

# Computergrafik

Universität Osnabrück, Henning Wenke, 2012-07-10

1

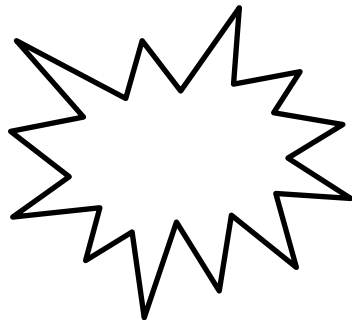
---

Probeklausur: 100 Punkte / 120 Minuten

# Aufgabe 1 (19 Punkte), Teil I

---

- Wie berechnet sich die Länge eines Vektors  $v = (x, y, z)$ ?
  - $|v| = \sqrt{x^2 + y^2 + z^2}$
- Welche Koordinaten hat der Vertex (4, 2, 6, 2) im  $R^3$ ?
  - (2,1,3)
- Geben Sie einen zu  $v = (1,2)$  orthogonalen Vektor an.
  - Z.B.: (2, -1)
- Geben Sie einen dunkelgrünen Farbton in RGB an.
  - Z.B.: (0,0.25,0)
- Zeichnen Sie ein konkaves Polygon.
  - Z.B.:



# Aufgabe 1 (19 Punkte), Teil II

---

- Nennen Sie eine feste Stage der OpenGL Graphics Pipeline.
  - Z.B.: Rasterizer
- In welchem Koordinatensystem wird Clipping in OpenGL ausgeführt?
  - Clipping Coordinates
- Nennen Sie einen Nachteil der lokalen Beleuchtung.
  - Z.B.: Keine impliziten Schatten
  - Oder z.B.: Kein indirektes Licht
- Nennen Sie einen Vorteil der lokalen Beleuchtung
  - Schnell zu berechnen

# Aufgabe 1 (19 Punkte), Teil III

---

- Welche Indizes bilden Dreiecke unter der Topologie `GL_TRIANGLE_STRIP` und den Indexdaten (0, 1, 2, 3, 4)?  
Achten Sie auch auf konsistente Orientierung.
  - 0, 1, 2
  - 1, 3, 2 oder z.B.: 2, 1, 3
  - 2, 3, 4 oder z.B.: 2, 3, 4
  - Hinweis: Auf Orientierung würden wir hier verzichten
- Wie viele Indizes benötigt man bei `GL_TRIANGLE_STRIP` um  $n$  Dreiecke zu erzeugen?
  - $2 + n$
- Was bewirkt `barrier(CLK_LOCAL_MEM_FENCE)` in einem Kernel?
  - Blockiert alle Zugriffe das Local memory, bis alle Work Items je einer Work Group diesen Punkt erreicht haben

# Aufgabe 1 (19 Punkte), Teil IV

---

- Was liefert die GLSL Methode reflect und was für Parameter bekommt sie?
  - Parameter: vec3 i, vec3 n. Liefert den vec3, der entsteht, wenn man n an i reflektiert.
  - Hinweis: Zu speziell, würden wir nicht abfragen
- Wie oft wird der Vertex Shader in einem Durchlauf der Graphics Pipeline ausgeführt?
  - Einmal pro Vertex
- Wie viele Fragments können pro Pixel minimal und maximal erzeugt werden?
  - Keine bis beliebig viele
- Geben Sie die Formel zur Winkelberechnung zweier Vektoren an, die das Kreuzprodukt verwendet.
  - $\sin \alpha = \frac{|v_1 \times v_2|}{|v_1| \cdot |v_2|}$

# Aufgabe 1 (19 Punkte), Teil V

---

- Nennen Sie *(die Namen)* zwei(er) verschiedene(r) Darstellungsformen von Ebenen.
  - HNF,
  - Implizit
- Welche Bedingung muss gelten, damit die Multiplikation  $M_1 \cdot M_2$  mit  $M_1 \in R^{n \times m}$  und  $M_2 \in R^{q \times p}$  möglich ist?
  - $m = q$
- Wodurch wird der Öffnungswinkel der Kamera eines Raytracers bestimmt?
  - Verhältnis von Entfernung der Kamera zur Projektionsebene zur deren Ausmaßen

## Aufgabe 2 (1+3 Punkte)

---

- Gegeben seien die beiden Vektoren  $v_1 = (3,0,4)$  und  $v_2 = (6,8,0)$ .
- Berechnen Sie den Kosinus des Schnittwinkels  $\alpha$  der beiden Vektoren.
- Geben Sie dabei Ihre Ausgangsformel und den Rechenweg an.
- Lösung:

- $\cos \alpha = \frac{v_1 \cdot v_2}{|v_1| \cdot |v_2|}$

- $\cos \alpha = \frac{v_1 \cdot v_2}{|v_1| \cdot |v_2|} = \frac{3 \cdot 6 + 0 \cdot 8 + 4 \cdot 0}{\sqrt{3^2 + 0^2 + 4^2} \cdot \sqrt{6^2 + 8^2 + 0^2}} = \frac{18}{5 \cdot 10} = \frac{9}{25}$



# Aufgabe 3 (1+1+3+5 Punkte)

---

- Stellen Sie jeweils die  $4 \times 4$ -Matrix auf, die durch die folgenden Transformationen beschrieben wird.
- $S\left(1, \frac{1}{2}, 2\right)$ , eine Skalierung mit den Faktoren  $1, \frac{1}{2}$  und  $2$ , jeweils in x-, y- und z-Richtung.
  - $T(-1, -2, -1)$ , eine Translation um den Vektor  $(-1, -2, -1)$ .
  - $R_z\left(\frac{\pi}{2}\right)$ , eine Rotation um die z-Achse mit dem Winkel  $\frac{\pi}{2}$ .
  - $M = S\left(1, \frac{1}{2}, 2\right) \cdot T(-1, -2, -1) \cdot R_z\left(\frac{\pi}{2}\right)$ , die Gesamttransformationsmatrix

# Aufgabe 3 (1+1+3+5 Punkte)

$$S\left(1, \frac{1}{2}, 2\right) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T(-1, -2, -1) = \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z\left(\frac{\pi}{2}\right) = \begin{pmatrix} \cos \frac{\pi}{2} & -\sin \frac{\pi}{2} & 0 & 0 \\ \sin \frac{\pi}{2} & \cos \frac{\pi}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

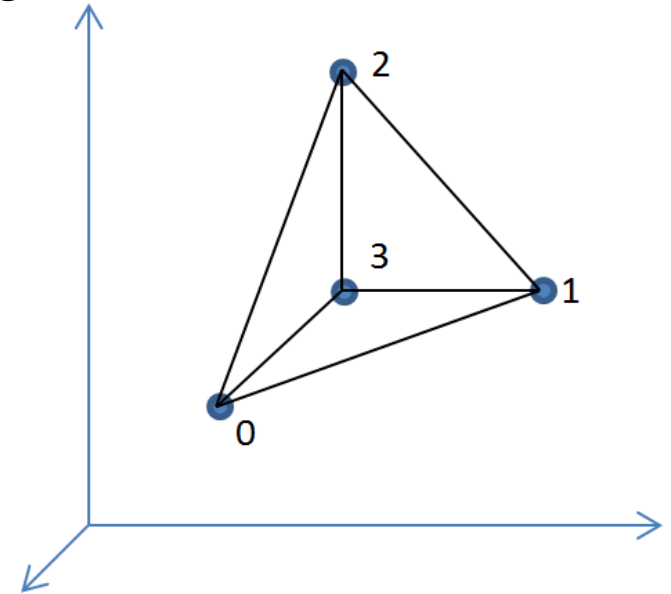
$$M = S\left(1, \frac{1}{2}, 2\right) \cdot T(-1, -2, -1) \cdot R_z\left(\frac{\pi}{2}\right) \\ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & \frac{1}{2} & 0 & -1 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & -1 \\ \frac{1}{2} & 0 & 0 & -1 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Aufgabe 4 (4+2+6 Punkte)

- Zeichnen Sie die folgenden vier Vertices in ein **rechtshändiges** Koordinatensystem ein (x-Achse zeigt nach rechts, y-Achse nach oben) und verbinden Sie sie zu einem Tetraeder. **Hinweis:** Ein Tetraeder ist eine Pyramide mit einer dreieckigen Grundfläche und besteht somit aus exakt **4 Dreiecken**. Jeder Punkt ist mit jedem anderen verbunden.

- Position:  $(0,0,-1)$ , Farbe:  $(1,0,0,1)$
- Position:  $(1,0,-2)$ , Farbe:  $(1,1,0,1)$
- Position:  $(0,1,-2)$ , Farbe:  $(0,0,1,1)$
- Position:  $(0,0,-2)$ , Farbe:  $(0,1,0,1)$



Hinweis: Zeichnung nicht maßstabsgetreu, muss in der richtigen Klausur auch nicht

# Aufgabe 4 (4+2+6 Punkte)

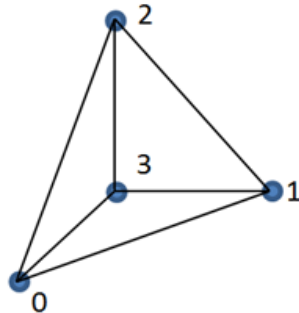
- Geben Sie den Inhalt eines geeigneten Floatbuffers und des zugehörigen Intbuffers an, die als Vertex- und Indexbuffer die Vertices zu einem Tetraeder verbinden.

- Vertexbuffer, Vertexlayout: { pos.x, pos.y, pos.z, color.r, color.g, color.b, color.a }

- |                   |                    |                   |
|-------------------|--------------------|-------------------|
| 0,0,-1, 1,0,0, 1, | 1,0, -2, 1, 1,0,1, | 0,1,-2, 0, 0,1,1, |
| 0,0,-2, 0,1,0,1   |                    |                   |

- Indexbuffer, Topologie: GL\_TRIANGLE\_STRIP (-2 Punkte, bei GL\_TRIANGLES)

- Triangle Strip: 0, 1, 3, 2, 0, 1  
(erzeugt: 0, 1, 3 und 1, 2, 3, und 3, 2, 0 und 2, 1, 0)
- Triangles: 0, 1, 3, 1, 2, 3, 3, 2, 0, 2, 1, 0

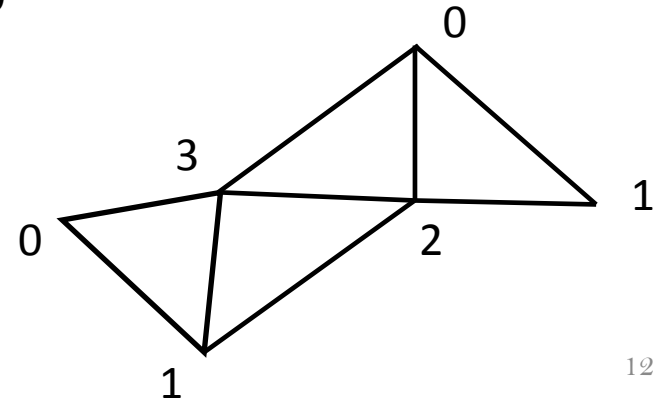


Position: (0,0,-1),  
Farbe: (1,0,0,1)

Position: (1,0,-2),  
Farbe: (1,1,0,1)

Position: (0,1,-2),  
Farbe: (0,0,1,1)

Position: (0,0,-2),  
Farbe: (0,1,0,1)



# Aufgabe 5a (3 2 Punkte)

- Erweitern Sie Abbildung 1 um diejenigen Vektoren, die zur Berechnung des diffusen Anteils des Phong Beleuchtungsmodells benötigt werden. Markieren Sie außerdem den Winkel, der letztendlich für die Stärke der diffusen Beleuchtung verantwortlich ist.

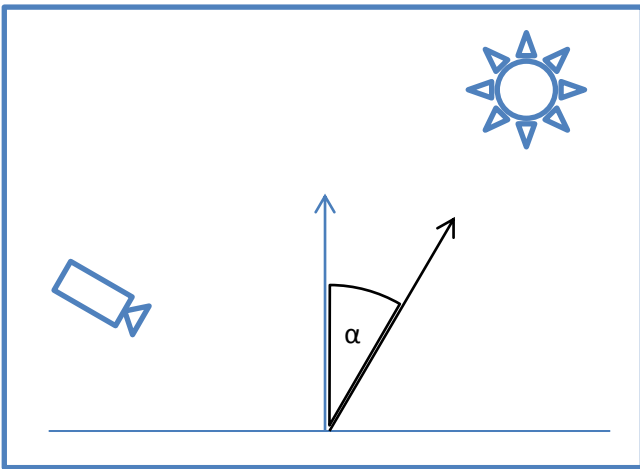


Abbildung 1

## Aufgabe 5b (4 3 Punkte)

- Erweitern Sie Abbildung 2 um diejenigen Vektoren, die zur Berechnung des spekularen Anteils des Phong Beleuchtungsmodells benötigt werden. Markieren Sie außerdem den Winkel, der letztendlich für die Stärke der spekularen Beleuchtung verantwortlich ist.

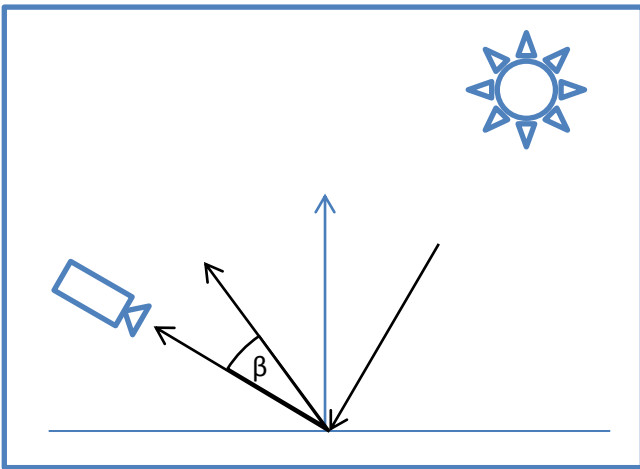


Abbildung 2

# Aufgabe 5c (5 7 Punkte)

- Erweitern Sie die GLSL Methode `vec4 Enlight(vec3 position, vec3 normal)`, die sowohl den diffusen als auch den spekularen Beleuchtungsanteil (*des Phong Modells*) berechnet und deren Summe zurückgibt. Die Entfernung und Art der Lichtquelle sollen dabei unberücksichtigt bleiben

```
uniform vec3 lightPosition;
uniform vec3 eyePosition;
uniform vec3 I_d, I_s, c_d, c_s, k_d, k_s;
uniform float es;

vec4 Enlight(vec3 position, vec3 normal){ // normal sei normiert
    vec3 world2light = normalize(lightPosition - position);
    float diffuse = max(0, dot(world2light, normal));

    vec3 light2world = -world2light;
    vec3 reflected = reflect(light2world, normal);
    vec3 world2eye = normalize(eyePosition - position);
    float specular = max(0, pow( dot(reflected, world2eye), es) );

    return I_d * c_d * k_d * diffuse + I_s * c_s * k_s * specular;
}
```

# Aufgabe 6 (3+3+2+2 Punkte)

---

- Wir betrachten ein bestimmtes Pixel. Folgender Ablauf von Ereignissen findet statt. Ergänzen Sie in den jeweiligen Kästen die aktuellen Pixeleigenschaften.
- *Zusätzlicher Hinweis: Verknüpfung sei “+”*  
(**glBlendEquation** (GL\_FUNC\_ADD) ;)



# Aufgabe 6 (3+3+2+2 Punkte)

---

1. Aufruf: `glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA); // SRC / DEST`
2. Aufruf: `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`  
`// z wird auf 1 und color auf (0, 0, 0, 0) gesetzt.`
3. Aufruf: `glEnable(GL_BLEND);`
4. Aufruf: `glEnable(GL_DEPTH_TEST);`
5. Drawaufruf erzeugt Fragment ( $z = 0.6$ ,  $\text{color} = (1, 1, 0, 0.5)$ )
  - Aktuelle Pixeleigenschaften ( $z$ ,  $\text{color}$ ):
  - $z = 0.6$
  - $\text{color} = (1, 1, 0, 0.5)$
6. Aufruf: `glDisable(GL_DEPTH_TEST);`
7. Drawaufruf erzeugt Fragment ( $z = 0.7$ ,  $\text{color} = (0, 1, 1, 0.5)$ )
  - Aktuelle Pixeleigenschaften ( $z$ ,  $\text{color}$ ):
  - $z = 0.7$
  - $\text{color} = (0, 1, 1, 0.5) + 0.5 * (1, 1, 0, 0.5) = (0.5, 1.5, 1, 0.75)$
8. Aufruf: `glEnable(GL_DEPTH_TEST);`
9. Aufruf: `glDisable(GL_BLEND);`
10. Drawaufruf erzeugt Fragment ( $z = 0.1$ ,  $\text{color} = (0.5, 0.5, 1, 0.2)$ )
  - Aktuelle Pixeleigenschaften ( $z$ ,  $\text{color}$ ):
  - $z = 0.1$ ,
  - $\text{color} = (0.5, 0.5, 1, 0.2)$
11. Drawaufruf erzeugt Fragment ( $z = 0.2$ ,  $\text{color} = (0.75, 0.15, 0, 0.3)$ )
  - $z = 0.1$ ,  $\text{color} = (0.5, 0.5, 1, 0.2)$

# Aufgabe 7 (3+3+3 Punkte)

---

- Wir sind uns in einem Programm nicht sicher, ob die Normalen einer Geometrie korrekt berechnet wurden. Um dies zu prüfen, würden wir gerne die Fragments entsprechend ihrer Normale einfärben. Implementieren Sie dazu den Vertex- und den Fragmentshader. **Achtung:** Die Einträge des Normalenvektors können negativ sein. Sorgen Sie für eine entsprechende Normierung.

# Aufgabe 7 (3+3+3 Punkte)

VS

```
uniform mat4 viewProjection, model;

uniform mat3 model_Normals;
in vec3 positionMC;
in vec3 normalMC;

out vec3 normalWC;

void main(){ // Klar, was im VS geschehen muss?
    gl_Position = viewProjection * model * vec4(positionMC, 1);
    normalWC = model_Normals * normalMC;
}
```

FS

```
in vec3 normalWC;
out vec4 fragColor;

void main(){
    fragColor = vec4(vec3(0.5) + 0.5 * normalize(normalWC), 1);
}
```

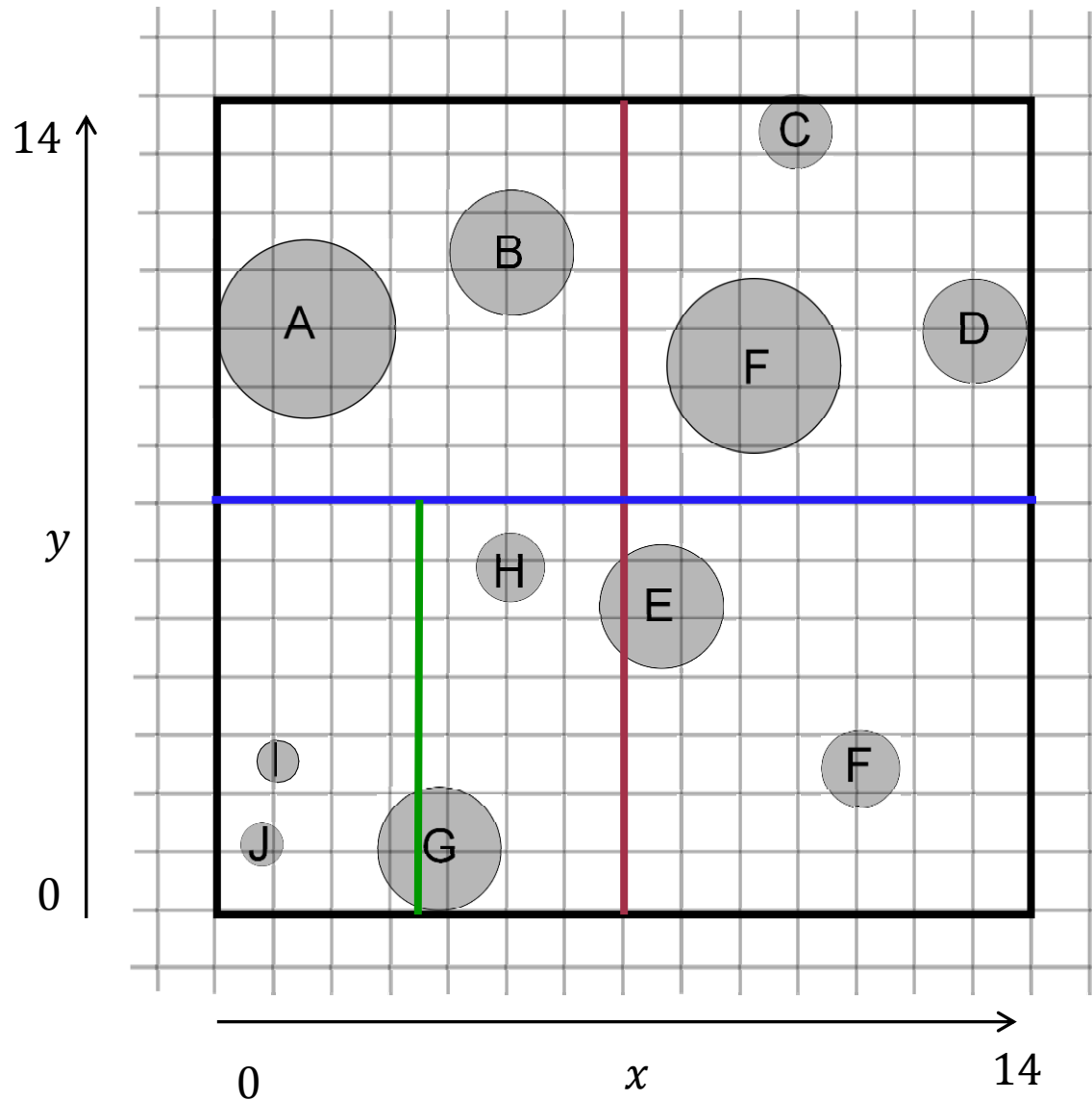
- Die Farben entsprechen der Richtung der Normalen. Sehen wir bspw. ein blaues Fragment, heißt das, dass die Normale genau in z-Richtung zeigt, also (0,0,1). Ein schwarzes Fragment würde aufgrund der Normierung einer Normale von (-1,-1,-1) entsprechen.

# Aufgabe 8 (5 Punkte)

---

- Zeichnen Sie in das Diagramm die Struktur eines darauf aufbauenden zweidimensionalen KD-Trees ein. Dabei sollen folgende Regeln befolgt werden.
- Maximale Tiefe: 4
- Angestrebte Anzahl der Elemente pro Node:  $\approx 3$
- ~~➤ Teilung in jedem Schritt wechselnd zwischen Parallele zur x- bzw. y-Achse~~
- ~~➤ Mit welcher Achse beginnen Sie und warum?~~
- *Teilen Sie, wenn Teilung nötig, den Raum senkrecht zur Richtung der jeweils längsten Achse mittig*
- *Falls nicht eindeutig, wählen Sie die x-Achse*

# Aufgabe 8 (5 Punkte)



— Tiefe: 1

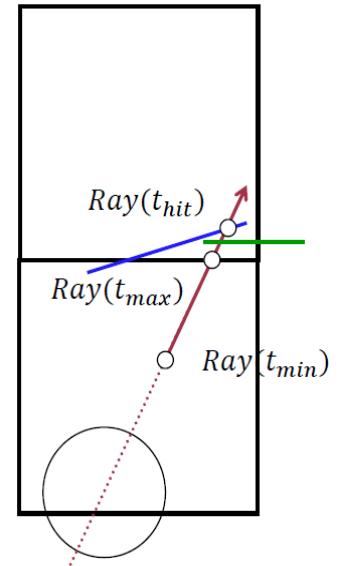
— Tiefe: 2

— Tiefe: 3

Maximale Tiefe: 4  
Angestrebte Anzahl der  
Elemente pro Node: 3

# Aufgabe 9 (6 Punkte)

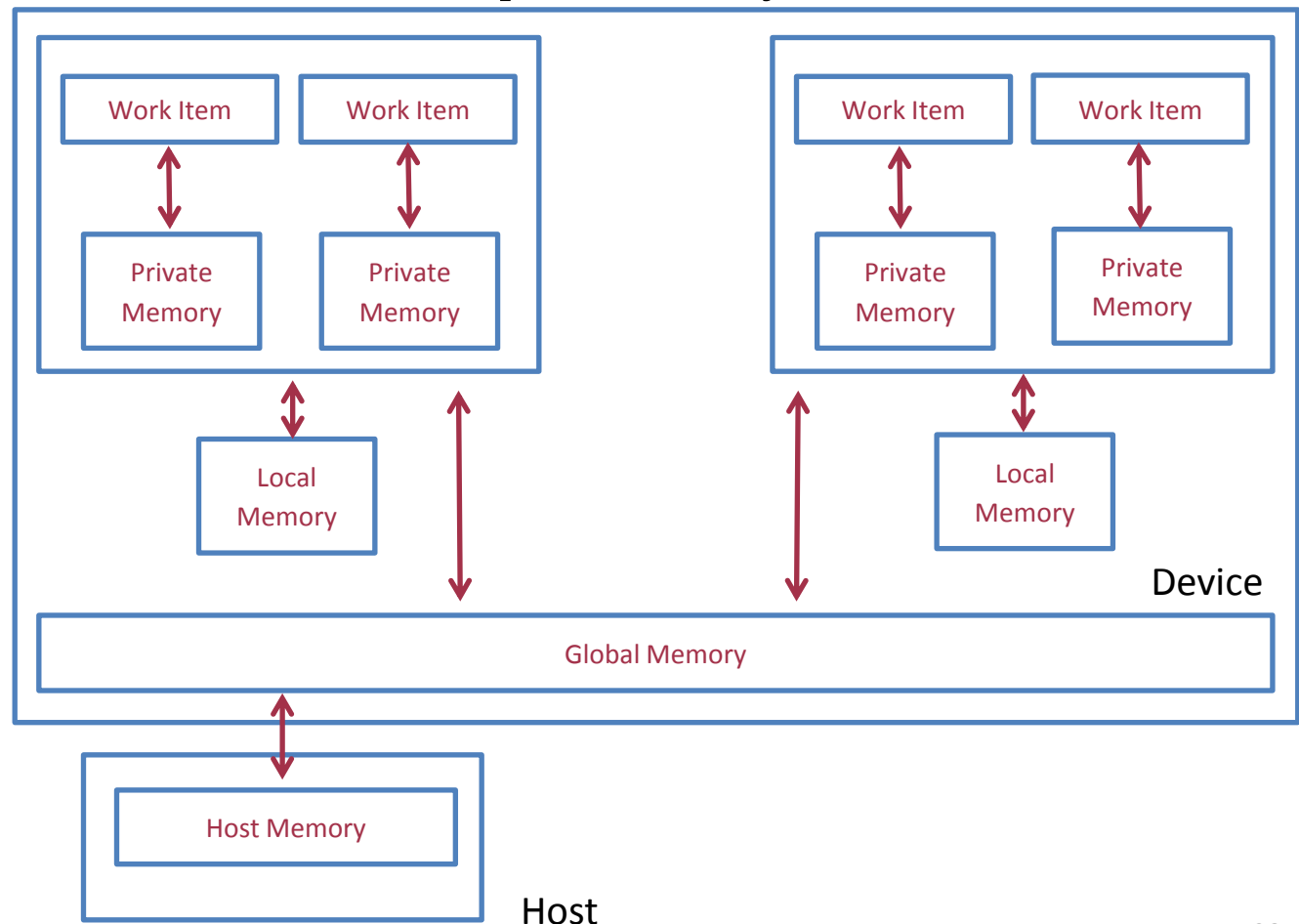
- Warum muss der Parameter eines Sichtstrahls in einem KD-Raytracer nach vorne ( $t_{min}$ ) und hinten ( $t_{max}$ ) beschränkt sein? Begründen Sie dies mithilfe einer Zeichnung.
- Die Begrenzung nach vorne bewirkt, dass keine Objekte hinter der Kamera berücksichtigt werden.
- Die Begrenzung nach hinten ist für das Traversieren durch den KD-Tree wichtig.
- Falls wir einen Treffer in der Near-Node bekommen, der Schnittpunkt jedoch weiter entfernt ist als  $t_{max}$ , liegt dieser Treffer in einer anderen Node. (Siehe Zeichnung blaue Linie). Weiterhin könnte es sein, dass ein weiteres Objekt in der Far-Node ist, welches jedoch näher an der Kamera liegt (Siehe Bild, grüne Linie). Dieses kann nicht gefunden werden, falls es nicht auch in der aktuell verarbeiteten Node enthalten ist. Da nach vollständigem Durchsuchen der aktuellen Node im Falle eines Treffers die Suche abgebrochen wird, würde das grüne Objekt in der Zeichnung nicht gefunden, obwohl es näher liegt als das Blaue.



# Aufgabe 10 (6 Punkte)

- Vervollständigen Sie das unten stehende Diagramm dahingehend, dass es das Speichermodell von OpenCL widerspiegelt. Verwenden Sie Pfeile um das Lesen und Schreiben der verschiedenen Speicher zu symbolisieren und folgende Begriffe.

- Host
- Device
- Work Item
- Work Group
- Global Memory
- Local Memory
- Private Memory
- Host Memory



# Aufgabe 11 (5 Punkte)

---

- Erläutern Sie die Möglichkeiten, in OpenCL zu synchronisieren.
  - Synchronisation nur innerhalb einer Workgroup möglich
  - Synchronisiert werden Speicherzugriffe auf lokalen `barrier(CLK_LOCAL_MEM_FENCE)` und/oder globalen `barrier(CLK_GLOBAL_MEM_FENCE)` Speicher
  - Globale Synchronisation ist innerhalb eines Kernels nicht möglich, sondern nur durch den Host



# Aufgabe 12 (10 Punkte)

---

```
kernel void Riddle(global float* h, local float* j) {  
    uint id = get_global_id(0);  
    j[get_local_id(0)] = h[id];  
    barrier(CLK_LOCAL_MEM_FENCE);  
    h[id] = j[get_local_size(0) - get_local_id(0) - 1];  
}
```

- *Zusatz: Wenden Sie den Kernel auf das Array {0, 1, 2, 3, 4, 5, 6, 7} mit Global\_Size 8 und Local\_Size 4 an:*
  - {3, 2, 1, 0, 7, 6, 5, 4}
- Erläutern Sie die Funktion des folgenden Kernels und begründen Sie den Synchronisationspunkt.
  - Kehrt innerhalb einer Work Group die Reihenfolge der Elemente um
  - Synchronisationspunkt stellt das Laden der unvertauschten Elemente sicher

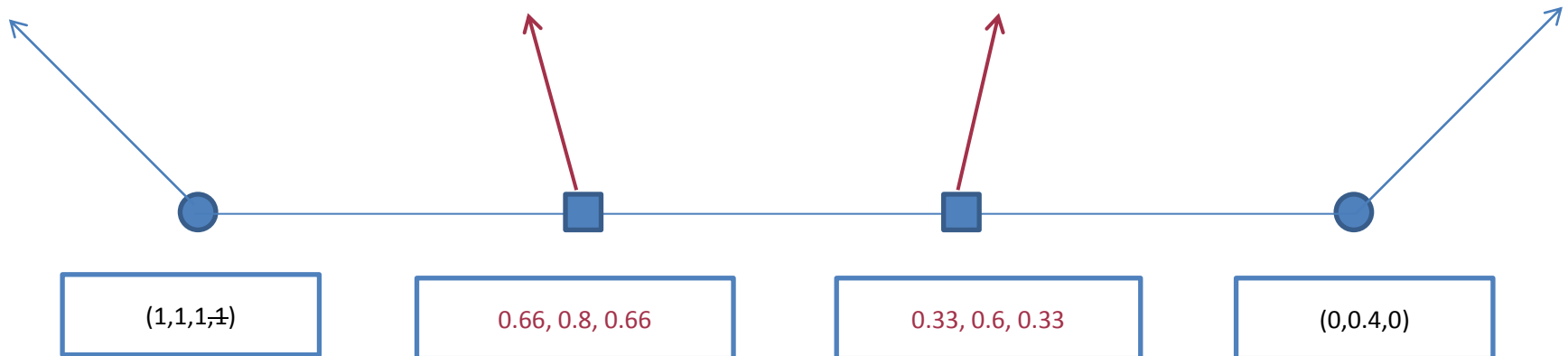
2

---

Alternative Aufgaben (Keine vollständige Klausur)



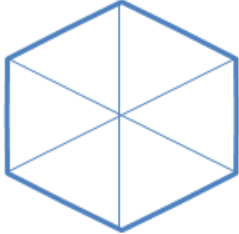
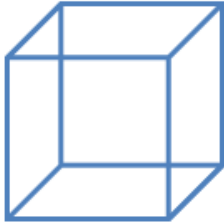

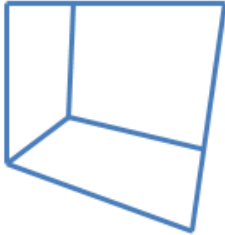
# Alternativaufgabe 1 (8 Punkte)

- In der Zeichnung unten sind zwei Vertices (●) jeweils mit einer Normalen und einer Farbe angegeben.
- Der Rasterizer erzeugt die beiden Fragments (■).
- Zeichnen Sie an diesen Stellen jeweils die interpolierte Normale ein und berechnen Sie die interpolierte Farbe.



# Alternativaufgabe 1 (6 Punkte)

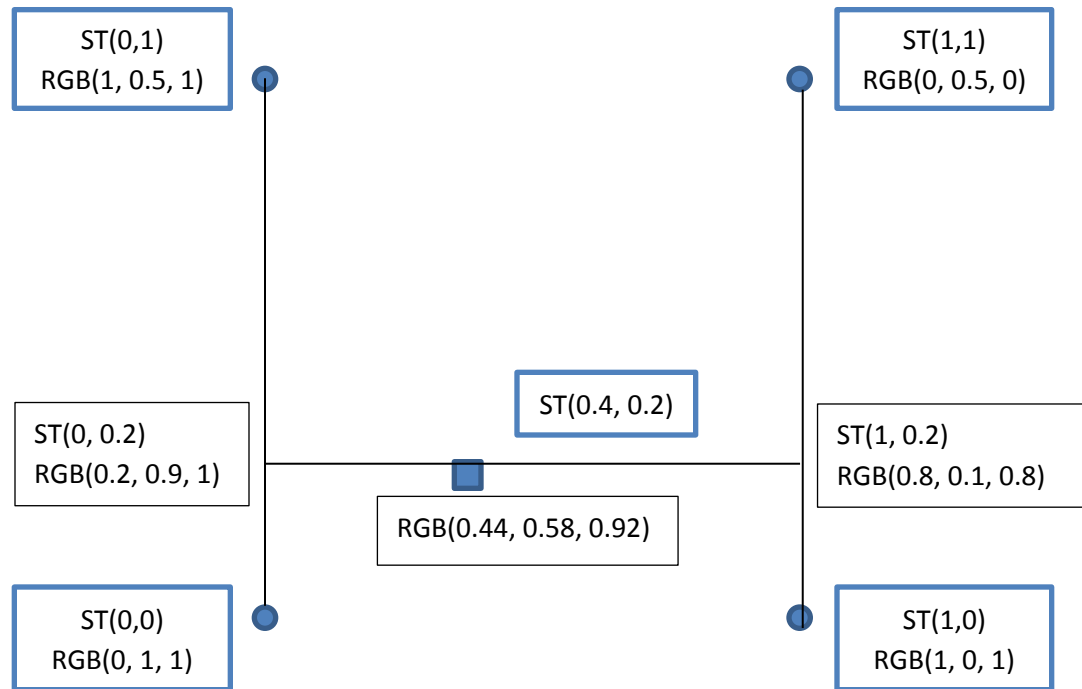
- Gegeben sei ein perfekter Würfel. Ordnen sie seinen Darstellungen als Drahtgitter unten jeweils die Art der Projektion zu. **Hinweis:** Genau eine Antwort ist jeweils korrekt.

 <div data-bbox="440 786 774 891"> Eindeutig orthogonal <input checked="" type="radio"/>  Eindeutig perspektivisch <input type="radio"/>  Nicht möglich <input type="radio"/> </div>	 <div data-bbox="813 786 1147 891"> Eindeutig orthogonal <input type="radio"/>  Eindeutig perspektivisch <input checked="" type="radio"/>  Nicht möglich <input type="radio"/> </div>	 <div data-bbox="1180 786 1514 891"> Eindeutig orthogonal <input checked="" type="radio"/>  Eindeutig perspektivisch <input type="radio"/>  Nicht möglich <input type="radio"/> </div>
 <div data-bbox="440 1296 774 1400"> Eindeutig orthogonal <input checked="" type="radio"/>  Eindeutig perspektivisch <input type="radio"/>  Nicht möglich <input type="radio"/> </div>	 <div data-bbox="813 1296 1147 1400"> Eindeutig orthogonal <input checked="" type="radio"/>  Eindeutig perspektivisch <input type="radio"/>  Nicht möglich <input type="radio"/> </div>	 <div data-bbox="1180 1296 1514 1400"> Eindeutig orthogonal <input type="radio"/>  Eindeutig perspektivisch <input checked="" type="radio"/>  Nicht möglich <input type="radio"/> </div>

# Alternativaufgabe 3 (10 Punkte)

- Führen Sie bilineares Filtern aus. ST gibt dabei die Koordinaten der Texel an und RGB ihre Farbe. Gesucht ist die Farbe des Quadrats. Zeichnen Sie die Linien ein, auf denen Sie linear interpolieren, sowie die interpolierten Werte, um den Farbwert des Quadrats zu erreichen. *Hinweis: Interpolieren Sie in t-Richtung zuerst*

$$\text{Lineare Interpolation: } C(t) = (1 - t) \cdot A + t \cdot B$$



# Alternativaufgabe 4 (16 Punkte)

FS

```
1 uniform sampler2D normalTex;
2 uniform sampler2D heightTex;
3
4 in vec2 texCoord;           // Texturkoordinaten des Vertex
5 in vec3 positionWC;         // Position des Vertex in der welt
6 out vec4 fragColor;
7
8 void main(void)
9 {
10  vec3 sampledNormal = texture(normalTex, texCoord).rgb;
11  // ob8: texturwerte zwischen 0 und 1
12  vec3 normal = normalize(2.0 * sampledNormal - vec3(1.0));
13
14  vec3 position = positionWC + normal * texture(heightTex, texCoord).r;
15
16  fragColor = doPhongLighting(position, normal);
17 }
```

Hinweis:  
Zu schwer,  
kommt so nicht vor

- Was steht in sampledNormal aus Zeile 10?
  - Der Farbwert der Textur normalTex an der Stelle der Fragment Texturkoordinaten.
- Erklären Sie stichpunktartig die Zeilen 10 und 12. Was steht in normal?
  - Zeile 10 interpretiert die Werte der Textur als Normale und Zeile 12 normiert die Werte so, dass sie im Bereich von den Werten von Normalenvektoren liegen. Anschließend werden sie normiert, sodass wir tatsächlich eine Normale erhalten

# Alternativaufgabe 4 (16 Punkte)

FS

```
1 uniform sampler2D normalTex;
2 uniform sampler2D heightTex;
3
4 in vec2 texCoord;           // Texturkoordinaten des Vertex
5 in vec3 positionWC;         // Position des Vertex in der welt
6 out vec4 fragColor;
7
8 void main(void)
9 {
10  vec3 sampledNormal = texture(normalTex, texCoord).rgb;
11  // ob8: texturwerte zwischen 0 und 1
12  vec3 normal = normalize(2.0 * sampledNormal - vec3(1.0));
13
14  vec3 position = positionWC + normal * texture(heightTex, texCoord).r;
15
16  fragColor = doPhongLighting(position, normal);
17 }
```

Hinweis:  
Zu schwer,  
kommt so nicht vor

- Erklären Sie stichpunktartig Zeile 14. Was steht in position?
- Hier wird ein Kanal aus der Textur heightTex gelesen und anhand dieses Wertes die Weltposition des Fragments entlang der Normalen verschoben. Ähnlich dem Displacementmapping der Erde für eine Hausaufgabe. Nur passiert es diesmal auf einer PerFragment Basis und nicht PerVertex.

# Alternativaufgabe 4 (16 Punkte)

FS

```
1 uniform sampler2D normalTex;
2 uniform sampler2D heightTex;
3
4 in vec2 texCoord;           // Texturkoordinaten des Vertex
5 in vec3 positionWC;         // Position des Vertex in der welt
6 out vec4 fragColor;
7
8 void main(void)
9 {
10  vec3 sampledNormal = texture(normalTex, texCoord).rgb;
11  // ob8: texturwerte zwischen 0 und 1
12  vec3 normal = normalize(2.0 * sampledNormal - vec3(1.0));
13
14  vec3 position = positionWC + normal * texture(heightTex, texCoord).r;
15
16  fragColor = doPhongLighting(position, normal);
17 }
```

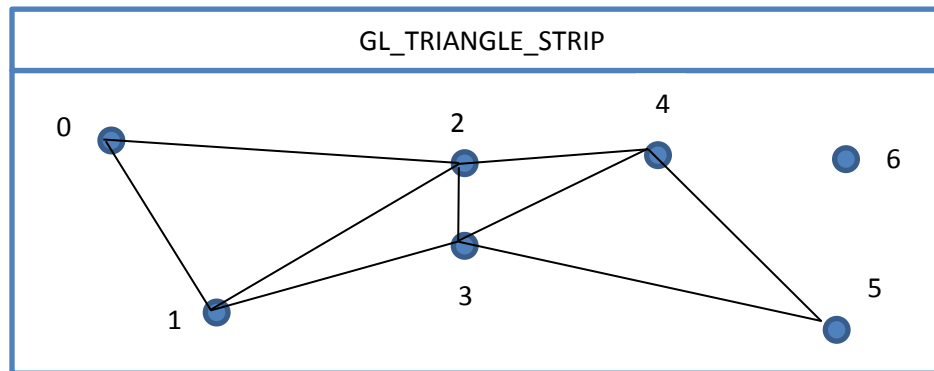
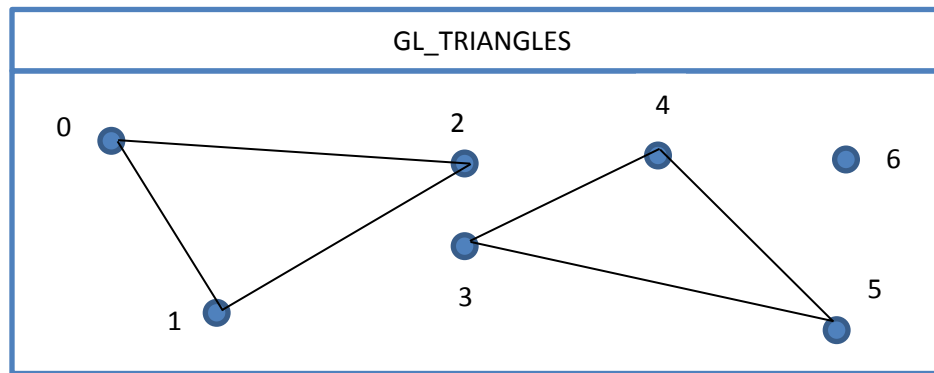
Hinweis:  
Zu schwer,  
kommt so nicht vor

- Fassen Sie ihre Antworten zusammen und beschreiben Sie stichpunktartig den Sinn des gesamten Shaders. Was steht in fragColor?
- In der letzten Zeile wird das Fragment mit dem Phongbeleuchtungsmodell eingefärbt. Als Basis werden dabei die Position und Normale benutzt, die aus den Texturen konstruiert wurden. Diese Techniken nennen sich Normalmapping und Bumpmapping (diese Info ist für die volle Punktzahl nicht nötig).



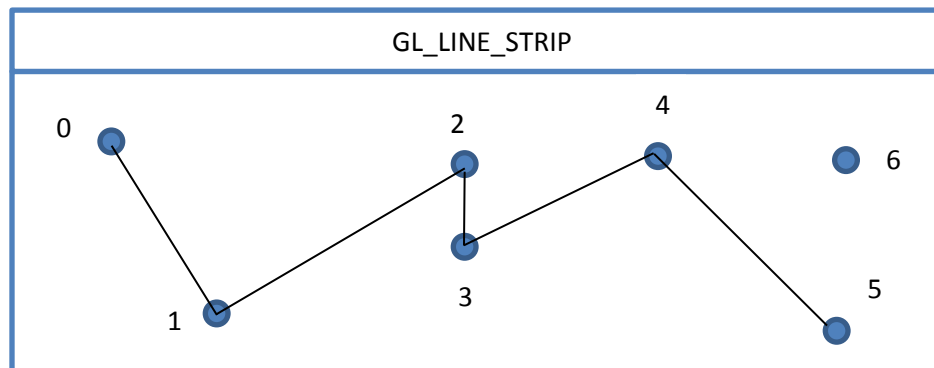
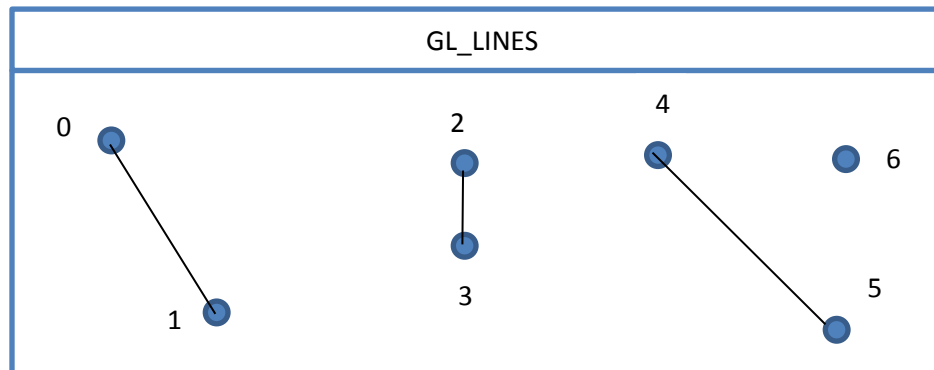
# Alternativaufgabe 5 (12 Punkte)

- Gegeben sei der Indexbuffer  $I = \{0, 1, 2, 3, 4, 5\}$ . Zeichnen Sie die entstehende Geometrie zu den jeweiligen Topologien in die zugehörigen Skizzen. Schraffieren Sie dabei entstehende Dreiecke.



# Alternativaufgabe 5 (12 Punkte)

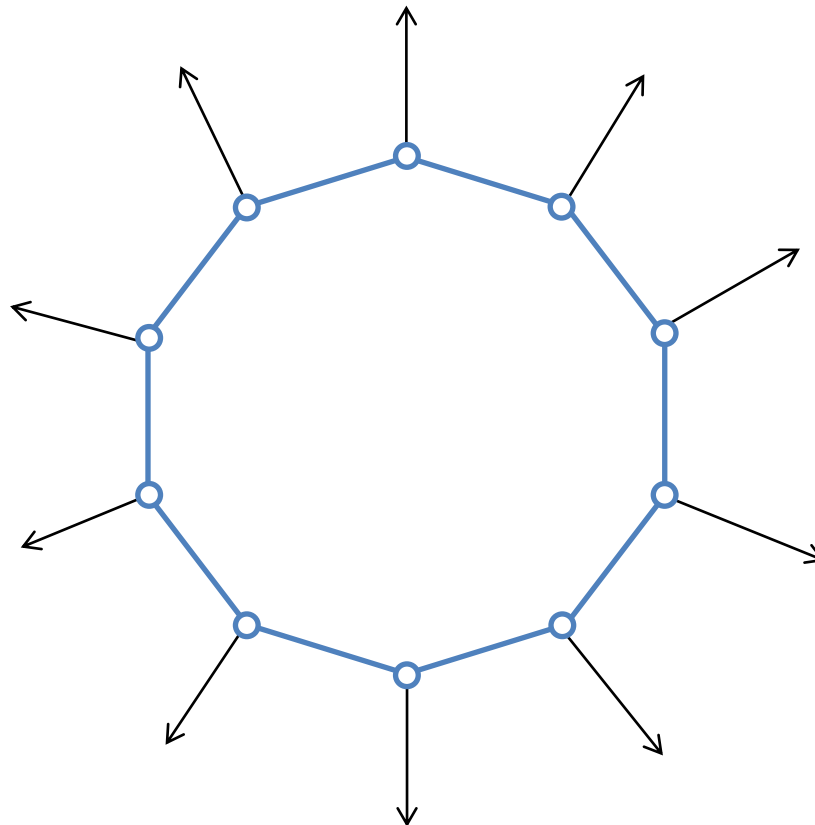
- Gegeben sei der Indexbuffer  $I = \{0, 1, 2, 3, 4, 5\}$ . Zeichnen Sie die entstehende Geometrie zu den jeweiligen Topologien in die zugehörigen Skizzen. Schraffieren Sie dabei entstehende Dreiecke.



# Zusatzaufgabe 6 (2 Punkte)

---

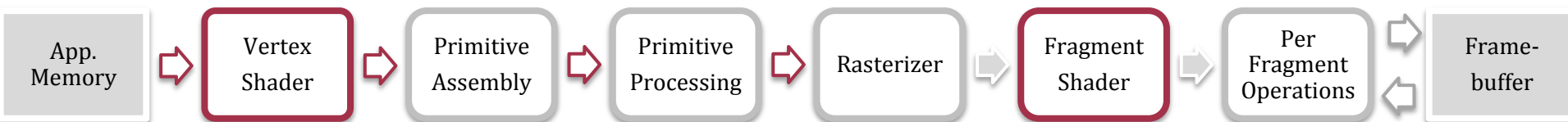
- Wir möchten einen Kreis mittels Rastergrafik darstellen. Dafür nähern wir die Geometrie durch 10 Vertices an.
- Zeichnen Sie an jedem Vertex seine zugehörige Normale ein, die der angestrebten Geometrie am ehesten entspricht.



# Zusatzaufgabe 7 (3 Punkte)

➤ Zeichnen Sie folgende Begriffe in die OpenGL Graphics Pipeline ein.

- Rasterizer
- Fragment Shader
- Vertex Shader
- Primitive Assembly
- Per Fragment Operations
- Primitive Processing





Legende


 Vertices

 Fragments

 Pixel Data

 Programmable Stage

 Fixed Stage

 Memory

3

---

Ansagen

# Was in der Klausur anders wird

---

- Eher etwas einfacher
- Kleine Formelsammlung
  - Trigonometrische Funktionen
  - Speziellere Funktionen wie reflect
- Keine Aufgaben mit aufeinander aufbauenden Teilaufgaben über 10 Punkte
- 3D-Zeichnungen / Indizierungen (Tetraederaufgabe)?
- Kleine Entschuldigung: Umfrage in Stud.IP, welche Aufgabe am wenigsten gefallen hat. "Sieger" kommt dann in der Klausur nicht vor