
let rotZRad = rotZ * Math.PI / 180;
let rotZMat = new Matrix(
    Math.cos(rotZRad), -Math.sin(rotZRad), 0, 0,
    Math.sin(rotZRad), Math.cos(rotZRad), 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1
);
let lookAtMat = new Matrix(
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, -camZ,
    0, 0, 0, 1
);
let modelViewMat = lookAtMat.mulMat(rotZMat).mulMat(rotYMat).mulMat(rotXMat);
```

Nachbesprechung Blatt 2

```
c) let projectionMat = new Matrix(  
    250, 0, 0, 0,  
    0, 250, 0, 0,  
    0, 0, 1, 0,  
    0, 0, -1, 0  
);  
let mat = projectionMat.mulMat(modelViewMat);  
let res = [];  
for (let i = 0; i < points.length; i++) {  
    let p = mat.mulVec(points[i]);  
    p.dehomogen();  
    res.push(p);  
}  
return res;
```

Projektion

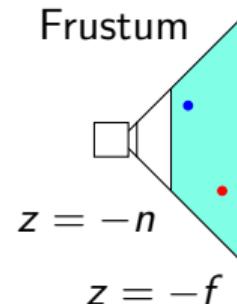
Projektion



$$P_{\text{std}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Tiefeninformation geht verloren

Frustum



$$M_{\text{Frustum}} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-f-n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Tiefeninformation bleibt bestehen

Bresenham-Algorithmus

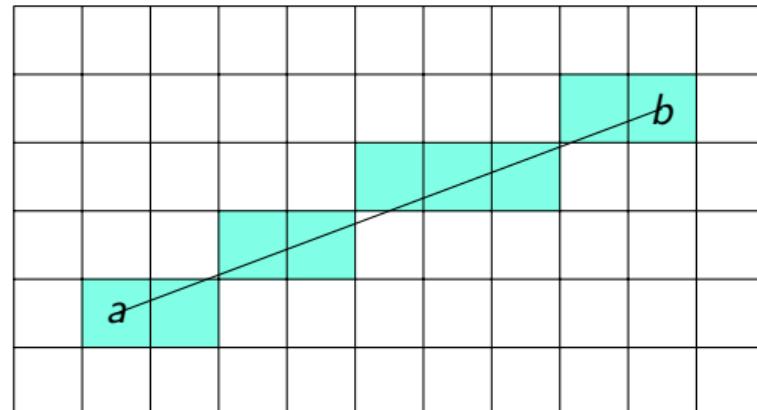
- Algorithmus zum Zeichnen von Linien von a nach b

- Voraussetzungen:

- $a_x, a_y, b_x, b_y \in \mathbb{Z}$
- $-1 \leq m = \frac{b_y - a_y}{b_x - a_x} \leq 1$
- $a_x < b_x$

- Effizienter Algorithmus:

```
y = ay;
for x = ax : bx do
    | setPixel(x, ⌊y⌋);
    | y += m;
end
```



- Antialiasing: Je zwei Pixel abhängig vom Linienabstand transparent einfärben oder Supersampling

Baryzentrische Koordinaten

- Punkt p als Linearkombination der Eckpunkte eines Dreiecks Δ_{abc}

$$p = \alpha a + \beta b + \gamma c$$

- $\alpha + \beta + \gamma = 1$ (Affinkombination),
damit p in der durch Δ_{abc} aufgespannten Ebene liegt
- Berechnung mit Flächeninhalten

$$\alpha = \frac{A(\Delta_{pbc})}{A(\Delta_{abc})} \quad \beta = \frac{A(\Delta_{pca})}{A(\Delta_{abc})} \quad \gamma = \frac{A(\Delta_{pab})}{A(\Delta_{abc})}$$

Achtung: Eckpunktreihefolge beachten!

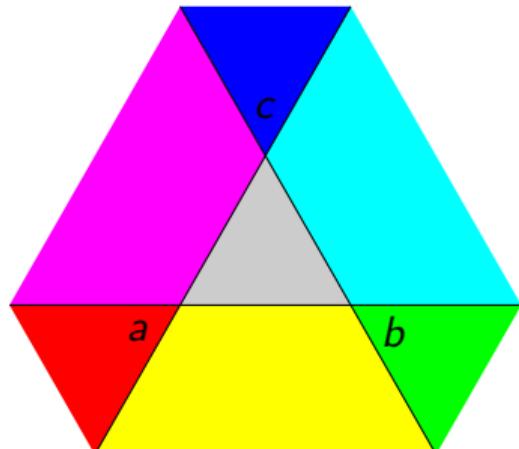
Eckpunkte gegen den Uhrzeigersinn \rightarrow Flächeninhalt positiv

Eckpunkte im Uhrzeigersinn \rightarrow Flächeninhalt negativ

- Flächeninhalt eines 2D Dreiecks Δ_{abc}

$$A(\Delta_{abc}) = \frac{1}{2} \det \begin{pmatrix} | & | \\ b-a & c-a \\ | & | \end{pmatrix}$$

Baryzentrische Koordinaten



- Zusammenhang der Vorzeichen der baryzentrischen Koordinaten und der Position des Punktes relativ zum Dreieck

α	β	γ	Bereich
+	+	+	grey
+	+	-	yellow
+	-	+	magenta
+	-	-	red
-	+	+	cyan
-	+	-	green
-	-	+	blue

Vorbesprechung Blatt 3

Theorie: Frustum, Linien, Dreiecke

- w und h aus Öffnungswinkeln berechnen
- Voraussetzungen beim Bresenham-Algorithmus prüfen

Praxis: Linien und Dreiecke

- Transformationen werden alle gestellt
- i.A. Endpunktkoordinaten $\notin \mathbb{Z}$, erst direkt vor `setPixel` runden beim Bresenham-Algorithmus möglich
- `image.data` enthält Pixel für Teiltransparenz beim Antialiasing

Computergrafik

Übung 4 – Beleuchtung

Steffen Hinderink / Ingmar Ludwig

7. Mai 2024

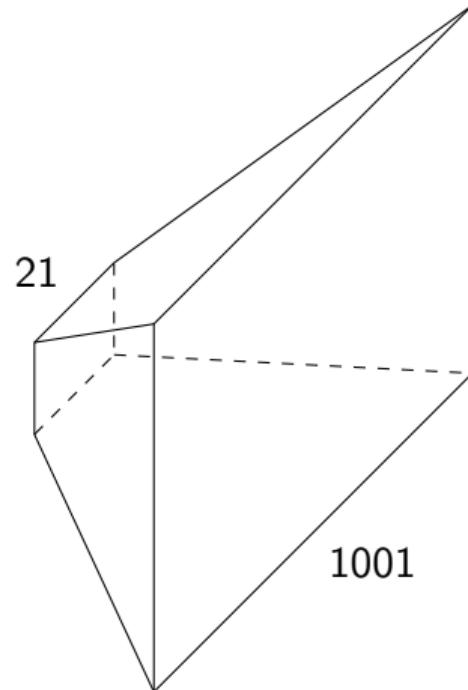
Nachbesprechung Blatt 3

a) 1. $M_{\text{Frustum}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \sqrt{3} & 0 & 0 \\ 0 & 0 & -\frac{1022}{980} & -\frac{42042}{980} \\ 0 & 0 & -1 & 0 \end{pmatrix}$

2. $p_{\text{near}} = \begin{pmatrix} \pm 21 \\ \pm 7\sqrt{3} \\ -21 \\ 1 \end{pmatrix} \quad p_{\text{far}} = \begin{pmatrix} \pm 1001 \\ \pm \frac{1001}{\sqrt{3}} \\ -1001 \\ 1 \end{pmatrix}$

3. $M_{\text{Frustum}} \cdot p_{\text{near}} = \begin{pmatrix} \pm 21 \\ \pm 21 \\ -21 \\ 21 \end{pmatrix} \stackrel{\wedge}{=} \begin{pmatrix} \pm 1 \\ \pm 1 \\ -1 \\ 1 \end{pmatrix}$

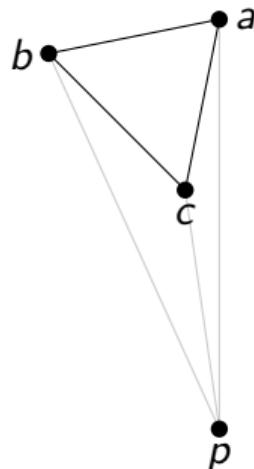
$M_{\text{Frustum}} \cdot p_{\text{far}} = \begin{pmatrix} \pm 1001 \\ \pm 1001 \\ -1001 \\ 1001 \end{pmatrix} \stackrel{\wedge}{=} \begin{pmatrix} \pm 1 \\ \pm 1 \\ 1 \\ 1 \end{pmatrix}$



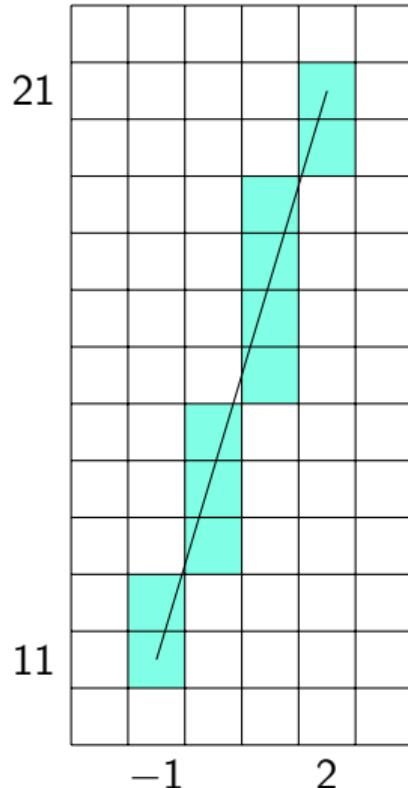
Nachbesprechung Blatt 3

b) $\begin{pmatrix} 8 \\ -2 \end{pmatrix} = \frac{2}{3} \begin{pmatrix} 15 \\ -8 \end{pmatrix} + \frac{1}{3} \begin{pmatrix} -6 \\ 10 \end{pmatrix}$

c) $\begin{pmatrix} 2 \\ -6 \end{pmatrix} = -1 \cdot \begin{pmatrix} 2 \\ 6 \end{pmatrix} - \frac{1}{2} \cdot \begin{pmatrix} -3 \\ 5 \end{pmatrix} + \frac{5}{2} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$



d)



Nachbesprechung Blatt 3

a)

```
let lines = [];
let a = new Point(100, 100, 100);
let b = new Point(100, -100, -100);
let c = new Point(-100, 100, -100);
let d = new Point(-100, -100, 100);
lines.push(new Line(a, b));
lines.push(new Line(a, c));
lines.push(new Line(a, d));
lines.push(new Line(b, c));
lines.push(new Line(b, d));
lines.push(new Line(c, d));
return lines;
```

```
let a = line.a;
let b = line.b;
let m = (b.y - a.y) / (b.x - a.x);
let swap = false;
if (m < -1 || m > 1) {
    let tmp = a.x;
    a.x = a.y;
    a.y = tmp;
    tmp = b.x;
    b.x = b.y;
    b.y = tmp;
    m = (b.y - a.y) / (b.x - a.x);
    swap = true;
}
if (a.x > b.x) {
    let tmp = a;
    a = b;
    b = tmp;
}
let t = a.y - m * a.x;
for (let x = Math.round(a.x); x <= Math.round(b.x); x++) {
    let y = Math.round(m * x + t);
    setPixel(swap ? y : x, swap ? x : y);
}
```

Nachbesprechung Blatt 3

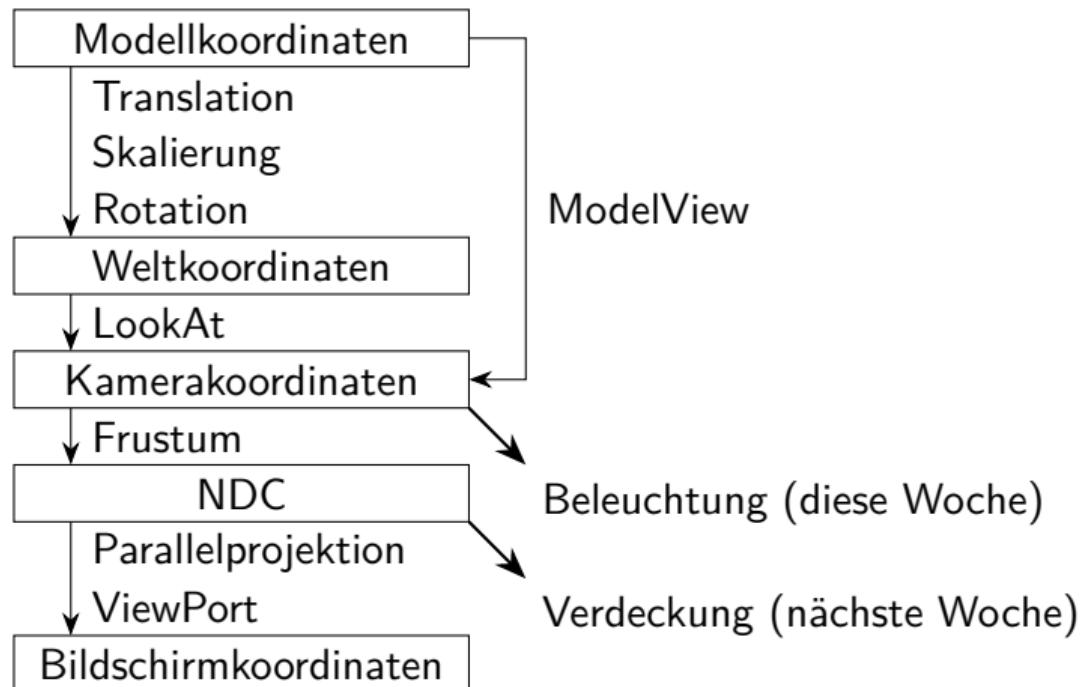
b)

```
let triangles = [];
let a = new Point(100, 100, 100);      let b = new Point(100, -100, -100);
let c = new Point(-100, 100, -100);    let d = new Point(-100, -100, 100);
triangles.push(new Triangle(a, b, c, [205, 255, 0]));
triangles.push(new Triangle(a, c, d, [0, 255, 204]));
triangles.push(new Triangle(a, d, b, [0, 204, 255]));
triangles.push(new Triangle(b, d, c, [204, 0, 255]));
return triangles;
```



```
function det(a, b, c, d) {
    return a * d - b * c;
}
function area(a, b, c) {
    return 0.5 * det(b.x - a.x, c.x - a.x, b.y - a.y, c.y - a.y);
}
let a = triangle.a;      let b = triangle.b;      let c = triangle.c;
let triangleArea = area(a, b, c);
for (let x = Math.round(Math.min(a.x, b.x, c.x)); x <= Math.round(Math.max(a.x, b.x, c.x)); x++) {
    for (let y = Math.round(Math.min(a.y, b.y, c.y)); y <= Math.round(Math.max(a.y, b.y, c.y)); y++) {
        let p = new Point(x, y);
        let alpha = area(p, b, c) / triangleArea;
        let beta = area(p, c, a) / triangleArea;
        let gamma = area(p, a, b) / triangleArea;
        if (alpha >= 0 && beta >= 0 && gamma >= 0) {
            setPixel(x, y, triangle.color);
        }
    }
}
```

Zusammenfassung Matrizen



Rendering Equation

$$L(x, \omega) = L_e(x, \omega) + \int_{\Omega} f(\omega', x, \omega) \cdot L(\text{ray}(x, \omega'), -\omega') \cdot \cos \theta d\omega'$$

- Punkt x
- Emittiertes Licht L_e
- Richtungen $\omega, \omega' \in \Omega$
- Bidirektionale Reflexionsverteilungsfunktion (BRDF) f
- Winkel θ zwischen ω' und Normale an x für Lambertsches Gesetz

Phong Lighting

$$L(x, v) = \sum_i \left(C_a + C_d \max(0, l_i^T n) + C_s \max \left(0, \begin{array}{l} \text{ohne Blinn: } v^T r_i \\ \text{mit Blinn: } h_i^T n \end{array} \right)^s \right) \cdot c_i$$

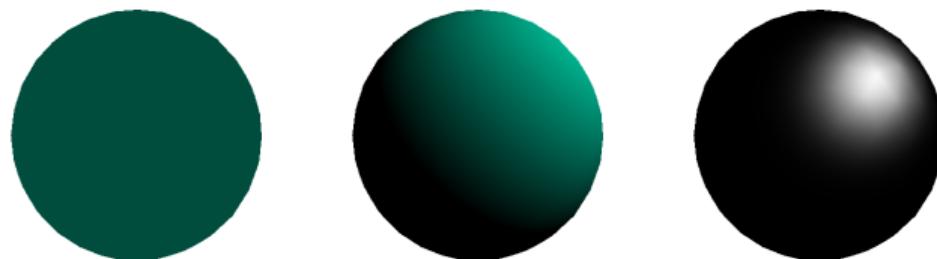
- Punkt x mit Normalen n
- Projektionszentrum eye
- View-Vektor $v = \frac{\text{eye}-x}{\|\text{eye}-x\|}$
- Lichtquellen an Positionen y_i mit Farben c_i
- Lichtvektoren $l_i = \frac{y_i-x}{\|y_i-x\|}$
- Reflexionsvektoren $r_i = 2l_i^T nn - l_i$
(ohne Blinn-Approximation)
- Halfway-Vektoren $h_i = \frac{v+l_i}{\|v+l_i\|}$
(mit Blinn-Approximation)

Phong Lighting

$$L(x, v) = \sum_i \left(C_a + C_d \max(0, l_i^T n) + C_s \max \left(0, \begin{array}{l} \text{ohne Blinn: } v^T r_i \\ \text{mit Blinn: } h_i^T n \end{array} \right)^s \right) \cdot c_i$$

Materialkonstanten:

- Ambient-Matrix C_a
- Diffuse-Matrix C_d
- Specular-Matrix C_s
- Shininess s



Shading

- Flat Shading:

Eine Farbe im Dreiecksmittelpunkt ausgewertet

$$L(x) = L\left(\frac{1}{3}(a + b + c)\right)$$

- Gouraud Shading:

An den Eckpunkten ausgewertete Farbe

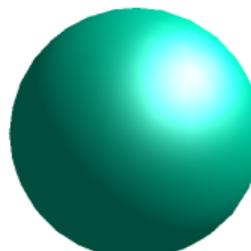
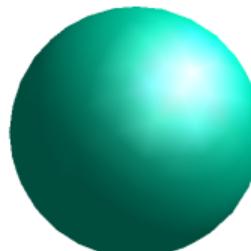
über das Dreieck interpoliert

$$L(x) = \alpha L(a) + \beta L(b) + \gamma L(c)$$

- Phong Shading:

Farbe an jedem Punkt des Dreiecks ausgewertet,

Interpolation der Normalen



Vorbesprechung Blatt 4

Theorie

- Grünes und blaues Licht auf ein graues Objekt
→ Ergebnis sollte türkis sein
- Runden ist erlaubt

Praxis

- Dreiecksdefinitionen für die Kugel nehmen viel Platz ein
- Normalen der Kugel lassen sich einfach bestimmen
- Bonus: Phong Shading; am besten eine Methode für Lighting schreiben, die von Flat und Phong Shading verwendet wird

Computergrafik

Übung 5 – Sichtbarkeit

Steffen Hinderink / Ingmar Ludwig

14. / 16. Mai 2024

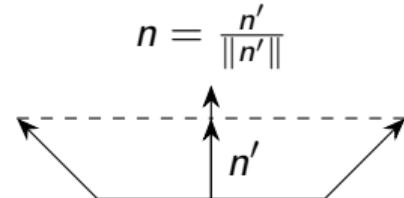
Nachbesprechung Blatt 4

$$a) \quad v = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad l_1 = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad l_2 = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad r_1 = \begin{pmatrix} 0 \\ -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \quad r_2 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$L(x, v) \approx \begin{pmatrix} 0 \\ 0.2418 \\ 0.2902 \end{pmatrix}$$

$$b) \quad h_1 \approx \begin{pmatrix} 0 \\ 0.3827 \\ 0.9239 \end{pmatrix} \quad h_2 \approx \begin{pmatrix} -0.3827 \\ 0 \\ 0.9239 \end{pmatrix} \quad L(x, v) \approx \begin{pmatrix} 0 \\ 0.4226 \\ 0.5072 \end{pmatrix}$$

$$c) \quad p = \begin{pmatrix} 1 \\ 4 \\ -6 \end{pmatrix} \quad n = \sqrt{\frac{13}{32}} \begin{pmatrix} -\frac{1}{8} \\ -\frac{3}{8} \\ \frac{1}{2} \end{pmatrix} \approx \begin{pmatrix} -0.1961 \\ -0.5883 \\ 0.7845 \end{pmatrix}$$



Nachbesprechung Blatt 4

a)

```
let a = triangle.a;
let b = triangle.b;
let c = triangle.c;
let triColor = triangle.color;

let x = new Point((a.x + b.x + c.x) / 3,
                  (a.y + b.y + c.y) / 3,
                  (a.z + b.z + c.z) / 3);
let n = new Vector(x.x, x.y, x.z + 2).normalize();
let v = new Point(0, 0, 0).sub(x).normalize();
let l = lightPos.sub(x).normalize();
let h = v.add(l).normalize();

let ca = triColor.mul(inputs["ambient"]);
let cd = triColor.mul(inputs["diffuse"]);
let cs = new Vector(1, 1, 1).mul(inputs["specular"]);
let s = inputs["shininess"];

let ambient = ca;
let diffuse = cd.mul(Math.max(0, l.dot(n)));
let specular = cs.mul(Math.pow(Math.max(0, h.dot(n)), s));
let phongColor = ambient.add(diffuse).add(specular).mulVec(lightColor);

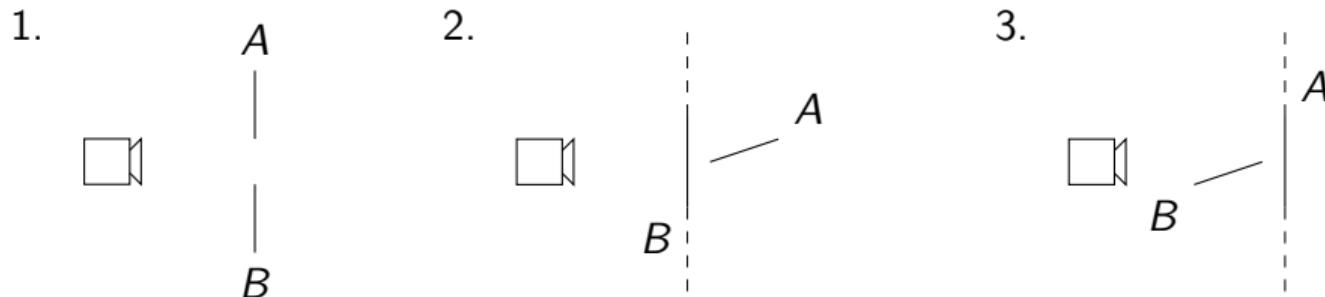
phongColor.x = Math.min(1, phongColor.x);
phongColor.y = Math.min(1, phongColor.y);
phongColor.z = Math.min(1, phongColor.z);
return phongColor;
```

Sichtbarkeit

- Bisher: Sortierung der Dreiecke nach z-Koordinate ihres Mittelpunktes
→ Kann zu Problemen führen
- Berechnung erfordert Tiefeninformationen
- Object Space Techniken
 - Verdeckungsvolumen
 - Painter's Algorithm
- Image Space Techniken
 - z-Buffer
 - α -Buffer
- Mixed Techniken
 - Raycasting

Painter's Algorithm

1. Keine Überlappung \rightarrow Reihenfolge egal
2. A hinter Ebene von B \rightarrow A vor B
3. B vor Ebene von A \rightarrow A vor B



Dreiecke schneiden sich oder es gibt zyklische Überlappungen
 \rightarrow Zerteilung der Dreiecke oder Screen Subdivision

z-Buffer

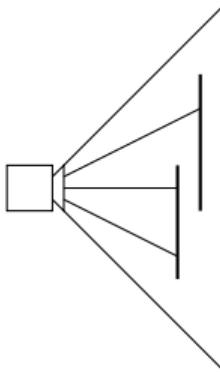
- Painter's Algorithm mit Screen Subdivision bis auf Pixelebene
- Pro Pixel z-Wert des nächsten Objektes, das darauf fällt
- Dazugehörige Farbe im Color-Buffer
- Finale Farbe des Pixels am Ende im Color-Buffer
- z-Fighting: Mehrere Objekte haben gleiche z-Werte



α -Buffer

- Pro Pixel Liste mit Einträgen für alle Objekte, die darauf fallen
 - z-Wert, Farbe, α -Wert
- Finale Farbe des Pixels durch Sortierung nach z-Werten und Akkumulation der Farben mit α -Werten
- $\alpha \in [0, 1]$ transparent bis opak

Raycasting



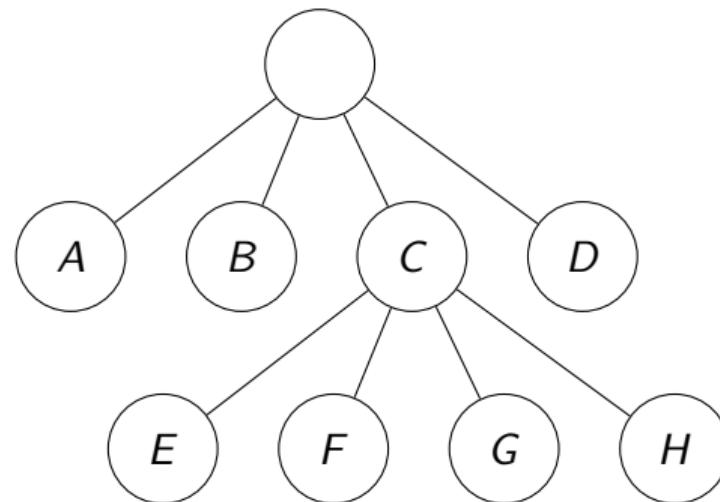
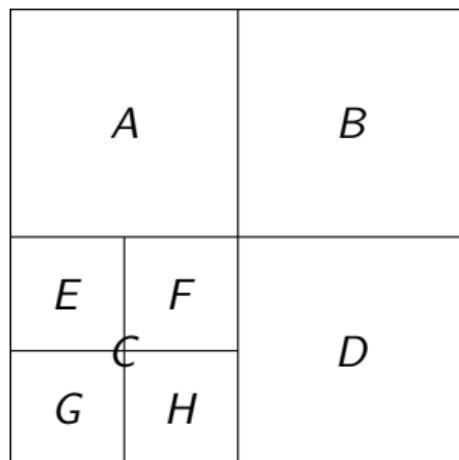
- Durch jeden Pixel ein Strahl $R = \{p + \lambda v\}$ in die Szene
- Sichtbares Objekt ist das Dreieck Δ_{abc} mit Normalen $n = \frac{(b-a) \times (c-a)}{\| \cdot \|}$, das R als erstes schneidet
- Strahl-Ebene-Schnitt:

$$s = p + \frac{n^T a - n^T p}{n^T v} v$$

- Mit baryzentrischen Koordinaten prüfen, ob $s \in \Delta_{abc}$

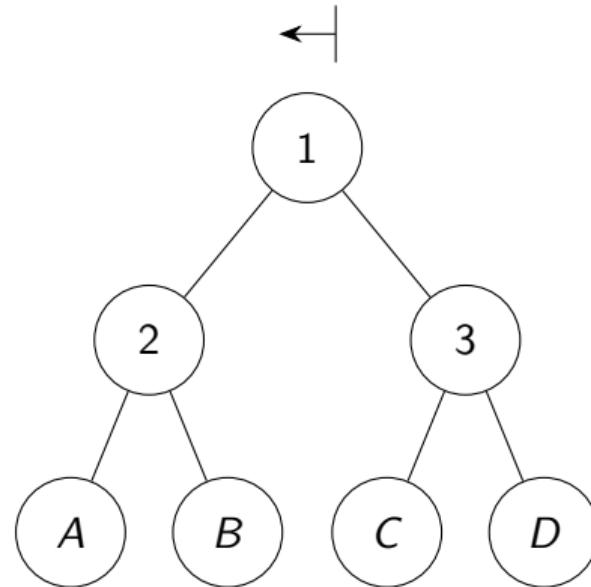
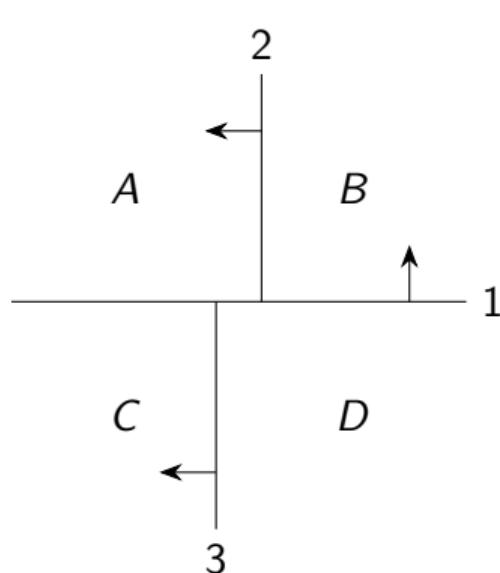
Octree

- Rekursive Aufteilung des Raumes in jeweils 8 gleiche Teile
- Quadtree in 2D (4 Teile)
- Ordnung der Kinder ist wichtig



BSP-Baum

- Binary Space Partitioning
- Rekursive, möglichst gleichmäßige Aufteilung des Raumes in jeweils 2 Teile
- Ordnung der Kinder anhand der Normalen der Trennebenen



Vorbesprechung Blatt 5

Theorie

- Begründungen beim Painter's Algorithm
- Farben sind weiterhin 3D Vektoren

Praxis

- Buffer sind bereits mit richtiger Dimensionalität und Größe aber noch leer initialisiert
- Rendering dauert mitunter ziemlich lange → size anpassen
(nächste Woche schafft Abhilfe 😊)
- z-Buffer Code nach der Bonusaufgabe nicht löschen

Computergrafik

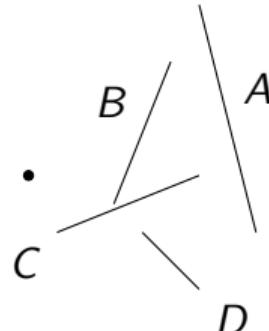
Übung 6 – Shader

Steffen Hinderink / Ingmar Ludwig

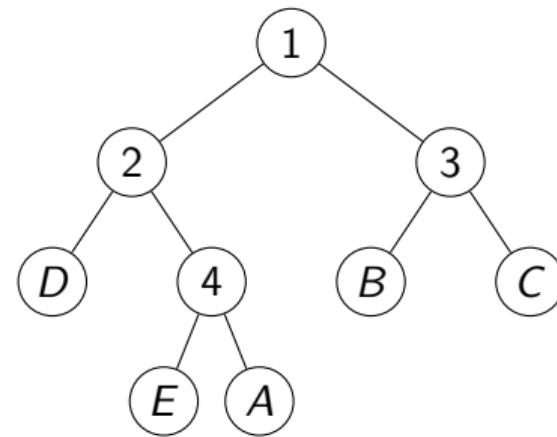
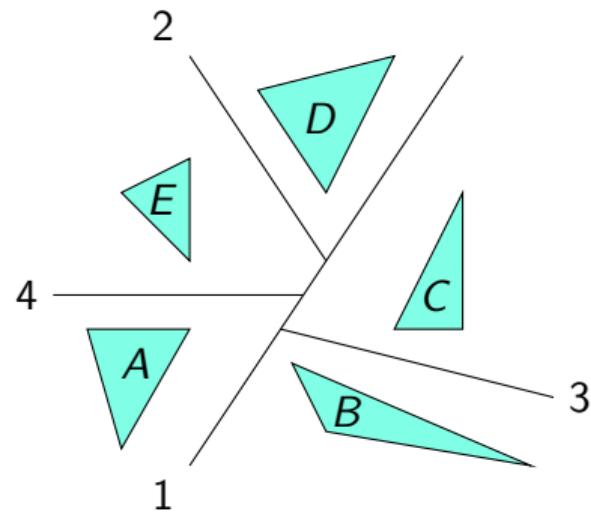
21. / 23. Mai 2024

Nachbesprechung Blatt 5

a) $ADC\dot{B}$



b)



Nachbesprechung Blatt 5

c) $n = \frac{1}{\sqrt{45}} \begin{pmatrix} 0 \\ 3 \\ 6 \end{pmatrix}$ $s = \begin{pmatrix} 0.5 \\ 1 \\ -0.5 \end{pmatrix}$

d)

z -Buffer	Color-Buffer
0.2	(0.7, 0.7, 0)
0.2	(0.7, 0.7, 0)
0.2	(0.7, 0.7, 0)
-0.3	(0.1, 0.9, 1)
-0.8	(0, 1, 0.8)

e)

	z	c	α
0	0.6	(0.7, 0.8, 0.9)	0.2
1	0.2	(0.1, 0.1, 0.1)	0.2
2	-0.2	(1, 1, 1)	0.5
3	-0.3	(0, 1, 0)	1
4	-1	(0, 1, 1)	0.8

$$C_4 = 0.8 \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + (1 - 0.8) \cdot \left(1 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + (1 - 1) \cdot \dots \right) = \begin{pmatrix} 0 \\ 1 \\ 0.8 \end{pmatrix}$$

Nachbesprechung Blatt 5

```
a) const backgroundColor = new Vector(1, 1, 1);
  for (let i = 0; i < size; i++) {
    for (let j = 0; j < size; j++) {
      colorBuffer[i][j] = backgroundColor;
      zBuffer[i][j] = 1;
    }
  }

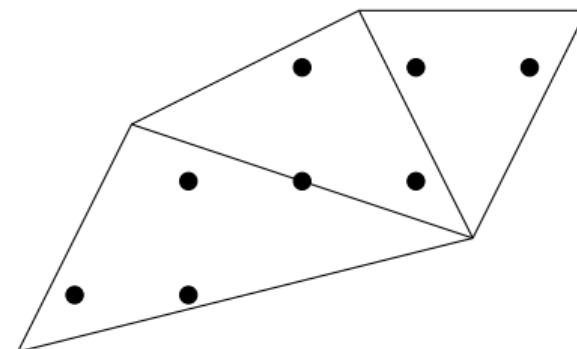
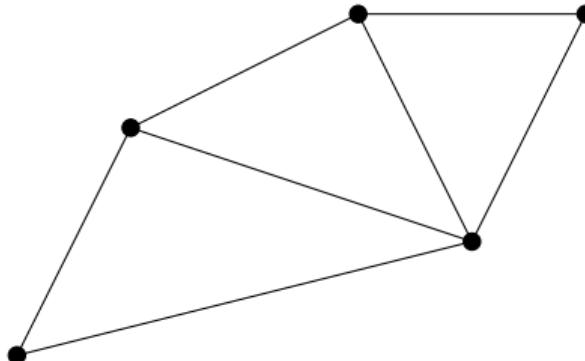
b) if (z < zBuffer[row][col]) {
  zBuffer[row][col] = z;
  colorBuffer[row][col] = color;
}
```

WebGL

- Kompilieren und Linken der Shader
- Buffer für alle pro Vertex Daten (Attributes)
 - Shared Vertex Datenstruktur: Ein Vertex kann Teil mehrerer Dreiecke sein
- Reset von Farben und z-Buffer
- Hinzufügen der Attribute-Buffer und globaler Daten (Uniforms)
- Draw Aufruf

Shader

- Programmierung in GLSL
- Vertex Shader
 - Wird für jeden Vertex aufgerufen
 - Berechnung der finalen Position
- Fragment Shader
 - Wird für jedes Dreieck für alle getroffenen Pixel aufgerufen
 - Berechnung der finalen Farbe
- Kommunikation über Varyings, Interpolation über Dreieck



GLSL

- OpenGL Shading Language
- Debugging schwierig, mit Hilfe von Farben
- Anders als JavaScript starke Typisierung
- Zahlen ohne Nachkommastellen werden nicht als float erkannt
→ 21.0 statt 21 schreiben
- Multi-Zugriff:
`vec3 v1 = vec3(0, 0.5, 1);
vec2 v2 = v1.yz;`
- Vektoren können in andere eingesetzt werden:
`vec3 v3 = vec3(v2, 1);`
- Swizzling:
`vec3 v4 = v1.brb;`
- * ist bei Vektoren die komponentenweise Multiplikation:
`vec3 v5 = v3 * v4;`
- Viele geometrische Funktionen, z.B. `dot`, `cross`, `normalize`
- `attribute vec4 ...` wird automatisch mit 1 aufgefüllt, wenn im Hauptprogramm nur 3 Werte mitgegeben werden
→ Homogene Koordinaten

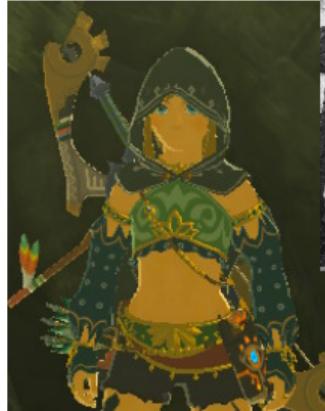
Vorbesprechung Blatt 6

Theorie

- Ausprobieren ist möglich

Praxis

- Erweiterungen auch außerhalb der Shader
für Bonusaufgabe, z.B. Facenormalen



shadertoy.com

Computergrafik

Übung 7 – Texturen

Steffen Hinderink / Ingmar Ludwig

28. / 30. Mai 2024

Nachbesprechung Blatt 6

a)	Attribute	Uniform	Varying
Lesen in VS	x	x	x
Schreiben in VS			x
Definition in Funktion in VS			
Gleicher Wert in VS		x	
Lesen in FS		x	x
Schreiben in FS			
Definition in Funktion in FS			
Gleicher Wert in FS		x	
Schreiben in HP	x	x	

b) Vertex, gl_Position, Fragment, gl_FragColor

- c)
- ModelView-Transformation
 - Projektion
 - Rasterisierung
 - z-Buffer-Test

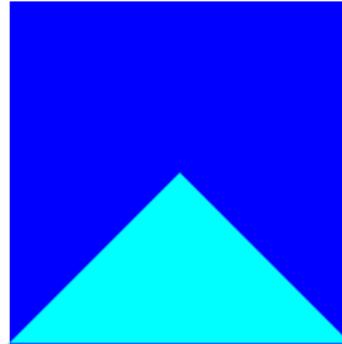
d) $[0, \infty)$

Nachbesprechung Blatt 6

e)

```
1  vec4 v = vec4(1.0, 2.0, 3.0, 4.0);
2  float x = 8;
3  vec3 v1 = v.xxx;
4  vec3 v2 = v.xyzw;
5  float f = v[1];
6  v[0] = f;
7  vec3 w = vec3(5.0, 6.0, 7.0);
8  mat2 m = mat2(2.0, 3.0, 1.0, 0.0, 0.1);
9  vec3 v3 = v.gb;
10 vec3 v4 = v.abc;
11 vec3 d = dot(w, v1);
12 vec3 e = dot(v.ab, w.xy);
13 vec3 r = cross(w, v);
14 v[1] = x;
```

f) 25%

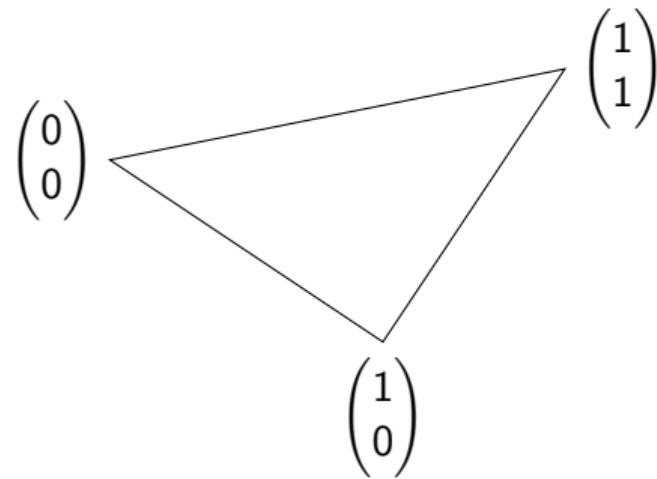
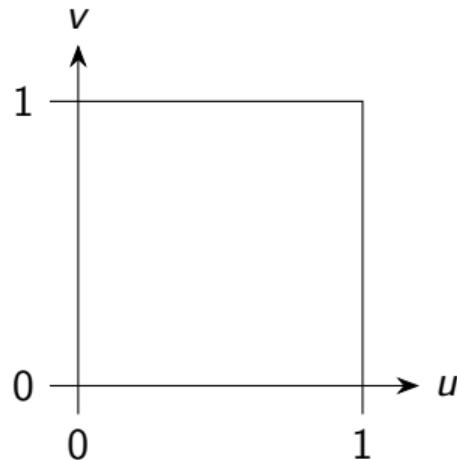


Nachbesprechung Blatt 6

- a)
- ```
let positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions), gl.STATIC_DRAW);
let normalBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normals), gl.STATIC_DRAW);
let colorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
let indexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), gl.STATIC_DRAW);
```
- b)
- ```
vec4 modelViewPosition = uModelViewMatrix * aPosition;
vPosition = modelViewPosition.xyz;
vNormal = normalize((uNormalMatrix * aNormal).xyz);
vColor = aColor;
gl_Position = uProjectionMatrix * modelViewPosition;
```
- c)
- ```
vec3 n = normalize(vNormal);
vec3 v = normalize(-vPosition);
vec3 l = normalize(uLightPos - vPosition);
vec3 r = 2.0 * n * dot(n, l) - l;
vec3 Ca = uAmbient * vColor;
vec3 Cd = uDiffuse * vColor;
vec3 Cs = uSpecular * vec3(1, 1, 1);
float s = uShininess;
vec3 color = (Ca + Cd * max(0.0, dot(l, n)) + Cs * pow(max(0.0, dot(v, r)), s)) * uLightColor;
gl_FragColor = vec4(color, 1);
```

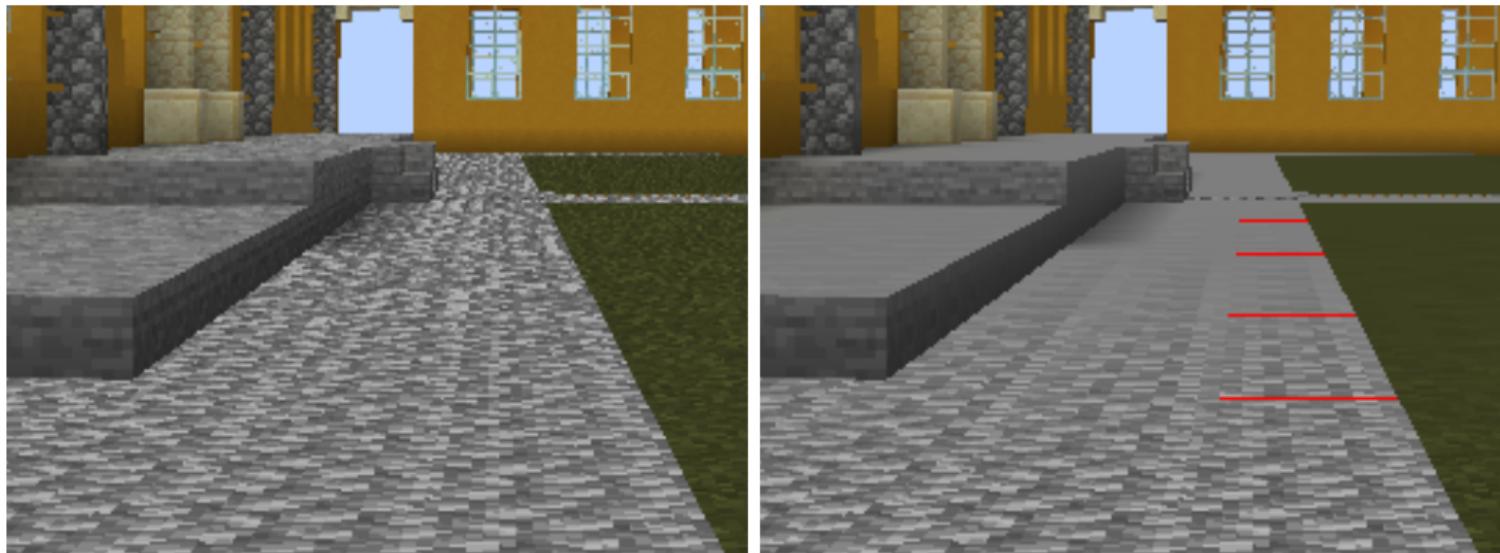
# Texturen

- Bilder auf den Oberflächen der Objekte
- Texturkoordinaten an den Vertices
- Interpolation mit baryzentrischen Koordinaten



# Textur-Filterung

- Magnification → Bilineare Interpolation
- Minification → MipMapping



- Trilineare Interpolation: Zusätzlich zwischen MipMap-Leveln interpolieren
- Anisotropic Filtering: MipMaps mit unterschiedlichen Leveln in  $u$  und  $v$

# WebGL

- Texturkoordinaten als Attribute
- Textur als Uniform
- Parameter für die Texturierung mit `gl.texParameteri`
  - Filterung: NEAREST, LINEAR,  $a\_MIPMAP\_b$   
( $a$ :  $u$ - $v$ -Filterung,  $b$ : MipMap-Level Filterung)
  - Wrapping: REPEAT, CLAMP\_TO\_EDGE, MIRRORED\_REPEAT
- Berechnung der Farbe in Shadern mit `texture2D`

# Nicht-Farb-Texturen

- Maps für Materialeigenschaften des Phong-Beleuchtungsmodells
- Normal-, Bump-, Parallax- oder Displacement-Map
- Shadow-Map
- Environment Mapping
  - Cube Mapping
  - Spherical Environment Mapping

# Vorbesprechung Blatt 7

## Theorie

- Pixelzentren haben weiterhin ganzzahlige Koordinaten

## Praxis

- Attributes, Uniforms und Varyings für Texturierung müssen noch deklariert werden
- Shader enthalten bereits Phong Lighting mit Phong Shading  
(Lösung von letzter Woche nur ohne Farb-Attributes)
- Texturen liegen komprimiert als String im Code vor
- Konvertieren von eigenen Bildern:  
[base64.guru/converter/encode/image](http://base64.guru/converter/encode/image)  
Output Format: Data URI

Computergrafik

# Übung 8 – Netze

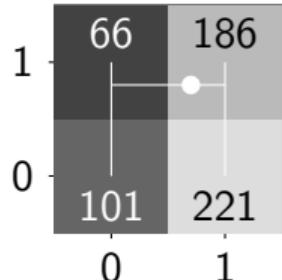
Steffen Hinderink / Ingmar Ludwig

4. / 6. Juni 2024

# Nachbesprechung Blatt 7

a) 186

b)  $0.3 \cdot (0.2 \cdot 101 + 0.8 \cdot 66) + 0.7 \cdot (0.2 \cdot 221 + 0.8 \cdot 186) = 157$



c)

|     |     |     |     |
|-----|-----|-----|-----|
| 204 | 42  | 41  | 151 |
| 21  | 69  | 187 | 145 |
| 196 | 158 | 243 | 51  |
| 159 | 51  | 32  | 42  |

|     |     |
|-----|-----|
| 84  | 131 |
| 141 | 92  |

|     |
|-----|
| 112 |
|-----|

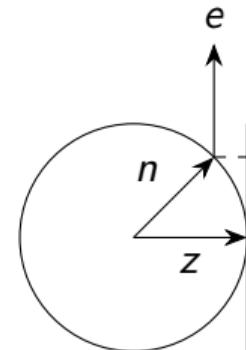
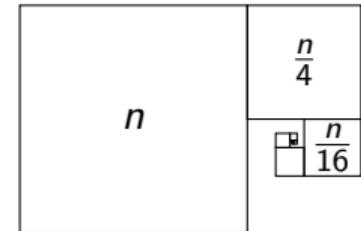
# Nachbesprechung Blatt 7

d)  $\frac{3}{4} \cdot \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} + \frac{1}{4} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.75 \\ 0.625 \end{pmatrix}$

e)  $\frac{n}{4} + \frac{n}{16} + \frac{n}{64} + \dots < n \cdot \sum_{i=1}^{\infty} \frac{1}{4^i} = \frac{n}{3}$

f)  $n = (e + z) \cdot \frac{1}{\|\cdot\|} = \left( \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right) \cdot \frac{1}{\|\cdot\|} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$

$$\Rightarrow \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{2} \cdot \left( \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.5 \\ \frac{1}{2\sqrt{2}} + 0.5 \end{pmatrix}$$



# Nachbesprechung Blatt 7

## a) Vertex Shader

```
attribute vec2 aTextureCoordinate;
varying vec2 vTextureCoordinate;
...
 vTextureCoordinate = aTextureCoordinate;
```

## Fragment Shader

```
varying vec2 vTextureCoordinate;
```

## b) uniform sampler2D uTexture;

```
...
 vec4 textureColor = texture2D(uTexture, vTextureCoordinate);
```

## c)

```
 vec3 Ca = uAmbient * textureColor.rgb;
 vec3 Cd = uDiffuse * textureColor.rgb;
```

# Euler-Formel

$$V - E + F = 2 \cdot (1 - g)$$

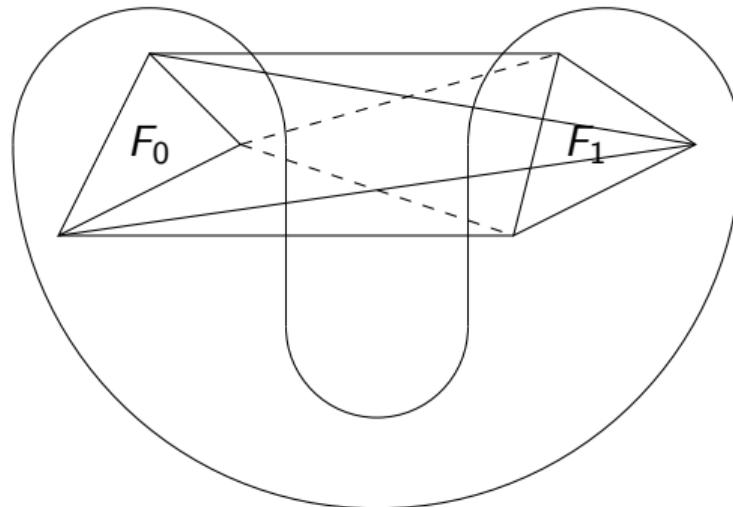
Induktive Beweisidee:

- Für ein Tetraeder gilt die Eulerformel:  $4 - 6 + 4 = 2 \cdot (1 - 0)$
- Durch die Modifikationsoperatoren behält die Euler-Formel ihre Gültigkeit
  - Face Split
  - Edge Split
  - Vertex Split
  - Edge Flip
  - Edge Collapse
- Alle Polygonnetze können mit den Modifikationsoperatoren aus einem Tetraeder erzeugt werden (hier ist der Beweis ungenau)

# Euler-Formel

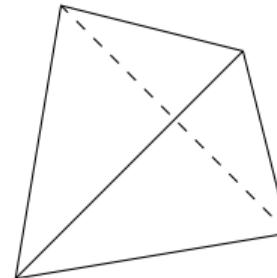
$$V - E + F = 2 \cdot (1 - g)$$

- Modifikationsoperator, der den Genus erhöht
- $F_0$  und  $F_1$  Faces, die sich keine Vertices teilen, mit freiem Raum zwischeneinander
- Entferne  $F_0$  und  $F_1$ , erzeuge neue Edges und Faces so:



- Differenz:  $V - E + F = 0 - 6 + 4 = -2 \checkmark$

# Datenstrukturen



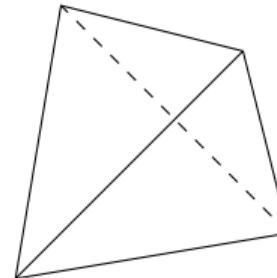
- Face List

$$F = [((1, 1, 1), (1, 0, 0), (0, 1, 0)), \\ ((1, 1, 1), (0, 1, 0), (0, 0, 1)), \\ ((1, 1, 1), (0, 0, 1), (1, 0, 0)), \\ ((1, 0, 0), (0, 0, 1), (0, 1, 0))]$$

- Shared Vertex

$$V = [(1, 1, 1), (1, 0, 0), (0, 1, 0), (0, 0, 1)] \\ F = [(0, 1, 2), (0, 2, 3), (0, 3, 1), (1, 3, 2)]$$

# Datenstrukturen



## Halfedge Mesh

$V = [(1, 1, 1), (1, 0, 0), (0, 1, 0), (0, 0, 1)]$

$\text{out} = [0, 1, 2, 5]$

$\text{next} = [1, 2, 0, 4, 5, 3, 7, 8, 6, 10, 11, 9]$

$\text{opposite} = [8, 11, 3, 2, 10, 6, 5, 9, 0, 7, 4, 1]$

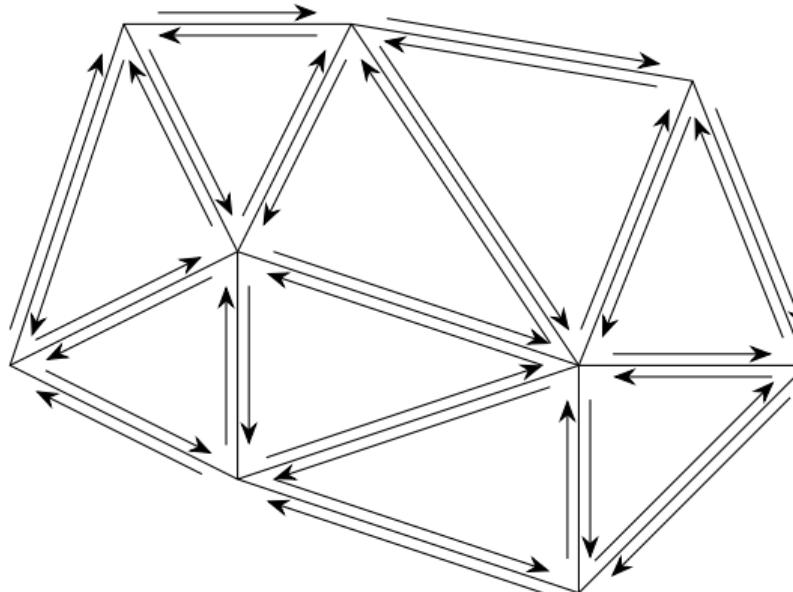
$\text{face} = [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3]$

$\text{to} = [1, 2, 0, 2, 3, 0, 3, 1, 0, 3, 2, 1]$

$\text{halfedge} = [0, 3, 6, 9]$

# Halfedge Mesh

- Beispiel für einen Algorithmus, der Halfedge Mesh Struktur nutzt:  
Bestimmung der Vertexnormalen von Vertex  $v$
- $n_v = \frac{1}{\|\cdot\|} \sum_f \alpha_f n_f$



# Vorbesprechung Blatt 8

## Theorie

- Algorithmen auf Halfedge Meshes benutzen oft do-while-Schleifen
- Halfedges sind auch außen bei Halfedge Meshes mit Rand
- Richtig Zählen
- Video zum Genus eines Menschen: [youtu.be/egEraZP9yXQ](https://youtu.be/egEraZP9yXQ)

## Praxis

- Halfedge Mesh durch Arrays implementiert
- Die Topologie soll unverändert bleiben

Computergrafik

# **Übung 9 – Freiformgeometrie**

Steffen Hinderink / Ingmar Ludwig

11. / 13. Juni 2024

# Nachbesprechung Blatt 8

a) Durchschnittliche Vertex-Valenz eines Hexagonnetzes:

$$H = 2E$$

$$H = 6F$$

$$g = 1$$

(1)

$$V - E + F = 2 \cdot (1 - g)$$

$$\Leftrightarrow 6V - 2H = 0$$

$$\Leftrightarrow \frac{H}{V} = 3$$

# Nachbesprechung Blatt 8

b) Halfedge  $hv = \text{out}(v)$

Halfedge  $sv = hv$

**do**

Vertex  $u = \text{to}(hv)$

Halfedge  $hu = \text{out}(u)$

Halfedge  $su = hu$

**do**

Vertex  $w = \text{to}(hu)$

bool  $\text{too\_close} = (w = v)$

Halfedge  $hw = \text{out}(w)$

Halfedge  $sw = hw$

**do**

if  $\text{to}(hw) = v$

|  $\text{too\_close} = \text{true}$

$hw = \text{next}(\text{opposite}(hw))$

**while**  $hw \neq sw$

if not  $\text{too\_close}$

| **return**  $w$

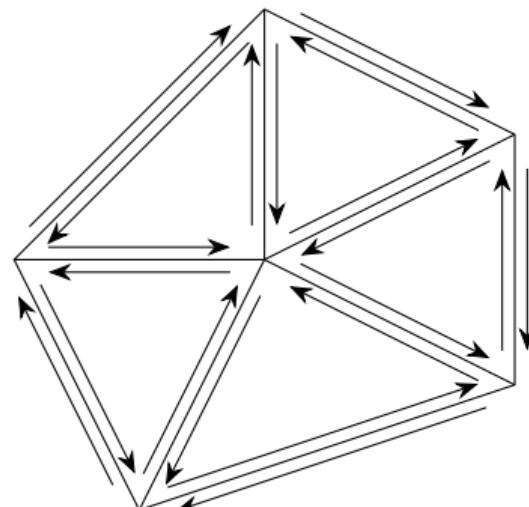
$hu = \text{next}(\text{opposite}(hu))$

**while**  $hu \neq su$

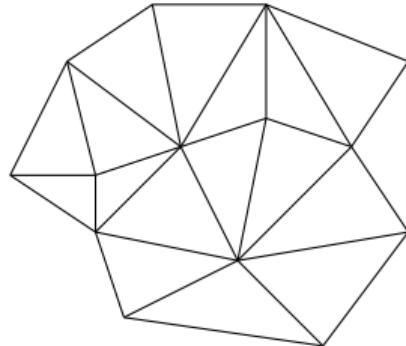
$hv = \text{next}(\text{opposite}(hv))$

**while**  $hv \neq sv$

**return** error (Es existiert kein Vertex 2 Kanten entfernt von  $v$ )



# Nachbesprechung Blatt 8



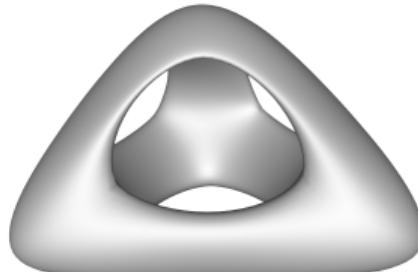
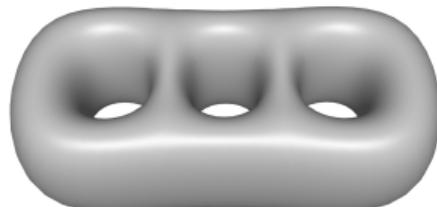
$$V = 14$$

$$E = 30$$

$$F = 17$$

$$H = 2E = 3F + \text{Ränder} = 60$$

- c)  $F \cdot 3 \cdot 3 \cdot 8 B = 1224 B$
- d)  $V \cdot 3 \cdot 8 B + F \cdot 3 \cdot 4 B = 540 B$
- e)  $V \cdot (3 \cdot 8 B + 4 B) + H \cdot 4 \cdot 4 B + F \cdot 4 B = 1420 B$
- f)  $g = 3$  bei beiden Objekten



# Nachbesprechung Blatt 8

```
a) for (let v = 0; v < numVertices; v++) {
 if (!(fixBoundary && isBoundary(v))) {
 let h = out[v];
 let s = h;
 let mean = new Vector(0, 0, 0);
 let cnt = 0;
 do {
 mean = mean.add(getCoordinates(to[h]));
 cnt++;
 h = next[opposite[h]];
 } while (h !== s);
 setCoordinates(v, mean.mul(1 / cnt).add(getCoordinates(v)).mul(0.5));
 }
}

b) function isBoundary(v) {
 let h = out[v];
 let s = h;
 do {
 if (face[h] === -1) {
 return true;
 }
 h = next[opposite[h]];
 } while (h !== s);
 return false;
}
```

# Bézierkurven

- Bernsteinpolynome als Basis

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

- Bézierkurve aus Kontrollpunkten  $b_i$

$$C : [0, 1] \rightarrow \mathbb{R}^d$$

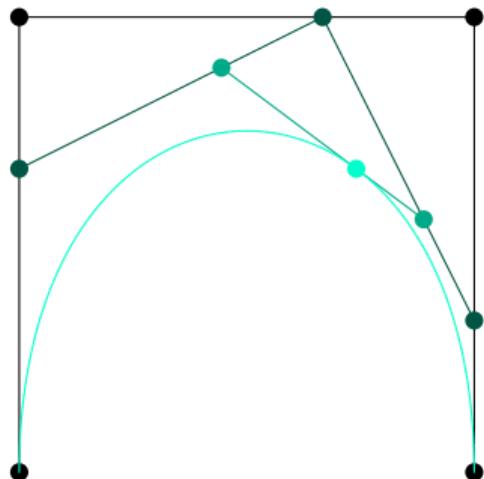
$$t \mapsto \sum_{i=0}^n b_i B_i^n(t)$$

- Schöne Eigenschaften für die Modellierung von Kurven:
  - Partition der Eins  $\rightarrow$  Unabhängig vom Koordinatensystem
  - Nicht-Negativität  $\rightarrow$  Verformung in Verschieberichtung
  - Gleichverteilung der Maxima  $\rightarrow$  Lokale Verformungen
- Beispiel: Bögen Tool in MS Paint
- Nicht nur zum Zeichnen  $\rightarrow$  Kamerafahrten

# De-Casteljau-Algorithmus

- Wiederholte lineare Interpolation zwischen Punkten
- Kubische Bézierkurve aus Kontrollpunkten

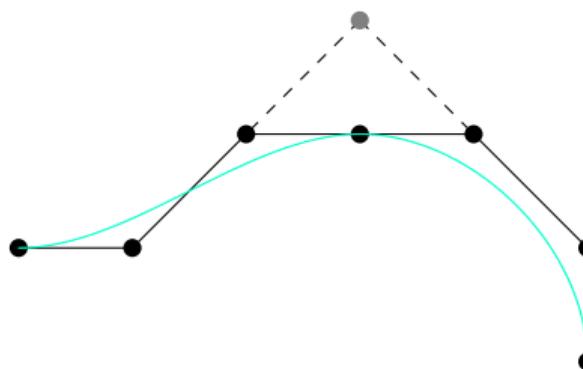
$$b_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 9 \end{pmatrix}, b_2 = \begin{pmatrix} 9 \\ 9 \end{pmatrix} \text{ und } b_3 = \begin{pmatrix} 9 \\ 0 \end{pmatrix}, \text{ ausgewertet bei } t = \frac{2}{3}$$



$$\begin{array}{c} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \xrightarrow{\cdot \frac{1}{3}} \begin{pmatrix} 0 \\ 6 \end{pmatrix} \\ \begin{pmatrix} 0 \\ 9 \end{pmatrix} \xrightarrow{\cdot \frac{2}{3}} \begin{pmatrix} 6 \\ 9 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 8 \end{pmatrix} \\ \begin{pmatrix} 9 \\ 9 \end{pmatrix} \quad \begin{pmatrix} 9 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 8 \\ 5 \end{pmatrix} \quad \begin{pmatrix} \frac{20}{3} \\ 6 \end{pmatrix} \\ \begin{pmatrix} 9 \\ 0 \end{pmatrix} \end{array}$$

# Splines

- Stückweise polynomiale Funktion (Grad  $n$ ) mit  $C^{n-1}$ -stetigen Übergängen
- $C^0$ -stetige Verbindung linearer Polynome  
→ Endpunkte gleich
- $C^1$ -stetige Verbindung quadratischer Polynome  
→ Tangenten an den Endpunkten gleich
- $C^2$ -stetige Verbindung kubischer Polynome  
→ A-Frame-Bedingung



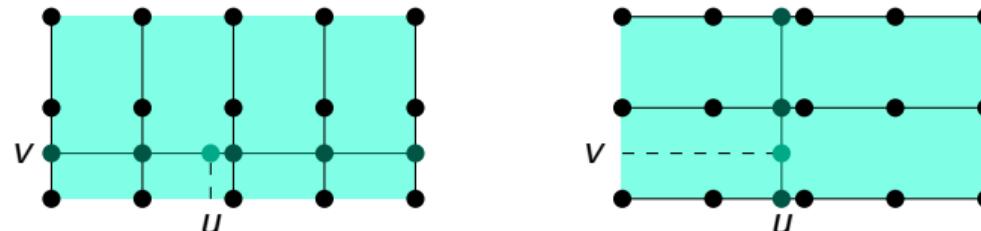
# Bézierflächen

- Gitter von Kontrollpunkten
- In beiden Dimensionen jeweils Bézierkurven

$$S : [0, 1] \times [0, 1] \rightarrow \mathbb{R}^d$$

$$\begin{pmatrix} u \\ v \end{pmatrix} \mapsto \sum_{i=0}^m \sum_{j=0}^n b_{ij} B_j^n(v) B_i^m(u)$$

- Auswertung auch durch wiederholten De-Casteljau-Algorithmus



- Spline Flächen sind auch möglich

# Vorbesprechung Blatt 9

## Theorie

- Unterschiedliche Zeichnungen für b und c
- Wichtige Eigenschaften (vorherige Aufgaben) sollten in der Skizze erkennbar sein

## Praxis

- JavaScript Array Kopie durch `slice`
- Übersichtlichkeit durch unterschiedliche Farben für verschiedene Tiefen des De-Casteljau-Algorithmus
- Zeichnen der Kurve/Fläche durch Punkte

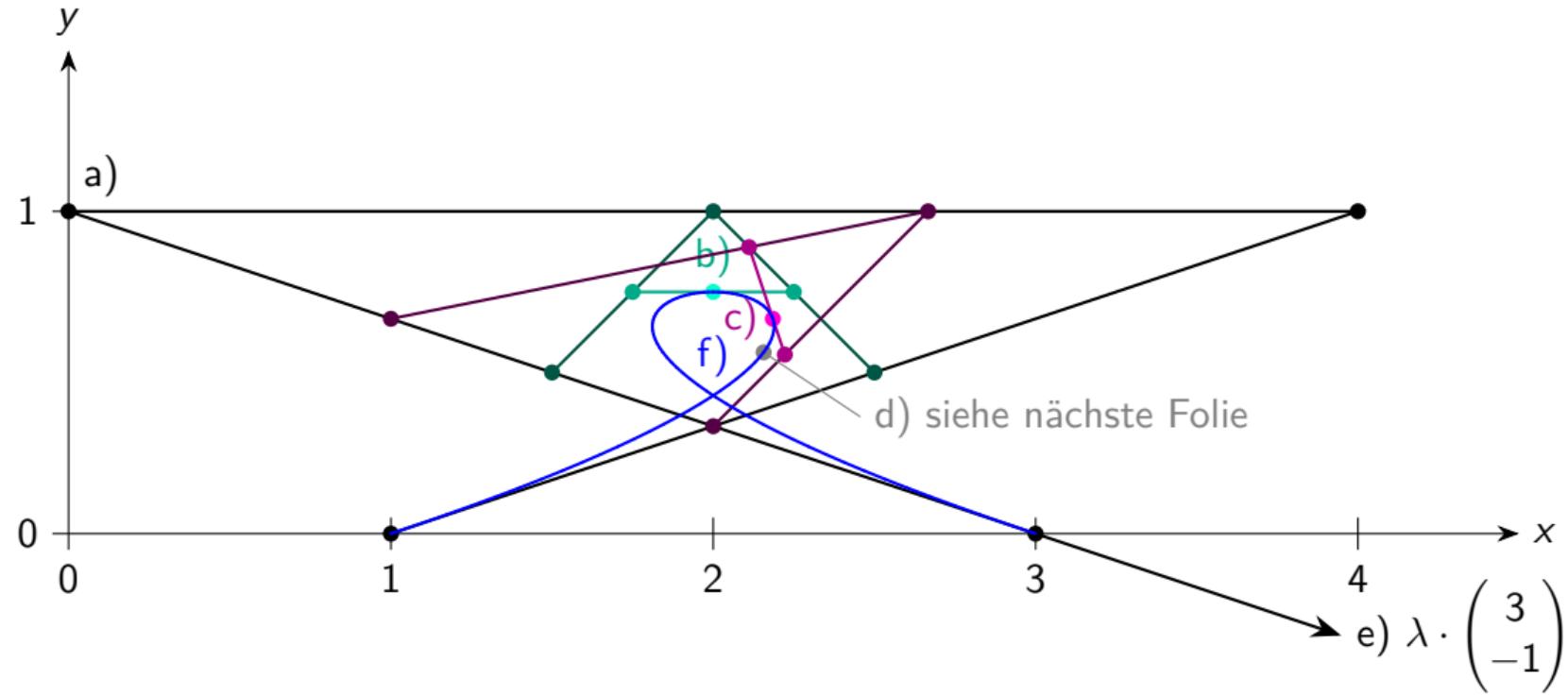
Computergrafik

# **Übung 10 – Globale Beleuchtung**

Steffen Hinderink / Ingmar Ludwig

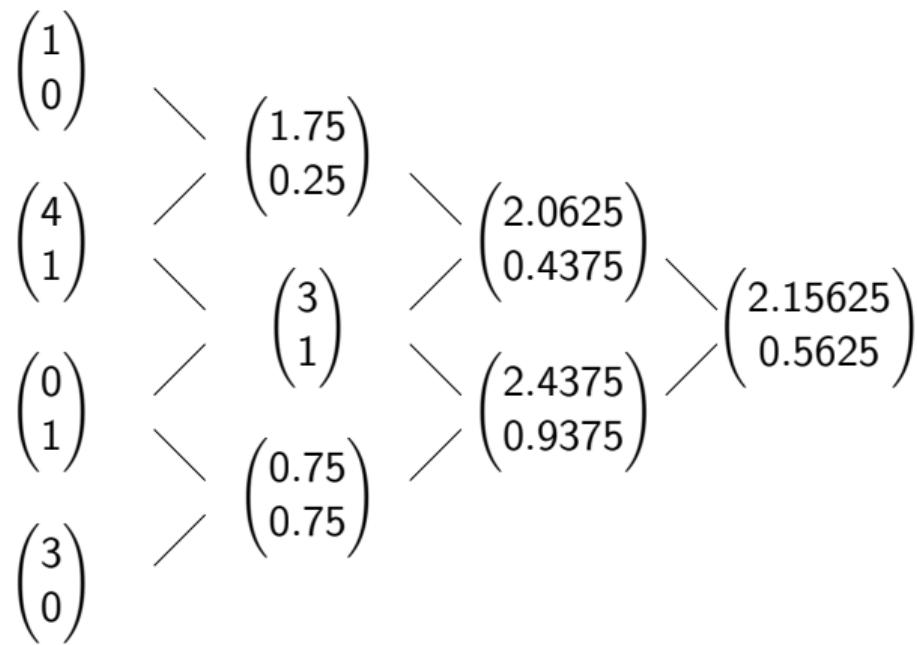
18. / 20. Juni 2024

# Nachbesprechung Blatt 9



# Nachbesprechung Blatt 9

d)



# Nachbesprechung Blatt 9

g)

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \quad \begin{matrix} \searrow \\ \swarrow \end{matrix} \quad \begin{pmatrix} 1 \\ \frac{8}{3} \\ 3 \\ 6 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ 3 \\ 3 \\ 7 \end{pmatrix}$$

h)

$$b_{00} = \begin{pmatrix} 0 \\ -4 \\ 0 \\ 4 \end{pmatrix} \quad \begin{matrix} \nearrow^{\cdot \frac{1}{2}} \\ \searrow^{\cdot \frac{1}{2}} \end{matrix} \quad \begin{pmatrix} 2 \\ -2 \\ 0 \end{pmatrix}$$

$$b_{10} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{matrix} \nearrow^{\cdot \frac{1}{4}} \\ \searrow^{\cdot \frac{3}{4}} \end{matrix} \quad \begin{pmatrix} -1 \\ 0.25 \\ 0.75 \end{pmatrix}$$

$$b_{01} = \begin{pmatrix} -4 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \begin{matrix} \nearrow^{\cdot \frac{1}{2}} \\ \searrow^{\cdot \frac{1}{2}} \end{matrix} \quad \begin{pmatrix} -2 \\ 1 \\ 1 \end{pmatrix}$$

$$b_{11} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 2 \end{pmatrix}$$

# Nachbesprechung Blatt 9

```
const colors = ["#005544", "#00AA88", "#00FFCC", "#808808"];
a) drawFrame(t) {
 for (let i = 0; i < n; i++) {
 drawLine(new Line(this.points[i], this.points[i + 1]));
 }
 let b = this.points.slice();
 for (let k = 0; k < n; k++) {
 for (let i = 0; i < n - k; i++) {
 b[i] = b[i].mul(1 - t).add(b[i + 1].mul(t));
 drawPoint(b[i], colors[k]);
 if (i > 0) {
 drawLine(new Line(b[i - 1], b[i]), colors[k]);
 }
 }
 }
}
```

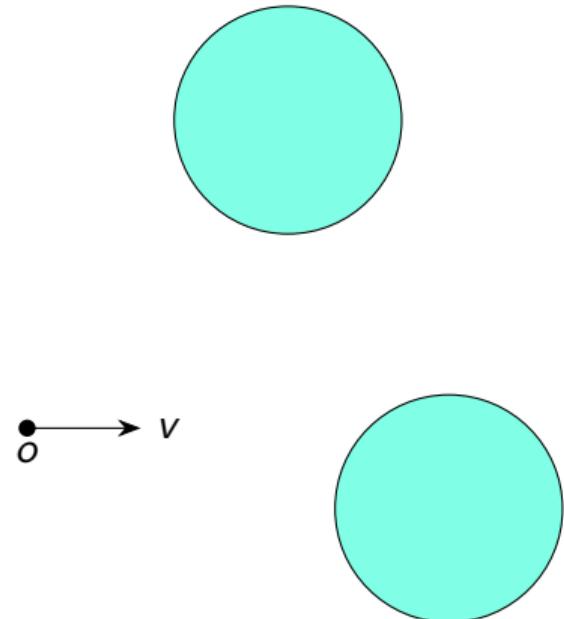
# Nachbesprechung Blatt 9

```
b) getPoint(t) {
 let b = this.points.slice();
 for (let k = 0; k < n; k++) {
 for (let i = 0; i < n - k; i++) {
 b[i] = b[i].mul(1 - t).add(b[i + 1].mul(t));
 }
 }
 return b[0];
}

drawCurve() {
 for (let t = 0; t <= 1; t += 0.001) {
 let p = this.getPoint(t);
 drawPoint(p, colors[n - 1], 1);
 }
}
```

# Raytracing

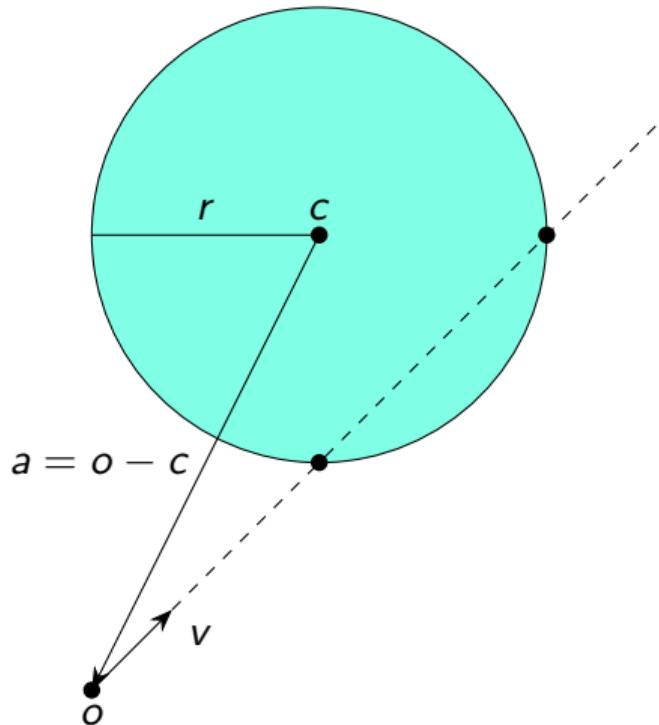
```
trace(o, v)
 if $p = \text{ffi}(o, v)$ *1*2
 $n = \text{surface normal at } p$
 $r = \text{reflection direction}$
 $t = \text{refraction direction}^{\ast 2}$
 $C_{\text{direct}} = \text{phong}(p, n, -v)$
 $C_{\text{reflect}} = \alpha_{\text{reflect}} \cdot \text{trace}(p, r)$
 $C_{\text{refract}} = \alpha_{\text{refract}} \cdot \text{trace}(p, t)$
 return $C_{\text{direct}} + C_{\text{reflect}} + C_{\text{refract}}$
 else
 return background color
```



<sup>\*1</sup>Beschleunigung durch räumliche Suchstrukturen

<sup>\*2</sup>Genauer auf den folgenden Folien

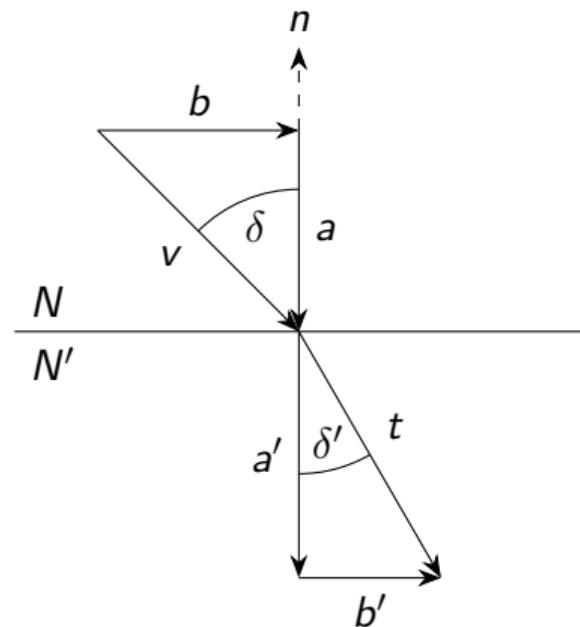
# Strahl-Kugel-Schnitt



$$\begin{aligned}\|a + \lambda v\| &= r \\ (a + \lambda v)^T(a + \lambda v) &= r^2 \\ \lambda^2 \cdot \underbrace{v^T v}_1 + \lambda \cdot \underbrace{2 \cdot a^T v}_p + \underbrace{a^T a - r^2}_q &= 0\end{aligned}$$

$$\begin{aligned}\lambda_{1,2} &= -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q} \\ &= -a^T v \pm \sqrt{(a^T v)^2 - a^T a + r^2}\end{aligned}$$

# Brechungsvektorberechnung



$$a = n^T v \cdot n$$

$$b = v - a$$

Snelliussches Brechungsgesetz:

$$N \sin \delta = N' \sin \delta'$$

$$\|b'\| = \sin \delta' = \frac{N}{N'} \sin \delta = \frac{N}{N'} \|b\|$$

$$\|a'\|^2 + \|b'\|^2 = \alpha^2 \|a\|^2 + \left(\frac{N}{N'}\right)^2 \|b\|^2 = 1$$

$$\alpha = \sqrt{\frac{1}{\|a\|^2} \left(1 - \left(\frac{N}{N'}\right)^2 \|b\|^2\right)}$$

$$t = a' + b' = \alpha a + \frac{N}{N'} b$$

# Radiosity

$$B = E + RF B$$

$B$ : Radiosity der Patches (richtungsunabhängiges Licht)

$E$ : Emittierte Radiosity der Patches

$R$ : Diagonalmatrix der Reflektivitäten der Patches

$F$ : Matrix der Formfaktoren

→ Lineares Gleichungssystem mit einer Gleichung pro Patch

# Vorbesprechung Blatt 10

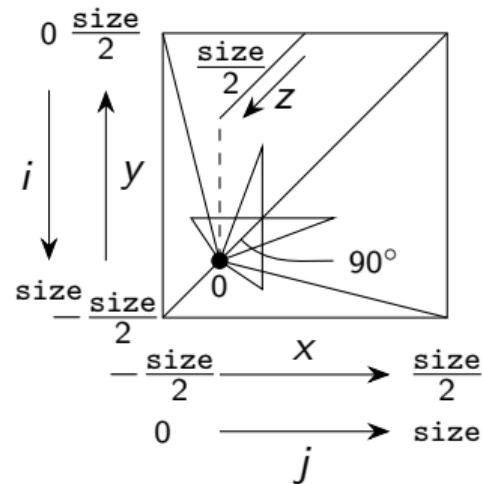
Klausur Anmeldung  
mit  $\geq 100$  Punkten  
bis zum 24. Juni 2024

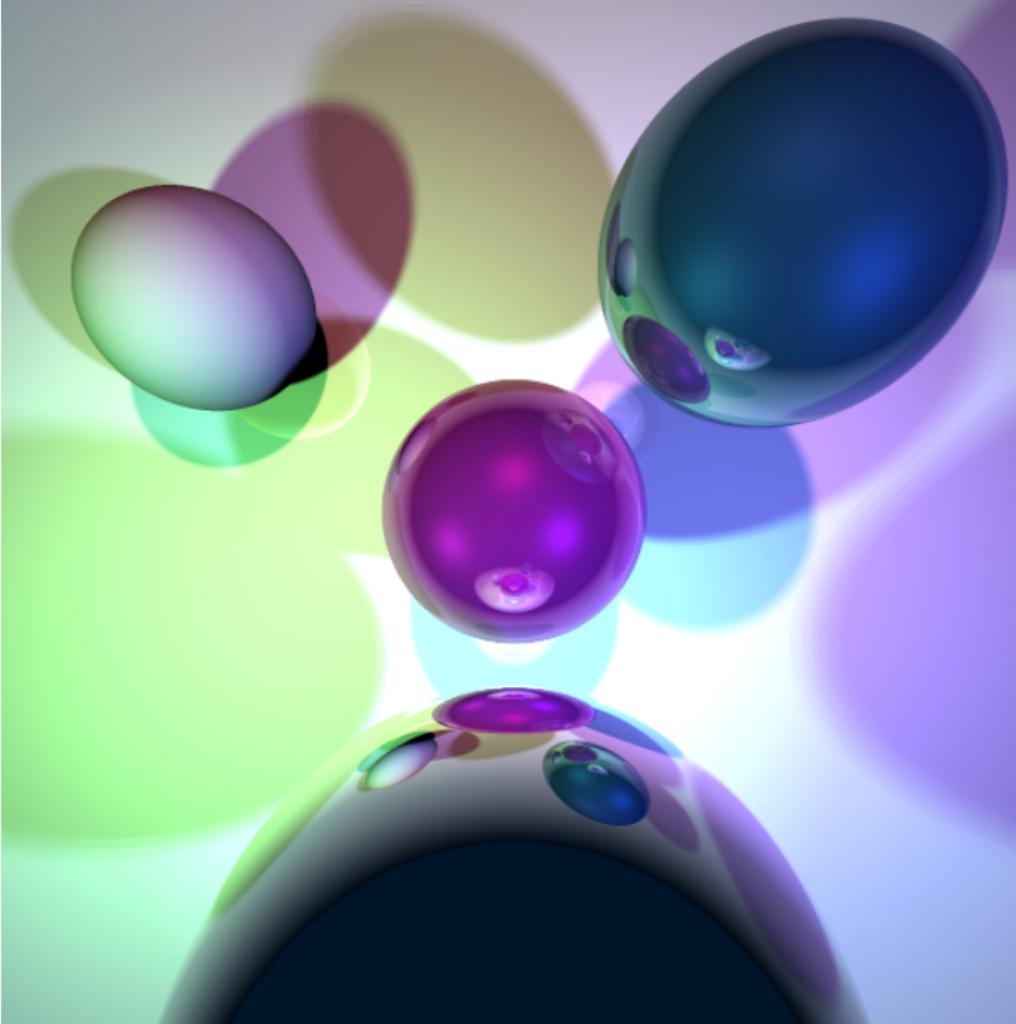
## Theorie

- Splines → Übung 9
- BSP-Bäume → Übung 5

## Praxis

- Reflexionsvektor → Übung 4
- Strahl-Ebene-Schnitt → Übung 5
- Keine Echtzeitfähigkeit, Rendering dauert lange
- Zwischenergebnis alle 10 000 Schritte
- Ergebnis mit höchsten Einstellungen auf nächster Folie





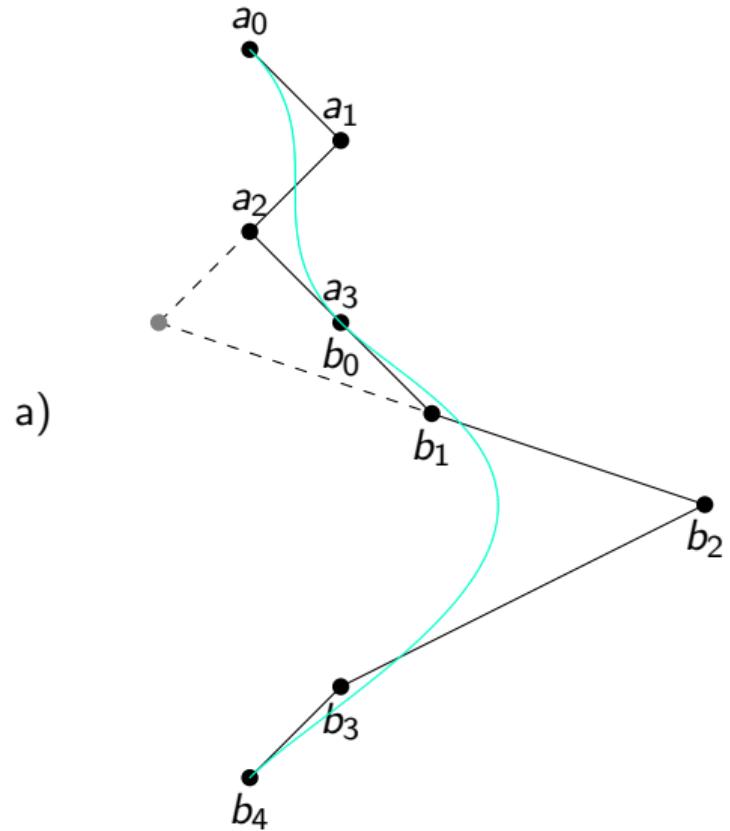
Computergrafik

# Übung 11

Steffen Hinderink / Ingmar Ludwig

25. / 27. Juni 2024

# Nachbesprechung Blatt 10



# Nachbesprechung Blatt 10

b)  $b_0 = a_1 = (4, 4)$

$b_1$  beliebig wählbar, z.B.  $(7, 4)$

$c_0 = b_1 = (7, 4)$

$c_1$  beliebig wählbar, z.B.  $(8, 7)$

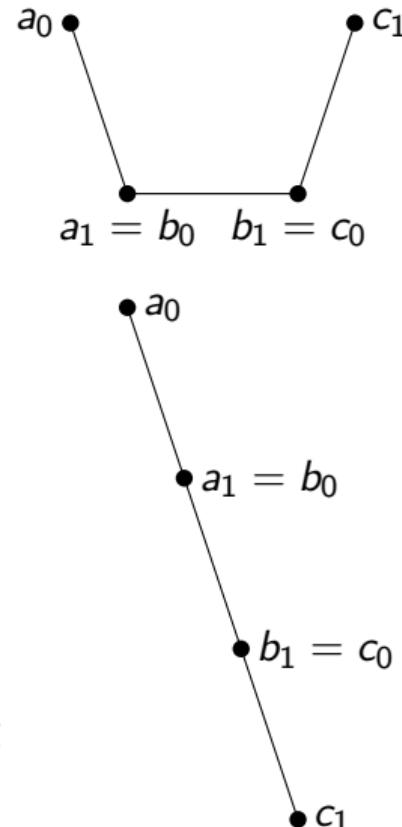
c)  $b_0 = a_1 = (4, 4)$

$b_1 = a_1 + a_1 - a_0 = (5, 1)$

$c_0 = b_1 = (5, 1)$

$c_1 = b_1 + b_1 - b_0 = (6, -2)$

d) Wie (c): Lineare Bézierkurven  $C^1$ -stetig verbunden  
→ Polynom,  $C^\infty$ -stetig und insbesondere  $C^2$ -stetig



# Nachbesprechung Blatt 10

e)  $\text{ffi}(a)$

inner node  
 $n_a^T v < 0$

$\text{ffi}(c)$

inner node  
 $n_c^T v \geq 0$

$\text{ffi}(C)$

leaf node  
`find_first_object_intersection({C})`

**return false**

$\text{ffi}(D)$

leaf node  
`find_first_object_intersection({D})`

**return false**

**return false**

$\text{ffi}(b)$

inner node  
 $n_b^T v < 0$

$\text{ffi}(B)$

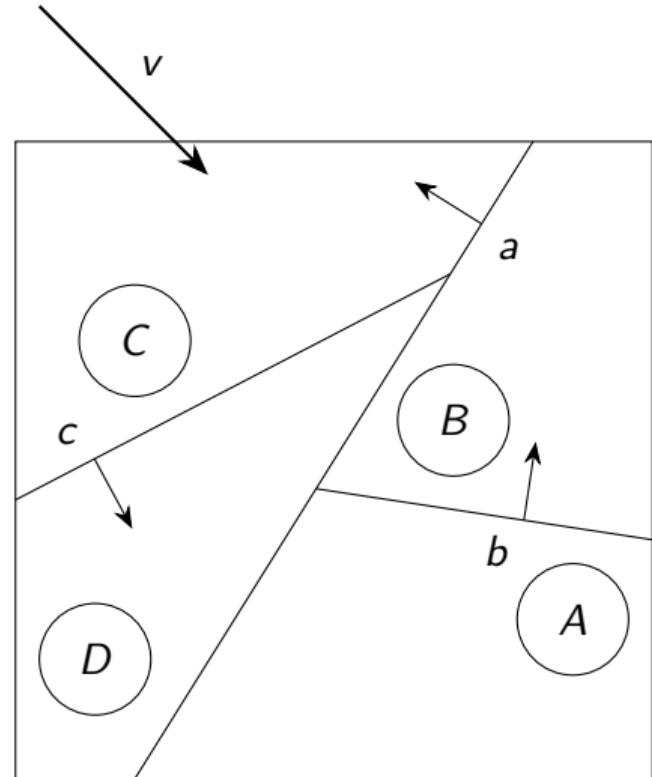
leaf node  
`p = find_first_object_intersection({B})`

**return p**

**return p**

**return p**

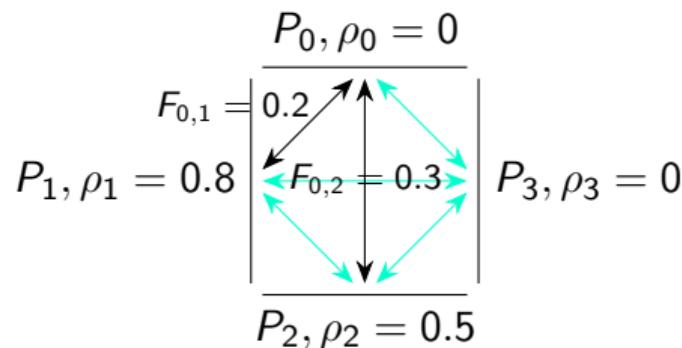
→ Schnittpunkt auf B



# Nachbesprechung Blatt 10

f)  $r = v - 2nn^T v = \frac{\sqrt{2}}{10} \begin{pmatrix} -1 \\ 7 \end{pmatrix} \approx \begin{pmatrix} -0.1414 \\ 0.9899 \end{pmatrix}$

g)



$$B = E + RF B = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0.16 & 0 & 0.16 & 0.24 \\ 0.15 & 0.1 & 0 & 0.1 \\ 0 & 0 & 0 & 0 \end{pmatrix} B \approx \begin{pmatrix} 1 \\ 0.1870 \\ 0.1687 \\ 0 \end{pmatrix}$$

# Nachbesprechung Blatt 10

a)

```
let x = j - size / 2;
let y = -(i - size / 2);
let z = -size / 2;
let v = new Vector(x, y, z).normalize();
setPixel(i, j, trace(new Vector(0, 0, 0), v));
```

b)

```
class Plane {
 constructor(a, b, c, material) {
 this.p = a;
 this.n = b.sub(a).cross(c.sub(a)).normalize();
 this.material = material;
 }

 normal() {
 return this.n;
 }

 intersect(o, v) {
 let nDotV = this.n.dot(v);
 if (nDotV === 0) {
 return [undefined, Infinity]; // Ebene und Strahl parallel
 }
 let lambda = (this.n.dot(this.p) - this.n.dot(o)) / nDotV;
 if (lambda < 0) {
 return [undefined, Infinity]; // Ebene hinter Strahl
 }
 return [o.add(v.mul(lambda)), lambda];
 }
}
```

# Nachbesprechung Blatt 10

c)

```
let n = objects[idx].normal(p);
let cDirect = phong(p, n, new Vector(0, 0, 0).sub(v), objects[idx].material);
if (depth >= maxDepth) {
 return cDirect;
}
let r = v.sub(n.mul(2 * n.dot(v)));
let cReflect = trace(p, r, depth + 1).mul(objects[idx].material.reflectivity);
return cDirect.add(cReflect);
```

d)

```
for (let j = 0; j < objects.length; j++) {
 let dist = objects[j].intersect(x, 1)[1];
 if (dist > epsilon && dist < lightPos.sub(x).norm()) {
 visibilityFactor = 0;
 }
}
```

# Beispielaufgaben

- |                               |                 |
|-------------------------------|-----------------|
| 1. Transformation             | Übungen 1 und 2 |
| 2. Projektion                 | Übungen 2 und 3 |
| 3. Netze                      | Übung 8         |
| 4. OpenGL                     | Übung 6         |
| 5. Euler-Formel               | Übung 8         |
| 6. Baryzentrische Koordinaten | Übung 3         |
| 7. Mapping                    | Übung 7         |

**Viel Erfolg bei der Klausur  
und im weiteren Studium!**